Developing with Concordance®

Concordance Developer's Guide

Concordance, version 10.20

- **About Concordance Programming Language (CPL)**
- **CPL Library**
- **Getting Started**
- **Concordance Programming Fundamentals**
- **Concordance Programming Language Reference**

Developing with Concordance

Concordance®, version 10.20
Concordance® Native Viewer, version 1.08
Concordance® Image, version 5.15
Concordance® FYI™ Server, version 5.13
Concordance® FYI™ Reviewer, version 5.16

Release Date: June 9, 2014

Table of Contents

Concordance® Programming Language (CPL)

User Guide

Developing with Concordance

Chapter

1

Developing with Concordance

About Concordance Programming Language (CPL)

The Concordance Programming Language (CPL) is a proprietary structured script programming language providing what programmers call extensibility: The ability of a program to stretch beyond its original capabilities. For example, let's say a paralegal has mistakenly entered the beginning and ending Bates numbers in the same field. Now he needs to get the ending number out and into a separate ENDBATES field. Or a litigation support analyst wants her search results to automatically include all cross-referenced documents as well. A CPL can be used to handle both situations.

Concordance ships with a number of CPLs for use with Concordance. For example, the CreateHyperlinks CPL creates an attachment or hyperlink to an external document so that search results automatically include all cross-referenced documents.

Concordance Scripts

A Concordance script is defined as code written in the Concordance Programming Language (CPL), stored in an ASCII-compatible plain text file, and saved with the .cpl extension. You can create, open, and modify a Concordance application using an ASCII-compatible text editor, or else through the Concordance UI.

While there are many scripts you can find, Concordance Technical Support supports only those that are shipped with the product and updated for a particular version. In the sample directory, CPL scripts are named with Concordance version release number extensions, indicating the software version in which they are they are supported.

Example: BlankField_v10.00.cpl

This example indicates that it is supported for Concordance version 10.00.

Unsupported scripts include the following:

- Those written for previous versions of Concordance.
- Those written by vendors or staff at other organizations who have taken a class in writing scripts for their own use.

For a complete list of Concordance version 10.*x* scripts, usage, and instructions for executing, see About CPL Scripts.

CPL Library

Concordance includes a number of sample CPL scripts in the installation. By default, the sample scripts are located in the following directory C:\Documents and Settings\All Users\Application Data\LexisNexis\Concordance 10\CPL (Windows XP) or C:\ProgramData\LexisNexis \Concordance\CPL (Windows 7).

If you do not have access to the Sample CPLs in the Concordance install directory, the CPLs are available for download by version:

- Concordance version 8.x

- Concordance version 9.x
- Concordance version 10.x

Once you have located a sample script, you can run it as you would any other script, see Running a Concordance Script. Before running a sample script on your live database, it is recommended you practice running the script on a sample database until you become familiar with how the sample works.

For more information about the sample CPL scripts, see About CPL Sample Scripts.

Getting Started

Where to Start

The following topics are a quick summary and introduction to working with and developing CPLs and associated resources.

Topic	Description
How the Concordance Development Documentation is Organized	Brief overview of the Concordance documentation, and where you should start reading.
What You Need to Know to Develop with Concordance	A small list of the subjects you should be familiar with before developing an application with Concordance.
What is the Concordance Programming Language	Brief overview of the Concordance Programming Language (CPL).
About the CPL Development Environment	A discussion of the programming environment, how to create and modify an application, and where the scripts are located.
Tutorial: Hello World! in CPL	Simple tutorial that demonstrated the basic features of the CPL.

How the Concordance Development Documentation is Organized

The Concordance development documentation is organized into the following sections:

- Getting Started

  A description of the development environment, what you need to know about developing CPLs and a quick-start tutorial. If you are completely new to developing with Concordance, these topics are designed to help you get started.

- Concordance Programming Fundamentals

An introduction to the basic programming techniques used to access Concordance: general programming structures, how to declare and use variables, creating functions, accessing the database, and using common CPL functions. These topics are written for the novice user who is looking to learn to program in CPL.

- Concordance Programming Language Reference

  A full description of the Concordance programming language. Also includes an alphabetical API reference list. These topics are designed for an experienced user who is looking for a specific method or answer to a technical programming question.

What You Need to Know to Develop with Concordance

In order to develop with Concordance, you may need to understand two technologies: Concordance itself, and general programming or scripting.

- **Concordance**

  The main features of Concordance Programming Language (CPL) are designed to automate Concordance tasks. Therefore, you should be familiar with Concordance features, such as tags, comments, and querying. You should also be familiar with the Concordance UI. In general, it is recommended that you receive the general training on using Concordance before you attempt to develop a CPL script.

- **C/Pascal Development**

  CPL is a scripting language in the C/Pascal family. While the Concordance Programming Fundamentals section can walk you through many of the basics of writing a script, it is recommended that you have at least a basic familiarity with scripting before you begin. Alternately, having a basic background in C or C++ would also be useful.

What is the Concordance Programming Language

The Concordance Programming Language (CPL) is a proprietary structured language designed to help you with your administrative tasks using pre-coded scripts to run the processes for you.

The CPL features promote clear and concise program design through the use of modularity, local and global variables, value returning functions, and predictable program flow. The CPL is written in a language similar to C/C++ in .CPL files. A CPL program consists of a sequence of instructions that tells Concordance how to solve a particular problem. The program will usually contain some amount of data. In CPL, the data can be database fields, text, characters, numbers, and dates. Program instructions are organized by functions. CPL function declarations establish the name of the function, values it will receive when it begins, and the local variables it will use while it is running.

The following example describes the structure of a typical CPL program.

```
int color;

   main()
   {
```

```
    char string[20];
    color = 15;
    string = "Hello, world";

    show(10, 20, string);
    return;
}

show(int row, column; char xyz[])
{
    puts(row, column, xyz, color);
}
```

The example program begins with the main() function. main() assigns a value to the global variable color, and to a local variable string. The application then calls the show() function to put the string on the screen. For a simplified tutorial of this code sample, see Tutorial: "Hello World!" in CPL. For more information on using CPL, see Concordance Programming Fundamentals. For the full definition of CPL, see Concordance Programming Language Reference.

About the CPL Development Environment

The Concordance development environment contains three main features you can use to develop a script:

- **Programming Tools** - to develop an application using CPL, Concordance includes a simple script editing and runtime environment. For more information, see Creating and Editing a Concordance Script.
- **The Concordance API** - in addition to the CPL language itself, Concordance has a number of additional, built-in functions you can use to access the database, sort queries, and more. For more information, see About Common CPL Functions , About the Advanced Programming Features, and the About CPL Functions topic in the CPL Reference.
- **Concordance CPL Scripts** - Finally, Concordance ships with a number of scripts. These scripts extend the capabilities of Concordance in a number of ways, including field formatting, database administration, and native image viewing. For more information, see About CPL Scripts.

Tutorial: "Hello World!" in CPL

The following topic describes the basic process for creating and running an application using the Concordance Programming Language (CPL.)

To create and run an application using the CPL

1. Open Concordance.
2. Select **Edit program** from the **File** menu.

3. Type *hello.cpl* as your file name.

   CPL is used as the file extension for CPLs. Note that a .CPT is created automatically after the first time you run your program. For more information about CPL files, see Creating and Editing a Concordance Application.

   An edit screen appears where you can type the following program.

4. Enter the following code into the edit screen:

```
main()
{
      puts(5,5,"Hello World");
      getkey();
}
```

   For more information about writing code, see About Concordance Programming Fundamentals.

5. Save your work, select **File**, and then and then select **Exit.**

6. On the **File** menu, select **Begin Program**.

7. Locate the ***hello.cpl*** and then click **Open**.

   Your program should run and you should see the text "Hello World" printed on the screen. Hit any key to return to Concordance. For more information on running an application, see Running a Concordance Application.

Concordance Programming Fundamentals

About Concordance Programming Fundamentals

The following topics cover the fundamentals of Concordance script development:

Topic	Description
Creating and Editing a Concordance Script	How to create scripts and use the samples.
About Variable Types	How to declare and use a variable.
About CPL Functions	How to declare a function, use variables, and return a value.
About Conditional Statements	How to use conditional operators, compound if-statements, switch statements, and loops.
About the Database	How to use a database handle, field information and loops.

About Common CPL Functions	How to use basic text manipulation, database searches and UI features.
About the Advanced Programming Features	Lists the functions in the CPL by feature area.

Working with the Concordance Development Environment

Creating and Editing a Concordance Script

To create or edit a .CPL file in Concordance

1. Open Concordance.
2. From the **File** menu, select **Edit Program.**
3. In the **Open** dialog, select the .CPL file you wish to edit, and click **OK**.

   Alternately, type the name of the .CPL file you wish to create, and click **OK**.
4. In the **CPL Editor**, write or modify your code.

   The CPL Editor is a simple text editor with search capabilities.

⚠ If you have created a script that calls another script, you will need to update the script name being called with the version number appended to the script name. For example, if your script uses Mark.cpl, you will need to update your script by changing Mark.cpl to Mark_v10.00.cpl in Concordance version 10.00.

Running a Concordance Application

Once you have created a Concordance .CPL file, you can run your script from within Concordance, or else run the optimized .CPT file from the command line.

When Concordance has finished running a program for the first time, it saves an optimized version of the file as a .CPT file. The next time you run the program, Concordance will run the .CPT file if it has a date and time later than the CPL file. A .CPT file runs faster than a CPL file, and can be run from the command line like a CPL file. However, they cannot be edited or modified with a text editor.

Note that scripts work on your current active query. Therefore, be sure to set up searches accordingly. Alternately, you can resort your database records by going to the **Standard** toolbar, and clicking the **All** button.

To execute a CPL script from within Concordance

1. Open Concordance.
2. If necessary, open the database and query you wish to execute the application on.

Unless specified otherwise using code, Concordance uses the current query as the target for a CPL application. For information on how to programmatically access multiple databases, see Opening and Closing a Database.

3. From the **File** menu, select **Begin Program**.
4. In the **Open** dialog, select the CPL you wish to run, and then select **OK**.

To execute a CPT file from the command line

1. Confirm that you have run the .CPL script at least once from within the Concordance UI.

   As stated above, you must first run the script, after which Concordance creates the .CPT file. The .CPT file allows you to run the script from the command line.

2. On the command line, type the path of Concordance as you would normally, and then include the full path and name of the .CPT file.

Declaring and using a Variable

About Variable Types

Before you declare a variable, you must determine what kind of data you wish to store in that variable. As with most programming languages, a basic CPL variable can contain either some kind of number, or else some kind of text. The following table describes the types of variables supported by CPL.

Type	Description
int	An integer value. The range of values it can store is from -2,147,483,648 to 2,147,483,647.
float	a floating point integer. Unlike an int, a float can store numbers with decimals. It can store a range of values from -2.2E-308 to 1.7E+308.
char	An ASCII character value from the range of -128 to 127.
short	A number value from -32,768 to 32,767.
text	Any text string including database field contents.

Once you have determined what kind of variable you want, you can create the variable by declaring it. For more information, see Declaring a Variable. For more information on variable types, see Data Types in the CPL Language Reference.

Declaring a Variable

The first step to using a variable in a CPL program is declaring the variable. Declaring a variable

is simply a way of informing the computer that you would like to use a variable with a specific name, and that you will be storing a specific type of data in that variable.

To declare a variable

1. Determine which part of your application needs to use the variable.

   Where you declare a variable is important to where you use it. This is known as the **scope** of the variable.

   - **Global scope** - any variable declared outside the main() function. Global variables can be accessed by any part of your application, although they cannot be accessed by other CPL's.
   - **local scope** - any variable declared within the beginning and ending {} of a function. These variables can be accessed only by code within that function.

   For more information, see Variable Declaration and Scope in the CPL language reference.

2. In your programming file, enter a variable type, and then follow the type with the variable name you wish to use.

   Variable types are described in About Variable Types. When naming your variables, be sure to conform to the following rules:

   - A variable can only contain letters, numbers and the underscore character.
   - A variable must begin with a letter.
   - A variable cannot contain any punctuation or spacing.

3. To declare multiple values of the same type on the same line, separate the variables by a comma.

   The following example show different types of variable declaration.

```
int x;
int y;
float myFloat;
text myString;
char myChar;
short a, b;
```

Once you have declared a variable, you can assign a value to that variable. For more information, see Assigning a Variable. You can also declare a type of variable called an **array** that can store multiple values of the same type. For more information, see Creating and using an Array.

Assigning a Variable

Once you have declared a variable, you may assign that variable a value.

To assign a value to a variable

1. Use the equals sign (=) to assign a value, as you would in mathematics.

   Be sure that the type of data you are attempting to assign to a variable matches up with the variable type. for example, you should not attempt to assign an integer to a variable that was declared as text.

2. When assigning a value to a text variable, be sure to use double-quotes at the beginning and ending.

3. When assigning a value to a char, be sure to use single-quotes at the beginning and ending.

The following examples describe various ways of assigning a value to a variable.

```
x = 2;
myString = "hello";
aFloat = 3.1415;
myChar = 'a';

x = "hello";   /* BAD! Don't do this */
```

In addition to directly assigning a value, you can perform basic mathematical operations, and assign the result to a variable. For more information, see Performing Math with Variables.

Performing Math with Variables

You can also assign numerical values to a variable by using mathematical operators such as addition(+), subtraction (-), multiplication (*) and division (/). Consider the following valid programming statements:

```
x = y + 2;
```

This statement adds the value of 2 and y, and assigns that value to x.

```
theAreaOfMyCircle = 3.1415 * r * r;
```

This statement multiplies the variable r to itself, multiplies that value to 3.1415, and then assigns that value to theAreaofMyCircle.

```
x = x + 17;
```

The third statement is somewhat more tricky. From a programming standpoint however, the statement merely adds 17 to the current value of x, and then stores that new value in x. After processing the statement, x now equals 17 more than it did previously.

Finally, you can assign a value to a variable using a function.

```
x = SomeRandomFunction(y, z, 17);
```

For more information, see About CPL Functions.

For more information on using arethmetic operations, see Operators and Operands in the CPL Language reference.

Creating and using an Array

In addition to variables that hold single values, you can also create variables that hold multiple values, called an **array**. You declare an array in a similar manner to a normal variable. However, in order to access the array, you must know where in the array your value exists. In this sense, an array is like an apartment building: in order to speak with someone who lives in an apartment, you need to know their apartment number.

To declare an array

1. Determine the type of data the array will hold, such as int or text.

   Like other variables, an array can hold data of only one type: an array can hold multiple values of that type. For more information on variable types, see About Variable Types and Data Types in the CPL Language Reference.

2. Declare the name of the variable, as you would any other variable.

   For more information on declaring a variable, see Declaring a Variable.

3. Follow the variable name with the size of the array, in square brackets [].

   You must define the size of the array before you start adding information to the array. Once you have declared the size, you may not change the size: this is known as a **static array**. (Other programming languages support **dynamic arrays**, which can change size.)

   The following example show several different types of array declarations:

   ```
   int daysOftheMonth[31];
   char myAlphabet[26];
   text myHaiku[15];
   ```

To assign a value to an array

Place the number of the array unit (or element) within the square brackets, and then assign the value as you would with any other variable.

The following examples show how to assign individual values to an array:

```
int myMonthlyWinnings[31]

myMonthlyWinnings[0] = 100;
```

```
myMonthlyWinnings[1] = -20;
myMonthlyWinnings[2] = 50;
...
```

Note that the numbering of array elements starts at 0, not 1. So in the previous example, valid entries for `myMonthlyWinnings[31]` would be `myMonthlyWinnings[0]` through `myMonthlyWinnings[30]`.

Writing a Function

About CPL Functions

In CPL, a function is a re-usable procedure within an application. Functions exist because, very often, you may need to perform certain tasks multiple times. Or, they exist because they can act as reasonable abstractions for certain tasks.

The following script declares and uses the `Multiply_X_by_Y` CPL function, which will be analyzed in the following topics:

```
main()
{
   int myValue;
   int valOne;
   int valTwo;

   valOne = 10;
   valTwo = 5;

   myValue = Multiply_X_by_Y(valOne, valTwo);
}

Multiply_X_by_Y(int x, int y)
{
   int z;
   z = x * y;
   return(z);
}
```

Topic	Description
About the Main() Function	Discussion of how the main() function starts off the entire CPL script.
Beginning and Ending a Function	How to begin and end a function.
Declaring Variables	How to declare and use variables in a function.
Writing a Function Body	How to write a function.
Returning a Value	How to return a value from a function call.

Calling a Function	How to call a function from another function, such as main().
About Built-in functions	Discusses the built-in CPL functions.

About the Main() Function

As the name implies, the main() function is the main function of any CPL script. You must have a main() function for Concordance to run your script, and as such you really cannot do anything without it.

When you start your script, the main() function is the first place Concordance looks to start the execution of your program. In that sense, the main() function is like the first page of a book. A good practice is to always start writing your program by typing the following three lines.

```
main()
{
}
```

After you create the main function, you can then proceed to write the rest of your CPL. You can declare information outside the main() function, such as variables or other functions. For example, it is customary to declare all your other functions after main(). however, you can write your other functions anywhere in the CPL as long as they reside outside the main function.

The following example shows the main() function, with the Multiply_X_by_Y function declared afterwards. Note the use of comments, which consists of text within the /* and */ characters. These lines are ignored by the compiler, and therefore allow you to make remarks in your own code.

```
/* Here is the main function */
main()
{
   int myValue;
   int valOne;
   int valTwo;

   valOne = 10;
   valTwo = 5;

   myValue = Multiply_X_by_Y(valOne, valTwo);
}

/*Other functions get declared down here */
Multiply_X_by_Y(int x, int y)
{
   int z;
   z = x * y;
   return(z);
}
```

Beginning and Ending a Function

After you have declared the main() function, your next task is to write the beginning and ending of the function.

To declare a function

1. Write the name of the function, followed by two parentheses (), followed by two brackets {}.

2. Optionally in the parentheses, write any values you wish to pass as parameters, separated by commas.

   As in math, parameters represent values that exist outside the function that you want to use inside the function. Note that parameters are different from global variables, A parameter is essentially a copy of a value, rather than the value itself. (In programming terminology, this is called **pass-by-value**.) Therefore, a function can modify the value of a parameter, without worrying about accidentally changing the original value outside the function. In contrast, any change to a global parameter will change that value for all subsequent parts of the script.

   If you do not have any parameters, you can simply have empty parentheses ().

The following example describes a basic function declaration with two parameters:

```
Multiply_x_by_y(int x, int y)
{
}
```

This example declares a function named `Multiply_x_by_y`, which takes two parameters: an integer `x`, and an integer `y`.

Once you have declared your function, you can declare any additional variables you may need within the function. For more information, see Declaring Variables. For more information on declaring a function, see Function Declaration in the CPL Language Reference.

Declaring Variables

Once you declare your function, you can then declare any additional variables that you may need.

To declare a variable in a function

Declaring a variable in a function is just like declaring them in other locations: you determine what type of variable you need, and then you give it a name.

As with normal variable declaration, you can declare any number of variables. For more information, see About Variable Types.

Note: When you begin to write a function, you will likely not know what variables you will need. Don't worry. Just add them in as you go. If you get to a piece of code that needs a variable, scroll back to the top of the function and insert the declaration. It is recommended to declare a variable as soon as you realize you need it, rather than wait until you are finished with the function. Doing so prevents "undeclared variable" errors.

The following example declares the integer variable z:

```
Multiply_X_by_Y(int x, int y)
{
   int z;
}
```

You can use the parameters as variables, as you would any other variable, to store and retrieve data. Just be careful you don't overwrite a value may need later.

Once you have declared your variables, you can begin writing your function. For more information, see Writing a Function Body.

Writing a Function Body

Once you have declared your function and the variables, you can write the rest of your function.

Functions exist mainly to help you organize your code, so you can put whatever you wish into one, as long as it seems logical to you. The main caveats to writing a function body are as follows:

- If you wish to return a value out of your function, you must use a return value. For more information, see Returning a Value.
- The parameters are considered variables within the function; however, they contain only a copy of the value you passed in. Therefore, nothing you do will affect the original value.

The following example code takes the two parameters x and y, multiplies them together, and assigns that value to z. So, if the value of x was 5 and y was 10, z would end up holding 50.

```
Multiply_X_by_Y(int x, int y)
{
   int z;
   z = x * y;
}
```

Once you have finished writing your function, you can use the function in a function call. For more information, see Calling a Function.

Returning a Value

A function can optionally return a single value. Returning a value means that, when a function ends, the function will give back a value. Most commonly, the value that is returned is then assigned to a variable of some sort, as described in Calling a Function.

To return a value from a function

At the end of the function, use the return() command to indicate what you wish to return.

Calling the return command ends the function; any additional code after the return() command will be ignored.

The following example shows how to return the value z from a function.

```
Multiply_X_by_Y(int x, int y)
{
   int z;
   z = x * y;
   return(z);
}
```

You do not need to use a return value. For example, if you designed a function to access the Concordance database or perform a calculation and display the result to the user, there may be no reason to return a value. However, many functions will return a value of some sort.

In most programming languages, you have to declare a function by also specifying the type of the parameter you are returning. You would find the function in the above example declared in C as:

```
int Multiply_X_by_Y(int x, int y)
```

This tells the C language interpreter to expect an integer as a return value. However, CPL does not use this convention. You do not have to specify the type of the return value. You must however be careful to remember the type of the return value, if any. If you call the function expecting to get a short and get an int instead, you may wind up with some unexpected results.

Once you have finished your function, you can use your function in a function call. For more information, see Calling a Function.

Calling a Function

In programming, when you call a function you are actually initiating the function. You are telling the function to begin and perform any tasks.

To call a function

1. Type the name of the function, followed by a list of parameters that the function is expecting.
2. If the function returns a value, be sure to assign that value to a variable.

The following example calls multiply_X_by_Y, and assigns the returned value to myValue.

```
int myValue;
myValue = Multiply_x_by_y(15, 10);
```

In addition, you can place variables in the parameter list. The function call will make a copy of

the values stored in the variables, and pass the copied value in through the parameters.

```
main()
{
   int myValue;
   int valOne;
   int valTwo;

   valOne = 10;
   valTwo = 5;

   myValue = Multiply_X_by_Y(valOne, valTwo);
}

Multiply_X_by_Y(int x, int y)
{
   int z;
   z = x * y;
   return(z);
}
```

About Built-in functions

There are two types of functions: built-in functions, and user-defined functions.The previous topics discuss user-defined functions in detail: they are functions that you write to perform tasks. However, there are entire libraries of built-in CPL functions that perform specific tasks, such as database handling, file handling, math, text manipulation, and more. You will likely find these functions an invaluable tool for your programming needs. However, such a discussion is beyond the scope of this section. Instead, the following topics will assist you in learning more about built-in functions:

Topic	Description
About CPL Functions	The Fundamentals topic that discusses built-in functions.
About the Advanced Programming Features	The section of the document that goes into detail about the different types of built-in functions.
About CPL Functions	The API reference listing of all the built-in functions.

Using Conditional Statements and Loops

About Conditional Statements

A loop is a programming structure that allows you to run a set of code multiple times. In contrast, a conditional statement is a line of code that check for a condition, and then runs a

set of code if the conditional statement is true. You use conditional statements and loops together to run code based on conditional values. For example, you could use a conditional statement to run different sections of code, depending on what the date was. Or perhaps you want to know whether the data you are processing contains a certain text phrase. You would use a conditional statement for either of those checks.

The basic conditional statement is the **if-statement**, which has the following syntax:

```
if (some conditional statement holds true)
{
      PerformSomeTask();
      PerformSomeOtherTask();
}
```

Note that a conditional statement uses the same opening and closing brackets as a function. The code you place in between the brackets will only be executed if the conditional statement holds true. This is a very important point: if the statement doesn't hold true then the section of code encapsulated between brackets is completely skipped.

The following topics discuss conditional statements and loops.

Topics	Description
Conditional Operators	About basic conditional operators, such as greater-than and less-than.
Else Statements	How to put together a branching if-then-else statement.
Compound If-Statements	How to link multiple if-statements together.
Switch Statement	How to use a switch statement.
Loops	How to loop code.

Conditional Operators

The first step in constructing an if-statement is a conditional. A conditional can be whether something equals something else, whether something is greater than something else, and so on. The operators you use to perform a comparison are very similar to the ones used in mathematics.

Operator	Description
==	Is equal (note the double equals sign)
<>	Does not equal
<=	less than or equal
<	Less than
>=	Greater than or equal
>	Greater than

Note: do not confuse the double equals sign with the single equals sign. Use the double equals sign (==) when comparing a value. Use the single equal sign (=) when assigning a value.

The following example checks if the variable age contains a value greater than or equal to 21.

```
if (age >= 21)
{
  //do something with code
}
```

The following example check if the companyName variable contains "LexisNexis".

```
if (companyName == "LexisNexis")
{
  //do something with code
}
```

Note the use of double quotes to encapsulate a piece of text (called a string).

Finally, CPL allows you to use numerical values as a trigger for a conditional statement, as shown in the following table

Value	Description
0	considered to be FALSE when used as a condition.
All other values (positive and negative)	Considered to be TRUE when used as a condition.

This allows you to use variables and function return values as conditional operators. For example, you may see code similar to the following:

```
if (someFunction())
{
   doStuff();
}
```

This code states "if the return value of someFunction() is non-zero, execute the following code." This allows you to create functions that check for information and perform some task; if that task was successful, you can use a conditional statement to act on it.

Else Statements

An **else-statement** is an addition you can place on an if-statement to branch your decision: that is, to execute one set of code if your conditional value is true, and another if the value is false. An else-statement is constructed using the following syntax:

```
if (conditionalValue)
{
      ExecuteThisCode();
}
else
```

```
{
      ExecuteSomeOtherCode();
}
```

The code in the if-statement is executed if the conditional value is **true**. The code contained within the else-statement is executed if the conditional value is **false**.

For example, you may wish to execute different one set of code if a target is at least 21, and another if they are less than 21. You could create two different if-statements, as shown in the following example:

```
if (age >= 21)
{
   ExecuteCodeForAdultCustomers();
}

if (age < 21)
{
   ExecuteCodeForJuvenileCustomers();
}
```

Alternately, you could use an else-statement, instead.

```
if (age >= 21)
{
   ExecuteCodeForAdultCustomers();
}
else
{
   ExecuteCodeForJuvenileCustomers();
}
```

Compound If-Statements

A compound if-statement occurs when two or more conditions must exist before you want to execute your code. For example,you may wish to know if a person was between the ages of 18 and 21 (inclusive). Perhaps you want to target that age group for some marketing campaign. You could write a **nested** if-statement that looks like the following:

```
if (age >= 18)
{
   if (age <= 21)
      {
         SendMarketingMaterials();
      }
}
```

Alternately, you could write both statements in **compound if-statement**. Compound if-statements use the keywords, **and** and **or**, to combine conditional statements. For example,

the previous nested if-statement is equivalent to the following compound if-statement:

```
if ((age >= 18) and (age <= 21))
{
      SendMarketingMaterials();
}
```

Both conditional statements must hold true for the function, **SendMarketingMaterials()**, to execute. Note the use of parentheses. As good practice, make sure all your conditional statements in your compound if-statements are enclosed within parentheses.

Switch Statement

A switch-statement allows you to compare a single value to a series of constants. You can think of a statement as the switch on a railroad. Flip the switch one way and your train goes to the left. Flip the switch the other way and your train goes to the right. Unlike a railroad switch, a switch-statement in programming can have more than two different alternatives.

The following sample describes the structure for a switch-statement:

```
switch (some value to check)
{
   case value1:
      doSomething();
      break;
   case value2:
      doSomethingElse();
      break;
   default:
      doTheRest();
      break;
}
```

Notice the four keywords here: **switch**, **case**, **break** and **default**:

- The first line contains the **switch** statement. This is the value that CPL will use to determine which of the case statements to execute.
- Each **case** statement contains one possibility for the switch value that you wish to write code about. You can have as many case-statements as you need in order to process each switch value. In the example below, there are 5 different case statements representing the five different vowels.
- The **break** statement tells CPL to stop processing the case statement and drop out of the switch-statement.
- The **default** statement is optional. If the value you are checking does not match any of the case statement values, CPL will execute the code referenced by the default-statement.

Note the use of colons (:) after the case-statements and default-statement. These are required.

the following example counts the number of times a vowel appears. You can write the following section of code, using a compound if-statement:

```
if ((myLetter == 'a') or (myLetter == 'e') or (myLetter == 'i') or (myLetter
{
      vowels = vowels + 1;
}
```

Alternately, you could also write the code using a switch-statement.

```
switch (myLetter)
{
   case 'a':
      vowels = vowels + 1;
      break;
   case 'e':
      vowels = vowels + 1;
      break;
   case 'i':
      vowels = vowels + 1;
      break;
   case 'o':
      vowels = vowels + 1;
      break;
   case 'u':
      vowels = vowels + 1;
      break;
}
```

Loops

Loops are a programming tool that allow you to repeat a section of code over and over again. For example, if you wrote a piece of code that formatted a page in a particular manner, you could use a loop to format all pages in a document in that particular manner. You use conditional statements with loops in order to instruct your computer as to when they should loop, and how many times they should loop. The following topics discuss the two different styles of loops.

For Loops

A For loop is a looping technique that contains three elements: an **initialization value**, a **test condition**, and a **loop increment** expression. They use the following format:

```
for (initialization statement; test condition; loop increment expression)
{
   DoLoopCode();
}
```

- The **initialization** value resets the test value you will use in the for loop.
- The **test condition** is the check on the test value to see if the loop should continue: as long as the condition holds true, the loop will continue to run. The check is made after the loop runs.
- The **loop increment** is the modification you make to the test value on each loop.

The following example uses a For-loop to represent a jogger running around a track 20 times:

```
for (laps = 0; laps < 20; laps = laps + 1)
{
      Run();
}
```

While Loops

A while-loop is a loop that contains a single "while" text expression. In order for a loop to proceed, the test statement must be true. A while loop uses the following syntax:

```
while (test condition)
{
   DoSomething();
}
```

In order for a while-loop to exit, the test condition must evaluate to false. What this means is that somewhere in your while-loop something must trigger this condition. For example, the following while-loop would run for 10 times, and then exit:

```
x = 10;
while (x > 0)
{
   DoSomeStuff();
   x = x-1;
}
```

Note that it is very easy to create an infinite while-loop, simply by not modifying your test condition.

Working with the Database

About the Database

Once you understand the basics of CPL programming, you can begin to use that knowledge to manipulate Concordance databases. Using CPLs you can perform a variety of tasks, such as searching a database, reading information out of fields, and writing information to fields. The following topic describe how to work with a Concordance database.

Topic	Description
Understanding Database Handles	About the basic way to identify a database in a CPL script.
Accessing Database Information	How to access information about the database, such as the number of fields or the database name.

Accessing Database Field Information	How to access information in a database field.
Looping through a Database	How to interact with all records in a database.
Opening and Closing a Database	How to open and close the database.

Understanding Database Handles

Before you open a database, you must first understand the concept of a database handle. You can think of a database handle as the handle of a briefcase: it is the part of the database you use to contact and use the database. CPL can have up to 16 handles at any given time; you can attach each of those handles to one and only one (ie, different) databases.

There are two ways to attach a handle to a database:

1. If you start a CPL with a database already open, Concordance will automatically attach the first handle (handle 0) to that database.

2. Alternately, you can explicitly open a database using the built-in **opendb** function.

Once you have an active handle, you can access database structure information such as field names and how many documents are in the database. You can also access the data within each field of each document.

A database handle is stored in an int variable, and is defined as an integer: 0 through 15. You will notice that many code samples in the following section contain the following arbitrary variable declaration:

```
int db;
```

Since all numeric variables in CPL are initialized with a value of 0, the variable db will by default contain a handle to the current database. You may of course use your own variable names, if you so choose:

```
int myDatabase;
int litSupportDatabase;
int database1, database2;
int youGetThePoint;
```

Accessing Database Information

There will be times when you'll need to access general database information such as the number of fields in the database, the names of the fields, how many records are currently in the database, and so on. For these situations, you use certain keywords in conjunction with the database handle. For more information about accessing fields in a database that are not part of the general database infrastructure, see Accessing Database Field Information.

To access database information

1. Create a handle to the database you wish to access.

   For many applications, this will be the currently-open database. The following example creates a handle to the current database. For more information, see Understanding Database Handles.

   ```
   int db;
   ```

2. Determine the keyword you wish to access.

   For example, one such keyword is `database`, which holds the name of the handle's database. For more information on the keywords you can use, see the table below.

3. Concatenate the keyword at the end of the handle, using a period to separate the two.

   The following example shows how to fill the text variable **myDatabasename** with the name of a database.

   ```
   text myDatabaseName;
   myDatabaseName = db.database;
   ```

4. If the keyword has brackets [] listed after it, treat the keyword as an array, and use an integer to indicate which part of the array you wish to access.

   Note that unlike normal arrays, database field arrays begin with 1, rather than 0. The following example checks to see if the 4th field is a paragraph field.

   ```
   if (db.type[4] == 'P')
   {
      doSomethingHere();
   }
   ```

   For more information on arrays, see Creating and using an Array.

The following table contains a list of keywords to use when accessing a database.

Keyword	Type	Description
access[ ]	int array	This provides the user's access rights to a field. This can be no-access, read access only, write access only, or read and write access.
activequery	int	Current active query.
database	text	Database name.
documents	int	Number of documents in the database.
edited	int	Non-zero if the database needs reindexing.
fields	int	True if the subscripted field is an image key.
image[ ]	int array	True if the subscripted field is a key field.
key[ ]	int array	True if the subscripted field is a key field.

length[ ]	int array	The defined length of the subscripted field. For date fields, this will return the format, i.e. 'Y', 'M' or 'D' for yyyy/mm/dd, mm/dd/yyyy, or dd/mm/yyyy respectively.
name[ ]	text	The name of the subscripted field.
order[ ]	int array	The order in which the subscripted field is used by load, unload, global, and other functions.
places[ ]	int array	Number of places in numeric fields.
query	int	The number of the last executed query.
type[ ]	text array	The field type of the subscripted field, either 'T' for text, 'P' for paragraph, 'D' for date, or 'N' for numeric fields.

Accessing Database Field Information

One of the most powerful abilities in CPL is the ability to read and write information stored in a field.

⚠ If you are ever going to write a CPL that writes data back to a field, remember to make a back up copy of your database.CPL is a very powerful tool and a simple mistake can cause complex headaches. However, note that Concordance database security is enforced in CPL. If the user running the CPL does not have read privileges to a particular field, instead of retrieving the data, Concordance will return nothing.

Referencing a database field

The previous section described how to use a period (.) to access database structure information. In order to access the information contained in a field, you must use a notation called a pointer, which is a minus sign (-) followed by a greater-than sign (>).

To access a database using a pointer

1. Start with the database handle.

   ```
   int db
   ```

2. Place the pointer directly after the database handle.

   ```
   db->
   ```

3. Type the name of the field directly after the pointer.

   ```
   db->OCR
   ```

Note that you can access fields in a variety of ways.

- You can **access fields directly** by spelling out the field name in ALL CAPS. If the field does not exist, CPL will give you an error. The following examples access the OCR and FIRST_NAME fields.

  ```
  db->OCR
  ```

```
db->FIRST_NAME
```

- You can also use an **integer value** to indicate the numerical order of the field to access. The following example uses an integer variable to access the third field in a database.

```
int x;
x = 3;
db->x
```

- You can also use **text variables**. The following example uses the myField string to access a field.

```
text myField;
myField = "OCR";
db->myField;
```

Identifying field types

Before you retrieve information out of a field, you must know the type of information the field contains. For example, you would not want to store the information from a text field into an integer variable.

The following sample code shows how to identify the type of information a field contains.

```
main()
{
   int db, numericVariable;
   text textVariable;

   for (i = 1; i <= db.fields; i = i + 1)
   {
      switch(db.type[i])
      {
         case 'P':
         case 'T':
            textVariable = db->i;
            break;
         case 'N':
         case 'D':
            numericVariable = db->i;
            break;
      }
   }
}
```

1. The first two lines are the variable declarations, which tell the CPL interpreter that you will use these variables:

```
int db, numericVariable;
text textVariable;
```

2. The next section is a for-loop.

```
for (i = 1; i <= db.fields; i = i + 1)
```

The initialization statement starts i as 1; the test condition checks if i is less than or equal to the number of fields in the database; the loop increments i by one. Recall that database fields start at 1 (rather than 0).

3. The next section of code is a switch-statement, which contains the type of the current field.

```
switch(db.type[i])
```

4. The next few statements are the case-statements.

Note that the code combines the case-statements for paragraph fields and text fields (P and T). This is done by not including the keyword, break, in between the case-statements. Depending on the field type, you set the value of a text variable or numeric variable to the contents of the field.

```
textVariable = db->i;
```

-or-

```
numericVariable = db->i;
```

You use the variable i to grab the contents of field i. If i equals 2, then db->i would equal the contents of field 2. This is an example of assigning a variable to the value of your field.

Assigning a value to a field

Once you have confirmed the type of information that a field contains, you can modify that information.

To set the contents of a field:

Use a pointer to reference a field, and then assign a value as you would a variable.

The following example sets the COMPANY text field value to "LexisNexis, Inc."

```
db->COMPANY = "LexisNexis, Inc.";
```

As with other fields, you can access a field through a numerical variable. The following example modifies the 3rd field in a database:

```
int n;
n = 3;
db->n = "LexisNexis, Inc.";
```

Here are you simply “assigning” a value. Remember the left side of the equals sign is the object you are storing information into. The following example stores the value of x into a field called PAGES.

```
db->PAGES = x;
```

Note you are not storing the letter "x" in the PAGES field. You are also not storing the value stored in the PAGES field into the variable x.

You can severely damage your database by assigning a value to it. For example, the following code deletes all data in a field

```
db->n = "";
```

Current record

The current record refers to the record that Concordance is interacting with. By default, this is usually the record being displayed in the UI. However, accessing multiple records may be more useful than accessing just one. For example, you may want to access all records in a database. In the UI, you would simply press the Next button, and continue with your work. This topic discusses how to access multiple records.

There are several ways to change the current record to a new record, using the following built-in functions:

- **next(int db)** - Using the database handle, the next command can move CPL to the next record in the database
- **prev(int db)** - Pass the database handle to this function to move CPL to the previous record in the database.
- **goto(int db, document)** - Pass a database handle along with a document number in the current query to move CPL to a particular document in the current query.

You can also use search features to locate additional records. For more information, see Searching Databases.

The following example opens a database and goes to field number 42.

```
main()
{
	int db, db2;
	db2 = opendb("c:\temp\support.dcb");
	goto(db2, 42);
}
```

Looping through a Database

Previous topics have discussed the two main looping structures: For Loops and While Loops. However, CPL supports a third structure that deals strictly with databases: the cycle-loop. A cycle-loop loops through the current Concordance query. If you performed a search prior to executing a CPL, that query will still be active unless you change the query using built-in functions. The following describes the cycle-loop syntax:

```
cycle(databaseHandle)
{
   CodeToRunOnEveryRecordInQuery();
}
```

Note the cycle statement looks very similar to a for- and while-loop. The cycle loop begins and ends with curly brackets, and has a body of instructions in between. The item in between the parentheses after the keyword, cycle, is the database handle.

For example, assume that you have already performed a search on your Concordance database, and therefore already have the results of a query. The following code will change the contents of the AUTHOR field in every record of your query to "John Smith".

```
main()
{
   cycle(db)
   {
      db->AUTHOR = "John Smith";
   }
}
```

Opening and Closing a Database

Most of the time, you will only be dealing with one database and one database handle. Eventually you may come across a situation where you need to access the information from two or more databases. Using the built-in function, **opendb**, you can open up to 16 databases.

The syntax for this function is as follows:

```
opendb(text databasePath);
```

This function returns a handle to the newly opened database. The following example opens a database whose path is "c:\temp\support.dcb".

```
main()
{
      int db, db2;
      db2 = opendb("c:\temp\support.dcb");
}
```

If you had a database open prior to running this CPL, the variable, db, would hold a handle to the current database. After running the opendb function, the variable, db2, would hold the handle to the newly opened database. You now have access to two databases.

You can write a script that accesses both databases. for example, suppose that record 42 in database 2 (db2) has a special page count in the PAGES field. You want to populate this value into the page count field (also called PAGES) in your currently open database (db). To start, you will need to access the first database:

```
main()
{
      int db, db2;
```

```
      db2 = opendb("c:\temp\support.dcb");
      goto(db2, 42);
}
```

Second, you would need to contain the page count:

```
main()
{
      int db, db2, pageCount;
      db2 = opendb("c:\temp\support.dcb");
      goto(db2, 42);
      pageCount = db2->PAGES;
}
```

Now, you can cycle through the original database and store the value stored in the pageCount variable into PAGES field of all the records in the current query.

```
main()
{
      int db, db2, pageCount;
      db2 = opendb("c:\temp\support.dcb");
      goto(db2, 42);
      pageCount = db2->PAGES;
      cycle(db)
      {
            db->PAGES = pageCount;
      }
}
```

Notice that even though the field name, PAGES, occurs in both database, we differentiate the two by using the individual database handles.

You can also add in one more line to close the database we opened. Don't worry if you forget this step. After a CPL finishes, Concordance makes sure to close any databases that you opened in the course of running the program.

```
main()
{
      int db, db2, pageCount;
      db2 = opendb("c:\temp\support.dcb");
      goto(db2, 42);
      pageCount = db2->PAGES;
      cycle(db)
      {
            db->PAGES = pageCount;
      }
      close(db2);
}
```

Using Common CPL Functions

About Common CPL Functions

The following topics discuss several of the built-in functions often used in CPL scripts. These include functions to manipulate text data, search databases, and interface with users.

Topics	Description
Text Manipulation	How to perform basic text manipulation.
Searching Databases	How to search a database.
User Interface	How to create basic message boxes and display text on the screen.

Text Manipulation

There are several built-in functions you can use to help manipulate text data in fields. Two of the more popular functions are **match** and **substr**.

match function

The **match** function searches for a text string within another text string. You can use it search for a specific phrase within a database field. The syntax for the match function is as follows:

```
int match(text target, search; int offset, length);
```

The following table describes the parameter values.

Parameter	Description
target	The line of text to search in, such as a text variable or text field.
search	The text to search for.
offset	The offset into the target to start looking in.
length	Optional. The length of text after the offset to search in. If not set, the function will search from the offset until the end of the string.

match returns the offset of the first character of the search string in the target string. Partial matches do not count with the match function. If there was no match, the function returns a value of 0. Note that **match** is case-sensitive.

For example, suppose you have the following target string:

```
targetString = "The quick brown fox jumped over the lazy dog.";
```

and had a search string such as:

```
searchString = “quick”;
```

The match function would return a value of 5 since the `searchString` appears at the 5th character into the `targetString`.

The following example counts how many times "lazy dog" appears in an OCR field.

```
main()
{
   int db, n, i;

   cycle(db)
   {
      i = match(db->OCR, "lazy dog", 1);
      while (i <> 0)
      {
         n = n + 1;
         i = match(db->OCR, "lazy dog", i + 1);
      }
   }
}
```

At the end of executing the cycle-loop, n equals the number of occurrences of the phrase “lazy dog.” Note that you exit the while loop when the value i equals 0. This indicates that the match function did not find anything. Remember to increment the starting offset by one or else you may be caught in an infinite-loop.

substr function

The **substr** function extracts a piece of text from another string. It makes a copy of this text and returns it as a text variable. The format for this function is as follows:

```
text substr(text string; int from, width);
```

Parameters	Description
string	The string to extract data from. Can be a text variable or database field. Note that to specify a database field, use the pointer notation (db->OCR).
from	The starting point from where to extract the text. This is a 1-based offset that starts from the first character.
width	How many character to extract, starting at the beginning offset.

substr returns the subset of the specified string.

For example, the following code sample extracts a subset from a string.

```
main()
```

```
{
   text targetString, myString;

   targetString = "The quick brown fox";

   myString = substr(targetString, 11, 5);
}
```

After executing the **substr** function, the text variable, **myString**, contains the value, "brown."

Searching Databases

You can utilize Concordance's powerful searching capabilities within CPL, using the **search** and **query** functions. It helps if you are proficient in constructing Concordance queries.

search function

The **search** function allows you to search through a database for a specified string. The syntax for the search function is as follows:

```
int search(int db; text searchString; int options);
```

The following table describes the parameter values.

parameter	Description
db	The database handle.
searchString	The search string to execute. Search strings are the same as the queries you would use in the standard search screen in Concordance.
options	Optional. For more information, see the CPL Language reference.

The results of the search become the current query. The return value contains an error code, if any.

The advanced features of CPL allow you to automate the creation of complex queries. For example, you can search for data in one database and use the results to populate the fields of another database. You can automate the searching in multiple databases to search and replace a misspelled word.

Consider the following example:

You are a paralegal and are preparing a database for trial. You suddenly get a call from your vendor telling you that the database they sent you two months ago was coded incorrectly. You hadn't realized it, but an important field was never coded, the DOCTYPE field. They send you an update to the database, but you are faced with a dilemma. For the past two months, the attorneys have been using the database and entering comments into the COMMENTS field. You obviously do not want to overwrite this database. You need to quickly whip up a

short CPL to do the job for you.

Let's assume for this example that you have a field called BEG_BATES and END_BATES which together uniquely identify the same document in both databases. We'll also assume your database is currently open and the database from your vendor is in the following directory:

```
c:\temp\vendor.dcb
```

Here's what the CPL would look like:

```
main()
{
   int db, db2;
   db2 = opendb("c:\temp\vendor.dcb");

   cycle(db)
   {
      search(db2, "BEG_BATES = " + db->BEG_BATES + " and END_BATES = " + d
      db->DOCTYPE = db2->DOCTYPE;
   }

   closedb(db2);
}
```

query function

The **query** function switches the current query to a new query. The syntax for the query function is as follows:

```
int query(int db, number; text string);
```

The following table describes the parameter values.

Parameter	Description
db	the database handle of the database to search.
query	The query number to switch to. Note that Concordance sequentially numbers the searches you execute. Query number 0 represents your entire database (with no query). In order for you to keep track of the queries you perform in CPL, you may wish to store the value of **db.query** or **db.activequery** after you perform a search.

If successful, the current query becomes the specified query. The return value contains an error code, if applicable.

User Interface

You can use the following user interface functions (**puts**, **messagseBox**, and **getkey**) to display text to the user as well as to get user input.

puts function

puts( ) is one of the most convienent ways of displaying text to the user. The following code displays the syntax for puts.

```
puts(int row, column; text string);
```

The first two parameters, **row** and **column**, specify the row and column number where you want to place your text. Imagine the CPL screen as a piece of graph paper. Depending on how large you make the window, the more squares on the graph paper you will see. Each square on the paper represents a place where you can place a character such as 'A' or 'd'. The top left corner of the screen is row 0 column 0. As you move to the right of the screen, the column number increases. As you move down the screen, the row number increases. The third parameter, string, is the text you want to display.

	0	1	2	3	4	5	6	7	8	9
0										
1			H	e	l	l	o			
2					W	o	r	l	d	
3										

The text above would have been placed on the CPL screen using the following code:

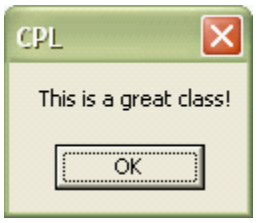
```
puts(1, 2, "Hello");
puts(2, 4, "World");
```

messageBox function

You are probably familiar already with Windows message boxes. CPL has a built-in function to display your own. The **messageBox** function has the following format:

```
int messageBox(text szText, szTitle; int style);
```

The first parameter, **szText**, is the text of the message box you wan to display. The second parameter, **szTitle**, is the title of the message box. The third parameter, **style**, can be several values. For more information on style, see About the Concordance Programming Language Reference.



This message box was produced using the following line:

```
messageBox("This is a great class!", "CPL", MB OK);
```

getkey function

getkey pauses the program and waits for the user to press a key. The syntax of getkey is as follows:

```
char getkey();
```

getkey does not take any parameters, but instead pauses the script until the user presses a key. Then, the function returns the key that the user pressed.

For example, suppose you want to write a program that accepts the letter 'A' or 'B' from the user. Here's how you would write it:

```
input()
{
   int value;

   value = 0;
   puts(5,10, "Enter code:");
   while(value == 0)
   {
      switch(getkey())
      {
         case 'A':
         case 'a':
            value = 1;
            break;
         case 'B':
         case 'b':
            value = 2;
            break;
         default:
            break;
         }
      }

   return(value);
}
```

Advanced Programming Features

About the Advanced Programming Features

The following topics list the features and functions that are described by the built-in functions of the Concordance programming language.

Topics	Description
About Annotation Functions	Access notes and attachments on Concordance records.
About Database Functions	Starts Concordance options that appear on the standard menus, or helps manage databases by opening, closing, and searching.
About Data Conversion Functions	Convert numbers to text and back again, dates to text and back again, and characters to text and back again.
About Data Editing Functions	Handles full screen editing, editing in windows, and input from the keyboard.
About Dictionary Btree List Management Functions	Creates, maintains, and searches btree list files.
About DDE Functions	Exchanges data with other DDE-enabled programs.
About File Handling Functions	Allows applications to open, close, read, write, create, and erase external files.
About Math Functions	Manipulates numeric data, providing functions for comparison and change.
About Query and Record Management Functions	Accesses the documents in the data base.
About Screen Control Functions	Maintains control over the program's presentation on the screen.
About System Functions	Provides access to external programs, operating system commands, and Concordance language features not available elsewhere.
About Text Manipulation and Classification Functions	Works with text and character strings.
About Time Functions	Determines the current time, date, and the elapsed time since the program started.

About Annotation Functions

The following table describes the Concordance annotation functions.

Function	Description
annotationAppend	Appends a new annotation to a record.
annotationCount	Count of annotations on the current record.
annotationDelete	Deletes an annotation.
annotationGoto	Makes an annotation the current record.
annotationIsTagged	Determines if an annotation is tagged.
annotationRetrieve	Retrieves data from an annotation record.
annotationTag	Adds or delete s a tag from an annotation.
annotationUpdate	Updates the annotation with new data.

About Database Functions

The following table describes the Concordance database functions.

Function	Description
browse	Concordance database browse.
closedb	Closes an open database.
createdb	Creates a new database.
createfs	Full screen database create options.
createReplica	Creates a replica of the database.
editfs	Concordance full screen editing.
exec	Executes a saved query file.
global	Invokes Concordance full screen global editing.
import	Loads a text file into the selected field.
importfs	Full screen Documents/Import menu option.
index	Indexes a data base.
keep	Saves the current query session to file.
load	Loads a delimited ASCII file.
loadfs	Invokes Concordance full screen load option.

lockdb	Locks database for exclusive use.
modify	Full screen data base modify.
opendb	Opens a data base for use.
operator	Changes the default search operator.
overlayfs	Invokes Concordance full screen overlay module.
pack	Removes documents marked for deletion.
print	Prints data base records using a print format file.
printfs	Invokes Concordance full screen print mode.
reindex	Reindexes edited and appended documents.
replicate	Replicates changes between databases.
report	Runs a saved report, printing a document range.
reportfs	Invokes Concordance full screen report writer.
resolve	Resolves collisions between replicated databases.
search	Searches a data base.
snapshot	Saves and restores a complete Concordance environment.
struc	Duplicates a database's structure.
table	Invokes Concordance full screen table view mode.
unload	Creates a delimited ASCII file from the current query.
unloadfs	Invokes Concordance full screen unload command.
unlockdb	Unlocks database from exclusive use.
zap	Zaps the data base, removes every document.

About Data Conversion Functions

The following table describes the Concordance data conversion functions.

Function	Description
asc	Converts a text variable into a numeric value.
chr	Converts a character into a text variable.
ctod	Converts a text representation of a date into an integer.
dtoc	Converts an integer date, or date field, into a string.
itoa	Converts an integer to a plain text in any base.
num	Converts a string into a number.

str	Converts a number into a string.

About Data Editing Functions

The following table describes the Concordance data editing functions.

Function	Description
edit	Edits text, dates, numbers in a window.
getkey	Retrieves a keystroke.
getline	Edit a line of text.
getnumber	Edits a number.
keypress	Status of next keystroke.

About Dictionary Btree List Management Functions

The following table describes the Concordance list management functions.

Function	Description
btclose	Closes a list file.
btcount	Returns the count of entries.
btcreate	Creates a new list file.
btcycle	Loads a list file from a data base field.
btdelete	Deletes an entry from a list file.
btexact	Locates a match for both the target and data value.
btfind	Locates an entry in a list file.
btfirst	Returns the first entry in the file.
btgt	Locates an entry greater than the target.
btgte	Locates an entry greater than or equal to the target.
btinsert	Adds a new entry to the list.
btinserta	Adds a new entry to the file in ascending order.
btlast	Locates the last entry in the file.
btlock	Locks the list file for exclusive use.

btlt	Locates an entry less than the target.
btmenu	Displays a list file list in a menu.
btnext	Returns the next entry in the file.
btopen	Opens an existing list file for use.
btprev	Locates the previous entry in the file.
btrebuild	Rebuilds a b+tree with 64-bit data values.
btunlock	Releases list file for multi-user use

About DDE Functions

The following table describes the Concordance Dynamic Data Exchange (DDE) functions.

Function	Description
ddeConnect	Connects to a DDE server application.
ddeDisconnect	Terminates a DDE conversation.
ddeExec	Executes a command in the DDE server.
ddePoke	Sends data to the DDE server.
ddeRequest	Retrieves data from the DDE server.

About File Handling Functions

The following table describes the Concordance file handling functions.

Function	Description
chdir	Changes the file system drive and directory.
close	Closes an open file.
diskspace	Determines the amount of disk space available.
erase	Erases a file.
exist	Determines if the file exists.
findfirst	Locates first matching file name.
findnext	Returns next matching file name.
getcwd	Determines the current working drive and directory.

getfile	Prompts the user for a file name.
lseek	Moves to the requested place in a file.
mapDevice	Maps a network drive or printer.
mkdir	Make a new directory.
open	Opens a file for use.
read	Reads data from a file.
readc	Retrieves a character from a file.
readln	Reads a line of text from file.
rename	Renames a file.
rmdir	Remove a directory.
unmapDevice	Unmaps a network drive or printer.
write	Writes data to file.
writec	Writes a character to file.
writeln	Write a line of text to file.

About Math Functions

The following table describes the Concordance math functions.

Function	Description
max	Returns the greater of two values
min	Returns the lesser of two values
not	Inverts the number, returning its one's compliment
rand	Generate random numbers
round	Rounds floating point numbers
sqrt	Calculates a square root

About Query and Record Management Functions

The following table describes the Concordance query and record management functions.

Function	Description

accession	Retrieves the records accession number.
append	Appends a record to the data base.
blank	Clears a record, used to append empty records.
concat	Concatenates data bases.
concatclear	Clears concatenated data bases.
count	Number of documents located by last search.
delete	Marks current document for deletion in the next pack.
deleted	Determines if document is marked for deletion.
docno	Returns the record's document number.
edited	Determines if document was edited since last indexed.
exec	Executes a saved query file.
first	Reads the first document in the current query.
fuzzy	Locates a list of fuzzy matches for a word.
getuuid	Retrieves the record’s UUID.
goto	Reads a requested document in the current query.
gotoaccession	Locates and reads the document the matches the accession number.
gotophysical	Locates the physical document number in the query.
gotouuid	Goes to a record by universal unique identifier.
hits	The number of search terms located in active query.
isdeleted	Determines if document is marked for deletion.
isedited	Determines if document was edited since last indexed.
isnexthit	Determines if there is another hit in the current document.
istagged	Determines if a document is tagged.
keep	Saves the current queries to file.
last	Retrieves the last document in the current query.
lockdoc	Prevents other users from editing the document.
locked	Determines if the document is locked.
markhits	Returns record's data, with search hits marked.
next	Retrieves the next document in the current query.
nexthit	Retrieves information on the next hit in the query.
prev	Retrieves the previous document in the current query.
prevhit	Retrieves information on the previous hit in the query.
query	Loads the requested query, and the search logic.
queryString	Retrieves the query string.

readdoc	Retrieves the requested document, regardless of query.
recall	Undeletes a document.
recno	Returns the document's record number.
reset	Returns data base document to prior state.
set	Changes data base environmental variables.
sort	Sorts the list of documents in the current query.
tag	Tags a document.
tagquery	Converts tagged documents into a query.
unlockdoc	Releases a locked document.

About Screen Control Functions

The following table describes the Concordance screen control functions.

Function	Description
box	Draws a box on the screen.
ccol	Retrieves the column number of the cursor.
cls	Clears the screen.
crow	Retrieves the row number of the cursor.
cursor	Positions the cursor on the screen.
cursoroff	Hides the cursor.
cursoron	Displays the cursor.
menu	Displays a menu.
messageBox	Displays a standard Windows message box.
puts	Puts a string on the screen.
putsl	Puts a string or substring on the screen.
restore	Restores a saved portion of the screen.
save	Saves a portion of the screen.
scroll	Scrolls the screen up or down.
show	Displays and highlights search words in a full text field.

About System Functions

The following table describes the Concordance system functions.

Function	Description
beep	Makes a beep on the speaker.
dc	Used in descending order sorts.
debug	Turns debugging on/off.
eval	Evaluates a Concordance expression.
exit	Ends the program, returns to the operating system.
func	Returns the name of the current function.
getarg	Retrieves Concordance command line arguments.
getenv	Returns environment string value.
getPrivateProfileString	Retrieves an entry from an initialization file.
memavl	Determines available memory.
program	Name of executing program.
putenv	Sets environment string value.
run	Runs a function located in another CPL program file.
shellExecute	Executes an external file, document, or program.
sizeof	Determines the space occupied by a variable.
sleep	Pauses execution of Concordance.
spawn	Executes an external program.
system	Executes an operating system command or program.
ver	Returns the Concordance version number.
writePrivateProfileString	Writes an entry to an initialization file.

About Text Manipulation and Classification Functions

The following table contains the functions that manipulate and classify text.

Function	Description
addr	Determines the address within a text variable.

capitalize	Duplicates the text with all words capitalized.
cut	Copies text to the cut and paste buffer.
deleteText	Deletes text from a database field.
findline	Locates the word wrapped line, and its length.
findnline	Locates the next word wrapped line, and its length.
findpline	Locates the preceding word wrapped line, and its length.
insertText	Inserts text into a database field while preserving annotations and formatting.
isalnum	Determines if the character is alphanumeric.
isalpha	Determines if the character is alphabetic.
isdigit	Determines if the character is a numeric.
isfield	Determines if the field exists in the data base.
islower	Determines if the character is lower case.
isspace	Determines if the character is a space character.
isupper	Determines if the character is upper case.
len	Calculates the length of the text.
lower	Duplicates the text in lower case.
ltrim	Removes leading white space from a string.
match	Locates a string within another string.
matchc	Locates a character within a string.
newline	Returns a carriage return and line feed string.
pad	Pads a string with spaces, centered, left or right justified.
paste	Retrieves text from the cut and paste buffer.
rep	Replicates a character into a string.
rtrim	Removes trailing white space from a string.
substr	Returns a partial string from a string.
trim	Removes leading and trailing spaces.
upper	Duplicates the text in upper case.
wordlen	Length of the word.
wrap	Word wraps text data.

About Time Functions

The following table lists the functions that deal with counting or manipulating time units.

Function	Description
clock	Milliseconds elapsed since Concordance started.
ctod	Converts character strings to date integers.
day	Day number, 1 - 7, for the date.
dtoc	Converts date integers to character strings.
month	Month number, 1 - 12, for the date.
time	Current time.
today	Today's date.
weekday	Day of the week, as a character string.
year	Year as an integer.

Concordance Programming Language Reference

About the Concordance Programming Language Reference

The following topics describe the Concordance Programming Language (CPL).

Topic	Description
Function Declaration	Function declaration, parameter, and return value syntax.
Identifiers	Naming conventions.
Data Types	Types of data, including char, short, int, and float.
Variable Declaration and Scope	How and where to use variables.
Reserved Words and Symbols	Words and symbols that have special meaning in CPL.
System Variables	Variables that are pre-declared by CPL.
Operators and Operands	Arithmetic, assignment, bitwise, conditional, and logical operators, as well as their precedence relative to each other.
Database Information	Database information and database fields.
Character Literals and Quoted Strings	How to write out strings.
Comments	How and where to write comments.
Program Flow and Control Structures	The structures used to control the execution of program logic.

About CPL Functions	An alphabetical list of the built-in functions.
About CPL Scripts	The scripts delivered with Concordance.

Function Declaration

The following topic describes function declaration, parameter, and return value syntax.

Function Declarations

Syntax:

```
name(optional parameters)
{
statements
}
```

A CPL function consists of a group of program statements that perform a job. Functions can optionally return values. CPL functions can call other functions to perform work without worrying how they do their jobs. Functions use discreet local variables that come into existence when the function runs, and go away when the function finishes. Functions facilitate complex programs by breaking them into small, easy to manage tasks.

CPL program execution always begins with a function called main. Every CPL program must have a main() function. It is executed by Concordance. main() may then call other functions for help in executing the program.

Function Parameters

Parameter declarations appear within the parentheses following the function's name. Parameters are declared exactly as all other variables are declared, by stating their type followed by their names. Multiple parameter declarations can be separated by commas if they are of the same type, or by semi-colons for different type declarations.

Example:

```
max(float a; float b)
{
float x;
   if (a > b)
      x = a;
   else
      x = b;
   return(x);
}
```

Concordance converts all parameters passed to max() into float types if necessary. The function parameters could also have been declared max(float a, b) with the same effect.

max() returns the largest of the two parameters passed. A statement that uses max() might look like this:

```
salary = salary * max(1.5,raise);
```

Since a and b become local variables, initialized to their passed values and accessible only within max(), max() can be rewritten without local variable x.

```
max(float a, b)
{
   if (a < b)
      a = b;
   return(a);
}
```

Return Statement

The value computed by max() is returned by the return statement. Any value can be computed within the return statement's parentheses. A return statement without the parentheses will cause the function to finish, but no value will be returned. Return statements are optional, a function without one will return automatically when the closing end statement is encountered.

CPL functions can contain multiple return statements. However, a single return statement at the end of the function is preferred. This makes the function easier to understand, and simpler to maintain.

Identifiers

A CPL identifier is any name. The name can represent a function, variable, reserved word, or data type.

Identifiers can contain any combination of letters, numbers, and the underscore character. However, an identifier can only begin with a letter. While the length of identifiers is not limited, Concordance will only consider the first 16 characters when comparing them.

CPL is case sensitive. Thus color, Color, and CoLoR constitute three different identifiers. Database field names, which are not case sensitive, are generally upper case to improve program readability.

Data Types

There are five basic data types in the Concordance Programming Language. Their types and limits are listed in the table below:

Type	Value Range	Size
char	-128 to 127	1 byte
text	any null terminated text	variable length
short	-32,768 to 32,767	2 bytes
int	-2,147,483,648 to 2,147,483,647	4 bytes
int64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	8 bytes
float	-2.2E-308 to 1.7E+308	8 bytes

The numeric data types—char, short, int, and float—can be freely assigned to each other and compared with each other. CPL will perform the necessary conversion so that the comparison or assignment is performed correctly.

Text values contain variable length strings. Their values are the values of the entire string. They can be assigned from or compared with character arrays, quoted strings, text or paragraph fields, or themselves. Database text fields, both fixed length and free text, are treated as text variables.

Date values, taken from date fields and functions which return dates, are handled as *int* types. Date math is performed in increments of days. Adding 5 to a date adds 5 days. Dividing or multiplying dates, while valid with any integer, will produce meaningless results.

Arrays

Any data type identifier followed by brackets, [ ], becomes an array. An array is a set of sequentially ordered elements which are accessed by subscripting the array. The first element in an array is element zero, thus an array with 100 elements contains elements 0 through 99.

A special array type is the char array. It can be used to store strings, a sequence of characters. In CPL, all character arrays are terminated by a zero. Character arrays can be assigned a quoted string, assigned the value of other char arrays or text variables, or compared with them. This is the only array type in CPL that allows the direct assignment or comparison of the entire array.

Example:

```
char today[10];
today = "Tuesday";
/* today[0] is T
today[1] is u
today[2] is e
today[3] is s
today[4] is d
today[5] is a
today[6] is y
today[7] is 0 */
```

Variable Declaration and Scope

Variables are declared by stating their type followed by their name. Several variables of the same type can be declared at once by stating their names separated by commas.

```
int i, count, WayDownTheRoad;
char string[20], ch;
```

Variable declarations, like all CPL statements, are terminated by a semi-colon. Variable names can be any CPL identifier.

By default, all variables start with the initial value of zero. Optionally, they can be declared and initialized in the same statement.

```
int x, y = 2, z, maximum = 25;
```

Global variable declarations must take place outside any function declaration. Globally declared variables are visible and accessible to all functions. Local variables, those declared between a function's opening { and closing } braces, are visible and accessible only to the declaring function. All variables must be declared before they are used.

Local variable declarations must take place immediately following a function's opening { brace and prior to the first executable statement. Local variables come into existence when the function is active and disappear when the function returns. Local variables can only be accessed by the function that declared them and only while it is executing. The one exception to this is when a function passes a locally declared array to another function. In this instance the receiving function can modify the original data. Arrays are passed by name alone, without subscription. In all other cases a called function gets a copy of the parameter, not the original variable.

Example:

```
something()
{
int a[1], b, c[1];
   a[0] = 1;
   b = 2;
   c[0] = 3;
   nothing(a,b,c[0]);
   /* a[0] now contains 5 */
   /* b still contains 2 */
   /* c[0] still contains 3 */
}
nothing(int x[], y, z)
{
   x[0] = 5;
   y = 10;
   z = 15;
}
```

A variable declared within a function with the same name as a global variable will cause the global variable to *disappear* from the function's point of view. References to the variable within the function only affects the local variable.

Likewise, a function within a CPL program with the same name as a Concordance function replaces the Concordance function.

Reserved Words and Symbols

Certain words in CPL have special meaning. These words are reserved and may not be used as function or variable names within a program. They are shown below. Reserved words are used to control program flow, logic, and structure.

and, begin, break, case, cycle, default, else, end, for, if, mod, or, return, switch, while

In addition to the reserved words, there are a number of reserved symbols. These symbols constitute the CPL math and comparison operators, the comments enclosures, and the character literal and string enclosures.

```
{ } /* */ ' " & = * + -> - . / <> <= < == >= > |
```

System Variables

Concordance has several variables which already exist when your program starts up. These are Concordance system variables, they contain values used and initialized by Concordance. Each of the following variables can be changed by your program. Changes to the color variables are stored by Concordance and are used as default values in Concordance displays and by various CPL functions.

Variable	Description
MaxRow	The bottom row on the screen, range 0 to MaxRow_
TextColor	Color of text used for most displays
TextBackground	Color used for background in Windows version
TextHighlight	Highlighted color used in most displays
MenuColor	Color of menu background and unselected items
MenuBackground	Background color used in Windows version
MenuHighlight	Color used to highlight selected menu item
MenuHighlightBackground	Background color for Windows version
HelpColor	Used to display help screens

Variable	Description
HelpHighlight	Highlighted text on help screens
QueryHighlight	Used in Browse mode to highlight search words
QueryHighlightBackground	Background color used in Windows version

Operators and Operands

The following topic discusses arithmetic, assignment, bitwise, conditional, and logical operators, as well as their precedence relative to each other.

Arithmetic Operators

The arithmetic operators are +, –, *, /, and mod. Unary minus is evaluated right to left, the other arithmetic expressions are evaluated left to right. Standard arithmetic precedence applies: division and multiplication are evaluated before addition and subtraction.

Operator	Description
–	Unary minus, negative numbers
*	Multiplication
/	Division
mod	The remainder of the first operand divided by the second
+	Addition
–	Subtraction

Assignment Operators

Assignments in CPL are done with =, the assignment operator. This should not be confused with ==, the comparison operator. The assignment operator assigns a value to a variable. The comparison operator determines if the two values are equal and returns a true or false as a result.

CPL assignments also produce a value as a result of the assignment. You can test the value of the assignment and make the assignment in one statement. This allows you to write more compact programs, but it can become confusing if you are not careful. There are times to take advantage of this feature, and times to avoid it.

Example:

```
if ((data = open("letter.txt","r")) <> -1)
{
/* ... */
```

```
close(data);
}
```

This program fragment opens a file and stores the file handle to the variable called data. The result of the assignment is the value stored in data. This is tested against -1 to see if the file was successfully opened. This construct is both efficient and common in CPL.

Since the assignment operator has a lower precedence than the comparison operator, the assignment to data from open() must be enclosed in parentheses. If the parentheses were not used, CPL would first compare the value of open() to -1. The result of that comparison would be a true or false. CPL would then store a true or false to data, not the file handle returned by open().

Bitwise Operators

CPL provides two operators that allow you to manipulate the individual bits in char, short, and int data types.

Operator	Description
&	Bitwise AND compares each bit of the first operand to the corresponding bit of the second operand. Returns a bit set to one for each bit set to one in both operands.
|	Bitwise OR compares each bit of the first operand to the corresponding bits in the second operand. Returns a bit set to one for each bit set to one in either operand.

Conditional Operator

The conditional operator "?:" provides a compact way to execute if-then logic. Consider the example used earlier to describe function parameters:

```
max(float a, b)
{
if (a < b)
a = b;
return(a);
}
```

It could be rewritten more compactly, by taking advantage of the conditional expression:

```
max(float a, b)
{
return(a > b ? a : b);
}
```

The conditional expression evaluates the expression preceding the question mark. It returns the value preceding the colon if it is true, and the value following the colon if it is false. The

syntax for the conditional expression is:

```
(test-expression) ? TRUE : FALSE
```

The conditional expression has the second lowest precedence of all operators, just above an assignment. Therefore, the test for the conditional expression should almost always be enclosed in parentheses.

Here's a bit of sample code that appends an *s* when the number of reported documents is plural.

```
string=str(count(db))+" Document"+(count(db)<>1)?"s":"";
writeln(handle,string,len(string));
```

Including the logic in the assignment creates a more concise program, though it may be somewhat terse and less readily understandable. Consider the extra code it would take to write this statement with if-then logic.

Logical Operators

The two logical operators are *and* and *or*. They each evaluate two operands, on the left and right, and return true or false values. They are usually used to connect individual relational comparisons into more complex relationships.

Operator	Result
and	Evaluates to true only if both operands are true
or	Evaluates to true if either operands are true

Precedence of Operators and Order of Evaluation

Operators are evaluated in order of precedence. Parentheses should be used to override precedence where desired. The operators are listed below from highest precedence to lowest. Operators on the same line have the same precedence.

Operator	Operands Evaluated
. –>	left to right
– (unary minus)	right to left
* / mod	left to right
+ –	left to right
< > <= >=	left to right
== <>	left to right
&	left to right

Operator	Operands Evaluated
|	left to right
and	left to right
or	left to right
?:	left to right
=	right to left

Database Information

The following topics discuss database information and database fields.

Database Information

A variety of information can be obtained about a database by using the database handle with the information selector and several keywords. Available information includes the number of documents in the database, the field types, length, and names, and the current query number.

Access the information by following the database handle with a period and one of the keywords listed below.

Keyword	Description
access	User's access rights to the field, read, write, or read-write. Subscript by field number, i.e., db.access[i].
activequery	Current active query.
database	Database name.
documents	Number of documents in the database.
edited	Non-zero if the database needs reindexing.
fields	Number of fields defined in the database.
image	True if the field is an image key, db.image[i].
key	True if the field is a key field, db.key[i].
length	The defined length of the subscripted field, this returns the left margin of paragraph fields, and the format of date fields, i.e. 'Y', 'M', or 'D' for yyyy/mm/dd, mm/dd/yyyy, or dd/mm/yyyy respectively.
name	The name of the subscripted field, db.name[i].

Keyword	Description
order	The order in which the subscripted field is used by Load, Unload, Global, and other functions.
places	Number of places in numeric fields.
query	The number of the last executed query.
type	The field type of the subscripted field, either *T* for text, *P* for paragraph, *D* for date, or *N* for numeric fields.

The access rights variable must be subscripted by the field's number, i.e. db.access[i]. It contains the following bit settings:

Bit Setting	End-user Rights
0	no access to this field
1	read permission is granted
2	write permission is granted

Example:

```
/* Display the names of each database field  */
/* in a menu, return the menu choice selected */
info(int db)
{
int i;
text choices[db.fields+1];
choices[0] = "Field Selection Menu";
for(i = 1; i <= db.fields; i = i + 1)
choices[i] = db.name[i];
return(menu(5,35,20,55,choices,1));
}
```

The access, name, length, places, type, and order entries must be subscripted by the field's number, as in the example above.

An interesting feature in the preceding example is the use of a calculation to initialize the number of elements in the choices array. This function guarantees that the array will always be large enough by using the parameter, db.fields+1, to calculate the number of elements. Function parameters become valid variables as soon as they are declared. Using parameters to size arrays builds flexibility into your CPL functions.

Database Fields

Databases consist of individual fields which contain varying types of data, including text, numbers, and dates. Individual fields are accessed with the database handle, the field

operator ->, and a field indicator. The database handle is any valid integer returned by a call to the opendb() function. A field indicator can be the field's number, its name, or a character array or text variable containing its name.

Example:

Assuming a database's second field is called AUTHOR, this program would replace the field contents five times, five different ways.

```
main()
{
char string[25];
int i, db;
   string = "Author";
   i = 2;
   if ((db = opendb("catalog")) <> -1) {
      db->string = "Marlowe";
      db->"author" = "O'Glue";
      db->AUTHOR = "Turley";
      db->2 = "McGee";
      db->i = "Stewart";
      closedb(db);
   }
}
```

When this program finishes, the value of the AUTHOR field will be *Stewart*. The variety of ways available to access fields provides a high degree of flexibility that should fit all situations. However, the preferred method is db->AUTHOR, with the field name in upper case.

Care should be taken when using the field name without enclosing it in quotes. Ambiguities can occur when variable names match field names. In the above example, if there were a variable called AUTHOR, Concordance would use it to determine the field name. An error would result if the variable contained something other than a valid field name. Worse yet, the AUTHOR variable could inadvertently contain a valid field name resulting in the replacement of the wrong field.

Character Literals and Quoted Strings

Individual characters used as data within a program are enclosed by apostrophes. Character strings can be enclosed within quotation marks or apostrophes. The character used to begin the enclosure must be used to terminate the enclosure. A single character enclosed within quotes is interpreted as a character string. A single character enclosed within apostrophes is interpreted as an integer.

Example:

```
trans()
```

```
{
int x;
char string[50];
x = 'a';
string = '"What," she said.';
/* ... */
string = "can't";
/* ... */
}
```

Comments

Comments can occur in a program preceding, following, or between statements. Comments cannot occur within a statement. Comments are introduced by the /* characters and are terminated by the */ characters. Comments can span several lines, but they cannot be nested.

Program Flow and Control Structures

Concordance statements provide for the control and execution of program logic. Several types of conditional execution and repetitive execution statements are available.

All simple CPL statements are terminated by a semi-colon. Individual statements can be continued over several lines before being terminated with a the semi-colon.

Begin and End

Syntax:

```
{
   statement
}
```

The { and } braces are used to create compound statements. Compound statements are treated as a single statement. { and } braces can be used after an if-else, while, or for to create a single block of statements that are executed if the test conditions are satisfied. Compound statements are terminated by the }, curly end-brace, a semi-colon does not follow the }. Individual statements within the compound statement must be terminated by semicolons.

Break

The break statement causes a cycle, while, or for loop to terminate before the loop control

statement ends the loop. break causes the innermost loop to exit if one loop is enclosed by another loop. It also causes a case statement to exit the switch.

Syntax:

```
break;
```

Example:

```
cycle(db)
   if (db->NAME == "Claire Ellen")
      break;
```

Cycle Loops

Syntax:

```
cycle(database-handle)
statement
```

Cycle loops execute the controlled statement once for each document in the current query set. After completion, the last document in the retrieved set will be the current document.

Example:

```
bonus(int db)
{
   cycle(db)
      if (db->COMMISSION >= 100000)
         db->BONUS = db->COMMISSION * 0.25;
      else
         db->BONUS = db->COMMISSION * 0.10;
}
```

For Loops

Syntax:

```
for(initialize; test-condition; increment expression)
statement
```

For loops contain three expressions: an initialization value, a test condition, and a loop increment expression. All three statements must be present. The initial loop value is set once by the initialization statement, the test condition is then evaluated. If the test condition evaluates to true, the controlled statement is executed. When the statement finishes, the

increment expression is executed, the test condition is then re-evaluated and the loop is re-executed if it is true.

Example:

```
for(i = 0; i < 100; i = i + 1)
   string[i] = 0;
```

This example would advance through the first 100 elements (from 0 to 99) of the array called string setting each element to zero.

```
for(i = goto(db,1); i > 0; i = next(db))
   if (db->COMMISSION >= 100000)
      db->BONUS = db->COMMISSION * 0.25;
   else
      db->BONUS = db->COMMISSION * 0.10;
```

The for loop used in this example produces the same results as the example used in the cycle statement on the previous page.

If-Else Statement

Syntax:

```
if(test-condition)
   statement
else
   statement
```

The if-else statement constitutes the most frequently used decision making syntax available in CPL. The statement evaluates the test condition and executes the first statement if it is true. If the test condition is false, the else statement is executed. else statements are optional.

Example:

```
   if (i < j)
i = j;
else {
   i = i - 1;
   j = j + 1;
}
```

In this example, if i is less than j, i will be set equal to j. If i is not less than j the else clause is executed which subtracts one from i and add one to j.

Switch Statement

Syntax:

```
switch(expression)
{
case expression: statement;
break;
case expression: statement;
break;
case expression: statement;
break;
/* ... */
default:
statement;
break;
}
```

Switch statements are similar to multiple nested if-else statements. A switch statement evaluates its test expression and begin executing the statements following the case statement that matches the test expression. If no case statement matches, then program execution begins at the default statement. Execution continues until the next case, break, default, or } statement is encountered. default statements are optional. If no match is made and a default is not present, no action is taken. The case statements must be enclosed within a { } pair of braces.

Example:

```
for(i = 0; string[i]  0; i = i + 1)
switch(string[i])
{
   case 'a':
   case 'e':
   case 'i':
   case 'o':
   case 'u': vowels = vowels + 1;
                  break;
   case '.': periods = periods + 1;
                  break;
   default: if (isalpha(string[i])
                        consonants = consonants + 1;
                  else
                        other = other + 1;
                  break;
}
```

While Loops

Syntax:

```
while(test-condition)
```

```
   statement
```

A while loop evaluates the test condition and executes the controlled statement zero or more times. The while loop executes the statement if the condition is true. Execution continues until the test condition becomes false.

Example:

```
intClear(int numbers[])
{
int i, elements;
   i = 0;
   elements = sizeof(numbers)/sizeof(numbers[0]);
   while(i < elements)
   {
         numbers[i] = 0;
         i = i + 1;
   }
}
```

This example would set every element of the numbers array to zero. One noteworthy feature is its use of sizeof(). intClear() does not know the number of elements in the array when it starts. It determines the number by dividing the size of the whole array by the size of one element. This function could be called in several places within a program to clear a variety of int arrays.

Functions

About CPL Functions

The following topics list are an alphabetical list of the built-in functions for Concordance Programming Language (CPL).

- A
- B
- C
- D
- E
- F
- G
- H
- I
- J
- K

- L
- M
- N
- O
- P
- Q
- R
- S
- T
- U
- V
- W
- X
- Y
- Z

A

The following topic discusses the Concordance Programming Language (CPL) functions that begin with the letter A. For more information on CPL functions, see Functions, About the Advanced Programming Features, and About CPL Functions.

accession

Summary

int accession(int db);

Description

Returns the document's accession number. This function only works on version 6.0 or later databases. Previous version databases do not support accession numbers.

Return Value

The document's accession number. Zero if the database version does not support accession numbers, or if the record has been blank()'ed but not yet append()'ed.

Example

```
/* See if we have a V6.x database opened */
 if (accession(db) > 0)
  /* It is V6.0 or later. */
 else
  /* It isn't V6.0 or later */
```

Version

Version 6.0 and later

addr

Summary

text addr(text string; int offset);

Description

Converts a text variable offset into an address that can be used by any of the Concordance functions. The offset begins with one, the first character in the field. This offset also applies to char arrays, even though they are subscripted starting with zero. addr() is faster than passing a copy of the text with the substr() function, and it will not use additional memory.

Return Value

Character index in internal format. Values returned by this function are treated as quoted strings for all practical purposes, i.e., they can only appear on the right side of an assignment.

See Also: substr()

Example

```
printField(int db, i, file)
{/* Print every line in the field to file. */
int offset, length;
 wrap(db->i, 40);
 offset = findline(db->i,1,length);
 while(offset > 0) {
  writeln(file, addr(db->i, offset), length);
  offset = findnline(db->i, offset, length);
 }
}
```

annotationAppend

Summary:

```
int annotationAppend( db->FIELD;
   int offset, length;
   text szNote;
   text szAttachment;
   int attachmentType;
   int attachmentAction);
```

Description:

Appends a new note to the current record in the field identified by db->FIELD. The note is

attached offset bytes from the beginning of the field, for length number of bytes. The remaining parameters are detailed in the table below.

szNote	The textual contents of the note.
szAttachment	The attachment, if any. This is the name of a file or web address that is launched.
attachmentType	Pass NOTEATTACHCLIPBOARD to have the attachment placed on the clipboard. Use NOTEATTACHEXTERNAL to have Concordance launch the application. NOTEATTACHNOTHING takes no action, and NOTEATTACHVIEWER sends the attachment to the current viewer.
attachmentAction	Pass TRUE to autolaunch the attachment

Return Value:

Returns zero if successful.

See Also: annotationCount(), annotationDelete(), annotationGoto(), annotationIsTagged(), annotationRetrieve(), annotationTag(), annotationUpdate()

Version:

Concordance version 7.0 and later.

annotationCount

Summary

```
int annotationCount(int db);
```

Description

Determines the number of annotations attached to the current record.

Return Value

Returns a count of the annotations on the current record.

See Also: annotationAppend(), annotationDelete(), annotationGoto(), annotationIsTagged(), annotationRetrieve(), annotationTag(), annotationUpdate()

Version

Concordance version 7.0 and later.

annotationDelete

Summary

int annotationDelete(int db);

Description

Deletes the current annotation. The note is marked for deletion and the first character of the NOTEPARENT field is set to a space. This immediately disassociates the note from the parent

record, however it is not physically removed from the database until the database is packed. Packing the parent database automatically packs the notes database.

When you delete a note with this function, it is immediately removed from the document. Your functions should reinitialize with annotationCount() and annotationGoto() to ensure that you are using the correct annotation.

Return Value

Returns zero on success.

See Also: annotationAppend(), annotationCount(), annotationGoto(), annotationIsTagged(), annotationRetrieve(), annotationTag(), annotationUpdate()

Version

Concordance version 7.0 and later.

annotationGoto

Summary

```
int annotationGoto(int db, recordNumber);
```

Description

Reads the requested annotation into memory.

Return Value

Returns zero if successful.

See Also: annotationAppend(), annotationCount(), annotationDelete(), annotationIsTagged(), annotationRetrieve(), annotationTag(), annotationUpdate()

Version

Concordance version 7.0 and later.

annotationIsTagged

Summary

```
int annotationIsTagged(int db; text tagName);
```

Description

Determines if the current annotation is tagged with the parameter tagName.

Return Value

Nonzero if the annotation contains the tag.

See Also: annotationAppend(), annotationCount(), annotationDelete(), annotationGoto(), annotationRetrieve(), annotationTag(), annotationUpdate()

Version

Concordance version 7.0 and later.

annotationRetrieve

Summary

text annotationRetrieve(int db; text "NOTETEXT");

Description

Retrieves the contents of the named field from the annotation. See annotationUpdate() for data formatting.

Return Value

All values are returned as text.

See Also: annotationAppend(), annotationCount(), annotationDelete(), annotationGoto(), annotationIsTagged(), annotationTag(), annotationUpdate()

Version

Concordance version 7.0 and later.

annotationTag

Summary

```
int annotationTag(int db; int TRUE|FALSE; [text tagName]);
```

Description

Tags an annotation, creating an issue, in the current note for the database whose handle is db. The tag is either applied or cleared according to the value of the second parameter:

TRUE Document is tagged.

FALSE Document is untagged.

If the optional tagName parameter is not passed, the operation is performed on the "default" tagged set. Passing a tagName will perform the tag or untag operation only for the specific tag.

Return Value

Zero indicates success.

See Also: annotationAppend(), annotationCount(), annotationDelete(), annotationGoto(), annotationIsTagged(), annotationRetrieve(), annotationUpdate()

Version

Concordance version 7.0 and later.

annotationUpdate

Summary

```
int annotationUpdate(int db; text FIELD, text DATA);
```

Description

Copies data to an annotation record field. All data must be passed as text. See below for examples. The FIELD parameter should be on of the fields in the annotation record.

NOTEPARENT	Unique identification assigned to the parent record. Do not modify this entry.
NOTEOWNER	The user logon of the person who created the note.
NOTETEXT	The text of the note.
NOTEATTACHED	An external file attached to the note. This is the file that is launched by a hyperlink.
ATTACHTYPE	The type of attachment can be "External", "Viewer", or "Clipboard", but this list may be expanded to support other types in the future. All three types take the contents of the NOTEATTACHED field as their parameter. A fourth value, "", is used to indicate that there is no attachment type.
AUTOATTACH	Either a "Y" or a "N". A "Y" indicates that the hyperlink is automatically launched when the user clicks on the link.
LINKFIELD	The field in the parent database that contains this note.
LINKOFFSET	The number of bytes from the beginning of the field where the note is attached. This is a zero based rich text offset; i.e., carriage returns are not counted.
LINKLENGTH	The length of the text at LINKOFFSET that contains the annotation. This is a rich text length. Carriage returns are not counted.
REPLICATION	The replication field that tracks edits.

All values must be passed as text, including dates and numbers. The function will make the appropriate conversion to store them in the database.

Dates must be passed in YYYYMMDD format, no punctuation, no spaces, fully padded and zero filled to eight characters.

Return Value

Returns the number of bytes copied.

See Also: annotationAppend(), annotationCount(), annotationDelete(), annotationGoto(), annotationIsTagged(), annotationRetrieve(), annotationTag(),

Version

Concordance version 7.0 and later.

append

Summary

```
int append(int handle);
```

Description

Appends a document to the end of the database. This document becomes the current document. Use append() with blank() to create and add a record to the end of the database. Append() used without blank() will duplicate the current record.

The Network Edition of Concordance will automatically lock the end of the file while it adds the document to the database. The document is locked when the append finishes.

Return Value

A -1 if unsuccessful.

See Also: blank()

asc

Summary

```
int asc(text info);
```

Description

Converts a character to the integer ASCII code that represents it. In the case of text fields, text variables, and unsubscripted character arrays, asc() returns the character code of the first character.

This function allows you to directly compare individual characters with numeric values and character constants. Integer comparisons can run faster than string comparisons.

Return Value

ASCII code that represents the character. The ASCII code is in the range 0 to 255. asc() will not return negative numbers.

See Also: chr()

Example

```
DoSomething(int db)
{
int code;
 if (asc(upper(db->sex)) == 'M')
  code = DoThis(db);
 else
  code = DoThat(db);
 return(code);
}
```

B

The following topic discusses the Concordance Programming Language (CPL) functions that begin with the letter B. For more information on CPL functions, see Functions, About the Advanced Programming Features, and About CPL Functions.

beep

Summary

```
beep(int frequency, duration);
```

Description

Emits a tone from the PC's speaker. Frequency is a value in hertz. Duration is the length of time in thousandths of a second. Beep will not make a sound if the beep option on the Set menu is disabled.

Return Value

None.

Example

```
main()
{
int i;
 /* Sound a siren, up scale and down scale. */
 /* Use a for-loop for the up scale tone, a */
 /* while loop for the down scale tone. */
 for(i = 450; i < 550; i = i + 1)
  beep(i,5);
 while(i > 450)
  beep(i = i - 1,5);
}
```

blank

Summary

```
blank(int db);
```

Description

Clears a document at the end of the file. The current document is not affected. Once a document is cleared it can be edited and appended to the database.

Return Value

None.

See Also: append()

Example

```
main()
```

```
{
int db1, db2, i;
 /* Copy records to another database. */
 db1 = opendb("recipes");
 db2 = opendb("cooking");
 /* Get a blank document.
 ** Copy the individual fields.
 ** Append the new document.
 ** Get the next record to copy.
 */
 cycle(db1) {
  blank(db2);
  for(i = 1; i <= db1.fields; i = i + 1)
   db2->i = db1->i;
  append(db2);
 }
 /* All done, close the databases. */
 closedb(db1);
 closedb(db2);
}
```

box

Summary

```
box(int  row,
 col,
 brow,
 bcol,
 format,
 [color,[background]]);
```

Description

Displays a box at the location described by the screen coordinates of the upper left corner and the lower right corner. The area inside the box is not cleared. Format is either 'S' or 'D', for a single line box or a double line box, case is ignored. If the format is SD or DD the box is created with a drop shadow, the dimensions of the box are decreased by two columns and one row for placement of the shadow. Drop shadows are not available in the Windows version, but the extra line and columns are accounted for before the box is drawn.

Windows will display a raised 3-D box if the 3U format style is selected and a depressed 3-D box if the 3D format style is selected. The background color is used to fill the box. A 3D box will produce a DD box in non-Windows versions of Concordance.

The color and background color parameters are optional. Color selected for normal text is used if the color parameter is left off. The background color is only used for Windows.

Return Value

None.

See Also: scroll()

browse

Summary

```
int browse(int db);
```

Description

Invokes the Concordance full screen browse mode. Browse will display the current document in the current query. The screen is automatically saved before entering this mode and restored after exiting.

Return Value

The key pressed to exit Browse mode. Either the [Esc] key or a function key.

See Also: table()

Example

```
main()
{
int db;
 db = opendb("recipes");
 if (db == -1) {
  puts(0,0,"Can't open recipes.");
  getkey();
 }
 else {
  search(db, "chocolate");
  browse(db);
  closedb(db);
 }
}
```

btclose

Summary

```
int btclose(int handle);
```

Description

Closes the dictionary file associated with handle.

Return Value

btclose returns 0 if successful, or -1 if an error was encountered.

See Also: btopen(), btcreate()

Example

```
ShowCount()
{
int i;
```

```
 i = btopen("recipes.dct");
 if (i == -1)
  puts(0,0,"Couldn't open the dictionary");
 else {
  puts(0,0,"There are "+
   str(btcount(i),6,0)+
   " words.",);
  btclose(i);
 }
}
```

btcount

Summary

```
int btcount(int handle);
```

Description

Returns a count of the number of keys in the file. Handle must be a valid handle returned by a call to btopen() or btcreate().

Return Value

The count of keys in the file, or -1 if an error occurs.

See Also: btopen(), btcreate()

btcreate

Summary

```
int btcreate(char string[]; int duplicate);
```

Description

Creates a new dictionary for use, will not erase or open an existing file. If duplicate is a nonzero value, duplicate key values will be allowed in the file. A zero value will reject duplicate keys.

The duplicate parameter is stored with the dictionary and will remain active when the file is reopened later. It cannot be changed once the file is created.

Return Value

btcreate() will return a file handle if successful, a -1 if it was unable to create the file. Reasons for failure are the file already exists, bad file name or directory path, diskette write protected, disk full.

See Also: btopen(), btclose()

btcycle

Summary

```
int btcycle(int file, dbHandle; text expression);
```

Description

Cycles through the database using the current query. Evaluates expression for each document in the database and inserts the result into the dictionary whose handle is file. The document's number is stored as the data value. The result of expression must be a text value.

The expression can be a database field, db->NAME, or a quoted string for evaluation, "pad(db->NAME,'L',40)+str(db->DATE)". The function will run faster with a simple field than it will with a quoted parameter that requires evaluation. Fixed fields will run the fastest.

Return Value

The number of entries placed into the dictionary. This number may not be the number of documents in the current query if duplicates are not allowed in the dictionary.

Example

```
ShowUniqueEntries(int db)
{
char string[80]; int document;
int dict = btcreate("erase.me",0);
 /* Create the list and show entries in a menu. */
 btcycle(dict,db,db->AUTHOR);
 while(btmenu(dict,7,0,22,79,"Authors",string,document))
 {
  /* Load the selected document and browse. */
  goto(db,document);
  browse(db);
  /* Get field to display at top of btmenu(). */
  string = trim(db->AUTHOR);
 }
 btclose(dict);
 erase("erase.me");
}
```

btdelete

Summary

```
int btdelete(int file; char key[]);
```

Description

Deletes the matching key from the dictionary.

Return Value

A 0 if successful, a -1 if not successful. Returns a -1 if the word was not in the file.

Example

```
remove(int btree; text string)
{
```

```
int key;
text line;
 /* Enclose string in quotes and ask */
 /* if the user is sure they want to */
 /* delete it from the file. */
 line = 'Delete "'+string+'"?';
 puts(0,0,line);
 cursor(0,len(line)+1);
 key = getkey();
 if ((key == 'Y') or (key == 'y'))
  btdelete(btree,string);
}
```

btexact

Summary

```
int btexact(int file; char key[]; int data);
```

Description

Locates a matching entry in the file. The entry must match both the key value and the data value.

Return Value

A zero if the entry was found, a nonzero value if an exact match could not be located.

See Also: btfirst(), btfind(), btgte(), btgt(), btlt(), btlast()

btfind

Summary

```
int btfind(int file; char key[]; int data);
```

Description

Locates the word and, if found, returns the associated integer value in data. The file handle must be a valid handle returned by a call to either btopen() or btcreate().

Return Value

The value of btfind() is 0. If the word was not in the file, the value of btfind() is -1.

See Also: btexact(), btfirst(), btgte(), btgt(), btlt(), btlast()

btfirst

Summary

```
int btfirst(int file; char key[]; int data);
```

Description

Locates the first word in the dictionary file and returns its integer value in "data." The word

is returned in key[], key[] must be large enough to hold the retrieved word.

Return Value

The value of btfirst() is 0. If the file is empty, the value of btfirst() is -1.

See Also: btlast()

btgt

Summary

```
int btgt(int file; char key[], buf[]; int data);
```

Description

Locates the dictionary entry greater than the search key. The word is returned in buf[], buf[] must be large enough to hold the retrieved word.

Return Value

A value less than zero if an error occurs, or if there is no value greater than the search key. If an entry is found, the entry is returned in buf[], its associated numeric value is returned in data.

See Also: btlt(), btgte()

btgte

Summary

```
int btgte(int file; char key[], buf[]; int data);
```

Description

Locates the dictionary entry greater than or equal to the search key. The entry is returned in the buf[] variable, and the key's value is returned in data.

Return Value

A -1 if an error occurs.

See Also: btgt(), btlt(), btfind()

btinsert

Summary

```
int btinsert(int handle; char key[]; int data);
```

Description

The key value is inserted into the dictionary along with the numeric value in data.

Return Value

A 0 indicates success, a value less than 0 indicates an error. If a negative value is returned, it may indicate either a disk full condition or a duplicate key value if duplicate keys are not

allowed.

See Also: btcreate(), btinserta(), btopen(), btclose()

Example

```
update(char string[]; int handle)
{
int UpperCase;
 /* Assume lower case word. */
 UpperCase = 0;
 if (isupper(string[0]))
 {
  /* Capitalized word. */
  UpperCase = 1;
  if (isupper(string[1]))
   /* Mixed case word. */
   UpperCase = 2;
 }
 /* Insert word in upper case, but */
 /* store the case indicator as the */
 /* data value. This can be used */
 /* later in a spell check program. */
 return(btinsert(handle, upper(string), UpperCase));
}
```

btinserta

Summary

```
int btinserta(int handle; char key[]; int data);
```

Description

The key value is inserted into the dictionary, however the dictionary will assume that what you are inserting is already sorted in ascending order. It will use an insertion scheme that optimizes for ascending order insertion and will produce a more compact dictionary.

Return Value

A 0 if the key was successfully added to the dictionary, a -1 if it was not.

See Also: btinsert(), btcreate(), btopen()

Example

```
main()
{
int OldError, NewError, old, new, data;
char buffer[60];
 new = btcreate("temp.xyz",0);
 old = btopen("oldfile.xyz");
 /* Read the existing dictionary and insert */
```

```
 /* all data into the new dictionary. */
 NewError = 0;
 OldError = btfirst(old,buffer,data);
 while((OldError <> -1) and (NewError <> -1)) {
  NewError = btinserta(new,buffer,data);
  OldError = btnext(old,buffer,data);
 }
 /* If the key counts are not the same in
 ** both files it means that an error occurred
 ** at some point, probably a full disk.
 */
 if (btcount(old) <> btcount(new))
  NewError = -1;
 /* Close the files and finish. */
 btclose(old);
 btclose(new);
 if (NewError == -1) {
  puts(0,0,"Error in new dictionary.");
  puts(1,0,"Disk may be full.");
  erase("temp.xyz");
 }
 else {
  erase("oldfile.xyz");
  rename("temp.xyz","oldfile.xyz");
 }
}
```

btlast

Summary

```
int btlast(int handle; char key[]; int data);
```

Description

Read the last entry in the dictionary, return the entry and its numeric value. The file is left positioned on the last entry after this call. A series of btprev() calls would then read the file in descending alphabetical order.

Return Value

A 0 if successfully read, a nonzero value if there was an error.

See Also: btfirst(), btprev(), btnext()

btlock

Summary

```
int btlock(int handle);
```

Description

Locks the btree file for exclusive use. All other network users are prevented from accessing the file. The file should be unlocked as quickly as possible to ensure access for other

network users.

Return Value

Returns a nonzero value if the file could not be locked.

See Also: btunlock()

btlt

Summary

```
int btlt(int handle; char key[], buf[]; int data);
```

Description

Locates the key entry less than the passed key. If an entry is found, it is returned in buf[] with its associated numeric value returned in data. The character string buf[] must be large enough to hold the returned value.

Return Value

Returns a -1 if an error occurred.

See Also: btprev(), btgt()

btmenu

Summary

```
int btmenu(int handle, row, col, brow, bcol; char title[], key[]; int data
```

Description

Displays the contents of the dictionary whose handle was returned earlier by a call to btopen() or btcreate(). The key values are displayed in a menu window. The title string is displayed at the top of the window.

The user can cursor through the dictionary, or type the letters of the word for lookup. As the letters are typed, btmenu() will display the list of words that match most closely.

The action parameter is optional. "U" or "u" converts all of the user's typed input to upper case. An "L" or "l" converts the input to lower case. Specifying an "E" or "UE" or "LE" enables the Insert and Delete buttons in the Windows version.

The user can select the current word by pressing the [Enter] key. btmenu() will copy the selected word to the key[] parameter, and the word's associated numeric value to the data parameter.

If the user presses [Ctrl-Enter], btmenu() will copy the word they are typing on the prompt line to the key[] parameter. Windows users should pass the "E" parameter mentioned above.

If key[] contains a value when btmenu() starts, then the key greater than or equal to that key becomes the first key displayed on the screen.

Return Value

A value of -1 indicates an error in the dictionary file.

A 0 return value indicates that the user pressed [Esc] to exit without making a selection.

A 1 indicates that the user pressed [Enter] to select a key, that key is returned in the key[] and data parameters.

A 2 indicates that the user pressed [Ctrl-Enter] to select the value they were entering on the prompt line, in this case key[] will contain the prompt line string.

btnext

Summary

```
int btnext(int handle; char key[]; int data);
```

Description

Locate the next entry in the file. The key and data parameters are set to the value of the key.

Return Value

A -1 is returned if there is no next key.

See Also: btprev(), btfirst()

btopen

Summary

```
int btopen(char string[]);
```

Description

Open a dictionary file for use.

Return Value

-1 if the file cannot be found, or if it is not recognized as a valid dictionary file.

See Also: btclose(), btcreate()

btprev

Summary

```
int btprev(int handle; char key[]; int data);
```

Description

Locates the previous entry in the file and returns it in key, and its numeric value in data.

Return Value

A -1 is returned if there is no previous entry.

See Also: btlast(), btnext(), btfirst()

btrebuild

Summary

```
int btrebuild(text filename);
```

Description

The btree file named by filename is converted to a Concordance 8 b+tree with 64-bit data values. The old b+tree is renamed with the .old file extension. The file must be closed for the conversion to work.

After the conversion, only Concordance V8 or later can open the b+tree.

Return Value

A -1 is returned if the file is open or if it is not a b+tree, otherwise the number of keys in the converted b+tree is returned.

Version

Concordance version 8 and later.

btunlock

Summary

```
int btunlock(int handle);
```

Description

Unlocks the file allowing other network users to access it. A file must be unlocked exactly as many times as it was locked to completely release it.

Return Value

A nonzero value if an error occurred while unlocking the file.

See Also: btlock()

C

The following topic discusses the Concordance Programming Language (CPL) functions that begin with the letter C. For more information on CPL functions, see Functions, About the Advanced Programming Features, and About CPL Functions.

capitalize

Summary

```
text capitalize(text string);
```

Description

Produces a duplicate of the parameter string capitalized. All words will begin with an upper case letter, all non-initial letters are lower case. Does not change the original.

Return Value

The parameter with capitalized text.

See Also: upper(), lower()

Example

```
main()
{
text author;
 author = "e e cummings";
 puts(0,0,capitalize(author));
 puts(1,0,author);
}
```

Output:

E E Cummings

e e cummings

ccol

Summary

```
int ccol();
```

Description

Current column position of the cursor.

Return Value

Cursor's column position.

See Also: crow(), cursor()

chdir

Summary

```
text chdir(text directoryPath);
```

Description

Changes the current drive and directory to the one specified by directoryPath.

Return Value

A zero if successful, non-zero if the directory or drive does not exist.

See Also: diskspace(), getcwd()

Example

```
main()
{
 /* Change to drive E: *\
 chdir("E:");
 /* Change directories. */
 chdir("concord5\DATA");
 /* Change drive and directories. */
 chdir("F:\apps\concord5");
 /* Display the current drive and path. */
 puts(10,40,"The current directory is "+getcwd());
}
```

chr

Summary

```
text chr(int c);
```

Description

Converts the value of c to a text variable.

Return Value

The single character converted to a text type variable.

See Also: asc()

Example

```
main
{
int handle;
text line;
 /* Send a form feed to the printer. */
 if ((handle = open("prn","w")) <> -1) {
  line = "Page ejected..."+chr(12);
  write(handle,line,len(line));
  close(handle);
 }
}
```

clock

Summary

```
int clock();
```

Description

The clock() function gets the number of thousandths of a second since the program was

started.

Return Value

Returns -1 if the clock is not available.

See Also: time(), today()

Example

```
main()
{
int i, start;
 start = clock();
 for(i = 0; i < 100; i = i + 1)
  puts(0,0,"Count ="+str(i,7,0,','));
 start = (clock() - start) / 1000;
 puts(1,0,"That took "+str(start,7,0,',')+" seconds.");
 getkey();
}
```

close

Summary

```
int close(int handle);
```

Description

The file associated with handle is closed. Handle must be a value returned earlier by a call to open().

Return Value

A -1 if an error is encountered while closing the file.

See Also: open()

closedb

Summary

```
int closedb(int db);
```

Description

Data base associated with db handle is closed.

Return Value

A 0 if successful, -1 if not successful.

Example

```
main()
```

```
{
int db;
 db = opendb("recipes");
 if (db >= 0) {
  puts(0,0,"There are "+str(db.documents,8,0,',')+" recipes in the cookboo
  if (closedb(db) == -1)
   puts(0,1,"Error closing database.");
 }
 else
  puts(0,0,"Can't open database.")
 getkey();
}
```

cls

Summary

```
cls([int color]);
```

Description

Clears the screen to TextColor_, the default color, or to the specified color if one is passed as a parameter. Under Windows, the color should be a background color.

Return Value

None.

See Also: scroll()

Example

```
colors()
{
int i, oldColor;
 /* Show the basic set of colors. Display the */
 /* color's number centered in the screen. */
 oldColor = TextColor_;
 for(i = 1; i <= 127; i = i + 1) {
  TextColor_ = i;
  cls();
  puts(MaxRow_/2,0,pad("Color "+str(i),'C',80));
  getkey();
 }
 TextColor_ = oldColor;
}
```

concat

Summary

```
concat(int db; text FileName);
```

Description

Concatenates a database to the opened database whose handle is db. FileName should be the path and name of a database file to concatenate.

Return Value

Zero if the database was concatenated, nonzero if an error occurred. Cause for error includes: too many concatenated databases, the maximum is sixteen; the requested database is already concatenated; file not found; database is in exclusive use by another user.

See Also: concatclear()

concatclear

Summary

```
concatclear(int db);
```

Description

All databases concatenated to the database whose handle is db are closed. The database is no longer concatenated after this call returns. All queries are reset and cleared by this call.

Return Value

None

See Also: concat()

count

Summary

```
int count(int db);
```

Description

Returns the number of documents in the current query. db should be a valid handle returned by a call to opendb(). To find the total number of documents in the database, use:

x = db.documents;

Return Value

Count of documents in the current query. A value less than zero if the database is not open.

See Also: hits()

createdb

Summary

```
int createdb( text  FileName;
  text  FieldNames[];
  int  types[];
  int  lengths[];
```

```
  int  places[];
  int  formats[];
  int  keyField[]);
```

Description

Creates a new database with the requested file name and structure. This function will not create the database if another database already exists by the same name.

The fields are defined by giving them a name, a type, a length, and optionally places and format entries. The type can be 'T', 'D', 'N', or 'P', for Text, Date, Numeric, and Paragraph field types respectively.

The length entry designates the total width of the field for text and numeric fields. Length indicates the display format for date fields, MDY, YMD, or DMY. It sets the left margin of paragraph fields.

The places array is only used for numeric field types, and its presence is optional. The format array can contain numeric display formats Z, $, or "," for zero filled, currency and comma numeric types. If the format array is passed, the places array must also be provided.

The keyField array specifies the key fields in the database. A non-zero entry creates a key field.

All rules that apply to creating databases in the Concordance full-screen Database Create mode apply to this function. See the reference manual for a full explanation.

This function is not implemented in the Concordance Runtime Module. It will always return a creation error in the Runtime module.

Return Value

Zero if the database structure was valid and a database was created. A nonzero return value is an error code in the reference manual error listings, or in the concordp.msg or runcpl.msg file. This function only creates the database, it does not open it.

See Also: createfs(), struc(), modify(), opendb()

Example

```
/* Create a new database. */
MakeNew()
{
char string[80];
int err, newHandle;
text names[4];
int lengths[4], types[4], key[4];
 newHandle = -1;
 getfile("New Database", "*.DCB", string);
 if (len(string) <> 0)
 {
  names[0] = "BIRTHDATE"; types[0] = 'D';
  lengths[0] = 'M';   key[0] = 1;
  names[1] = "EARNINGS"; types[1] = 'N';
  lengths[1] = 10;   key[1] = 0;
```

```
  names[2] = "NAME";  types[2] = 'P';
  lengths[2] = 16;    key[2] = 0;
  names[3] = "AUTHORED"; types[3] = 'P';
  lengths[3] = 16;    key[3] = 0;
  err = createdb(string,names,types,lengths);
  if (err == 0)
   newHandle = opendb(string);
 }
 return(newHandle);
}
```

createfs

Summary

```
int createfs();
```

Description

Concordance full screen create command. It first prompts the user for a database name, then presents the database create screen. See the Concordance Reference Manual for a full description of database create.

This function is not implemented in the Concordance Runtime Module. It will always return an error when called from the Runtime module.

Return Value

A handle to the newly created and opened database. A value less than zero if the user canceled the database create.

See Also: modify(), createdb(), struc()

createReplica

Summary

```
int createReplica(int db; text databaseName; int copyAttachments);
```

Description

Creates a replica database using all records in the current query, in the current sorted order. The replica has all security settings and document tags in the current database, but it is not indexed. The optional parameter, copyAttachments, will copy attachments with the database if set to TRUE.

Return Value

A zero signals success

See Also: replicate(), resolve()

crow

Summary

```
int crow();
```

Description

Current cursor row.

Return Value

Row position of the cursor.

See Also: ccol(), cursor()

ctod

Summary

```
int ctod(char string[][; char format]);
```

Description

Convert character string or text variable to date format. The format, if specified, can be 'Y', 'M' or 'D' to select YYMMDD, MMDDYY and DDMMYY formats of the string parameter. The default is 'M' format if none is specified.

Return Value

The date as an integer.

See Also: dtoc()

Example

```
days()
{
int DaysBefore2000;
 DaysBefore2000 = ctod("1/1/2000") - today();
 return(DaysBefore2000);
}
```

cursor

Summary

```
cursor(int row, column);
```

Description

Moves the cursor to the specified row and column coordinates. The screen begins in the upper left corner with row 0, column 0. An 80 column screen goes from row 0 to row 24, and from column 0 to column 79.

Return Value

None.

See Also: ccol(), crow(), cursoron(), cursoroff()

cursoroff

Summary

```
cursoroff();
```

Description

Hides the cursor.

Return Value

None.

See Also: cursoron()

Example

```
main()
{
 cursoroff();
 puts(0,0,"The cursor has disappeared.");
 puts(1,0,"Hit any key to show the cursor.");
 getkey();
 cursoron();
}
```

cursoron

Summary

```
cursoron();
```

Description

Enables the display of the cursor.

Return Value

None.

See Also: cursoroff()

cut

Summary

```
text cut(text data);
```

Description

Copies the data to the cut and paste buffer. The data replaces the current contents of the buffer. It remains in the buffer until another cut() replaces it. The buffer is shared by the programming language with the editor's cut & paste functions. Cutting text with the editor

will also flush the buffer and replace the data.

The Windows version cuts to the system clipboard. Note that other programs have access to the clipboard, and they can clear or replace the data.

Return Value

The text value that was cut.

See Also: paste()

D

The following topic discusses the Concordance Programming Language (CPL) functions that begin with the letter D. For more information on CPL functions, see Functions, About the Advanced Programming Features, and About CPL Functions.

day

Summary

```
int day(int birthday);
```

Description

Extacts the day of the month from the date. The date can be a numeric representation of the date, as returned by ctod(), or a date field.

Return Value

The day of the month (1 - 31).

See Also: month(), year(), weekday()

Example

```
oneHundred(int billday)
{
text line;
 billday = billday + 100;
 puts(0,0,"100 days overdue is:");
 line = str(day(billday),2,0,'Z')+"/"+str(month(billday)2,0,'Z')+"/"+str(
 puts(1,0,line);
}
```

dc

Summary

```
text dc(text string);
```

Description

dc() is used within a sort parameter string to specify a descending order sort. Enclose the entire sort string, or specific fields, within the dc() function to perform the descending or mixed order sort. Any data not enclosed within the dc() function is sorted in ascending order.

Return Value

The parameter string inverted for a descending order sort.

See Also: sort()

Example

See sort() for an example.

ddeConnect

Summary

```
int ddeConnect(text Service, text Topic);
```

Description

Begins a dynamic data exchange (DDE) conversation with another Windows application. Service is the name of an application which can respond to DDE messages. This is usually the program name without the .EXE extension. Topic is the name of a topic recognized by the server. This function is available only in Concordance for Windows.

Return Value

Returns a nonzero handle to that conversation. An error can occur if the application is not running, or if it is running but does not recognize the topic or support DDE.

See Also: ddeDisconnect(), ddeExec(), ddePoke(), ddeRequest()

Example

```
makeWorksheet()
{
int ddeHandle, i;
text topicList, sheetName;
 /* Establish a link to the spreadsheet. */
 ddeHandle = ddeConnect("Excel","System");
 /* Create a new work sheet. */
 ddeExec(ddeHandle,"[New(1)]");
 /* Retrieve a list of available spreadsheets. */
 topicList = ddeRequest(ddeHandle,"Selection");
 /* Disconnect */
 ddeDisconnect(ddeHandle);
 /* Isolate the name of the first spreadsheet. */
 /* Establish a new conversation based on the spreadsheet, */
 /* this will allow us to modify the spreadsheet.*/
 sheetName = substr(topicList,1,match(topicList,"!",1) - 1);
 ddeHandle = ddeConnect("Excel",sheetName);
 /* Fill it with data. */
```

```
for(i = 1; i < 10; i = i + 1)
 ddePoke(ddeHandle,"R1C"+str(i),str(i));
/* Make a chart. */
ddeExec(ddeHandle,'[Select("R1C1:R1C10")]');
ddeExec(ddeHandle,'[New(2,1)]');
/* Terminate the DDE conversation. */
ddeDisconnect(ddeHandle);
```

ddeDisconnect

Summary

```
ddeDisconnect(int ddeHandle);
```

Description

Disconnects from the dynamic data exchange (DDE) conversation previously opened by using ddeConnect().

Return Value

None.

See Also: ddeConnect(), ddeExec(), ddePoke(), ddeRequest()

ddeExec

Summary

```
ddeExec(int ddeHandle, text command);
```

Description

Uses an open dynamic data exchange conversation handle to send a command to another application. The command depends on the application and topic specified when the conversation was initiated.

Return Value

Success value returned by the server. Generally 0 means the ddeExec() was not processed, 16384 means the server was busy, and 32768 signifies success.

See Also: ddeConnect(), ddeDisconnect(), ddePoke(), ddeRequest()

ddePoke

Summary

```
ddePoke(int ddeHandle, text item, text data);
```

Description

Sends data to the DDE server application. The item is a value, such as a spreadsheet row and column or database field name, which is recognized by the server. The data is a string containing the data sent to the other application.

Return Value

Success value returned by the server. Generally 0 means the ddeExec() was not processed, 16384 means the server was busy, and 32768 signifies success.

See Also: ddeConnect(), ddeDisconnect(), ddeExec(), ddeRequest()

ddeRequest

Summary

```
ddeRequest(int ddeHandle, text item);
```

Description

Sends a dynamic data exchange (DDE) request to the open conversation. This generally retrieves data from the server, though the action taken is dependant on the server, the topic, and the item.

Return Value

The DDE server must return a text value.

See Also: ddeConnect(), ddeDisconnect(), ddeExec(), ddePoke()

Example

```
int dh;
text listOfFields, field, fieldData;
 /* Establish a link to the database */
 dh = ddeConnect("Concordance","Resumes");
 /* Retrieve a list of fields in the database. */
 listOfFields = ddeRequest(dh,"List of Fields");
 /* Retrieve data from the first field */
 field = substr(listOfFields,1,match(listOfFields,',',1)-1);
 fieldData = ddeRequest(dh,field);
 /* Disconnect from the database */
 ddeDisconnect(dh);
```

debug

Summary

```
int debug(int mode);
```

Description

Turns single step debugging on or off. Mode should be 1 to turn debugging on, 0 to turn it off, and -1 to disable it. Debugging can also be turned on and off by pressing [Alt-F1], unless debug(-1) has disabled it.

Return Value

Value of previous setting.

delete

Summary

```
delete(int db);
```

Description

Marks the current document for deletion. The document isn't removed from the database until it is packed.

Return Value

None.

See Also: deleted(), recall(), pack()

isdeleted

Summary

```
delete(int db);
```

Description

Marks the current document for deletion. The document isn't removed from the database until it is packed.

Return Value

None.

See Also: deleted(), recall(), pack()

deleteText

Summary

```
int deleteText(db->FIELD; int offset; int length)
```

Description

The first parameter identifies a field in the database. The deleteText() function deletes a block of text at offset for length number of bytes. This function preserves rich text formatting and properly processes annotations. Any annotations contained by the block of deleted text are also deleted.

Return Value

The number of bytes deleted.

See Also: insertText()

Version

Concordance version 7 and later.

diskspace

Summary

```
int diskspace(int drive, totalSpace, freeSpace);
```

Description

Determines the amount of total and free space on the drive specified. The drive should be a letter, such as 'A', 'C', etc.

Return Value

Zero if successful, non-zero if an error occurred or the drive does not exist.

Upon returning, the variables totalSpace and freeSpace will contain the total bytes available on the disk and the free space remaining, respectively. Allocated space can be calculated with:

```
int allocatedSpace, totalSpace, freeSpace;
 if (diskspace('C',totalSpace,freeSpace) == 0)
  allocatedSpace = totalSpace - freeSpace.
```

See Also: chdir(), getcwd()

Version

Version 5.32 and later

docno

Summary

```
int docno(int db, document, field, offset);
```

Description

Returns the relative document number in the current query. After issuing a first(), docno() would return the number 1, even though the first document retrieved in the query may be the 25th document in the database. Docno() always returns a number relative to the current query set, not a physical document number in the underlying database.

db must be a value returned by a call to opendb().

If the optional parameters document, field, and offset are provided they will contain the same information documented in, and returned by, the first() function.

Return Value

A relative document number. Returns a value less than or equal to zero if there are no documents in the current query set.

See Also: goto(), first(), last(), prev(), next(), prevhit(), nexthit(), recno()

Example

```
main()
{
int db, document;
 db = opendb(" recipes.dat");
 if (db <> EOF) {
```

```
  search(db," chocolat*");
   goto(db,5);
  document = docno(db);
  /* document now equals 5, the fifth
   document located in the search.
   The document's physical position
   in the underlying database may
   be almost any number. */
 }
}
```

dtoc

Summary

```
text dtoc(int birthday[; char format]);
```

Description

Converts the date value to a character string. This function will return the exact date contained in a date field, even if the date is invalid, i.e., 12/00/81 or 99/99/9999. To test for a valid date use:

```
if (dtoc(db->DATE) <> dtoc(db->DATE+0))
  badDate(db);
else
  goodDate(db);
```

The optional format parameter can be 'Y', 'M' or 'D' to indicate the date format for conversion: YYMMDD, MMDDYY, or DDMMYY. The default format 'M' is used if the parameter is not explicitly set.

Return Value

A character string representing the date.

See Also: ctod()

Example

```
whatDayIsToday()
{
 puts(12,35,"Today is "+ dtoc(today()));
}
```

E

The following topic discusses the Concordance Programming Language (CPL) functions that begin with the letter E. For more information on CPL functions, see Functions, About the Advanced Programming Features, and About CPL Functions.

edit

Summary

int edit(data-value; int TR, TC, BR, BC; text mode;

...

...

int element; text iMode; int SO, EO, color);

Description

Edits a data-value, which can be a database field, char array, integer, floating point value, or date, in an edit window. The window is described by its top row, TR, and column, TC, and its bottom corner row, BR, and column, BC. The mode string tells the edit function how to edit the data. Mode attributes are described later. Multiple data values can be passed to the edit function in one call.

The last line in edit() uses additional variables to tell edit() which element to edit first, what initial mode to use, what offset into the data to start editing, and what color to use.

element is the data element to edit first. If multiple values are passed you can tell edit() to start editing with any selected value, instead of the first. Edit elements are numbered starting with one at the top. Setting element to a value greater than the total number of elements will cause edit() to use the last element. Upon return, element contains the current number of the element being edited.

The iMode parameter specifies the edit attribute to use for this first element the initial time it is edited. See the modes below for additional information.

SO and EO are the Screen Offset and Edit Offset. These are used for full text paragraph fields. They specify the offset into a full text field to begin displaying data, and the offset into the field to place the cursor when edit() is first called. Upon return, they contain the current offsets of the user's cursor.

The last parameter, color, is the color used to edit all values passed to the edit() function.

The following mode string attributes are available:

A	Alpha only mode, numeric values are rejected.
U	Upper case conversion.
N	Numeric only mode, for text or char array variables.
N:w.d	Numeric data is edited in a field w characters wide, with d decimals.
Y	YMD mode for dates, can be a date field or integer value.
M	MDY mode for dates, can be a date field or integer value.
D	DMY mode for dates, can be a date field or integer value.

C	Cut and paste mode. This mode allows the user to move with the cursor control keys, mark text with [Alt-M], and copy it to the cut and paste buffer using the [Enter] key. Use the paste() function to access the marked text.
S	Scroll field left and right, no wordwrapping. Use for fixed fields and char arrays.
E	Return on [Enter] key, no hard carriage returns allowed in data. Useful with fixed fields and char arrays.
T	Always edit data starting from the top. If T and B are not specified, edit() assumes you wish to continue editing using the screen and edit offset parameters.
B	Edit data starting from the bottom. This mode is useful as an initial mode value overriding default value. If T and B are not specified, edit() assumes you wish to continue editing using the screen and edit offset parameters.
@	Display only this value, no editing allowed.
!	Return immediately when this field is entered, don't edit it. This is how required authority lists or other special handling can be added to the edit function. Note that the element parameter must be changed upon calling edit() again or an endless loop is created when edit() immediately returns again and again.
Button	Creates a button under Windows. Returns CR if selected.

The following additional keys can be used while editing:

[Alt-M] will begin marking text for cut and paste. Pressing [Alt-M] again will unmark the text.

[Shift-Del] copies marked text to the cut and paste buffer. This requires [NumLock] to be off.

[Shift-Ins] copies text from the buffer to the edit window. This requires [NumLock] to be off.

[Del] deletes marked text and copies it to the paste buffer.

[Ctrl-Y] deletes text from the cursor up to and including the end of the line. [Ctrl-Y] ignores marked text and does not save the deleted text in the cut and paste buffer.

Return Value

Edit will automatically process all cursor control keys that move between edit elements, unless the user tries to cursor above the first element or below the last element. Edit returns the keystroke that caused termination. In the case of mode @, display mode, the returned value is always a 0.

Upon returning, the parameter element will be set to the number of the data element currently being edited. The parameters SO, Screen Offset, and EO, Edit Offset, will contain the offsets into the text of the line displayed at the top of the current window and of the character under the cursor, respectively.

The value 16, a [Ctrl-P], is returned if the user types past the last character in a field.

The value -1 indicates an insufficient memory error.

See Also: getline(), getnumber(), show()

Example

```
/* name:  editPhone
** synopsis: Edits a phone number in 999/999-9999 format.
*/
editPhone(int db)
{

char  areacode[4],
  prefix[4],
  phone[5];
int  key,
  CR = 13,  /* Value of [Enter] key. */
  ESC = 27,  /* Value of [Esc] key. */
  element = 2,  /* Element to edit first. */
  SO = 1,   /* Screen offset */
  EO = 1;   /* Edit offset */

 /* Get the three elements of the phone number, areacode, prefix, and phoı
 areacode = trim(substr(db->PHONE,1,3));
 prefix  = trim(substr(db->PHONE,5,3));
 phone  = trim(substr(db->PHONE,9));


 /* Edit them until [Enter] or [Esc] is hit. */
 while((key <> ESC) and (key <> CR))
  key = edit("Phone:", 3, 1, 3, 7, "@",
   areacode, 3, 8, 3, 10, "NET",
   "/",  3, 11, 3, 11, "@",
   "prefix,  3, 12, 3, 14, "NET",
   "-",  3, 15, 3, 15, "@",
   "phone,  3, 16, 3, 19, "NET",
   db->NAME, 2, 8, 2, 60, "@",
   element, "", SO, EO, TextColor_);

 /* If the user pressed [Enter], save the number. */
 if (key == CR)
  db->PHONE = pad(areacode,'L',3)+"/"+pad(prefix,'L',3)+"-"+pad(phone,'L',
 return(key);
}
```

edited

Summary

```
int edited(int db);
```

Description

Determines if the paragraph fields in the current document have been edited or appended to the database since the last time the database was reindexed. Documents located in a full text search that have been edited may not actually match the search conditions. Attempting to highlight the keywords on the screen, or to underline them on a printer, may result in the wrong words being highlighted.

Return Value

A nonzero value if the document has been edited or modified since the last reindex.

editfs

Summary

```
editfs(int db);
```

Description

Invokes the Concordance full screen editor. The current document in the current query is edited. Users can change documents, but they cannot change queries from the editor. All fields in the database are edited. The screen is automatically saved before entering this mode. It is restored upon exiting.

Return Value

None.

erase

Summary

```
int erase(char string[]);
```

Description

Erases the file. The file name can be a character array or a text variable.

Return Value

If successful, erase returns a 0. Otherwise it returns a -1; erase() will not delete a read-only file or a directory.

Example

```
remove(char FileName[])
{
 if ( erase(FileName) == -1)
  puts(0,0,"Couldn't erase "+ FileName+".");
 else
  puts(0,0, FileName+" has been erased.");
}
```

eval

Summary

```
eval(text string; int code, line);
```

Description

Executes the Concordance expression in the string and returns the result as the value of eval(). This function allows end users to enter commands into strings and have them

evaluated by the command processor.

This function is useful in allowing the user to enter a report format, field, or combination, and passing them to another function for report creation. Local variables declared in the function which executes eval() are accessible to the eval() function.

Return Value

The value and type returned by eval() is the result of the string that is evaluated. For instance, if the string contains "2+2" eval() will return an integer. However if the string contains "weekday(today())" eval() returns the day of the week as a character string. Enclosing the eval() function within the str() function guarantees that the result is always of type text.

The second parameter, code, will contain an error code if the expression could not be evaluated. Otherwise it will contain zero. The error code corresponds to one of the numbered error messages in this manual.

The third parameter, line, is optional. It returns the line number that produced the error. This is useful when using eval() within a run() function to determine the error code and line number which produced the error. Using eval() to execute a run() also keeps your main program from crashing if the external program does not exist or has an error.

See Also: run()

Example

```
/* Read a command from the user and execute it.
** Display an error code if an error code is
** returned. Scroll each line up the screen
** after each command.
*/
main()
{
int code, key, db;
int CR, ESC;
char string[81];
 db = opendb("c:\concord\database\research");
 CR = 13;
 ESC = 27;
 while(key <> ESC)
 {
  key = 0;
  string[0] = 0;
  /* Get the command from the user. */
  while((key <> CR) and (key <> ESC))
   key = getline(MaxRow_,0,81,string);
  /* Scroll the screen and display the command. */
  scroll( 1, 0, MaxRow_ - 2, 79, 1, 1);
  puts(MaxRow_ - 2,0,pad(string,'L',80));
  scroll( 1, 0, MaxRow_ - 2, 79, 1, 1);
  string = trim(string);
  if (string[0]) {
   if (upper(string) == 'QUIT')
    return;
```

```
    /* Evaluate string, convert all results */
    /* to strings. Display result or error. */
    string = str(eval(string,code));
    if (code)
     puts(MaxRow_ - 2,0,"Error "+str(code));
    else
     puts(MaxRow_ - 2,0,string);
   }
  }
}
```

_exec

Summary

```
int exec(int db; text filename);
```

Description

Executes the saved query file. Unlike the Execute option on the Concordance Search menu, exec() does not display the searches as they are executed.

Concordance normally appends the file extension .QRY to executed query files. The exec() function does change the file name passed to it. Your program must ensure that the file extension is correct before calling the exec() function.

Return Value

A nonzero value if the file could not be found.

See Also: keep(), snapshot()

_exist

Summary

```
int exist(char string[]);
```

Description

Looks for the file name and determines if it exists.

Return Value

Exist returns a nonzero value if the file was found, and a zero if it was not found. In addition, the returned value has the following bit settings to indicate:

1 file exists

2 write permission is granted

4 read permission is granted

Example

```
main()
{
int status;
```

```
 if (status = exist("c:\config.sys")) {
  puts(0,0,"config.sys exists.");
  if (status & 2)
   puts(1,0,"config.sys has write access.");
  if (status & 4)
   puts(2,0,"config.sys has read access.");
 }
 else {
  puts(0,0,"config.sys file not found.");
  puts(1,0,"Creating one for you...");
  CreateConfig();
 }
 puts(3,0,"Press any key to continue...");
 getkey();
}
```

_exit

Summary

exit()

Description

Stops execution of the CPL program and of Concordance. Quits Concordance and returns to the operating system. Any open databases are closed automatically.

Return Value

No return value, this function does not return to the caller.

F

The following topic discusses the Concordance Programming Language (CPL) functions that begin with the letter F. For more information on CPL functions, see Functions, About the Advanced Programming Features, and About CPL Functions.

_findfirst

Summary

```
text findfirst(text fileMatch; int attribute);
```

Description

Locates the first matching file name for the fileMatch string parameter that has matching file attributes. All DOS wildcards are valid.

The attribute parameter specifies DOS file attributes that are included in the search. They include:

0 Normal files - no read/write restrictions.
1 Read only files.
2 Hidden files.
4 Systems files.
8 Volume ID file.
16 Subdirectory.
32 Archived file.

The file attributes can be or'ed together to find various file types. For example: findfirst("C:*.*", 1 | 2 | 4);

Return Value

The name of the first file found that matches the file specification string and the attribute value.

See Also: findnext()

Example

```
main()
{
char f[128];
 /* Erase all matching files. */
 for(f=findfirst("\*.TMP",0); f[0]<>0; f=findnext())
  erase(f);
}
```

findline

Summary

int findline(text line; int offset, length);

Description

Locates a line of text which contains the character at "offset" bytes from the beginning. The line is terminated by an end-of-line character. The end-of-line character is handled internally to Concordance and may not be compatible with other programs.

The "line" parameter can be a database field, a character array, or a text variable. It cannot be a numeric or date formatted field, or a numeric variable.

Text begins with character 1. An 11 character char array would contain characters at offsets 1 through 11, though it would be indexed with values 0 through 10.

Return Value

Returns the offset into the text variable where the line begins, the length entry will contain the length of the line. The length is only for the text on the line, it does not include the end-of-line terminator.

If the line isn't found, the field is empty, or offset is out of range, then findline will return 0

and length will be undefined. Remember that a length of 0 is legal since some lines may be empty.

See Also: findpline(), findnline()

Example

```
main()
{
int file, i, width, db;
 /* Open the printer as a file. */
 if ((file = open("prn","w")) == -1) {
  puts(0,0,"Couldn't access printer.");
  return;
 }
 /* Open a database for business. */
 if ((db = opendb("c:\concord\research")) < 0) {
  puts(0,0,"Couldn't open the database.");
  close(file);
  return;
 }
 /* Find and print every line in the */
 /* note field that mentions stock. */
 i = match(db->NOTE,"stocks",1);
 while (i > 0) {
  i = findline(db->NOTE,i,width);
  writeln(file,substr(db->NOTE,i,width),width);
  /* Look for next hit in the next line. */
  i = match(db->NOTE,"stocks",i+width);
 }
 /* Close all files before exiting. */
 close(file);
 closedb(db);
}
```

findnext

Summary

```
text findnext();
```

Description

Locates the name of the next matching file first located using findfirst().

Return Value

A file name, or an empty string if no file name is available.

See Also: findfirst()

Example

```
main()
{
char f[256];
 /* Erase all matching files. */
 for(f=findfirst("\*.TMP",0); f[0]<>0; f=findnext())
  erase(f);
}
```

findnline

Summary

```
int findnline(text line; int offset, length);
```

Description

Locates the next line of text following the line indicated by offset. Offset is an index into the field, text variable, or character array.

Return Value

findnline() returns the offset of the first character in the next line or 0 if there isn't a following line. Length will contain the length of the line when the function returns.

See Also: findline(), findpline()

Example

```
/* Print every line in the text field to file.
** Watch for disk full errors in writeln(). */
PrintField(int db, i, fh)
{
int offset, length;
int error;

 /* Wrap the field to fit in 40 columns. */
 /* Get the offset of the first line. */
 /* Enter the loop, print the line, then */
 /* continue writing lines until there */
 /* aren't any left to process. */
 wrap(db->i, 40);
 offset = findline(db->i,1,length);
 while((offset > 0) and (error == 0)) {
  if(writeln(fh,addr(db->i,offset),length)<length)
   error = 1;
  offset = findnline(db->i, offset, length);
 }
 return(error);
}
```

findpline

Summary

```
int findpline(text line; int offset, length);
```

Description

Locates the line of text preceding the line indicated by offset. Offset is an index into the field, text variable, or character array.

Return Value

The value of findpline() is the offset of the first character of the preceding line, or 0 if there is no preceding line. The length parameter will contain the length of the preceding line.

See Also: findline(), findnline()

Example

```
/* Scan up several lines in the field.
** Returns offset of the nth line above or
** a 0 if there aren't enough lines.
*/
scanup(int db, i, offset, lines)
{
int length;
 while((offset > 0) and (lines > 0)) {
  lines = lines - 1;
  offset = findpline(db->i, offset, length);
 }
 return(offset);
}
```

first

Summary

```
int first(int db, document, field, offset);
```

Description

Reads the first document in the current query. If no query exists, the first document in the database is used.

The variables document, field, and offset are optional. If one or more are provided, they will contain the document's number, the field number that contains the search word, and the offset into the field where the word occurs. The document number is the relative document number in the query, not the physical document number in the database. For instance, the 100th document in the database may be the first in the query. In this case the value of the document variable after issuing a first() would be 1.

Use the recno() function to get the physical document number.

Select queries, and query 0, will return 0's for the values of field and offset. Mixing select and search mode queries will result in some values for field and offset containing 0's within the retrieved query set.

Return Value

Returns the physical document number of the first document retrieved in the query. Returns a value less than or equal to zero if the query or database is empty, or if document could not be read.

See Also: last(), next(), nexthit(), prev(), prevhit(), count(), docno(), hits()

func

Summary

```
text func(int level);
```

Description

Returns the name of the currently executing function, and all previous call functions. Level is a number determining which function name is returned. Level 0 returns the name of the current function, level -1 returns the name of the function that called the current function, level -2 returns the name of the function two functions back, and so on.

Return Value

The name of a currently executing function, or an empty string if the level parameter is out of range.

Example

```
/* Display the names of all functions
** (currently called) on the screen.
*/
ListFunctions()
{
int con, i;

 if ((con = open("con","w")) >= 0) {
  for(i = 0; string = func(i); i = i - 1)
   writeln(con,string,len(string));
  close(con);
 }
}
```

fuzzy

Summary

```
int fuzzy( int db,
   text szWord,
   text szDestination,
   int certainty, flag);
```

Description

This function locates words that match the szWord parameter either in spelling or sound. The matches are placed in a file, whose name and path is provided in szDestination.

Concordance will append to the file if it exists. certainty is the percent of a match for spelled-like words. A value of 70 (70%) or higher is recommended.

The following flag parameters are integer values that are predefined for you. Do not declare them. Do not pass them in quotes. Never assign values to them. Simply use them. They can be or'ed together with the | operator to combine functions.

Flag Parameter	Function
CPLSOUNDSLIKE	Retrieves words that sound like the passed szWord parameter. Concordance uses a sounds-like algorithm similar to soundex.
CPLSPELLEDLIKE	Retrieves words that are spelled like szWord. This takes longer than a sounds-like search. The certainty parameter is used to determine if a word has similar spelling.
CPLSYNONYMSORT	Words are returned with the best matches first.
CPLSYNONYMS	Words are returned in quotes.

Return Value

Returns a count of the words found to match the passed parameter.

Version

Version 7.30 and later.

G

The following topic discusses the Concordance Programming Language (CPL) functions that begin with the letter G. For more information on CPL functions, see Functions, About the Advanced Programming Features, and About CPL Functions.

getarg

Summary

```
text getarg(int argNumber);
```

Description

Retrieves the command line argument passed to Concordance at startup. Concordance will ignore any arguments beginning with @, allowing a CPL program to retrieve them for exclusive use. Note that this function will also retrieve all command line arguments, not just those beginning with @.

Return Value

A text string containing the argument. An empty string indicates the end of arguments.

See Also: getenv() , putenv()

Example

```
main()
{
   text     pszArgs,
            pszCurrentArg;
   int      i;

   /* Loop for each arg */
   while ((pszCurrentArg = getarg(i)) <> "")
   {
      /* Add the arg to the list */
      if (pszArgs == "")
         pszArgs = "arg(0) = " + pszCurrentArg;
      else
         pszArgs = pszArgs + chr(13) + chr(10) + "arg(" + str(i) + ") = "

      /* Increment to the next arg */
      i = i + 1;
   }

   /* Display the args */
   if (pszArgs == "")
      messageBox("No args found on the command line.", "getargs", MB_OK);
   else
      messageBox(pszArgs, "getargs", MB_OK);
}
```

Version

Version 5.32 and later

getcwd

Summary

```
text getcwd();
```

Description

Determines the full file path of the current working directory.

Return Value

A text string containing the current working drive and directory.

See Also: chdir(), diskspace()

Example

```
main()
{
 chdir("C:\DATA");
 puts(10,40,"The current directory is "+getcwd());
 return(getkey());
}
```

getenv

Summary

```
int getenv(text name);
```

Description

The getenv() function searches the environment list for an entry matching name. The search is not case-sensitive.

Entries are added to the environment list with the DOS SET command, or with the CPL putenv() command. You can view the current environmental settings by typing SET from the DOS command prompt.

Return Value

A text value containing the string assigned to the environment variable specified by name. The string is empty if there is no entry for name.

See Also: getarg(); putenv()

Example

```
main()
{
text tempDir;
 tempDir = getenv("TEMP");
 if(tempDir == "")
  puts( 10, 40, "No temporary directory exists.");
 else
  puts( 10, 40, "The temporary directory is "+tempDir);
 return(getkey());
}
```

getfile

Summary

```
int getfile(char prompt[], fileMask[], fileName[]);
```

Description

A file selection window is displayed that shows the names of files that match the file mask. The mask can contain any characters valid in a file name, including wildcard characters. The prompt string is displayed as the title of the file selection window. If the user selects a file, the file's name is returned in the filename[] array.

Return Value

getfile() returns a carriage return and a file name if the user selects a file. It returns an ASCII escape character, value 27, if the user presses escape to quit file selection without making a selection.

Example

```
OpenADataBase()
{
int db = -1,
  CR = 13;
char string[60];

 if (getfile("Database","*.DCB",string) == CR)
  db = opendb(string);
 return(db);
}
```

getkey

Summary

```
int getkey();
```

Description

Read a key from the keyboard. This function will not return until a key is ready.

Return Value

Key pressed, there is no error value.

See Also: keypress()

Example

```
/* Prompt the user for the input code. */
input()
{
int value;
 value = 0;
 puts(5,10,"Enter code:");
 cursor(5,22);
 while(value == 0)
  switch(getkey()) {
   case 'A':
   case 'a': value = 1;
     break;
   case 'S':
   case 's': value = 2;
     break;
   case 27: value = -1; /* [Esc] */
```

```
      break;
    default:  beep(45,100);
      break;
   }
 return(value);
}
```

getline

Summary

```
int getline( int row, column, width; char string[]; [int color[,backgroun
```

Description

Retrieves a line of input text from the user. The maximum number of characters it will retrieve is equal to width - 1, since all character strings must be terminated by a zero. Color and background color are optional parameters. The background color only has meaning under Windows.

Return Value

The keystroke that caused the function to terminate. Any key that it cannot process itself will cause it to return. It will handle END, HOME, Ctrl-Y (Delete to end of line), LEFT, and RIGHT. All other keys cause it to return. The LEFT and RIGHT cursor control keys will also cause it to return if the user tries to move out of the text they are editing.

The value 16, a Ctrl-P, is returned if the user types past the last character in the field.

See Also: edit(), getnumber()

Example

```
/* Get a line of input from the user, return */
/* the input and the exiting key stroke. */
huh(char txt[])
{
int CR, ESC, key;
 key = 0;  txt[0] = 0;
 CR = 13;  ESC = 27;
 puts(15, 20, "Huh?");
 while((key <> CR) and (key <> ESC))
  key = getline(15, 25, sizeof(txt), txt);
 return(key);
}
```

getnumber

Summary

```
int getnumber( int row, col; float number, min, max; int width, places, [
```

Description

Prompts the user for a number at the row and column coordinates on the screen. The number will be between the min and max values. It will be no more than width characters wide, and will contain only places number of decimal places. The number, min, and max parameters can be any numeric value type, not just float.

The color and background color parameters are optional. TextColor_ will be used if the color parameters are left off.

Return Value

The key that caused the function to return. The number parameter will contain the user's entry when the function returns. If the user presses the [Esc] key, number will contain the original value when getnumber() was called.

The value 16, a Ctrl-P, is returned if the user types past the last character in the field.

See Also: getline()

Example

```
/* Get the number of lines per page to print. */
GetLinesPerPage(int lpp)
{
int CR, ESC, key;
CR = 13;
ESC = 27;
puts(15, 1, "Lines per page:");
while((key <> CR) and (key <> ESC))
key = getnumber(15, 17, lpp, 0, 255, 5, 0);
return(lpp);
}
```

getPrivateProfileString

Summary

```
int getPrivateProfileString( text szSection;
   text szKey;
   text szDefault;
   char szDestination[];
   int size;
   text szFile);
```

Description

The getPrivateProfileString() function returns a string from the specified section in an initialization file.

Parameter	Function
szSection	The name of the section containing the key name. If this parameter is NULL, the function copies all section names in the file to the supplied buffer.

Parameter	Function
szKey	The name of the key whose associated string is to be retrieved. If this parameter is NULL, all key names in the section specified by the szSection parameter are copied to the return buffer.
szDefault	If the szKey entry cannot be found in the initialization file, then the default string is returned. This parameter cannot be NULL.
szDestination	The data is returned in this array of char. It must be large enough to hold the data.
size	The length of szDestination in byes, for instance, use sizeof(szDestination).
szFile	The fully qualified name of the .ini file. If this parameter does not contai a full path, the system searches for the file in the Windows directory.

Avoid specifying a default string with trailing blank characters. The function can insert a null character in the returned text to strip any trailing blanks, depending on the operating system used.

You can create a NULL parameter by declaring NULL as a text variable and not assigning any value to it.

Return Value

The return value is the number of characters copied to the buffer, not including the terminating null character.

If neither szSection nor szKey is NULL and the supplied destination buffer is too small to hold the requested string, the string is truncated and followed by a null character, and the return value is equal to size minus one.

If either szSection or szKey is NULL and the supplied destination buffer is too small to hold all the strings, the last string is truncated and followed by two null characters. In this case, the return value is equal to size minus two.

See Also: writePrivateProfileString()

Version

Version 7.0 and later.

gettags

Summary

```
text gettags(int db; text szDelimiter);
```

Description

Retrieves all tags applied to the current document. Each tag is separated by the szDelimiter parameter.

Return Value

A text string containing the document's tags. An empty string if the document is not tagged.

See Also: istagged()

Example

```
/* Locate tags. Delimit with semi-colon & newline. */
text szResult;
szResult = gettags(db, ";" + newline());
```

Version

Version 8.0 and later.

getuuid

Summary

```
text getuuid(int db);
```

Description

Retrieves the document's unique universal identifier. This is a unique serial number assigned to each record in a Concordance database. Unlike accession numbers, the UUID is unique across databases.

Return Value

A text string containing the document's UUID. Only V7.0 and later databases contain UUIDs. Records in earlier databases will return empty strings.

See Also: accession(), gotouuid()

Version

Version 7.30 and later.

global

Summary

```
global(int db);
```

Description

Invokes Concordance Global Edit mode. See the Concordance manual for a description of Global Editing. The screen is automatically saved before entering this mode and restored after exiting.

Return Value

None.

goto

Summary

```
int goto(int db, document, field, offset);
```

Description

Reads the specified document in the current query set. On return, the document, field, and offset parameters, if they are passed, will contain the relative document number, field number and word offset. The offset indicates the offset of the retrieved word from the beginning of the field. Fixed field searches will return offset values of zero.

The hit list is positioned on the first hit for the document. Additional calls to nexthit() will retrieve information about the other items in the list.

Return Value

Returns a value less than or equal to zero if you to try to move past the last document in the current query.

The document, field, and offset parameters will contain information about the first word in the document.

See Also: recno(), docno(), first(), last(), next(), prev(), readdoc()

gotoaccession

Summary

```
int gotoaccession(int db, accessionNumber);
```

Description

Locates and reads the document that matches the accession number. Accession numbers are returned by the accession() function. They are used by Concordance internally in place of physical record numbers for record tagging and full text searching in V6.0 and later databases.

Return Value

Returns zero if successful.

See Also: accession()**,** goto(), gotophysical(), gotouuid(), recno(), docno(), readdoc()

Version

Version 6.0 and later.

gotophysical

Summary

```
int gotophysical(int db, document, field, offset);
```

Description

Locates the specified document in the current query set. The document number is the physical document number in the underlying database, not the relative document number in the current query.

On return, the document, field, and offset parameters, if they are passed, will contain the relative document number, field number and word offset. The offset indicates the offset of the retrieved word from the beginning of the field. Fixed field searches will return offset

values of zero.

The hit list is positioned on the first hit for the document. Additional calls to nexthit() will retrieve information about the other items in the hit list. Only the database handle and document number parameters are required.

Return Value

Returns the physical document number if successful. Returns a value less than or equal to zero if the physical document can't be located in the current query. In this case, the last document in the list becomes the current document.

The document, field, and offset parameters will contain information about the first hit word in the document if it was located successfully.

This function should not be used with concatenated databases as more than one database can have the same physical document number.

See Also: goto()

gotouuid

Summary

```
int gotouuid(int db, text szUUID);
```

Description

Locates and reads the document that matches the unique universal identifier text. UUIDs are returned by the getuuid() function for any V7.0 or later database. They are used by Concordance internally in place of physical record numbers to link an annotation to a record. Use the NOTEPARENT field in a database-notes database to obtain the parent record's UUID. Then use the UUID with gotouuid() to retrieve the parent record. The note is attached to this record.

Return Value

Returns zero if successful.

See Also: getuuid(), gotoaccession()

Version

Version 7.30 and later.

H

The following topic discusses the Concordance Programming Language (CPL) functions that begin with the letter H. For more information on CPL functions, see Functions, About the Advanced Programming Features, and About CPL Functions.

hits

Summary

```
int hits(int db);
```

Description

Returns the count of words located by the active query.

Return Value

Number of hits in currently retrieved search list.

See Also: count()

I

The following topic discusses the Concordance Programming Language (CPL) functions that begin with the letter I. For more information on CPL functions, see Functions, About the Advanced Programming Features, and About CPL Functions.

import

Summary

```
int import( text db->field;
   char filename[];
   char delimiter[];
   int anchored;
   int keepfile;
   int blanks;
   int wrap;
   [int maxBytes]);
```

Description

Loads the text file, whose name is passed in filename[], into the selected field and appends the new documents to the end of the database. If the file contains data for several documents, they are separated from each other by a line with the delimiter string on it. Concordance will assume the delimiter string is flush left if the anchored parameter is an 'A'.

The text file is a plain text file, i.e., ASCII format. Import will break the file into several documents if it is larger than 65,000 characters or maxBytes, and it cannot flow the text into following paragraphs.

If the keepfile parameter is 'K', Concordance will store the file's name in parentheses on the first line of each document created. The file name will appear on the line by itself.

Concordance will keep blank lines if the blanks parameter is 'B'. This parameter should be set to zero to remove blank lines. Concordance will automatically trim any trailing spaces from each line it loads.

The wrap parameter should be set to a 'W' if your text has indented paragraphs, or paragraphs separated by blank lines. This will cause Concordance to treat all carriage returns as soft returns, word wrapping all text until it finds an indented or empty line.

Concordance loads up to 60,000 characters per field unless the optional maxBytes parameter specifies another maximum.

Return Value

A zero if no error occurred. A nonzero value would indicate that the user aborted the import by pressing the [Esc] key, a disk or database full condition, or an error reading or locating the import file.

See Also: load()

Example

```
LoadDeposition(int db)
{
int CR;
char string[60];
CR = 13;
if ( getfile("Deposition","*.*",string) == CR)
import(db->depo,string,"",0,'K',0,0);
return;
}
```

importfs

Summary

```
importfs(int db);
```

Description

Full screen import menu as described in the Concordance Reference Manual.

Return Value

None.

See Also: loadfs()

index

Summary

```
int index(int db);
```

Description

Database is indexed from scratch. The dictionary file and inverted text files are erased before this command begins. The user is not prompted before the files are erased.

The db parameter must be a value returned by a call to opendb(). The screen is automatically saved before entering this mode and restored after exiting.

Return Value

Zero if successful.

See Also: reindex(), opendb()

insertText

Summary

```
insertText(db->FIELD; text szText; int offset);
```

Description

The first parameter identifies a field in the database. The insertText() function inserts a block of text, the szText parameter, into the field. This function preserves annotations and rich text formatting. Simply assigning text to a field does not preserve annotations or rich text, i.e., db->FIELD = db->FIELD + szText may cause all annotations to be misplaced and rich text formatting to disappear.

Return Value

The number of bytes inserted.

See Also: deleteText()

Version

Concordance version 7 and later.

isalnum

Summary

```
int isalnum(char ch);
```

Description

Tests the character value to determine if it is in the set a to z, or A to Z, or 0 to 9. ch can be any numeric value, char, int, float, or short.

Return Value

Returns a nonzero value if the character is in the alphanumeric set of characters. Returns a zero if it is not.

See Also: isalpha(), isdigit()

isalpha

Summary

```
int isalpha(char ch);
```

Description

Tests the character value to determine if it is in the set a to z, or A to Z.

Return Value

Returns a nonzero value if the character is alphabetic, or a zero if it is not alphabetic.

See Also: isalnum(), isdigit()

isdeleted

Summary

```
int isdeleted(int db);
```

Description

Checks if the document is marked for deletion.

Return Value

Nonzero if the document is marked for deletion, zero if it is not marked for deletion.

See Also: delete(), recall(), pack()

Example

```
/* Count the documents marked for deletion. */
DelCount(int db)
{
int count;
cycle(db)
if (isdeleted(db))
count = count + 1;
return(count);
}
```

isdigit

Summary

```
int deleteText(db->FIELD; int offset; int length)
```

Description

The first parameter identifies a field in the database. The deleteText() function deletes a block of text at offset for length number of bytes. This function preserves rich text formatting and properly processes annotations. Any annotations contained by the block of deleted text are also deleted.

Return Value

The number of bytes deleted.

See Also: insertText()

Version

Concordance version 7 and later.

isedited

Summary

```
int isedited(int db);
```

Description

Determines if the full-text indexable fields in the current document have been edited or appended to the database since the last time the database was reindexed. Documents located in a full text search that have been edited may not actually match the search conditions. Attempting to highlight the keywords on the screen, or to underline them on a printer may result in the wrong words being highlighted.

Return Value

A nonzero value if the document has been edited or modified since the last reindex.

isfield

Summary

```
int isfield(int db; text fieldName);
```

Description

Determines if the named field exists in the database. Useful with concatenated databases and the report writer.

Return Value

Returns the nonzero field number if the field exists in the current database, otherwise zero.

Example

```
printSomeData(int db, fh)
{
int i;
text someData;
cycle(db) {
someData = ((i=isfield(db,"SUMMARY")) ? db->i : "");
writeln(fh, someData);
}
}
```

islower

Summary

```
int islower(char ch);
```

Description

Tests the character to see if it is a lower case letter, in the set a to z.

Return Value

Returns a nonzero value if the character is lower case, and a zero if it is not a lower case letter.

See Also: isupper(), lower(), upper()

isnexthit

Summary

```
int isnexthit(int db);
```

Description

Determines if there is another hit in the current document. A hit is a word located in a search.

Return Value

Returns a nonzero value if there is a hit.

Example

```
/* Count the hits in this record */
for (i = 1; isnexthit(db); i = i + 1)
nexthit(db);
```

See Also: nexthit()

Version

Version 5.33 and later

isspace

Summary

```
int isspace(char ch);
```

Description

Tests the character to see if it is a white space character. White space characters include the space, horizontal tab, line feed, vertical tab, form feed, and carriage return; ASCII codes 32, 9, 10, 11, 12, and 13, respectively.

Return Value

A nonzero value is returned if the character is a space, and a zero is returned if it is not a space character.

istagged

Summary

```
int istagged(int db[, text tagName]);
```

Description

Checks if the document is currently tagged. The parameter db is a handle an open database. If the optional tagString is passed, the return value only applies to that tag. Calling istagged() without a tagString determines if the default tag, "", has been applied to the

document. To determine if ANY tag has been applied use the following code:

```
text NULL;
if (tagged(db,NULL))
```

Return Value

Returns a zero if the document is not tagged, nonzero if it is tagged.

See Also: gettags(), tag(), tagquery()

Version

The tagName parameter is only valid in version 5.31 and later.

isupper

Summary

```
int isupper(char ch);
```

Description

Tests the character to determine if it is an upper case letter, in the set A to Z.

Return Value

Returns a nonzero value if the character is upper case, and a zero if the character is not an upper case letter.

See Also: islower(), lower(), upper()

itoa

Summary

```
int itoa(int value; int radix);
```

Description

Converts an integer to a plain text, ASCII, value in base radix notation. If the value of radix is 10, and the number is negative, then itoa() will prepend a minus sign to the result.

Return Value

The number as a string in the requested base.

See Also: str(), num()

Example

The following program displays the values 25, 19, and 11001, which is the decimal 25 in base 10, 16 and base 2.

```
main()
{
```

```
puts( 10, 10, itoa( 25, 10 ));
puts( 11, 10, itoa( 25, 16 ));
puts( 12, 10, itoa( 25, 2 ));
getkey();
}
```

J

There are currently no Concordance Programming Language (CPL) functions that begin with J. For more information on CPL functions, see Functions, About the Advanced Programming Features, and About CPL Functions.

K

The following topic discusses the Concordance Programming Language (CPL) functions that begin with the letter K. For more information on CPL functions, see Functions, About the Advanced Programming Features, and About CPL Functions.

keep

Summary

```
int keep(int db; text filename);
```

Description

Saves all queries in the current session to the named file. Concord ance normally appends the file extension . QRY to saved query files. The keep() function does change the file name passed to it.

Return Value

Returns a nonzero value if an error was encountered while saving the queries. An error return indicates a bad file name, or a full disk.

See Also: exec(), snapshot()

keypress

Summary

```
int keypress();
```

Description

Checks to see if there is a key in the input buffer. keypress does not read the key from the buffer if one is ready. It immediately returns to the caller, whether or not a key is ready.

Return Value

Returns a zero if no key is ready, otherwise it returns the value of the key that will be returned by the next getkey() call.

See Also: getkey()

Example

```
main()
{
int ESC, finished;
ESC = 27;
finished = 0;
/* Loop until finished or until */
/* someone hits the ESC key. */
while((keypress() <> ESC) and (finished == 0))
finished = DoSomething();
/* If the user hit the ESC key */
/* to terminate processing early, */
/* read it from the buffer. */
if (keypress() == ESC)
getkey();
}
```

L

The following topic discusses the Concordance Programming Language (CPL) functions that begin with the letter L. For more information on CPL functions, see Functions, About the Advanced Programming Features, and About CPL Functions.

last

Summary

```
int last(int db, document, field, offset);
```

Description

Advances to the last document retrieved in the current query set. On return the document, field, and offset parameters, if they are passed, will contain the relative document number, field number and offset into that field that indicates the offset of the retrieved word. The document's number is its position relative to other documents in the query. Some searches, such as query zero, will return field values of zero. Check the field parameter before using it to access a field's data.

The hit list is positioned on the first hit for the last document. Additional calls to nexthit() will retrieve information about the other items in the hit list.

Return Value

Returns the physical document number of the last document retrieved by the query. Information about the hit word is returned only if one or more of the document, field, and offset parameters are passed.

Returns a value less than or equal to zero if the query or database is empty, or if the document could not be read.

See Also: first(), next(), nexthit(), prev(), prevhit(), count(), docno()

len

Summary

```
int len(text note);
```

Description

Determines the length, in characters, of the text variable, character array, or database field. The results are not valid for numeric or date fields.

Return Value

Integer length in bytes of the parameter.

See Also: sizeof()

load

Summary

```
int load( int db; char filename[]; [int comma, quote, newline; [int row, (
```

Description

Loads the database from the parameter file name. The file must contain database records in delimited ASCII format.

The fields loaded, and the order in which they are loaded, is set by assigning a number to the "order" entry of the field definition. You can assign any number from -128 to 127, but Concordance will only load fields with consecutive numbers from 1 to the last field number defined in the database. You should always renumber the fields before loading the database. Concordance uses, and changes, the field order in several full screen modes including Print, Load, and Unload. See the for-loop example below.

The optional parameters, comma, quote and newline, are used to delimit the fields during the load process. If they are left off Concord ance will use its internal ASCII default values, 20, 254 and 174 respectively. It is recommended that you leave the comma, quote and newline parameters off if transferring data between Concordance databases. However, they are required if the screen status parameters are passed.

Return Value

A count of the number of documents loaded from the delimited ASCII file. A count less than expected can indicate a disk full condition, or that the user pressed the [Esc] key to cancel the operation.

See Also: unload(), loadfs(), unloadfs(), import()

Example

```
main()
{
int db, i, count;
/* Permanately sort all records in the */
/* database in ascending order. */
if ((db = opendb("helpdesk")) <> -1)
{
sort(db,"db->DATE");
/* Renumber all fields to unload in */
/* order. This step is necessary */
/* since the order previously set, */
/* or set in Print, is retained */
/* until reset. Never assume the */
/* order includes all fields. */
for(i = 1; i <= db.fields; i = i + 1)
db.order[i] = i;
count = db.documents;
if (unload(db,"helpdesk.dat") <> count)
puts(0,0,"Error unloading database.");
else
{
zap(db);
if (load(db,"helpdesk.dat") <> count)
puts(0,0,"Error reloading data.");
else
{
erase("helpdesk.dat");
index(db);
}
}
closedb(db);
puts(0,1,"Press any key to continue...");
getkey();
}
}
```

loadfs

Summary

```
loadfs(int db);
```

Description

Invokes Concordance full screen load mode. The screen is automatically saved before entering Load, it is restored after returning.

Return Value

None.

lockdb

Summary

```
int lockdb(int db);
```

Description

Will attempt to lock the current database for exclusive use. This will always succeed in single user versions of Concordance. Network versions will fail if another user is accessing the database. The database cannot be accessed by other network users while it is locked.

The database is automatically locked and unlocked by Concordance during pack and modify procedures.

Return Value

Returns a zero if the lock operation was successful, a nonzero value if unsuccessful.

See Also: unlockdb()

lockdoc

Summary

```
int lockdoc(int db);
```

Description

Will attempt to lock the current document in the network version of Concordance. This has no effect in non-network versions of the program. A locked document cannot be changed and saved to file by other users. Reading another document, or using one of the Concordance full screen functions, such as loadfs() or global(), will unlock the document. However, the document is automatically locked again when it is reloaded.

Documents are automatically locked by Concordance when read into memory. Your program should unlock them as a courtesy to other network users if you do not intend to edit or modify them.

Return Value

Returns a zero if the lock operation was successful. Concordance will make ten attempts to lock the document before giving up.

See Also: unlockdoc(), locked()

locked

Summary

```
int locked(int db);
```

Description

Determines if the current document is locked. A locked document can be used, edited, and modified, but none of the changes will be saved to file. Documents are automatically locked in network versions of Concordance, unless they have been unlocked by a call to the unlockdoc() function.

Return Value

A zero if the document is not locked. A value of one if you have locked the document by a call to lockdoc(), and a value greater than one if the document is locked by another user on the network. Non-network versions of Concordance will always return zero.

See Also: lockdoc()

Example

```
getdoc(db,document)
{
int trys;
/* Try to get document up to 10 times. */
trys = 10;
readdoc(db,document);
while((trys > 0) and locked(db)) {
readdoc(db,document);
trys = trys - 1;
}
lockdoc(db);
/* Return a nonzero if we got a lock. */
return(locked(db) == 1);
}
```

lower

Summary

```
text lower(text author);
```

Description

Produces a duplicate of the variable in lower case. Does not change the original.

Return Value

The parameter in lower case.

See Also: capitalize(), upper()

Example

```
main()
{
text author;
author = "E E Cummings";
puts(0,0,lower(author));
puts(1,0,author);
}
```

Output:

```
e e cummings
```

E E Cummings

lseek

Summary

```
int lseek(int handle, offset, mode);
```

Description

Moves the next read or write position in the file to the specified offset. The mode indicator tells lseek how to move, as follows:

Mode 'B' Move relative to the beginning of file.

Mode 'P' Move relative to the present position.

Mode 'E' Move relative to the end of the file.

Use offset = lseek(handle, 0, 'P') to determine the present location in the file.

Return Value

The new position in the file, as an offset in bytes from the beginning of the file. The first byte in the file is at offset zero. Returns a -1 if the move was unsuccessful.

ltrim

Summary

```
text ltrim(text string);
```

Description

Removes all leading white space from the text variable, character array, or field. White space consists of spaces, tabs, carriage returns, and line feeds.

Return Value

Returns a duplicate of the string, does not modify the original.

See Also: rtrim(), trim()

Version

Version 6.0 and later

M

The following topic discusses the Concordance Programming Language (CPL) functions that begin with the letter M. For more information on CPL functions, see Functions, About the Advanced Programming Features, and About CPL Functions.

mapDevice

Summary

```
int mapDevice(text localName, remoteName, user, password);
```

Description

Maps a network printer or disk drive. If user and password are NULL parameters then mapDevice() will use the currently logged user. If that is unsuccessful then the system may prompt the user for a password. Create a NULL parameter by declaring a text variable and not assigning a value to it. See the example below.

localName is the local device name such as "lpt1" or "f:" that you will use to access the device. remoteName is the network device name, such as "\\MyServer\sys".

Return Value

Zero indicates success.

See Also: unmapDevice()

Version

Version 7.30 and later

Example

```
main()
{
text NULL;
mapDevice("T:", "\\MyServer\sys", NULL, NULL);
}
```

markhits

Summary

text markhits(text field, beginMark, endMark);

Description

Retrieves the database field, marking hits with the beginMark and endMark text sequence. This is used primarily for adding word processing bold-on and bold-off sequences to search terms located in a query. This would include situations where Concordance is being used with an Internet Server to return data in Hypertext Markup Language (HTML) for Internet Web Browsers.

The function should be called with a database field:

text markedText;

markedText = markhits(db->TEXT, "<B>", "</B>");

Return Value

The database field as a text value.

Version

5.42 and later.

Example

```
/* Write fields from db to fh (file handle.) */
/* Output the text using Internet HTML (Hyper */
/* Text Markup Language.) */
OutputHTML(int db; int fh)
{
int i, j, k, ln;
text szText;
for (j = 1;j <= db.fields; j = j + 1) {
switch(db.type[j]) {
case 'P':
szText = markhits(wrap(db->j, 65000), "<B>", "</B>");
for (k = findline(szText, 1, ln);
k > 0;
k = findnline(szText, k, ln)) {
write(fh, addr(szText, k), ln));
write(fh, "<BR>", 4);
}
break;
case 'T':
writeln(fh, szText="<B>"+db.name[j]+" = </B>" +
trim(db->j) + "<BR>", len(szText));
break;
case 'D':
writeln(fh, szText="<B>"+db.name[j]+" = </B>" +
dtoc(db->j, db.length[j]) + "<BR>",
len(szText));
break;
case 'N':
writeln(fh, szText="<B>"+db.name[j]+" = </B>" +
str( db->j, db.length[j], db.places[j]) +
"<BR>",
len(szText));
break;
}
}
} /* OutputHTML() */
```

_match

Summary

```
int match( text target, search; int offset, length );
```

Description

Locates the search string in the target beginning offset bytes into the target and looking for, at most, length bytes. The length parameter is optional, if it is left out match will look through the entire target string from offset to the end.

Return Value

The index to the location of the search string in the target string. A zero if no match was made.

See Also: matchc()

matchc

Summary

int matchc(text target, letterString);

Description

Locates the first occurrence of any character from the letter string in the target search string.

Return Value

The offset of the first matching letter from letterString found in the target. Returns a 0 if no match was found. Note that in the following example, if the character C is found to occur in the target search string before the letters b and a, then the offset returned will be that of the letter C.

See Also: match()

Example

```
if (i = matchc(db->TEXT,"abABC"))
...
```

max

Summary

```
float max(float x, y);
```

Description

Determines which value is greater. The parameters x and y can be char, short, int, or float, max() will make the proper comparison regardless of type.

Return Value

The greater of the two values.

See Also: min()

memavl

Summary

```
int memavl();
```

Description

Determines the maximum amount of memory that is available for use by your program. This

can be important if your program concatenates large character strings. Concordance uses available memory to manipulate the large blocks of text and will produce an insufficient memory error if it cannot allocate enough room.

Return Value

The largest block of memory that Concordance can allocate for use by your program. This value will change while your program runs.

Example

```
concatenate(int db, first, second)
{
text data;
/* Check if there is enough memory to join */
/* two large text fields. This test avoids */
/* a FATAL "Insufficient Memory" error. */
if (memavl()<len(db->first)+len(db->second))
{
/* There wasn't enough space. */
/* Display a non-fatal message. */
beep(45,200);
puts(0,0,"Can't join the two fields.");
getkey();
}
else
/* Enough space, join the fields. */
data = db->first+db->second;
return(data);
}
```

memcpy

Summary

```
memcpy(destination, source; int bytes);
```

Description

Copies the source data value to the destination for the specified number of bytes. Source and destination should be the same data type, but that is not required. This function should be used with caution as data type and range parameter checking that is performed with direct assignments, a = substr(b,1,10), is not performed with memcpy(). Consequently memcpy() will perform faster in some cases, but it will not provide you with zero terminated strings.

Return Value

The destination value.

Example

```
/* Copy to a date field to avoid math conversions */
/* on invalid entries, including zeroed dates. */
memcpy(db->DATE,"19930900",8);
See Also
```

memset()

memset

Summary

```
memset(destination; char value; int length);
```

Description

Sets bytes in destination to value for length specified. Destination can be any left-side-value including char, short, int, float, text, or a database field. This function offers a fast way to initialize a string or other variable to a set value.

Return Value

The destination value.

Example

```
/* Clear an array before reading from disk. */
memset(string,0,sizeof(string));
read(fh,string,sizeof(string));
See Also
memcpy()
```

menu

Summary

```
int menu( int row, col, brow, bcol; text array[]; int next; text "QUICK");
```

Description

Displays a menu in a window whose screen coordinates are specified by the two row and column parameter pairs.

The "next" parameter specifies the first option in the text array[] to highlight when the menu is displayed. The first item in the menu, item zero, is displayed as the title of the menu.

The quick keys parameter, which is optional, will cause the menu to return if any of the keys in the list are pressed. The menu is case sensitive, only upper case letters should be in the quick keys list. Keystrokes read by menu() are converted to upper case before processing.

The menu() function will leave the selected menu item highlighted. If the user pressed a quick key, the menu() function will attempt to highlight the item which corresponds with the key. Therefore, the quick key list must contain only one key for each item in the menu, and they must be in the same order as the menu items. Additional keys, which do not correspond to displayed menu items, can be included in the list, but they should appear at the end of the list.

Return Value

The number of the menu item selected or 0 if the ESCape key was pressed. The menu is not cleared from the screen when the user makes a choice.

See Also: btmenu()

Example

```
main()
{
text mymenu[4];
int db, ESC, next;
ESC = 27;
mymenu[0] = "Research Technology Library";
mymenu[1] = "Search";
mymenu[2] = "Browse";
mymenu[3] = "Edit";
if ((db = opendb("library")) >= 0)
next = 1;
while (next > 0)
{
next = menu(5,30,10,50,mymenu,next,"SBEX");
switch(next)
{
case 1: /* Do a search. */
search(db);
break;
case 2: /* Browse the database. */
browse(db);
break;
case 3: /* Edit retrieved documents. */
editfs(db);
break;
case 'X': /* Option X isn't ON the menu, */
/* it is in the quick key list. */
/* Display available memory. */
puts(0,0,"Memory: "+str(memavl()));
break;
}
}
closedb(db);
}
```

messageBox

Summary

```
int messageBox(text szText, szTitle; int style);
```

Description

Displays a standard Windows message box with the szText parameter as the message and szTitle as the message box title.

The style parameter determines the return value. Combine the style parameters together with the | operator, for instance MB_YESNO | MB_ICONQUESTION. All parameters are

predefined by the CPL interpreter. You do not need to declare them to use them. They are all integers. Style is any one of the following values:

Parameter	Description
MB_ABORTRETRYIGNORE	The message box contains three push buttons: Abort, Retry, and Ignore.
MB_DEFBUTTON1	The first button is the default. The first button is always the default unless MB_DEFBUTTON2 or MB_DEFBUTTON3 is specified.
MB_DEFBUTTON2	The second button is the default.
MB_DEFBUTTON3	The third button is the default.
MB_ICONEXCLAMATION	The message box displays an exclamation-point icon.
MB_ICONINFORMATION	The information icon, a lowercase letter "I" in a circle, is displayed
MB_ICONQUESTION	A question-mark icon appears in the message box.
MB_ICONSTOP	The message box displays a stop-sign icon.
MB_OK	The message box contains one push button: OK.
MB_OKCANCEL	The message box contains two push buttons: OK and Cancel.
MB_RETRYCANCEL	The message box contains two push buttons: Retry and Cancel.
MB_YESNO	The message box contains two push buttons: Yes and No.
MB_YESNOCANCEL	The message box contains three push buttons: Yes, No, and Cancel.

Return Value

A zero is returned if Windows could not create the message box, generally as a result of a low memory condition. Depending on the style parameter, the return values are listed below. All return value parameters are predefined by the CPL interpreter. You do not need to declare them to use them. If a cancel button is available and the user presses the [Esc] key, IDCANCEL is returned.

Parameter	Description
IDABORT	Abort button was selected.
IDCANCEL	Cancel button was selected.
IDIGNORE	Ignore button was selected.
IDNO	No button was selected.
IDOK	OK button was selected.
IDRETRY	Retry button was selected.
IDYES	Yes button was selected.

Version

Version 6.53 and later.

min

Summary

```
float min(float x, y);
```

Description

Determines which value is smaller. The parameters x and y can be char, short, int, or float, min() will make the proper comparison regardless of type.

Return Value

The lesser of the two values.

See Also: max()

mkdir

Summary

```
int mkdir(text directoryPath);
```

Description

The mkdir() function creates a new subdirectory with directoryPath name. directoryPath can be either and absolute path name or relative to the current working directory.

Return Value

mkdir returns zero if successful, nonzero otherwise.

See Also: chdir(), getcwd(), rmdir(), rename(), erase()

modify

Summary

```
modify(int db);
```

Description

Concordance full screen database modify command. It allows the user to modify the structure of the database whose handle is db. See the Concordance User Guide for a full description. The screen is automatically saved before entering modify and restored after exiting.

This function is not implemented in the Concordance Runtime Module. It will return immediately when called from the Runtime module.

Return Value

None.

See Also: createfs(), createdb(), struc()

month

Summary

```
int month(int db->field);
```

Description

Extracts the month from the date. The date must be an integer value representing the date, though any passed numeric value will be converted by month() to integer format before processing. The parameter can be a date formatted field, the return value of the ctod() function, or either of these values stored as an integer.

Return Value

The month of the year as an integer value, 1 - 12.

See Also: day(), year(), weekday()

N

The following topic discusses the Concordance Programming Language (CPL) functions that begin with the letter N. For more information on CPL functions, see Functions, About the Advanced Programming Features, and About CPL Functions.

netuser

Summary

```
text netuser();
```

Description

Determines the user ID, or workstation ID, for the network.

Return Value

Text string containing the user's sign-on ID.

newline

Summary

```
text newline();
```

Description

Returns a hard carriage return. Primarily used in the report writer.

Return Value

A carriage return/line feed combination.

next

Summary

```
int next(int db, document, field, index);
```

Description

Concordance advances to the next document in the current query. The hit list is positioned on the first hit for the document. Additional calls to nexthit() will retrieve information about the other items in the hit list.

Return Value

Returns the physical document number of the next document retrieved in the query. Returns a value less than or equal to zero if the current document is the last in the query, or if an error was encountered reading the document.

See last() for more information about returned values.

See Also: last(), first(), nexthit(), prev(), prevhit(), count(), docno(), hits()

Example

```
/* Cycle through every document in the data
** base, starting with the current document,
** converting the STATE field to upper case.
*/
StateToUpper(db)
{
int i;
for(i = docno(db); i > 0; i = next(db))
db->state = upper(db->state);
}
```

nexthit

Summary

```
int nexthit(int db, document, field, index);
```

Description

Moves to the next hit in the current query. If this moves to the next document, then that document is read. Information about the hit list item is returned.

db is a valid database handle returned by a call to opendb(). The document, field, and offset parameters are optional.

Return Value

Information about the hit word is returned only if one or more of the document, field, and offset parameters are passed.

Returns a value less than or equal to zero if the document could not be read.

See Also: goto(), first(), isnexthit(), last(), prev(), next(), prevhit(), count(), hits()

not

Summary

```
int not(int number);
```

Description

Logically inverts the number, converts the number to its one's compliment. The binary representation of 2 is 0000000000000010, not(2) returns the value 1111111111111101. The number is converted to int type before the inversion.

Return Value

The number's one's compliment.

See Also: Operators and Operands | and &.

num

Summary

```
float num(char string[]);
```

Description

Converts a character string to a number. The number is converted to a float type, but the result of num() can be assigned to any numeric type.

Blank spaces are ignored in the conversion. The conversion begins with the first numeric character, and it ends when the first nonnumeric character is encountered.

Return Value

Numeric representation of character string.

See Also: itoa(), str()

Example

```
main()
{
text something;
int i;
something = " 1982 was a good year.";
i = num(something) + 1;
/* i now equals 1983. */
}
```

O

The following topic discusses the Concordance Programming Language (CPL) functions that begin with the letter O. For more information on CPL functions, see Functions, About the Advanced Programming Features, and About CPL Functions.

onexit

Summary

```
int onexit(text exitFunction);
```

Description

Runs a Concordance menu command after the CPL program terminates. The exitFunction parameter is a string containing any standard item that appears in the Concordance menus, such as "Browse" or "Table." Use an empty string, "", to reset the onexit() function.

Return Value

A nonzero value if successful, zero if onexit() has been reset. The function is reset and zero is returned if the exitFunction menu string was not matched.

Version

Version 8.0 and later.

open

Summary

```
int open(char string[], type[]);
```

Description

Opens a file for use and returns that file's handle. The handle must be used to access the file.

Type can be:

"ru" or "rU" Open existing Unicode file for read only access.
"wu" or "wU" Create empty Unicode file for writing, erases file if it exists.
"au" or "aU" Open or create Unicode file for appending, no reading.
"ru+" or "rU+" Open existing Unicode file for reading and writing.
"au+" or "aU+" Open or create Unicode file for appending and reading.
"wu+" or "wU+" Open Unicode file for reading and writing. This erases the file if it exists.

Use caution with "w+" and "w" as they will reset a file and destroy its contents if the file already exists.

Note that both parameters must be either text or char arrays, they can not be character literals as in 'r'. Always use quotes to enclose the parameters, never use apostrophes.

Return Value

A file handle if successful, -1 if not successful.

See Also: read(), write(), readln(), writeln(), close(), exist(), erase()

opendb

Summary

```
int opendb(char string[]);
```

Description

Opens the database and returns a file handle which must be used to access the database documents, its fields, and statistics. The first document in the database is automatically read into memory by the opendb() command.

Return Value

A -1 indicates an error opening the database. Any other positive value is a database handle. Up to 16 databases may be open at one time, if available memory.

See Also: closedb()

opendbconvert

Summary

```
int opendbconvert(text database path);
```

Description

Converts version 8 or 9 databases to Concordance version 10. Opens the database and returns a file handle to the database, which must be used to access the database documents, its fields, and statistics. The first document in the database is automatically read into memory by the opendb() command.

The opendbconvert function does not validate or prompt for administrator username and password credentials. To prevent databases from inadvertently being converted to version 10, please ensure that you have only chosen directories and file paths to databases you wish to convert to version 10.

Return Value

A -1 indicates an error opening the database. Any other positive value is a database handle. Up to 16 databases may be open at one time, if available memory.

See Also: closedb()

operator

Summary

```
operator("ADJ");
```

Description

Sets the default operator recognized by search(). The operator can be any valid search operator: ADJ, ADJ1 - ADJ99, NEAR, NEAR1 - NEAR99, OR, AND, NOT, XOR.

Return Value

None.

overlayfs

Summary

```
overlayfs(int db);
```

Description

Invokes Concordance full screen Overlay mode. The screen is automatically saved before entering Overlay and restored after exiting.

Return Value

None.

P

The following topic discusses the Concordance Programming Language (CPL) functions that begin with the letter P. For more information on CPL functions, see Functions, About the Advanced Programming Features, and About CPL Functions.

pack

Summary

```
int pack(int db);
```

Description

Invokes Concordance full screen pack mode. The database is packed, all documents marked for deletion are removed. The search files are optimized if there is enough room on the disk. The screen is automatically saved before entering Pack and restored after exiting.

Return Value

Zero if successful.

pad

Summary

```
text pad(char string[], align; int width);
```

Description

Duplicates the string and pads it with spaces to width number of characters. The string will be left, center, or right justified within the spaces depending on the value of align:

L String is left justified

C String is centered

R String is right justified

V Vertical alignment, adds or truncates lines to maximum height

Return Value

A duplicate of the original string padded with the specified number of spaces. The string is truncated if the width is less than the length of the string. The returned text value will never be longer than width.

Example

```
main()
{
puts(0,0,"|"+pad("hello",'L',21)+"|");
puts(1,0,"|"+pad("hello",'C',21)+"|");
puts(2,0,"|"+pad("hello",'R',21)+"|");
}
```

Output:

```
|hello                |
|        hello        |
|                hello|
```

paste

Summary

```
text paste();
```

Description

Returns a copy of the data in the cut and paste buffer. The same data is returned over and over again until it is replace by a cut() or by the user from the editor. The Windows version uses the system clipboard.

Return Value

The contents of the cut and paste buffer.

See Also: cut()

prev

Summary

```
int prev(int db, document, field, index);
```

Description

Concordance moves to the previous document in the current query.

Return Value

Returns the physical document number of the previous document retrieved in the query. Returns a value less than or equal to zero if the current document is the first in the query, i.e., there is no previous document, or if the query set is empty. An error reading the document from file will also return a value less than or equal to zero.

Information about the first hit in the document is returned in the document, field, and index parameters. These parameters are optional. See first() for more detailed information on the return values.

See Also: goto(), first(), last(), next(), prevhit(), nexthit(), count(), hits()

prevhit

Summary

```
int prevhit(int db, document, field, offset);
```

Description

Moves to the previous hit in the current query. If this moves to the previous document, then that document is read. Information about the hit list item is returned.

db is a valid database handle returned by a call to opendb(). The document, field, and offset parameters are optional.

Return Value

Returns a value less than or equal to zero if the document could not be read or if the current hit is the first in the database, i.e., there is no previous document.

Information about the hit word is returned in the document, field, and offset parameters.

See Also: goto(), first(), last(), prev(), next(), nexthit(), count(), hits()

print

Summary

```
print( int db,
outputFileHandle,
firstDocument, lastDocument;
text printFormatFileName;
[int statusRow[,
statusColumn[,
statusColor]]]);
```

Description

Loads the print format file, file extension .FMT, and prints the document range in the current active query. The output is sent to the open file whose handle is outputFileHandle. The document count is displayed on the screen at statusRow, statusColumn in statusColor. The Windows version displays the print status in a dialog box.

Print format files store all settings that can be set in full screen print mode. The settings are stored as plain text, and are created with the Documents/Print/Save menu option. They can be edited with any text editor. The print format file parameter can be a file name or an empty string, "", to accept the current settings. Any option not specified in the print format file is left alone, it is not reset by its absence. For instance, the file could contain only print margins, page length, and the setup string.

The Windows version requires a valid print format file parameter. It ignores the outputFileHandle parameter and prints to the current device.

Return Value

None.

See Also: printfs(), report(), reportfs()

Example

```
PrintThisList(int db)
{ /* Example for DOS and OS/2 */
int fileHandle;
if ((fileHandle = open("LPT1","w")) >= 0) {
print(db,fileHandle,1,count(db),"HPLASER",10,40);
close(fileHandle);
}
}
```

printfs

Summary

```
printfs(int db);
```

Description

Invokes the Concordance full screen print menu. See description in the Concordance Reference Manual. The screen is automatically saved before entering this mode and restored after exiting.

Return Value

None.

See Also: print(), reportfs()

program

Summary

```
text program();
```

Description

Returns the name and DOS path of the currently running CPL program. The full program path may not be available if the program was executed as a command line prompt with Concordance.

Return Value

The name of the CPL program with the full DOS path if available.

putenv

Summary

```
int putenv(text nameAndValue);
```

Description

This function places the value into environment list. The environment list consists of a name and a value. These can be displayed from the DOS prompt by typing SET. The following is an example of a set value.

PATH=C:\;C:\DOS

Values are placed into the environment by passing the name followed by an equal sign and the value. Items are deleted from the environment by passing the name followed by an equal sign only.

Space for environment variables is limited. Consequently, the putenv() function may fail if there isn't enough room to add another value.

Return Value

Zero indicates success, -1 indicates failure.

See Also: getenv()

Example

The following program assigns a string to an environmental value and deletes another value.

```
main()
{
putenv("USERID=SMITHJ");
putenv("UPDATE=");
}
```

puts

Summary

```
puts(int row, column; char string[]; [int color[, background]]);
```

Description

Displays the text variable or character string on the screen at the row and column, in the specified color.

Color and background color are optional parameters. For DOS and OS/2 the color can be any value from 0 to 255, though values 127 and over may blink. Only Windows will use the background color parameter.

Windows uses colors in the range 0 - 16,777,215. The colors consist of three values: red, green, and blue. They are created by the RGB() function, defined in your program, as follows:

int white = RGB(255,255,255);

RGB(char red, grn, blu)

```
{
return(((blu*65536)|(grn*256)|red);
}
```

Return Value

None.

See Also: putsl()

Example

```
main()
{
int row, col, color;
/* Fill the screen with colors. */
for(col = 0; col < 74; col = col + 6)
for(row = 0; row < 25; row = row + 1) {
puts(row, col, "Color ", color, RGB(190,190,190));
color = color + 1;
}
}
```

putsl

Summary

```
putsl(int row, col, length; char string[]; [int color[, background]]);
```

Description

Displays the string at the row and column position on the screen. The length parameter specifies the number of characters in the string to be displayed. The color for Concordance normal text is used if the color and background parameters are omitted. Background color is only used by Windows.

Return Value

None.

See Also: puts()

Example

```
/* Display one screen of the text field. */
showfield(int db, i, offset)
{
int length, row;
wrap(db->i, 75);
offset = findline(db->i, offset, length);
row = 0;
while((offset > 0) and (line < 25)) {
putsl(row, 0, length, addr(db->i, offset));
offset = findnline(db->i, offset, length);
row = row + 1;
}
/* Return the offset of the next line. */
return(offset);
}
```

Q

The following topic discusses the Concordance Programming Language (CPL) functions that begin with the letter Q. For more information on CPL functions, see Functions, About the Advanced Programming Features, and About CPL Functions.

query

Summary

```
int query(int db, number; char string[]);
```

Description

Retrieves and makes the selected query number the current query. The query is used by all functions until it is changed by a call to search(), select(), or query(). Queries manually changed in full screen functions, such as browse() and global(), are restored when the function returns.

The string parameter is optional. When it is supplied, the function will fill it with the search string used to create the query.

To use the entire database use query(db, 0). Query 0 is defined to contain every document in the database.

To clear all queries use query(db, -1). This will reset the query counter to 0, and erase the temporary query file.

Return Value

A -1 if the query is out of range, or if an error was encountered reading the query from disk.

See Also: Database information variables db.query and db.activequery.

queryString

Summary

```
text queryString(int db, query);
```

Description

Retrieves the query logic string for the query. Unlike the query() function, it does not make the selected query number the current query.

Return Value

The text of the query or an empty string if either parameter is invalid or out of range.

See Also: query(), database information variables db.query and db.activequery.

Version

Version 7.30 and later.

R

The following topic discusses the Concordance Programming Language (CPL) functions that begin with the letter R. For more information on CPL functions, see Functions, About the Advanced Programming Features, and About CPL Functions.

rand

Summary

```
int rand([int seed]);
```

Description

Creates a random number sequence for any given seed number. The sequence will always be the same for a particular seed number. rand() should be called with a seed value to begin the random number sequence, for instance, rand(clock()). From then on it will return random numbers.

Return Value

Zero when called with a seed value. A random number otherwise.

Version

Version 6.0 and later

read

Summary

```
int read(int handle; char buffer[]; int length);
```

Description

Reads length bytes from the file referenced by handle into the variable buffer. You can read/write numeric values by passing their names and using the sizeof() function to determine their lengths.

Return Value

Returns the number of bytes read, which may be less than that requested if the end-of-file was encountered during the read.

See Also: readln(), readc(), write(), writeln(), open(), close()

Example

```
copyfiles(text from, to)
{
char buffer[512];
int i, oldfile, newfile;
 /* Open the old file, in read only mode. */
 if ((oldfile = open(from,"r")) < 0)
  return(-1);
```

```
 /* Now open the new file, create it. */
 if ((newfile = open(to,"w+")) < 0) {
  close(oldfile);
  return(-1);
 }
 /* Copy the old file to the new file. */
 while((i = read(oldfile,buffer,512)) > 0)
  write(newfile,buffer,i);
 close(oldfile);
 close(newfile);
 return(0);
}
```

readc

Summary

```
int readc(int handle)
```

Description

Retrieves one byte from the file referenced by handle. Handle must be a value returned by a call to open().

Return Value

Character read from file, or -1 if end of file encountered.

See Also: open(), close(), writec(), read(), readln()

Example

```
skip(int handle)
{
int c, EOF, LF;
 /* Skip past one line of input. Read
 ** until the end-of-file is encountered,
 ** or a line feed is found.
 */
 EOF = -1;
 LF = 10;
 c = 0;
 while((c <> LF) and (c <> EOF))
  c = readc(handle);
}
```

readdoc

Summary

```
int readdoc(int db, document);
```

Description

Reads the document from the database. The document becomes the current document. This function does not use the query list, but reads the document in the underlying database. If the current query contains 3 documents, and you request

```
readdoc(db, 100);
```

then the 100th document in the database is read.

Using next() after this command will retrieve the next document in the query, not document 101. The same goes for prev(), both functions work on the current query.

Return Value

A -1 if the document is out of range.

See Also: recno()

readln

Summary

```
int readln(int handle; char buffer[]);
```

Description

Reads one line of text from the file referenced by handle. The read continues until a line feed is encountered, carriage returns are ignored. The text is stored in the buffer without the terminating line feed. If the buffer fills before the full line is read, then the partial line is returned. The buffer is terminated with a zero.

Return Value

The number of characters read, without counting carriage returns and line feeds. A -1 indicates an attempt to read beyond the end of the file, the buffer will not contain any characters in this case.

See Also: writeln(), read(), write()

recall

Summary

recall(int db);

Description

The current document is unmarked for deletion.

Return Value

None.

See Also: delete()

Example

```
main()
```

```
{
int db;
 if ((db = opendb("recipes")) <> 0) {
  /* Recall all documents from deletion. */
  cycle(db)
   recall(db);
  closedb(db);
 }
}
```

recno

Summary

```
int recno(int db);
```

Description

Retrieves the physical record number of the current document. This function does not use the current query. recno() can be used with readdoc(). Both readdoc() and recno() ignore the current query and operate on the underlying database.

A related function, docno(), always returns information from the current query. Using docno() after readdoc() will cause the current document in the query set to be read. The document read by readdoc() will be flushed.

Use docno() with the functions that manipulate queries, i.e. first(), last(), next(), etc. recno() can be used with either readdoc() or the query functions.

Return Value

The current document's physical record number, or a value less than or equal to zero if no document is ready, i.e. after a blank(), or if the database is empty.

See Also: readdoc(), docno()

reindex

Summary

```
int reindex(int database);
```

Description

The database is reindexed. reindex() will call index() if the database has not yet been indexed. The screen is automatically saved before entering this mode and restored after exiting.

Return Value

Zero if successful.

See Also: index()

rename

Summary

```
int rename(char oldname[], newname[]);
```

Description

The file name is changed from oldname to newname.

Return Value

A return value of -1 indicates an error. Errors are caused by a failure to find the file, a poorly formed DOS file name, or a file that already exists with the new name.

See Also: exist(), erase()

rep

Summary

```
text rep(char ch | text string; int length);
```

Description

Creates a new string with the character ch or text string repeated length number of times.

Return Value

A text variable.

Example

```
DrawLine(int row; char ch);
{
 /* Draw a line across the screen. */
 puts(row, 0, rep(ch,80), TextHighlight_);
}
```

replicate

Summary

```
int replicate( int publisher;
  text subscriber;
  text synchMethod;
  int appendToPublisher;
  int appendToSubscriber;
  text deletionTag;
  int copyDeletionMarks;
  int restoreDeletedDocs;
  text collisionTag;
int copyAttachments);
```

Description

Synchronizes two database. The databases must be from the same replication set, for instance they were created from the same databases using createReplica(). Security, record

edits, deletions, and tags are replicated. Replication does not reindex or pack the databases.

Concordance will attempt to open the subscriber database in shared, multi-user mode. This enables laptop clients to synchronize with the network database even if they are running a single user. However, if the single user program already has the database open on the File menu, the shared open will fail. The example program takes this into account by closing any open database handles.

replicate() can synchronize all fields or just selected fields. Use the db.order[i] value to select fields for replication. Only selected fields are replicated. See the example for more information.

Any records in collision are tagged with the collisionTag. A collision occurs when the same field in the same record in both databases has been edited. Concordance will not overwrite either edit, but will flag the records as being in collision. See the resolve() function for information on scripting collision resolution.

Parameter	Type	Function
publisher	int	Handle of publisher database.
subscriber	text	Full path and file name of subscriber database.
synchMethod	text	Specify one of the three following options for synchronization: "Synchronize" for full bi-directional synchronize between databases. "POverwrites" for the publisher to overwrite the subscriber when differences are detected, regardless of which database has the most up-to-date information. "Soverwrites" for the subscriber to overwrite the publisher when differences are detected.
appendToPublisher	int	Pass any nonzero value to allow appending of new records to the publisher from the subscriber. Use zero to prevent appending of new records to the publisher database. This is a Boolean value.
appendToSubscriber	int	Pass any nonzero value to allow appending of new records to the subscriber from the publisher. Use zero to prevent appending of new records to the subscriber database.
deletionTag	text	The optional tag is applied to records when the flag marking a record for deletion is copied from one database to another. Use "" if you do not want to tag these records.
copyDeletionMarks	int	Use any nonzero value to allow replication to copy deletion flags. This flag indicates that a record should be deleted during the next database pack. It does not delete the record during replication.
restoreDeletedDocs	int	Use any nonzero value to restore deleted records. This restores the record if it exists in one database but has been deleted from the other. Note that it may be restored and still flagged for deletion.

collisionTag	text	This tag is applied to records in each database when a collision occurs. Use this tag after replication to resolve collisions via the resolve() function. Use "" if you don't need to track collisions.
copyAttachments	int	This optional parameter copies attachments if set to TRUE.

Return Value

A zero indicates success. Any nonzero value indicates failure. Failure can occur if the subscriber database cannot be opened, if the databases are not from the same replication set, if the replication fields are not present or are not both system and key fields.

Example

```
/* Fragment of code to synchronize two databases.
** The user's ID and date are used to flag any
** collisions. The tag could be used later to
** resolve the collisions with resolve().
*/

int i, TRUE = 1, FALSE = 0;
/* Replicate every field. This is a required step. */
for (i = 0; i <= db.fields; i = i + 1)
db.order[i] = i;
replicate( db,
"\\LexisNexis\Nome\Alaska\Catalog.dcb",
"Synchronize",
TRUE,   /* Append to publisher. */
FALSE,  /* Don't append to subscriber. */
"",  /* Don't tag deletions. */
TRUE,  /* Replicate deletions. */
FALSE,  /* Don't restore deletions. */
netuser() + " " + dtoc(today());
```

See Also: createReplica(), resolve()

report

Summary

```
int report(int db; text reportName; int first, last);
```

Description

Runs the named report for the document range. The report is sent to the current default printer.

Return Value

Zero if the report ran, nonzero otherwise.

Example

```
MonthlyReport(int db)
{
char  szNow[10];
int rc;

 /* Print a report for every record this month */
 if (db.documents > 0) {
  szNow = str(month(today())) + "/??/" + str(year(today()))
  search(db, "DATE = " + szNow);
  if (count(db) > 0) {
   sort(db, "dtoc(db->DATE, 'Y')");
   rc = report(db, "C:\concord5\taxs.arp", 1, count(db));
  }
 }
 return(rc);
}
```

See Also: print(), printfs(), reportfs()

Version

Version 6.0 and later.

reportfs

Summary

```
reportfs(int db);
```

Description

Invokes the Concordance full screen report writer. The screen is automatically saved before entering this mode and restored after exiting.

Return Value

None.

reset

Summary

```
reset(int db);
```

Description

Resets the current document, reloading the record from disk without saving any changes made by the programming language application. Note that a call to any full screen function, such as browse() or editfs(), will automatically save the edits. The database functions, next(), prev(), etc., will also save the edits.

Return Value

None.

resolve

Summary

```
int resolve(int toDatabase, fromDatabase);
```

Description

This function is passed handles to two databases. It resovles a collision between records in these databases by copying all fields from one database to another database, making the records identical and eliminating the collision.

Collisions occur when the same field in the same record has been edited in both the subscriber and publisher databases. Since Concordance cannot determine which edit is the "correct" edit, it flags these records as being in collision.

The resolve() function is used to automate replication. It provides a powerful mechanism to script collision resolution based on edit dates, user log-in name, and other user criterion stored in the replication fields.

Return Value

A zero signifies success.

Example

```
/* Fragment of code to synchronize two databases
** and resolve any collisions between them.
*/

int i, dbS, TRUE = 1, FALSE = 0;
text szTag;  /* Tag used to locate collisions. */
char string[200]; /* Used to create a search string. */

/* Replicate every field. This is a required step. */
for (i = 1; i <= db.fields; i = i + 1)
db.order[i] = i;

/* Create a unique tag for collisions */
szTag = netuser() + " " + dtoc(today());

/* Replicate the databases, tagging any collisions. */
replicate( db,
  "\\Williamson\case\Production.dcb",
  "Synchronize",
  TRUE,  /* Append to publisher. */
  TRUE,  /* Append to subscriber. */
  "",  /* Don't tag deletions. */
  TRUE,  /* Replicate deletions. */
  FALSE,  /* Don't restore deletions. */
 netuser() + " " + dtoc(today());

/* Databases are replicated. Now handle collisions. */
/* First open the subscriber database. */
if ((dbS = opendb("\\Williamson\Case\Production.dcb")) >= 0)
```

```
{
 /* We need to see the system fields used for */
 /* replication. Make them visible to us. They */
 /* link the records in the databases. */
 set(db, "System fields", "Show");
 set(dbS, "System fields", "Show");

 /* Locate all records in the publisher database */
 /* that were tagged for collision, then process. */
 tagquery(db, szTag);

 /* Loop through the tagged collision records. */
 cycle(db)
 {
  /* Find this record in the subscriber. */
  string = 'CREATIONID = "'+trim(db->CREATIONID) + chr('"');
  if ((search(dbS, string) == 0) and (count(dbS) > 0))
  {
   /* We found the record. For the example */
   /* we will keep the NOTES field from the */
   /* publisher database, and use all other */
   /* fields from the subscriber database. */
   dbS->NOTES = db->NOTES;
   resolve(db, dbS);
  }
 }

 /* Clear the temporary tags we used for collisions. */
 tag(db, -1, szTag);
 tag(dbS, -1, szTag);

 /* Close the subscriber, we're done */
 closedb(dbS);

 /* Hide the system fields, we're done. */
 set(db, "System fields", "Hide");

}
```

See Also: createReplica(), replicate()

restore

Summary

```
restore(row, column, text screen);
```

Description

Places the screen image on the screen that was previously created by a call to the save() function. The image is placed at the location whose upper left corner is described by the row and column coordinates.

Return Value

None.

See Also: save()

rmdir

Summary

```
int rmdir(text directoryPath);
```

Description

Removes (deletes) the specified directory. The directory must not contain any files or subdirectories. directoryPath can be either an absolute path name or relative to the current working directory.

Return Value

rmdir returns zero if successful, -1 otherwise.

See Also: chdir(), getcwd(), mkdir(), rename(), erase()

round

Summary

```
float round(float number; int decimals);
```

Description

Rounds the fractional portion of a floating point number to the specified number of decimals.

Return Value

The rounded floating point number.

Example

```
main()
{
float f, pi;
 pi = 3.1459;
 f = round(pi,2);
 /* f now contains 3.15 */
 ...
}
```

rtrim

Summary

```
text rtrim(text string);
```

Description

Removes all trailing white space from the text variable, character array, or field. White space consists of spaces, tabs, carriage returns, and line feeds.

Return Value

Returns a duplicate of the string, does not modify the original.

See Also: ltrim(), trim()

Version

Version 6.0 and later

run

Summary

```
run(text program, function);
```

Description

This function is used to execute external CPL programs. The program parameter is the name of a .CPL or .CPT file which contains the function. Inclusion of the .CPL or .CPT file name suffix is optional.

The called function can access and change any globally declared variable in the calling program. It can also execute any function in its own file or in the calling function's file. Global variables declared in the called module are only accessible while the module is loaded and running. Functions and variables in the module will replace pre-existing functions and variables of the same name.

Quoted strings that are used as parameters should be assigned to a text or char array before being passed. See the example below.

Return Value

The value returned by run() is the value returned by the function that is executed. It can be any Concordance value.

See Also: eval()

Example

```
LoadSpeller(int db)
{
char prompt[50]; int ok;
 /* Load an external program to spell check. */
 prompt = "Look up word: ";
 if (ok = run("spellchk.cpl","Check(db,prompt)"))
 {
  prompt = "Enter correction: ";
  ok = run("spellchk.cpl","Correct(db,prompt)");
 }
 return(ok);
}
```

S

The following topic discusses the Concordance Programming Language (CPL) functions that begin with the letter S. For more information on CPL functions, see Functions, About the Advanced Programming Features, and About CPL Functions.

The function searchfs() has been removed from Concordance version 9.0 and later.

save

Summary

```
text save(row, col, brow, bcol);
```

Description

Returns an image of the screen window described by the upper left corner and lower right corner coordinates. The image is stored in text variable format, but it is not a standard text variable.

Return Value

A text value containing the image of the screen.

See Also: restore()

scroll

Summary

```
scroll(int tr, tc, br, bc, rows, up [, bkcolor]);
```

Description

Scrolls a window in the screen. The window is described by the upper left corner row and column coordinates (tr and tc) and the bottom right corner row and column coordinates (br and bc).

The window is scrolled up if the up parameter is 'U' or 'u', and down if it is 'D' or 'd'. It is scrolled rows number of lines, the new lines (at the top or bottom) appear empty and in the background color. If the rows parameter is zero the entire window is cleared. Bkcolor is an optional parameter and specifies the background color.

Return Value

None.

See Also: cls()

Example

```
/* Clear the screen in the passed color. */
clsc(int color)
```

```
{
 scroll(0,0,24,79,0,'U',color);
}
```

search

Summary

```
int search(int db; char string[]);
```

Description

The string contains a search, it is searched in the database and the results become the current query. The search screen is not displayed. Query error messages are not displayed.

Return Value

Zero if no error occurred, an error number if the search could not be completed due to query logic or file error, or -1 if the user pressed [Esc] to cancel the search. The error number corresponds to an error message in the reference manual.

Programs should test both the return code and the query number to guarantee that a search was successful.

Example

```
int rc,   /* Return code from search(). */
 oldQueryNumber; /* Previous query number. */
 ...
 /* Note that XYZ CO, a fixed field search, is enclosed in quotes. This is
 oldQueryNumber = db.query;
 if(rc=search(db,'CUSTNO="XYZCO"andPAID="F"'))
  puts(10,30,"Error"+str(rc)+"duringsearch.");
 else
  if (oldQueryNumber <> db.query)
   processSuccessfulQuery(db);
```

select

Summary

```
int select(int db; char string[]);
```

Description

This function is only included for compatiblity with releases prior to 5.20. All programs should now use the search() function.

Return Value

See search()

See Also: search()

selectfs

Summary

```
int selectfs(int db; char string[]);
```

Description

This function is only included for compatiblity with releases prior to 5.20.

Return Value

See search()

See Also: search()

set

Summary

```
int | text set(int db; text option [,int|text value]);
```

Description

This is a family of functions which set or retrieve various environmental options in Concordance. Optional parameters are enclosed in brackets [...]. If the optional parameters are passed, the set option is changed to the new value. If the parameters are not passed, no change takes place. In either case, the previous value of the set option is returned by set().

The database handle is only required by Margin, Punctuation, and Empties. For all other set options the database handle is a dummy parameter.

```
text set(db,"Punctuation" [,"&-,"]);
int  set(db,"Margin" [,79]);
int set(db,"Empties" [,TRUE | FALSE]);
int set(db,"Wildcard" [,'*']);
int set(db,"Quote" [,'"']);
int set(db,"Bell" [,TRUE | FALSE]);
```

Example

```
status(int db)
{
 cls();
 puts(2,2,"Margin "+str(set(db,"Margin")));
 puts(3,2,"Punctuation "+set(db,"Punctuation"));
 puts(4,2,"Wildcard "+chr(set(0,"Wildcard")));
 puts(5,2,"Quote  "+chr(set(0,"Quote")));
 puts(6,2,"Bell is  "+(set(0,"Bell") ? "On":"Off"));
 puts(7,2,"Empties are "+(set(db,"Empties")?"On":"Off"));
 getkey();
}
```

shellExecute

Summary

```
int shellExecute(text szOp, szFile, szParams, szDir; int fsShowComd);
```

Description

shellExecute() launches a program, or opens or prints a file or a document. The file specified by the szFile parameter can be a document file or an executable file. If it is a document file, shellExecute() opens or prints it, depending on the value of the szOp parameter. If it is an executable file, this function opens it, even if the szOp string is "print."

Create a NULL parameter by declaring NULL as a text variable. Do not assign a value to it.

Parameter	Meaning
szOp	A string specifying the operation to perform. This string can be "open" or "print." If this parameter is NULL, "open" is the default value.
szFile	A string specifying the file to open.
szParams	A string specifying parameters passed to the application when the szFile parameter specifies an executable file. If szFile specifies a document file, this parameter is NULL.
szDir	A string specifying the default directory.
fsShowCmd	Specifies whether the application window is to be shown when the application is opened. This parameter can be on of the values from the table below.

The fsShowCmd controls the method used to display the application window. Use one of the following parameters for the fsShowCmd.

fsShowCmd Value	Meaning
SW_HIDE	Hides the window and passes activation to another window.
SW_MINIMIZE	Minimizes the specified window and activates the top-level window in the system's list.
SW_RESTORE	Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position (same as SW_SHOWNORMAL).
SW_SHOW	Activates a window and displays it in its current size and position.
SW_SHOWMAXIMIZED	Activates a window and displays it as a maximized window.
SW_SHOWMINIMIZED	Activates a window and displays it as an icon.
SW_SHOWMINNOACTIVE	Displays a window as an icon. The window that is currently active remains active.
SW_SHOWNA	Displays a window in its current state. The window that is currently active remains active.

fsShowCmd Value	Meaning
SW_SHOWNOACTIVATE	Displays a window in it most recent size and position. The window that is currently active remains active.
SW_SHOWNORMAL	Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position (same as SW_RESTORE).

Returns

The program's instance or DDE server handle is returned. A return value less than or equal to 32 indicates an error. Error values are listed in the following table.

Return Value	Meaning
0	System was out of memory, executable file was corrupt, or relocations were invalid
2	File was not found.
3	Path was not found.
5	Attempt was made to dynamically link to a task, or there was a sharing or network-protection error.
6	Library required separate data segments for each task.
8	There was insufficient memory to start the application.
10	Windows version was incorrect.
11	Executable file was invalid. Either it was not a Windows application or there was an internal error in the .EXE image.
12	Application was designed for a different operating system.
13	Application was designed for MS-DOS 4.0.
14	Type of executable file was unknown.
15	Attempt was made to load a real-mode application (developed for an earlier version of Windows).
16	Attempt was made to load a second instance of an executable file containing multiple data segments that were not marked read-only.
19	Attempt was made to load a compressed executable file. The file must be decompressed before it can be loaded.
20	Dynamic-link library (DLL) file was invalid. One of the DLLs required to run this file was corrupt.
21	Application requires Microsoft Windows 32-bit extensions.
31	There is no association for the specified file type or there is no association for the specified action within this file type.

Version

Version 6.62 and later.

See Also: spawn(), system()

show

Summary

```
show(db->field; int TRow, TCol, BRow, BCol, offset, lastRow);
```

Description

This function displays the passed field in a window described by the row and column coordinates. It will highlight all words located in the current query that appear in a paragraph field.

The query list will be positioned on the next item in the list to be highlighted when show() returns. If the last item in the current document is already highlighted on the screen, then the hit list will be positioned on that item. Show() will not cause the hit list to advance to the next document.

Any field can be passed to show() for display, but only paragraph fields will receive highlighting. The text is not wordwrapped before it is displayed as with edit() mode -1.

The lastRow parameter is optional. If provided, it will contain the number of the last screen row used to display data when show() finishes. Use this value to determine the next screen line available for display.

Return Value

The offset of the last line displayed in the text window.

See Also: edit(), mode @

sizeof

Summary

```
int sizeof(value);
```

Description

Determines the size of the variable in bytes. If the variable is an array, it returns the declared number of elements in the array times the size of an individual element. The size of a subscripted array element is the size of the base type.

Taking the size of a database field is invalid.

Sizeof should not be confused with len(), which returns the number of characters stored in a character array or text variable.

Return Value

Integer length of value in bytes.

See Also: len()

Example

```
main()
{
short i, list[20];
puts(0,0,"str(sizeof(i),5,0)+str(sizeof(list),5,0)+str(sizeof(list[7]),5,(
}
```

Output: 2 40 2

sleep

Summary

```
sleep(int winks);
```

Description

Pauses the program for as many thousandths of a second as the winks parameter specifies. If the program is an OS/2 or Windows application, the sleep function returns control to the operating system for at least that amount of time.

Return Value

None

snapshot

Summary

```
int snapshot(int db; text fileName; int takeShot);
```

Description

Saves or restores a full snapshot of the current state of Concordance, which includes all concatenated databases, all searches for the databases, the current record, and the current sort in effect.

takeShot can be any nonzero value to save the snapshot to the fileName parameter. A zero for takeShot causes the named snapshot to be restored, the handle to the newly opened database is returned.

Return Value

A handle to the newly restored database if the snapshot is being restored, or a zero to signify success when saving a snapshot. A nonzero value indicates failure when saving a database, typically caused by a disk full or insufficient write/create rights on a network drive.

See Also: exec(), keep()

Version

Version 5.43 and later

sort

Summary

```
int sort( int db; char string[]; [int row, column; [int color[, backgrounc
```

Description

Sorts the current query according to the string parameter. The string parameter can be a character array, a text variable, or quoted text. The query is sorted in ascending order unless the dc() function is used within the string, see example below. Sort() will display a percentage of progress if the optional row and column parameters are used. The color parameters, which affect the percentage display, are also optional.

Return Value

A -1 if the sort failed, most likely due to a disk full condition or to the user pressing the [Esc] key. If the sort succeeds the current query is sorted, not the physical documents in the database. The sort will stay in effect until the database is closed, until another query is executed, or until the query is reloaded with the query() function.

See Also: dc()

Example

```
main()
{
int db;
/* Sort the current query in ascending order
** by the author field, in descending order
** by the publication date, most recent
** publication first. The sort routine will
** display its percentage of progress on the
** screen at row 5, column 11, in the default
** color.
*/
if ((db = opendb("library")) >= 0)
{
puts(5,2,"Sorting:);
sort(db,"db->author+
dc(str(year(db->date),4)+
str(month(db->date),2)+
str(day(db->date),2))",5,11);
browse(db);
closedb(db);
}
return(0);
}
```

spawn

Summary

```
int spawn(text "program.exe", "parameters");
```

Description

Starts the external program and returns any exit value or completion code returned by the program. This differs from the system() command in that system() calls the operating system command interpreter to invoke the command. system() will handle external programs, .bat files, and internal commands such as dir and copy, but it will not return completion codes.

spawn() will use the current environment path to locate the program if it is not in the current directory. The .exe and .com file extensions are optional. spawn() will not run .bat files.

Return Value

-1 if the program could not be executed. The return code from the executed program if it was run.

See Also: system()

sqrt

Summary

```
float sqrt(float v);
```

Description

Determines the square root of the value.

Return Value

The square root as a floating-point value.

Version

Version 6.0 and later

str

Summary

```
text str(int x[, width[, decimals; [char format]]]);
```

Description

Converts numeric value to character string, with optional commas, dollar signs, decimal point, and width specification. Numeric value, x, can be char or int or float. Format specification can be '$', 'Z', a blank space, or ','. The comma places a comma every third significant digit, the dollar sign format is the comma format with a dollar sign preceding the number, a Z zero fills leading blank spaces, and a blank space formats the number without comma, dollar signs, or zero padding.

The number is formatted right justified within the specified width. If there isn't enough room for the number, a series of asterisks are returned.

Return Value

The number as a character string.

See Also: itoa(), num(), trim()

Example

```
main()
{
int width, decimals, i, format;
width = 5; decimals = 2; format = '$';
for(i = 0; i <= 10; i = i + 2)
puts(i,0,str(i,width,decimals,format));
}
Output:
$0.00
$2.00
$4.00
$6.00
$8.00
***** /* No room to print $10.00 */
```

struc

Summary

```
int struc(int db; text FileName);
```

Description

Creates a new database with the same structure as the currently opened database. The database isn't created if a database already exists with the same name.

The db parameter must be a handle to a currently opened database. The FileName parameter is the name of the new database created by this function. struc() creates the database, but it does not open it for use.

Return Value

Zero if the new database was successfully created.

See Also: createdb()

substr

Summary

```
text substr(text string; int from, width);
```

Description

Makes a copy of the characters in the string, beginning at from characters, and continuing for width characters. From must be greater than or equal to 1.

Width is optional. If it is left off, substr will return a copy of the entire string beginning at from.

Return Value

A copy of the characters in the string.

See Also: addr()

Example

```
main()
{
text a, b;
a = "Joe's Diner in the Park";
b = substr(a,match(a,"Diner",1));
/* b is now "Diner in the Park" */
}
```

system

Summary

```
int system(text command);
```

Description

Passes the command string to the operating system for execution. This can be used to execute operating system commands such as DIR, or COPY, or to run external programs.

Return Value

A 0 if successful, a nonzero value if not successful. Reasons for failure are insufficient memory or the system was not able to find the program in the command string for execution.

Example

```
/* Program fragment to display a document on the screen
** and show a graphics image when ALT-V is pressed. */
ViewDocument(int db, document)
{
int key;
goto(db,document);
/*Call subordinate function to display browse screen.*/
ShowDoc(db);
while(key <> ESC)
{
switch(key = getkey())
{
case ALTV: system("view "+db->graphic);
case '+': if (next(db) > 0)
ShowDoc(db);
case '-': if (prev(db) > 0)
ShowDoc(db);
}
}
}
```

See Also: spawn()

T

The following topic discusses the Concordance Programming Language (CPL) functions that begin with the letter T. For more information on CPL functions, see Functions, About the Advanced Programming Features, and About CPL Functions.

table

Summary

```
table(int db);
```

Description

Invokes Concordance full screen Table View mode.

Return value

None.

See Also: browse()

Version

Version 5.32 and later

tag

Summary

```
int tag(int db, ON|OFF|CLEAR [, text tagString]);
```

Description

Tags a document in the database whose handle is db according to the value of the second parameter:

ON 1 Document is tagged.

OFF 0 Document is untagged.

CLEAR -1 All documents in the database are untagged.

Remember to clear all tagged queries before beginning a tagging cycle. Otherwise, documents previously tagged will be included in the current tagging operation. Document tags are stored in a file called database.trk, they are retained with the database and do not disappear if the database is closed and reopened.

If the optional tagString parameter is passed, the supplied tag is applied to the document. Otherwise the default tag, "", is applied.

Return Value

Returns a zero if no error occurred. Returns a nonzero value if an error occurred while tagging the document.

See Also: istagged(), tagquery()

tagquery

Summary

```
tagquery(int db[, text tagString]);
```

Description

Collects all tagged documents in the database whose handle is db into a single query. If the optional tagString is supplied, only documents tagged to the tagString are selected. The set of tagged documents becomes the current active query.

Return Value

None. Check the database information variable db.query to see if a tagged query was actually created. The function will create a tagged query with zero documents if no tagged documents were found.

See Also: tag(), istagged()

time

Summary

```
time(int hours, minutes, seconds);
```

Description

Returns the current time in the passed parameters. The hours are in military format, 1 - 24 hours. Minutes and seconds are optional parameters.

Return Value

Returns the total seconds elapsed since midnight, (hours * 3600) + (minutes * 60) + seconds. The passed parameters are set to the current time when the function returns.

See Also: clock(), today()

Example

```
ShowTime()
{
int hours, minutes, seconds;
text AMorPM;
char string[10];
/* Display time on the screen. */
/* Get the time in a string. */
time(hours, minutes, seconds);
AMorPM = " p.m.";
if (hours < 12)
AMorPM = " a.m.";
if (hours > 12)
hours = hours - 12;
string = str(hours,2,0,'Z')+":"+str(minutes,2,0,'Z')+":"+str(seconds,2,0,
```

```
puts(0,0,string+AMorPM);
}
```

today

Summary

```
int today();
```

Description

Gets today's date in internal numeric format.

Return Value

Integer representing today's date.

See Also: dtoc(), ctod(), clock(), time()

Example

```
WhatDayIsIt()
{
puts(0,0,"Today is "+dtoc(today()));
}
```

trim

Summary

```
text trim(text string);
```

Description

Removes all trailing and leading blanks from the text variable, character array, or field.

Return Value

Returns a duplicate of the string, does not modify the original.

See Also: ltrim(), rtrim()

Example

```
main()
{
text line;
line = " This is a fine mess ";
puts(0, 0, "|"+trim(line)+"|");
puts(1, 0, "|"+line+"|");
}
Output:
|This is a fine mess|
| This is a fine mess |
```

U

The following topic discusses the Concordance Programming Language (CPL) functions that begin with the letter U. For more information on CPL functions, see Functions, About the Advanced Programming Features, and About CPL Functions.

unload

Summary

```
int unload(int db; char string[]; [int comma, quote, newline; [int row, c
```

Description

Unloads the current query to the file specified by the string file name. The string must be a valid file name. The documents are unloaded in delimited ASCII format as documented in the Concordance Reference Manual.

The fields unloaded, and the order in which they are unloaded, is set by assigning a number to the order entry of the field definition. You can assign any number from -128 to 127, but Concordance will only unload fields with consecutive numbers from 1 to the last field number defined in the database. You should always renumber the fields before any unload. Concordance allows the user to change the field order in several full screen modes including Print, Load, and Unload. See the for-loop example below.

The optional parameters, comma, quote and newline are used to delimit the fields during the unload process. If they are left off Concordance will use its internal ASCII default values, 20, 254, and 174 respectively. It is recommended that you leave these parameters off if transferring data between Concordance databases, however they are required if row, column, and color are passed.

Return Value

The number of documents unloaded. This should be checked against the number of documents that should have been unloaded. Any inequality will indicate full disk or other disk error condition, or that the user pressed [Esc] to cancel the unload.

See Also: unloadfs(), load(), loadfs()

Example

```
main()
{
int db, i, count;
/* Permanately sort all records in the */
/* database in ascending order. */
if ((db = opendb("helpdesk")) <> -1)
{
sort(db,"db->DATE");
/* Renumber all fields to unload in */
/* order. This step is necessary */
```

```
/* since the order previously set, */
/* or set in Print, is retained */
/* until reset. Never assume the */
/* order includes all fields. */
for(i = 1; i <= db.fields; i = i + 1)
db.order[i] = i;
count = db.documents;
if (unload(db,"helpdesk.dat") <> count)
puts(0,0,"Error unloading database.");
else
{
zap(db);
if (load(db,"helpdesk.dat") <> count)
puts(0,0,"Error reloading data.");
else
{
erase("helpdesk.dat");
index(db);
}
}
closedb(db);
puts(0,1,"Press any key to continue...");
getkey();
}
}
```

unloadfs

Summary

```
unloadfs(int db);
```

Description

Invokes Concordance full screen unload mode. The screen is automatically saved before entering this mode and restored after exiting.

Return Value

None.

unlockdb

Summary

```
unlockdb(int db);
```

Description

The database whose handle is db is unlocked and released from exclusive use.

Return Value

None.

See Also: lockdb()

unlockdoc

Summary

```
unlockdoc(int db);
```

Description

Will unlock the current document in the network version of Concordance. Unlocking the record will force an immediate write to disk if it has been edited. This has no effect in non-network versions of the program. A locked document cannot be changed and saved to file by other users. Reading another document will automatically unlock a document.

Documents are automatically locked by Concordance when read into memory. Your program should unlock them as a courtesy to other network users if you do not intend to edit or modify them.

Return Value

None.

See Also: lockdoc(), locked()

unmapDevice

Summary

```
int unmapDevice(text Device, int force);
```

Description

Removes the network's mapping of the disk drive or printer. The unmapping fails if the device has a file or lock open. Set the force parameter to TRUE to force the device to unmap regardless of the open file status. (TRUE is a predefined integer. You do not need to declare it. Do not assign a value to it. Do not pass it in quotes. Simply use it as you would any integer.)

Return Value

Zero indicates success.

See Also: mapDevice()

Example

```
unmapDevice("F:", FALSE);
Version
Version 7.30 and later
```

upper

Summary

```
text upper(text string);
```

Description

Converts the parameter string to all upper case. The parameter can be a text value or a character array.

Return Value

Returns a duplicate of the string in upper case letters, does not modify the original.

See Also: capitalize(), lower()

V

The following topic discusses the Concordance Programming Language (CPL) functions that begin with the letter V. For more information on CPL functions, see Functions, About the Advanced Programming Features, and About CPL Functions.

ver

Summary

```
float ver([char string[]]);
```

Description

Version number of Concordance.

Return Value

The release version number of Concordance. If the optional string parameter is provided, it will contain a detailed description of the version, Concordance Runtime, or Concordance/386 Network.

W

The following topic discusses the Concordance Programming Language (CPL) functions that begin with the letter W. For more information on CPL functions, see Functions, About the Advanced Programming Features, and About CPL Functions.

weekday

Summary

```
text weekday(int d);
```

Description

Computes the day of the week for the parameter date and converts it to a text string. The parameter can be a date field or the result of the ctod() function.

Return Value

The name of the weekday.

See Also: day(), month(), year(), ctod(), dtoc()

wordlen

Summary

```
int wordlen(int db; text field);
```

Description

Determines the length of the word in the character array, text variable, or database field. The length is determined by using the data base embedded punctuation.

Return Value

Length of the word, or 0 if the first letter of the word isn't alphanumeric.

wrap

Summary

```
text wrap(text string; int width);
```

Description

Wordwraps the text within a column of width characters. The text can be a database field, character array, or text variable.

Return Value

The parameter text passed to wrap() is returned. wrap() does not return a duplicate of the text, but the actual text parameter fully wrapped. The statement

wrap(db->SUMMARY,20)

is equivalent to

db->SUMMARY = wrap(db->SUMMARY,20)

except that the second statement provides additional processing work for Concordance without any effect whatsoever. wrap() returns the parameter textas opposed to a copyso that it can be passed as a parameter to other functions, such as show() or write(). Returning the original parameter also avoids insufficient memory errors when wordwrapping large text fields.

See Also: findline(), findnline(), findpline()

write

Summary

```
int write(int handle; text buffer; int length);
```

Description

Writes length number of bytes to the file referenced by handle from the buffer. Handle must be a valid file handle returned by a call to open().

Return Value

The number of bytes written to file. If this value does not equal length, it indicates a disk full condition. A value of -1 indicates an error.

See Also: writeln(), read(), readln(), open(), close()

writec

Summary

```
int writec(int handle; char ch);
```

Description

The character is written to the file referenced by handle. The handle must have been returned by the open() function.

Return Value

Returns a nonzero value if successful, and a -1 if an error is encountered.

See Also: readc(), open(), close()

writeln

Summary

```
int writeln(int handle; text buffer; int length);
```

Description

Writes the contents of the buffer to file and prints a carriage return-line feed after the text.

Return Value

Returns the number of characters written to file, or a -1 if an error was encountered. If the number of characters written is less than the length parameter, the disk is probably full.

See Also: write(), read(), readln(), open(), close

writePrivateProfileString

Summary

```
int writePrivateProfileString( text szSection;
text szKey;
text szValue;
text szFile);
```

Description

The writePrivateProfileString() copies a string into the specified section of the specified initialization file.

Parameter	Function
szSection	The name of the section to which the string is copied. If the section does not exist, it is created. The name of the section is not case sensitive.
szKey	The name of the key to be associated string with a string. If the key does not exist in the specified section, it is created. If this parameter is NULL, the entire section, including all entries within the section, is deleted. Create a NULL value by declaring NULL as a text variable, but do not assign any value to it.
szValue	szValue is written to the file. If this parameter is NULL, the key pointed to by the szKey parameter is deleted.
szFile	The name of the initialization file. This must be the fully qualified name, with the .ini file extension.

Return Value

Nonzero if successful.

See Also: getPrivateProfileString()

Version

Version 7.0 and later.

X

There are currently no Concordance Programming Language (CPL) functions that begin with X. For more information on CPL functions, see Functions, About the Advanced Programming Features, and About CPL Functions.

Y

The following topic discusses the Concordance Programming Language (CPL) functions that begin with the letter Y. For more information on CPL functions, see Functions, About the Advanced Programming Features, and About CPL Functions.

year

Summary

```
int year(int duedate);
```

Description

Extracts the year from the date. The date can be a value returned by ctod() or a date field.

Return Value

The year in integer format.

See Also: day(), month(), weekday()

Z

The following topic discusses the Concordance Programming Language (CPL) functions that begin with the letter Z. For more information on CPL functions, see Functions, About the Advanced Programming Features, and About CPL Functions.

zap

Summary

```
int zap(int db);
```

Description

Removes every document from the database, erases all associated database files except for the stopword file, .key and .dcb files. Use with caution, this is a destructive command.

Return Value

Zero if zap() was successful, nonzero if zap() was not able to clear the database. This can happen if the database has read-only access, or if it is in use by other network users.

Concordance Scripts

About CPL Scripts

The Concordance scripts listed below are delivered with Concordance. The customer support team can help resolve issues or answer questions regarding these scripts. For more information about editing and running CPL scripts, see Creating and Editing a Concordance Script and Running a Concordance Application.

If you do not have access to the CPLs in the Concordance install directory, the CPLs are available for download here:

- Concordance version 8.x
- Concordance version 9.x
- Concordance version 10.x

CPL Name	Description	Usage
AppendOneFieldToAnother_v10.00	Uses a field based on options the user selects and appends, prepends, or copies data from one field to another. Append: Copying one field to the end of another.	Global variables are capitalized in this program. Local variables are in lower or mixed case. Variable names in all upper case, i.e., LEFT, are initialized once and should not be changed afterwards.
AppendTextToField_v10.00	Appends or prepends data to a field, leaving existing data intact.	Global variables are capitalized in this program. Local variables are in lower or mixed case. Variable names in all upper case, i.e., LEFT, are initialized once and should not be changed afterwards.
BlankField_v10.00	Erases data from a user-specified field.	To start, edit the FIELDNAME entry to represent the name of the field that you want to blank out. The FIELDNAME needs to be entered exactly as it displays in the database. The field specified should be a Text or a Paragraph field. Once this is done, you will need to save the CPL and launch it from within Concordance. **Note:** Running this on a Numeric or a Date field will not blank the field. Date fields will continue to hold a date of 00/00/0000 and Numeric fields will simply display 0.
BulkConvertImagesbases_v10.00	Converts all imagebases to the selected version in the directory you specify.	This program can bulk convert imagebases to version 5. Imagebase versions will be recorded in whole numbers and not decimals. For example, imagebase versions 3 and 4 will display as 3 or 4, but not 3.x or 4.x. Once you select a directory, the program will convert all imagebases to the selected version in the directory you specify and all its subdirectories.
ConvertTextToDate_v10.00	Converts dates in a text field to a valid date in a date field. Preserves invalid dates, like 12/00/92 or 00/00/00. All dates	

CPL Name	Description	Usage
	are assumed to be in MM/DD/YY format.	
CreateHyperlinks_v10.00	Cycles through the current query of an e-mail database and creates hyperlinks from the file paths listed in the ATTACHMENT field.	Does not need user modifications to run properly.
EDocView_v10.00.cpl	Uses the Concordance View Image button to launch the native document in its native application from within an e-document database. This CPL looks for a field named "FILEPATH", so if you renamed the field to e.g., "ATTACHMENT", then the CPL will need to be edited to replace all instances of "FILEPATH" with "ATTACHMENT." **Note:** This CPL cannot be used on a database with an existing imagebase of TIFF images.	
FieldToTag_v10.00	Copies the contents in a field to a tag, as identified by the user.	There is a line in the CPL that defines "pszField". As delivered, this CPL assigns the value "MYFIELD" to pszField. Then the CPL creates a tag with the same name as the contents of MYFIELD. To use this CPL, alter "MYFIELD" to the field name that contains the tagging information you want to convert and run the CPL.
FindAttachements_v10.00	Uses an attachment range field to find attachments for the current query as opposed to the current document. **Note:** Attach.cpl and Attach2.cpl have been replaced by FindAttachments.cpl and FindAttachments2.cpl and are not installed in Concordance version 9.52 and higher.	There is a line in the CPL that defines "ATTACHFIELD" and "ATTACHTAG". As delivered, this CPL assigns the value "ATTACHMENT" to ATTACHFIELD and "Attachments" to ATTACHTAG. For each document in the query, this CPL then tags each document that has an "ATTACHMENT" field and tags it with "Attachments". This CPL runs a series of relational searches (equal to the number of records in the current query), using the information contained in the

CPL Name	Description	Usage
		user defined "ATTACHFIELD", across the contents of the entire database. A temporary tag is created by the CPL with a user defined name taken from the "ATTACHTAG" entry. At the completion of this CPL the results are displayed in the new current query and the temporary tag is removed. The user needs to change the "ATTACHMENT" entry in the CPL to reflect the name of the field that holds the attachment range.
FindAttachements2_v10.00	Uses a beginning and an ending attach field to bring together attachment ranges from the current query as opposed to the current document.	
IssueToTag_v10.00	Converts all issues from the current query into tags.	Should not need modification. It prompts the user for the information it needs using message boxes. After the information is collected, this program performs the conversion on the current search list.
LoadOCRFromOpticonLog_v10.00	Used when you have single-page OCR .txt files in the same directory as the .tif files. With the Concordance Image log file pointing to the images, the OCR text files can be loaded into the database. The CPL prompts for the log file and the field to contain the OCR text. This script automatically overflows to the next field if the first field's content exceeds 12MB and you have sequentially numbered the OCR fields.	This CPL requires that the following directory exist on the machine running the script: C:\temp\convert. A log file named ocrdcb.txt is written to the "convert" folder as the script finishes.
Mark_v10.00.cpl	Considered a *sub-program* by other CPL scripts, like Spell.cpl. This script allows users to select fields from the currently opened database for subsequent action by the main program. Use Mark.cpl if you encounter the message prompt, *Couldn't find Mark.cpl* when running another	There is nothing the user needs to change in this program to get it to perform. Menus will guide the user through all aspects of this program's functionality.

CPL Name	Description	Usage
	program. **Note:** You will need to update Mark.cpl with the version number appended to the CPL name. For example, for version 10.00, change Mark.cpl to Mark_v10.00.cpl.	
PDFPrint_v10.00.cpl	Automatically prints the current query of a Concordance E-documents database using an Adobe Acrobat viewer, such as Adobe Reader, as the print application. This utility is part of the Concordance E-Documents template, and is launched from the Adobe print menu item.	
PrintWithAttachments_v10.00	Prints documents and their attachments. This script uses the command shellExecute() to prompt the operating system to use the program associated with the file type. If a program is not associated with the file type, then it will not print the document. Errors are logged to a file called Print-With-Attachments Error.log.	You must create and save a print format file to the same directory as this program. Call the print format file: Print-With-Attachments.fmt. It is used to specify the print formatting options.
READOCR1 (singlePage)_v10.00	Cycles through the current query and locates the image field, then translates the image into a filename for the corresponding OCR text. This script reads the file and writes content into the specified paragraph field.	Does not need user modifications to run properly. This revision is for databases that contain the full path, file name, and extension for the OCR.txt files.
readOCR1_v10.00	Cycles through the current query and locates the image field, then translates the image into a filename for the corresponding OCR text. This script reads the file and writes content into the specified paragraph field.	Does not need to be modified to run properly.
readocr_v10.00	Cycles through the current query and locates the image field, then translates the image into a filename for the corresponding OCR text. This script reads the file and writes content into the specified paragraph field.	Does not need to be modified to run properly.

CPL Name	Description	Usage
ReindexingDaemon_v10.00	Reindexes all databases that have the file path location written into a log file.	There are several lines in the CPL that define szUserID, szPassword, and LogFile. Put your user id, password, and the log file name between the quotation marks in the proper variables. Then run the program.
Renumber_v10.00	Assumes that a search has already been performed to isolate documents that need to be numbered. This script prompts for a field name, a starting numeric value, a range, and an optional alphabetic prefix. It assigns values to the database field for every document in the current query. ***Example:*** ABC00005 through ABC0949	Does not need to be modified to run properly.
ShowSystemFields_v10.00	Displays system fields.	Does not need user modifications to run properly.
Spell_v10.00	Scans documents and prompts the user to correct misspelled words. Creates a file to store skipped words. The file name is created by combining the database path/name with the file extension: .SPL. The file is stored in the same directory as the database.	Does not need modification to run properly. Can run from Concordance, or called with the Run() function by passing Spell() the database handle.
Synonym_v10.00.CPL	Loads a text file containing synonyms into a Concordance .syn B-tree file for Search mode. 1. A blank line separates one synonym group from another. 2. Words in a group (not preceded by anything) are stored as synonyms. 3. Words preceded by a minus sign are sub-categories of the main word. 4. Words preceded by a plus sign are stored as synonyms AND have their own sub-	Does not need modifications to run properly.

CPL Name	Description	Usage
	groups. ***Examples:*** Words preceded by a "-" are stored as related terms but are not stored as synonyms. In the following example the word "colors" would pull up red, green, and blue. But searching for red would only search for red. colors -red -green -blue This allows you to create additional synonym groups as subcategories. Use the "+" sign if a word in a synonym group has its own group. colors +red -green -blue red reddish -ochre -brick Now *red* has its own group and its own set of narrower terms. In this case searching for *colors* would pull up red, which in turn would pull up the red group. Searching *red* or *reddish* would pull up ochre and brick as well. However, searching for ochre, brick, green, or blue would not pull up any additional synonyms. The synonyms are not limited to single word entries, they can contain any query logic that can be entered in the Search task pane. This includes parenthetical logic, search operators, and even fixed field searches. Note that synonyms on different lines are OR'ed	

CPL Name	Description	Usage
	together during the search processing.	
TagHistoryAndStoreIt_v10.00	Writes the tag history held in the .trk file to a paragraph field in the database named TAGINFO.	
TAGSAVER_v10.00	Writes the current list of tags from the database into a .gat file. The .gat file can be stored for future use, in case tags are lost due to corruption or user error. When prompted to select a field, please be sure to select one that contains unique values in your database, as this is the linking field that associates each record with its tags. For example, in the CALFCO database, this field would be "STARTPAGE", but in other databases you might need to select e.g., "BEGDOC".	Does not need modifications to run properly. **Note:** If the database is new and has never been indexed, index the database before running the TAGSAVER CPL.
TagToField_v10.00	Copies the a tag name to a field, as identified by the user.	Before launching the CPL, create or designate a database field to hold the tag information. When the CPL is started, the user is prompted for the following information. 1. **Open database:** Database name 2. **Tag field:** Field designated to hold the tags 3. **Delimiter:** Character used to separate the tags After this information is entered, the user selects Go to run the CPL.
TextFileToQuery_v10.00	Reads an ASCII text file and runs a query on each line of the file. Each line can be written for a full-text or relational search. When the process is completed, the text file results are concatenated into a single query. There is no known limit to the number of lines that can	There is a line in the CPL that defines "ATTACHTAG". As delivered, this CPL assigns the value "Attachments" to ATTACHTAG. Then the CPL tags the new query results from the ASCII file with the contents of ATTACHTAG.

CPL Name	Description	Usage
	be run from the text file.	
UpperCase_v10.00	Converts text to upper case. **Note:** Running UpperCase_v10.00.cpl changes all field text to the default font settings. For example, if the default font is Arial 9 and the field text is Times New Roman 12 bold, running UpperCase_v10.00.cpl changes the field text to upper case and also changes the font to Arial 9 regular.	Does not need modifications to run properly.

AppendOneFieldToAnother_v10.00

Use the AppendOneFieldToAnother CPL to copy the contents of one field into another. The contents can be added to the beginning or end of the content in the destination field.

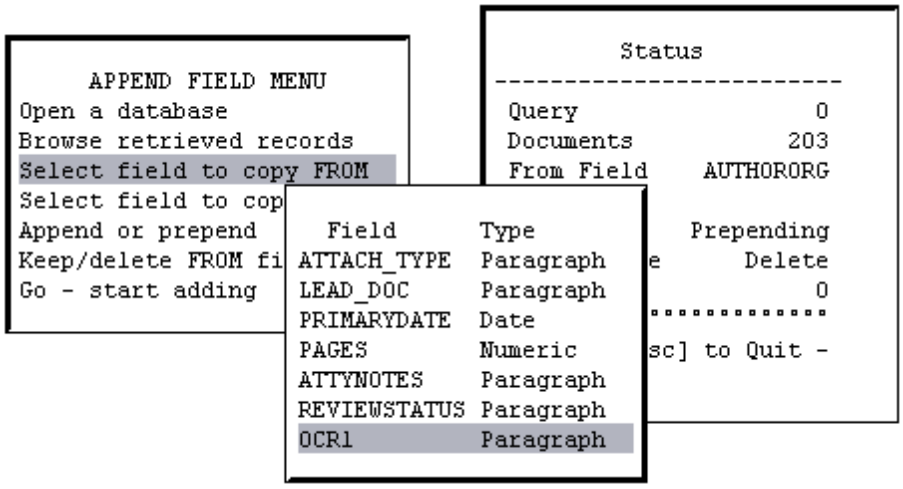
The destination field must be of type TEXT or PARAGRAPH.

The AppendOneFieldToAnother CPL works with the following versions of Concordance:

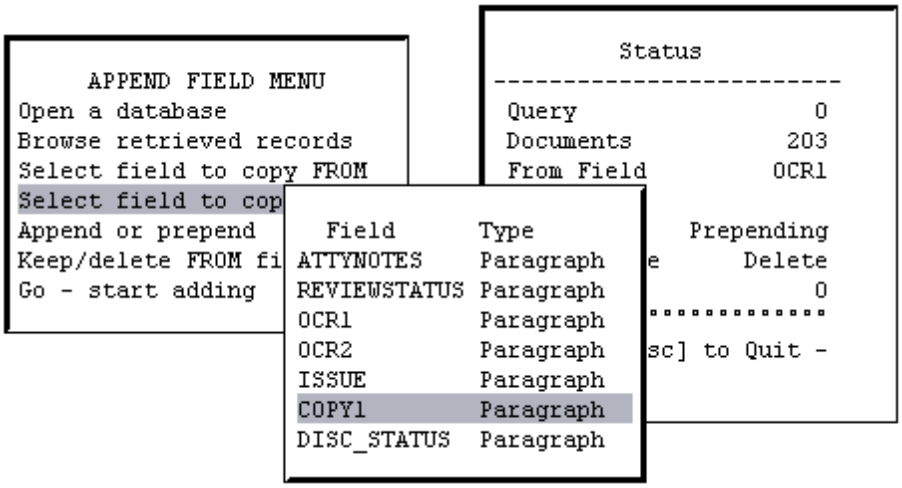
- 8.*x*
- 9.5*x*
- 10

To run the AppendOneFieldToAnother_v#.cpl:

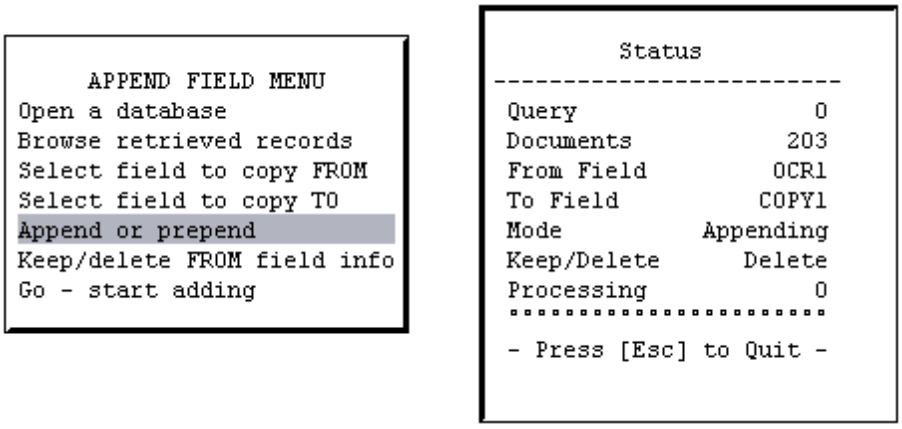
1. On the **File** menu, click Begin Program.
2. Locate and open the **AppendOneFieldToAnother_v[version #].cpl** file.
3. Click **Select field to copy FROM**, use the arrow keys to select the field name that the data is copied from, and then press Enter.



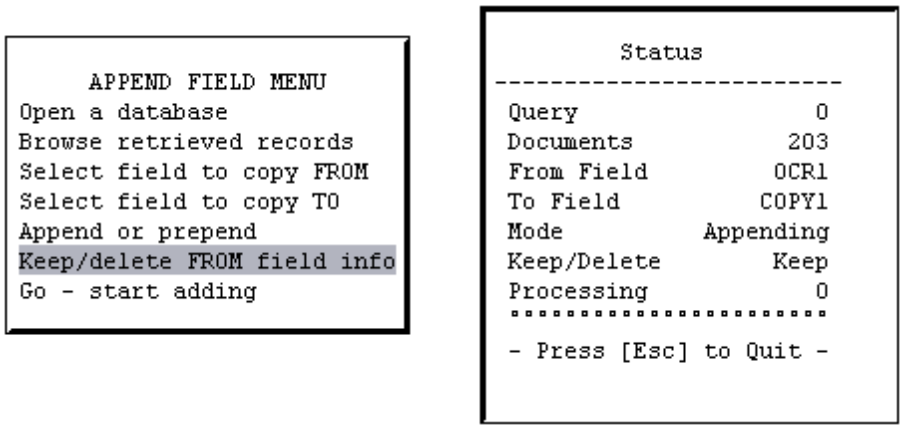
4. Click **Select field to copy TO**, use the arrow keys to select the field name that the data is copied into, and then press Enter.



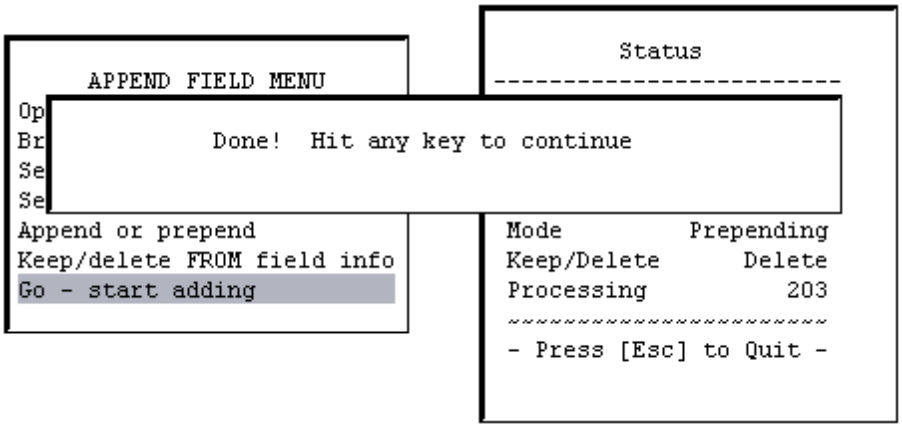
5. Click **Append or prepend** to toggle the Mode field of the Status dialog box to define whether the data is added to the end (**Append**) or the beginning (**Prepend**) of the designated field contents.



6. Click **Keep/delete FROM** field info to retain or remove the data from the initial source field when the CPL is executed, and then click Go.



7. After the CPL is finished, press any key to continue, and then verify that the data from the initial field is copied correctly into the destination field.



AppendTextToField_v10.00

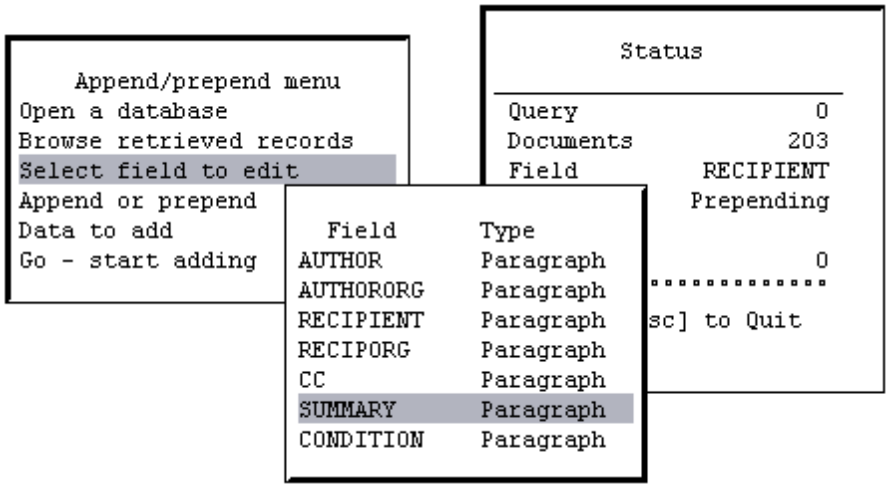
Use the AppendTextToField CPL to add new data to the existing contents of a specified field. The new data can be added to the beginning or end of the content in the destination field.

The AppendTextToField CPL works with the following versions of Concordance:

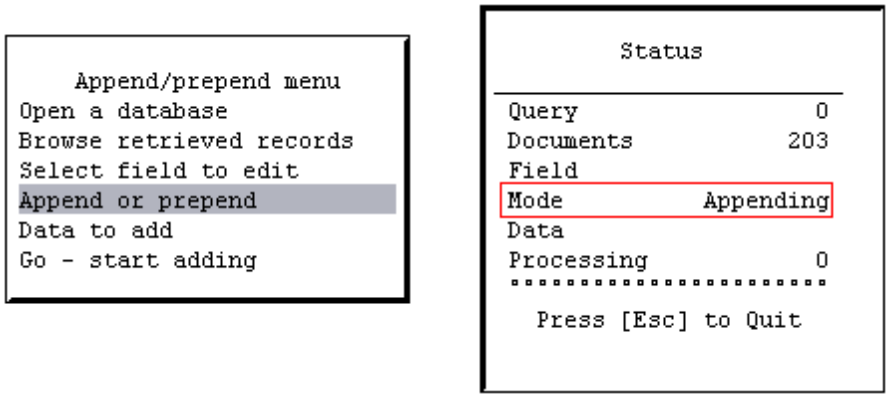
- 8.*x*
- 9.5*x*
- 10

To run the AppendTextToField_v[version #].cpl:

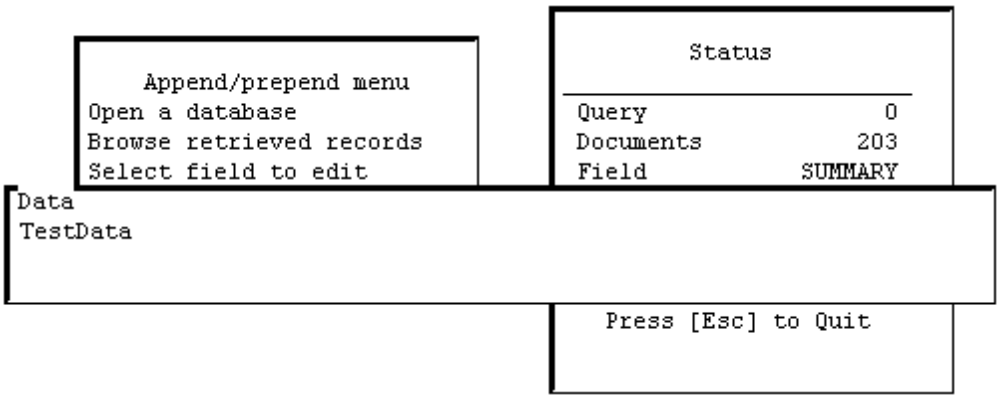
1. On the **File** menu, click **Begin Program**.
2. Locate and open the **AppendTextToField_v[version #].cpl** file.
3. Click **Select field to edit**, use the arrow keys to select the field name to add the data, and then press Enter.



4. To define whether the data is added to the end (**Append**) or the beginning (**Prepend**) of the designated field contents, click **Append or prepend** to toggle the **Mode** field of the **Status** dialog box



5. Click **Select data to add**, enter the data you want to append/prepend to the designated field, and then press Enter.



6. When finished, click **GO - start adding**.

7. After the CPL is finished, press ESC, and then verify that the data you entered is added correctly to the specified field.

BlankField_v10.00

Use the BlankField CPL to remove data from a specified field leaving it empty.

> Because this CPL removes data, it is highly recommended that you backup your database before running the CPL.

The BlankField CPL works with the following versions of Concordance:

- 8.*x*
- 9.5*x*
- 10

To run the Blankfield_v[version #].cpl:

1. Using any text editing application, locate and open the **BlankField_v[version #].cpl** file.
2. Locate the text **FIELDNAME**, replace the text with the name of the field you want to make a blank field, and then save the file.

```
main() {
int db;
   cycle(db)  {
   puts(0,0, "Now processing document " + str(docno(db),8,0,',') + " of " + str(count(db),8,0,','));
         FIELDNAME = "";
  }
}
```

Before

```
main() {
int db;
   cycle(db)  {
   puts(0,0, "Now processing document " + str(docno(db),8,0,',') + " of " + str(count(db),8,0,','));
         OCR2 = "";
  }
}
```

After

3. On the **File** menu, click **Begin Program**.
4. Locate and open the **BlankField_v[version #].cpl** file.

   As soon as you open the CPL file, it automatically executes the script.
5. When the CPL is finished, verify that the field you specified is empty.

CreateHyperlinks_v10.00

Use the CreateHyperlinks CPL to create a Concordance attachment or hyperlink to an external document. This CPL supports multiple attachments.

The CreateHyperlinks CPL works with the following versions of Concordance:

- 8.*x*
- 9.5*x*
- 10

To run the CreateHyperlinks_v[version #].cpl:

1. Using any text editing application, locate and open the **CreateHyperlinks_v[version #].cpl** file.

2. Using the text editor's Find and Replace features, replace the name of the current field specified in the CPL (default value "ATTACHMENT") with the name of the field you want to convert to a hyperlink.

```
main()
{
int db, i, length, nAnnotationCount;

   cycle(db) {

      Message("Processing "+str(docno(db))+" of "+str(count(db)), FALSE);

      if (len(db->ATTACHMENT) > 0) {

         /* Delete any existing annotations */
         nAnnotationCount = annotationCount(db);
         for (i = nAnnotationCount; i >= 1; i = i - 1) {
            if (annotationGoto(db, i) == 0) {
               if (annotationRetrieve(db, "ATTACHMENTFIELD") == "ATTACHMENT")
                  annotationDelete(db);
            }
         }

         i = findline(db->ATTACHMENT, 1, length);
         while(i > 0) {
            annotationAppend(db->ATTACHMENT, i, length, "Launch "+capitalize(FileName(substr(db->ATTACHMENT, i,
            length))), capitalize(substr(db->ATTACHMENT, i, length)), TRUE, NOTEATTACHEXTERNAL);
            i = findnline(db->ATTACHMENT, i, length);
         }
      }
   }
   return;

} /* main() */
```

Before

```
main()
{
int db, i, length, nAnnotationCount;

   cycle(db) {

      Message("Processing "+str(docno(db))+" of "+str(count(db)), FALSE);

      if (len(db->LINK) > 0) {

         /* Delete any existing annotations */
         nAnnotationCount = annotationCount(db);
         for (i = nAnnotationCount; i >= 1; i = i - 1) {
            if (annotationGoto(db, i) == 0) {
               if (annotationRetrieve(db, "LINKFIELD") == "LINK")
                  annotationDelete(db);
            }
         }

         i = findline(db->LINK, 1, length);
         while(i > 0) {
            annotationAppend(db->LINK, i, length, "Launch "+capitalize(FileName(substr(db->LINK, i, length))),
            capitalize(substr(db->LINK, i, length)), TRUE, NOTEATTACHEXTERNAL);
            i = findnline(db->LINK, i, length);
         }
      }
   }
   return;

} /* main() */
```

After

In this example, *ATTACHMENT* is replaced with the field name *LINK.*

3. Review the changes to make sure that all the modifications were made.
4. In Concordance, verify that the field contains the full path to the locations of the document you are linking to. If you are setting up links to multiple files, make sure that the links are separated by a hard return.

   LINK D:\Data
   \sample.txt
   D:\Data
   \sample2.txt

5. On the **File** menu, click **Begin Program**.
6. Locate and open the **CreateHyperlinks_v[version #].cpl** file. The CPL automatically executes the code.
7. When the CPL is finished, verify that the following changes are true for the field:
   - Links are highlighted.
   - If you right-click the link, click **Edit Note**, and then click Selecting the Attachment the document path is displayed.
   - If you click the link, the document opens.

   LINK D:\Data
   \sample.txt
   D:\Data
   \sample2.txt

EDocView_v10.00

Use the EDocView CPL to view native files using the camera button.

The EDocView CPL works with the following versions of Concordance:

- 8.*x*
- 9.5*x*
- 10

This CPL requires a field containing a full file path to a native file..

To run the EDocView_v[version #].cpl:

1. Open the **EDocView_v[version#].cpl** in an text editor (Notepad, TextPad, UltraEdit)

2. In the text editor, replace the name of the current field specified in the CPL with that of the field you will be running the EdocView CPL on. By default this field is called *FILEPATH* and this name needs to be replaced on line 40.

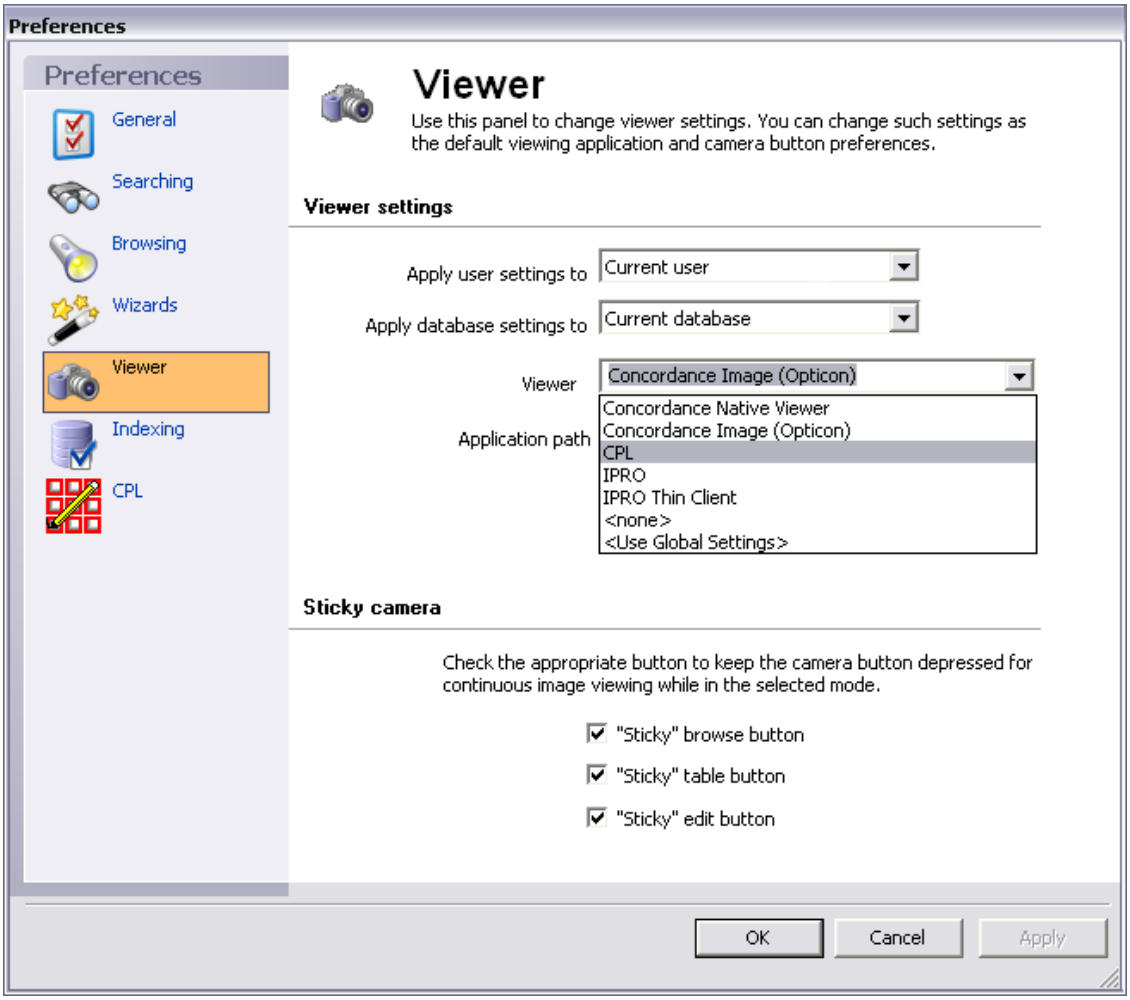
```
39      /* Get the field containing the path */
40      if ((f = isfield(db, "FILEPATH")) == 0)
41          f = isfield(db, "PDFFILE");
```

Before

```
39      /* Get the field containing the path */
40      if ((f = isfield(db, "NATIVEFILE")) == 0)
41          f = isfield(db, "PDFFILE");
```

After

3. In Concordance, on the **Standard** toolbar, click the **Tools** button.

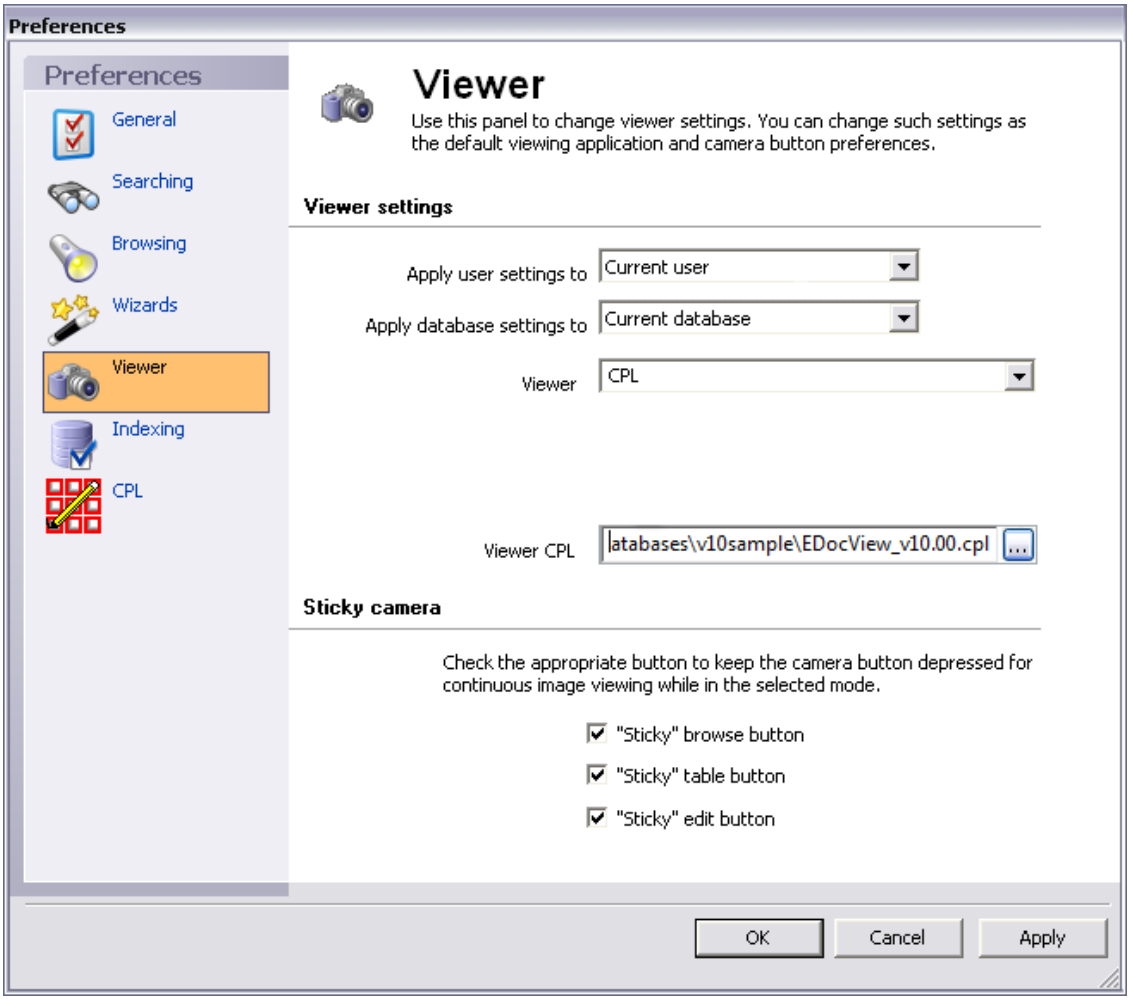


4. In the Preferences dialog box, click the **Viewer** tab.

5. In the **Viewer settings** section, from the **Viewer** list, click **CPL**.

6. In the **Viewer CPL** field, click the **Browse** button, and navigate to and select the edited

CPL file.



7. When finished, click OK to close the Preferences dialog box.
8. Restart Concordance.
9. Click the **View Image** (camera) button. The document associated with the current record should open in Concordance Image.

FieldToTag_v10.00

Use the FieldToTag CPL to create tags from entries in a specified field.

The FieldToTag CPL works with the following versions of Concordance:

- 8.*x*

- 9.5*x*
- 10

Make sure that you are using the correct CPL for your version of Concordance.

To run the FieldToTag_v[version #].cpl:

1. Using any text editing application, locate and open the **FieldToTag_v[version #].cpl** file.
2. Locate and replace the value for **pszField** (default value "MYFIELD") with the name of the field you want the contents converted to a tags.

```
text pszField = "MYFIELD";
text pszDelimiter = ";";
```

Before

```
text pszField = "TAGS";
text pszDelimiter = ";";
```

After

3. Locate and replace the delimiter value for **pszDelimiter** (default value ";").

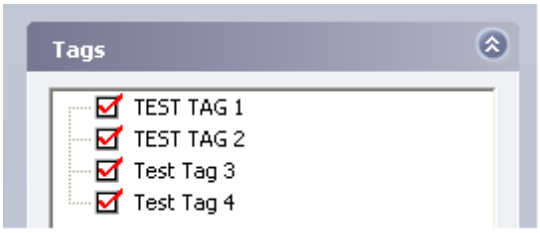
```
text pszField = "TAGS";
text pszDelimiter = ";";
```

4. When finished, save the file.
5. On the **File** menu, click **Begin Program**.
6. Locate and open the **FieldToTag_v[version #].cpl** file. The file automatically executes the script.
7. When the CPL is finished, verify that the **Tags** pane contains tags that match the values in the specified field. For every value in the field, you should have a matching tag.

   For example, if the field you specified contains the following values:

   TAGS TEST TAG 1; TEST TAG 2; TEST Tag 3; TEST Tag 4

   The Tags pane displays the following tags:



FindAttachements_v10.00

Use the FindAttachments CPL to execute an active query that retrieves all attachment documents for the specified attachment range field.

This CPL requires one field containing the attachment range information.

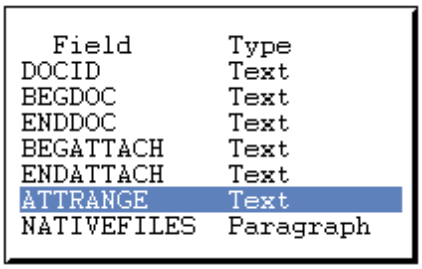
The FindAttachments CPL works with the following versions of Concordance:

- 8.*x*
- 9.5*x*
- 10.19 or earlier

Make sure that you are using the correct CPL for your version of Concordance.

To run the FindAttachments_v[version #].cpl:

1. On the **File** menu, click **Begin Program**.
2. Locate and open the **FindAttachments_v[version #].cpl** file.
3. Select the field that contains the attachment range, and then press Enter.



   The CPL automatically executes and then returns you to the database.

4. When the CPL is finished, verify that it performed correctly and the attachments are returned when the query is executed. This is easily identified when the query returns more results than when it was previously executed.

FindAttachements2_v10.00

Use the FindAttachments2 CPL to execute an active query that retrieves all attachment documents for the specified attachment fields.

This CPL requires two fields containing the Beginning Attachment and Ending Attachment information.

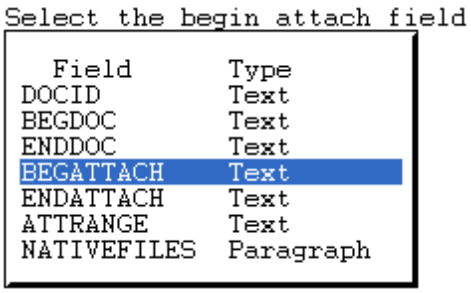
The FindAttachments2 CPL works with the following versions of Concordance:

- 8.*x*
- 9.5*x*
- 10.19 or earlier

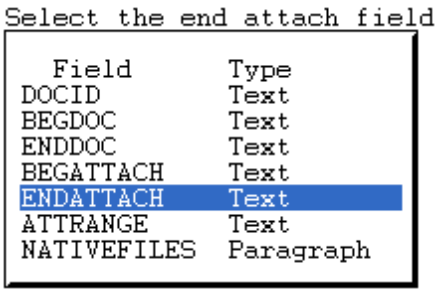
Make sure that you are using the correct CPL for your version of Concordance.

To run the FindAttachments2_v[version #].cpl:

1. On the **File** menu, click **Begin Program**.
2. Locate and open the **FindAttachments2_v[version #].cpl** file.
3. Select the field that contains the beginning attachment number, and then press Enter.



4. Select the field that contains the ending attachment number, and then press Enter.



5. When the CPL is finished, verify that it performed correctly and the attachments are returned when the query is executed. This is easily identified when the query returns more results than when it was previously executed.

IssueToTag_v10.00

Use the IssueToTag CPL to convert issue tags into document level tags. This is useful when users have attempted to add document tags while text is highlighted in the Browse View.

The IssueToTag CPL works with the following versions of Concordance:

- 8.*x*

- 9.5*x*
- 10

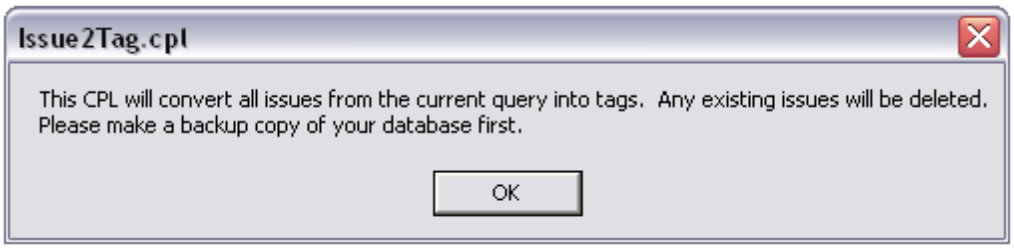
Make sure that you are using the correct CPL for your version of Concordance.

To run the IssueToTag_v[version #].cpl:

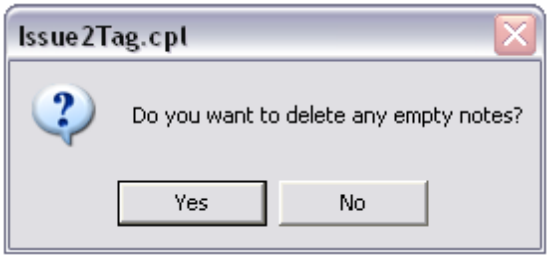
1. Locate the record containing the issue tag you want to convert.

   In version 8, issue tags are shown in the Tags pane with grey check boxes that contain a red check mark. In v. 9.5 and above, issue tags are displayed in the Tags pane with red text.

2. On the **File** menu, click **Begin Program**.
3. Locate and open the **IssueToTag_v[version #].cpl** file.
4. When prompted, click **OK**.



5. When prompted, click **Yes** or **No** to delete any empty notes.



Selecting No does not remove the highlight; however, the issue tag is converted to a document-level tag.

6. When the CPL is finished, verify that the record no longer displays the issue tag. Also, if you selected Yes for step 5, any highlight associated with the issue tag should no longer appear unless you added data to it.

LoadOCRFromOpticonLog_v10.00

Use the LoadOCRFromOpticonLog CPL to load OCR.txt file into the current database utilizing a matching image path in the .opt file.

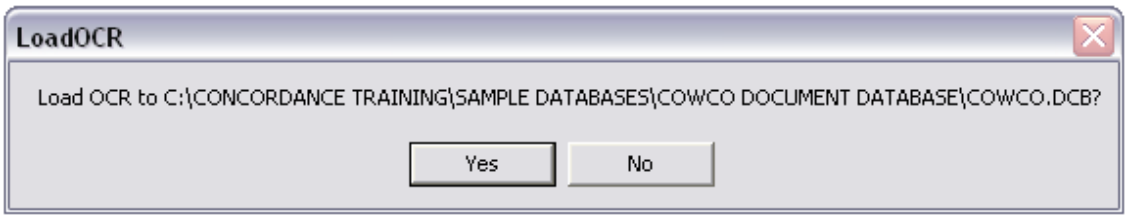
The LoadOCRFromOpticonLog CPL works with the following versions of Concordance:

- 8.*x*
- 9.5*x*
- 10

Make sure that you are using the correct CPL for your version of Concordance.

To run the LoadOCRFromOpticonLog_v[version #].cpl:

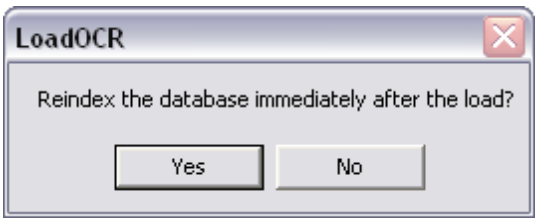
1. On the **File** menu, click **Begin Program**.
2. Locate and open the **LoadOCRFromOpticonLog_v[version #].cpl** file.
3. When prompted, click **Yes** to load OCR into the current database.



4. Using the arrow keys, select the field name where you want to place the OCR text, and then press **Enter**.

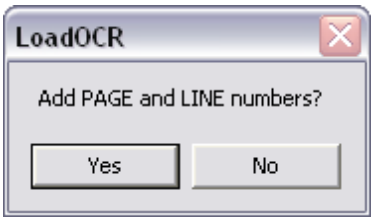
Field	Type
BEGNO	Text
ENDNO	Text
PRODBEG1	Text
PRODEND1	Text
DOCDATE	Date
DOCTYPE	Paragraph
DOCTITLE	Paragraph
AUTHOR	Paragraph
RECIPIENT	Paragraph
PAGES	Numeric
OCR1	Paragraph
OCR2	Paragraph
OCR3	Paragraph
CREATEDATE	Date
EDITTRAIL	Paragraph
PRODNOTES1	Paragraph
PRODNOTES2	Paragraph
PRODDATE1	Date
TAGINFO	Paragraph
CUSTODIAN	Paragraph
REVIEWSTATUS	Paragraph
ATTYNOTES	Paragraph
LOADEDBY	Paragraph
ADMIN2	Paragraph
ADMIN3	Paragraph
ADMIN4	Paragraph
ADMIN5	Paragraph
TAGHISTORY	Paragraph

5. When prompted, click **Yes** or **No** to reindex the database immediately after the OCR text is loaded.

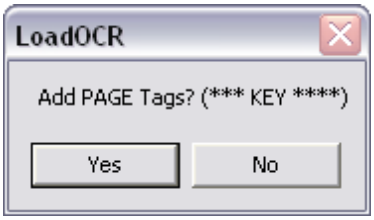


The OCR data is not viewable until the database is reindexed.

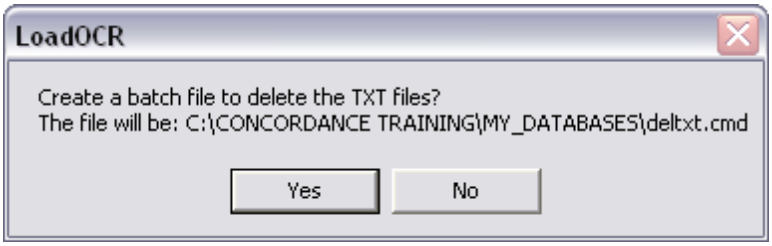
6. When prompted, click **Yes** or **No** to add page and line numbers to the OCR text.



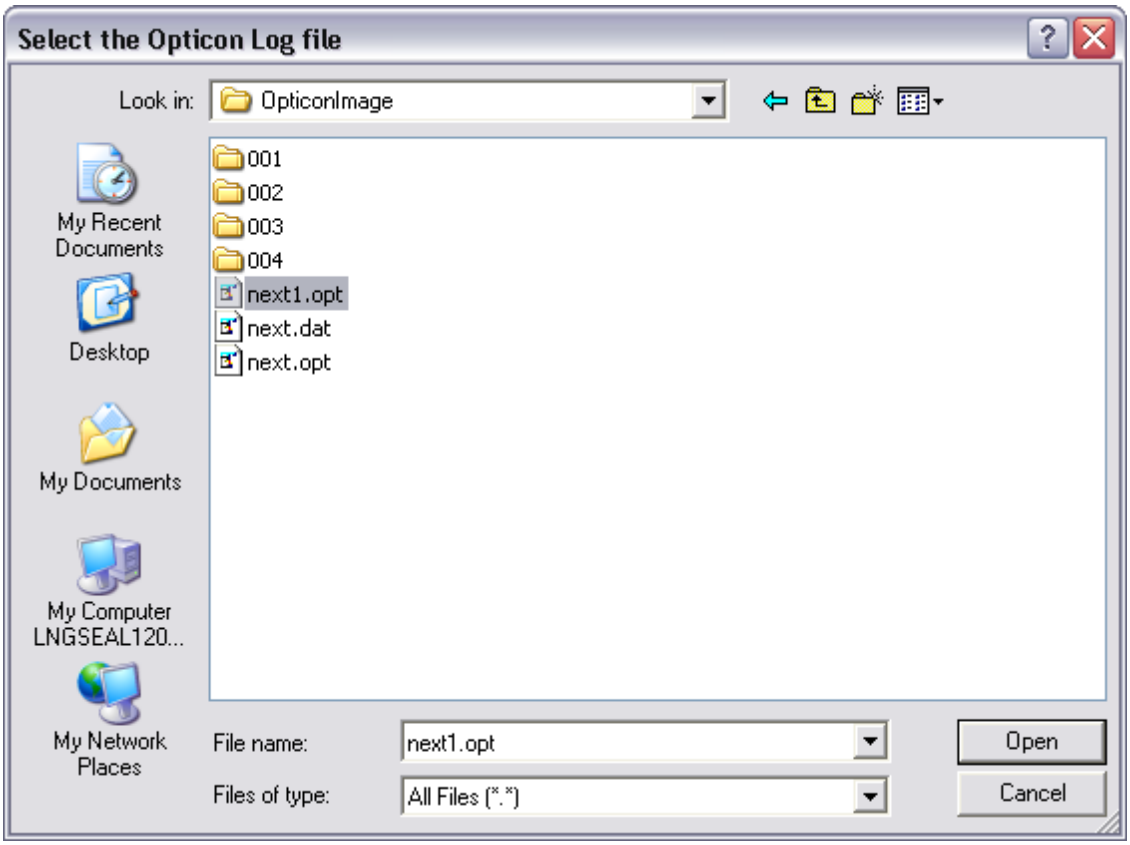
7. When prompted, click **Yes** or **No** to add page tags to the OCR text for quick navigation to the images.



8. When prompted, click **Yes** or **No** to create a batch file to delete the .txt files.



9. Locate and open the .opt or .log file that contains the links to your text location.



10. When the CPL is finished, verify that it performed correctly and the OCR text is displayed in the specified field.

PrintWithAttachments_v10.00

Use the PrintWithAttachments CPL to print Concordance records and any externally attached files through their native application.

Attachments that cannot be printed are logged to the Printwithattachments Error.log file in the CPL directory. The log file name varies depending on the version of Concordance in use.

The PrintWithAttachments CPL works with the following versions of Concordance:

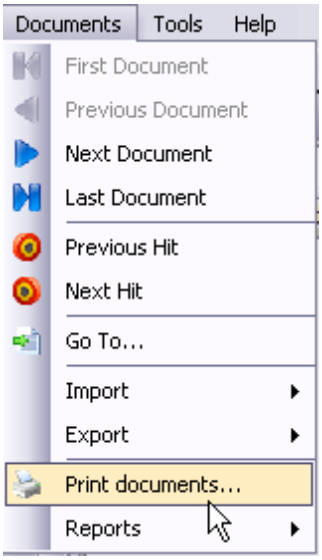
- 8.*x*
- 9.5*x*
- 10

Concordance versions 8.*x* and 9.5*x* print only the attachment; the record is not printed.

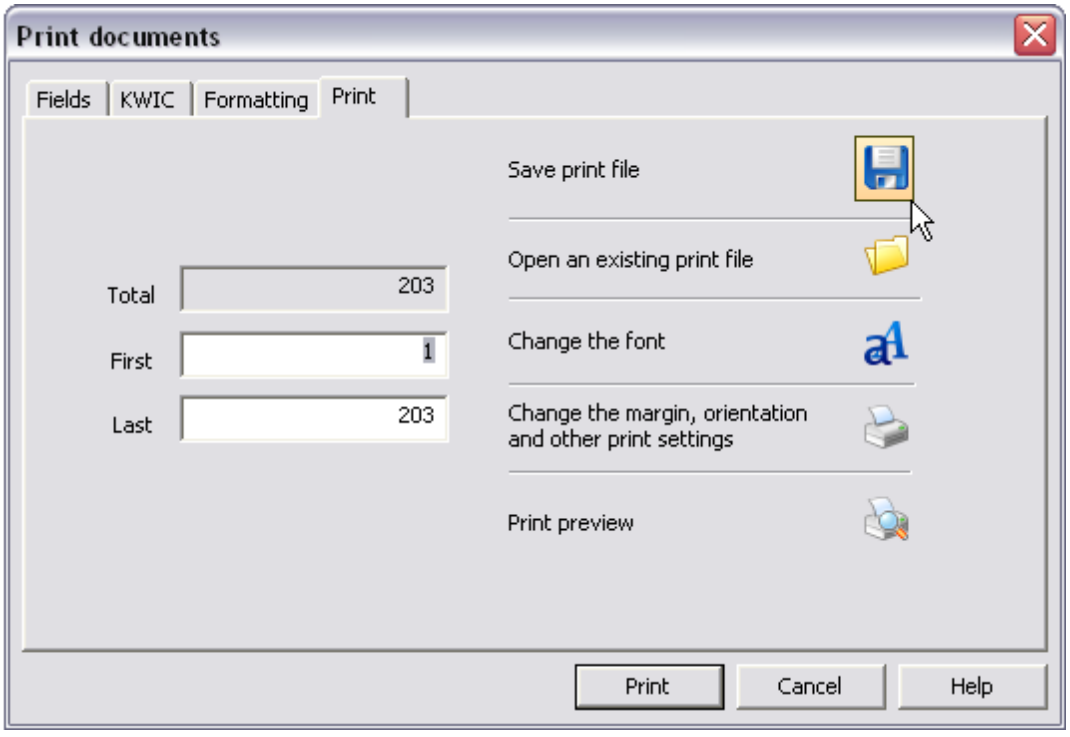
To run the PrintWithAttachment_v[version #].cpl:

1. If this is the first time running the PrintWithAttachments CPL, you must create a print format file first. If not, go to step 6.

2. In Concordance, on the **Documents** menu, click **Print Documents**.



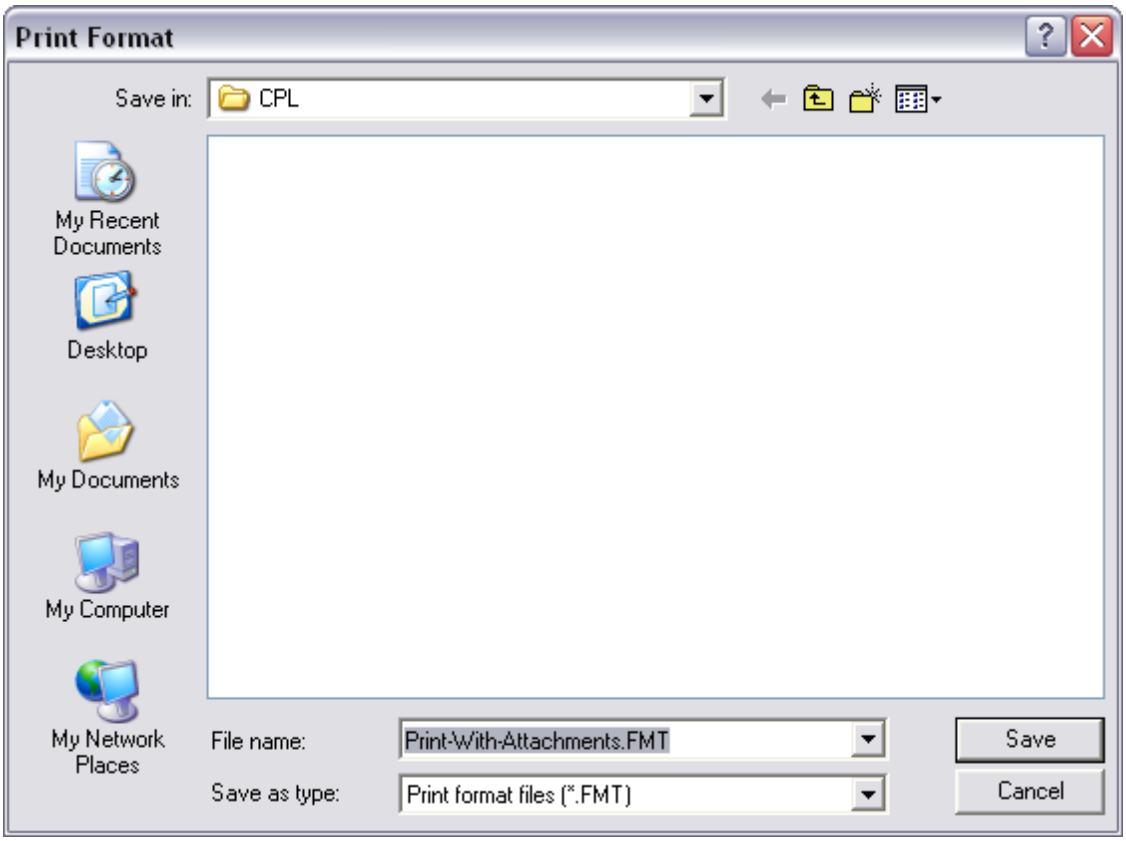
3. Click **Save Print File**.



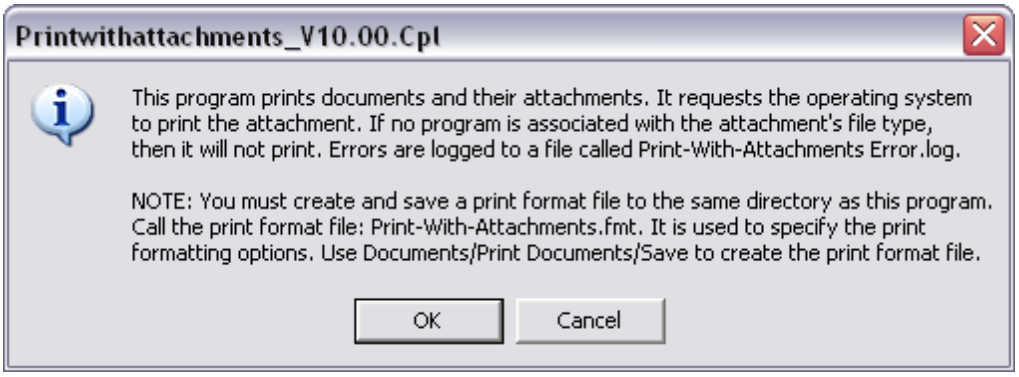
4. Locate the directory that contains the PrintWithAttachments CPL, and then do the following:

   - In the **File Name** text box, type ***Print-With-Attachments***.

- In the **Save as Type** box, select **Print Format Files (*.FMT)**.



5. When finished click **Save**, and then close the **Print Documents** dialog box.
6. On the **File** menu, click **Begin Program**.
7. Locate and open the **PrintWithAttachment_v[version #].cpl** file.
8. When prompted, click **OK** to print all the documents and attachments.



READOCR1 (singlePage)_v10.00

Use the READOCR1(single page) CPL to import multipage text into an existing database utilizing a given file path name.

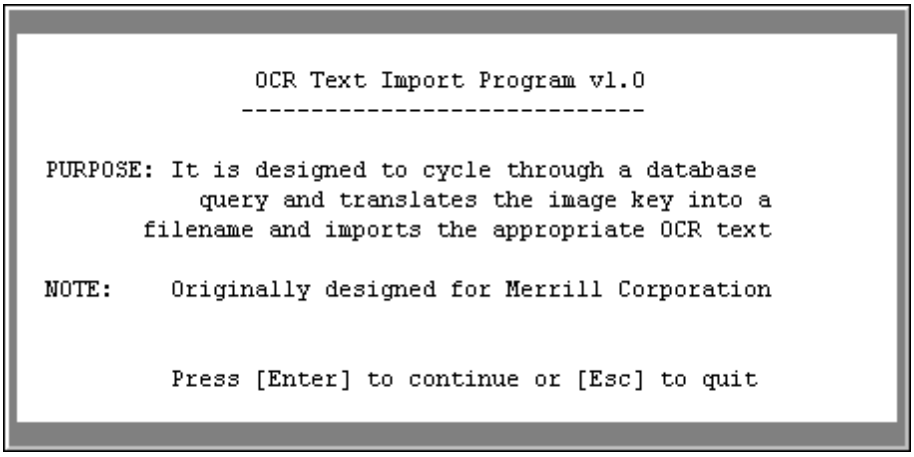
This CPL is used for databases that contain the full path, file name, and extension for the OCR.txt files. Make sure that the destination field contains the full path to the text file.

The READOCR1(single page) CPL works with the following versions of Concordance:

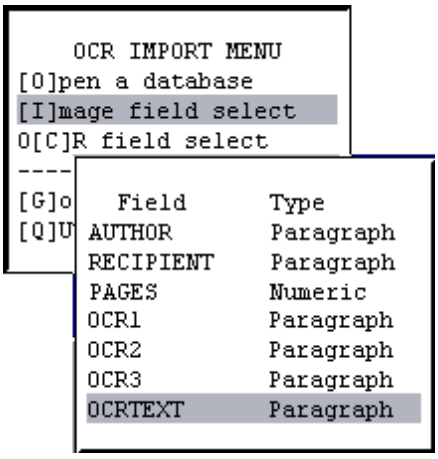
- 8.*x*
- 9.5*x*
- 10

To run the ReadOCR1(Single Page)_v[version #].cpl:

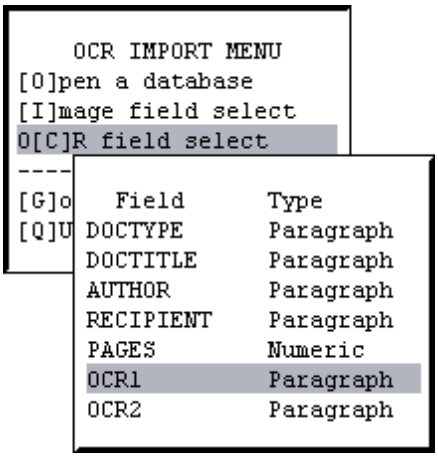
1. On the **File** menu, click **Begin Program**.
2. Locate and open the **ReadOCR1(Single Page)_v[version #].cpl** file.
3. When prompted, press Enter.



4. Click **[I]mage Field Select**, using the arrow keys, locate the field that contains the full path to the OCR text, and then press Enter.

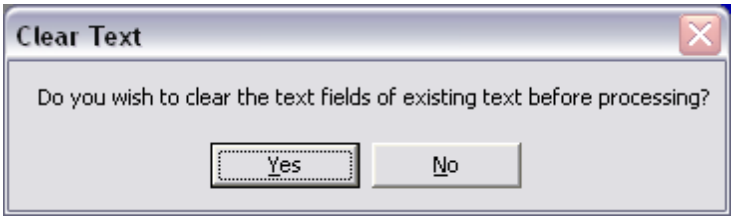


5. Click **O[C]R Field Select**, using the arrow keys, locate the destination field you want to place the OCR text, and then press Enter.

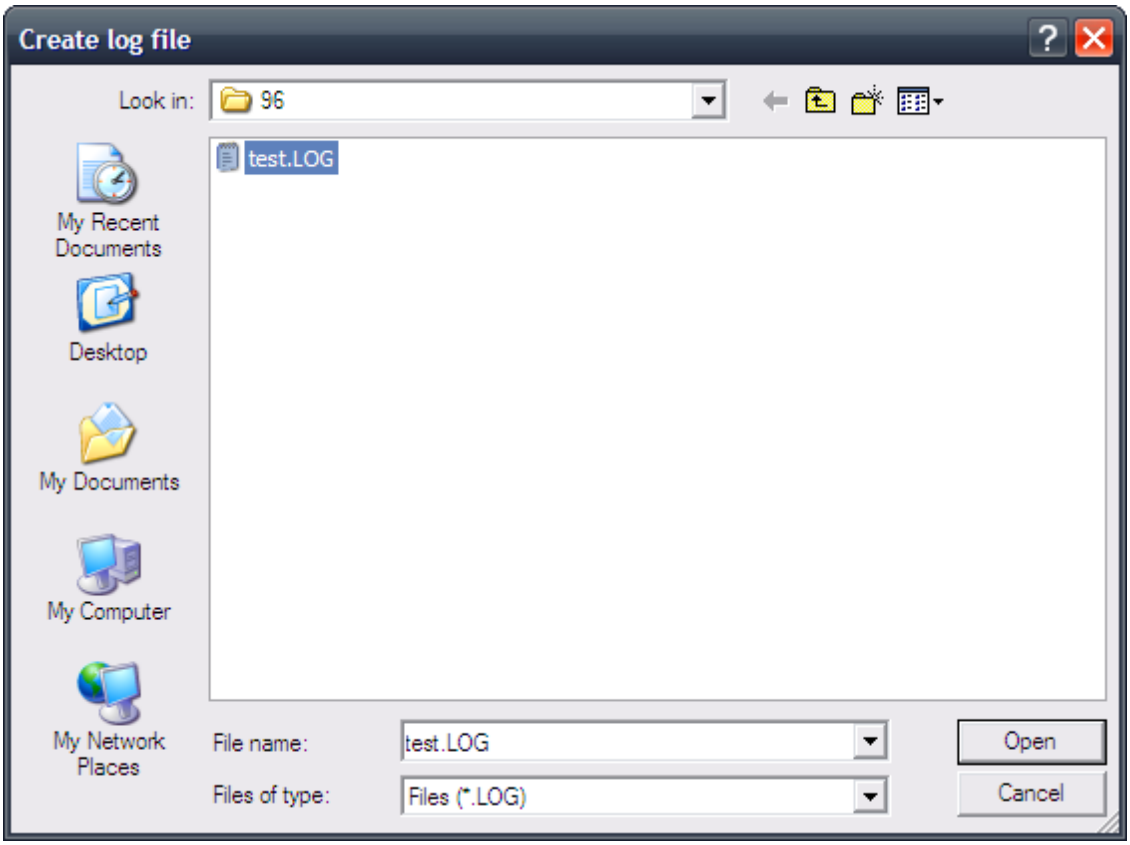


6. Click **[G]o!**.

7. When prompted, click **Yes** to clear the existing data in the destination field.



8. When prompted, designate a location and file name for the log file.



9. When the CPL is finished, click **Quit**, and then verify that the CPL performed correctly. The targeted field should be populated with the data from the text file.

readOCR1_v10.00

Use the READOCR1 CPL to import multipage text into an existing database utilizing a given file path name.

This CPL requires a field that contains the full path to each text file.

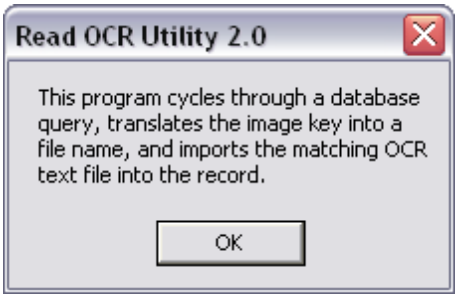
The READOCR1 CPL works with the following versions of Concordance:

- 8.*x*
- 9.5*x*
- 10

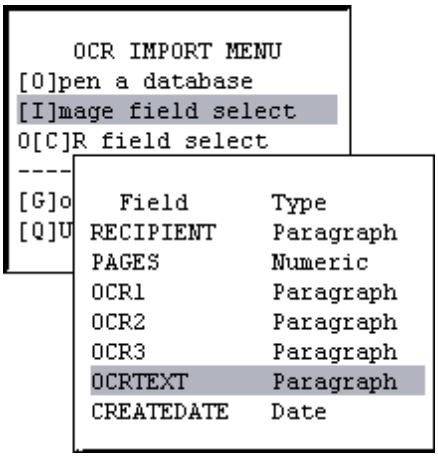
Make sure that you are using the correct CPL for your version of Concordance.

To run the ReadOCR1_v[version #].cpl:

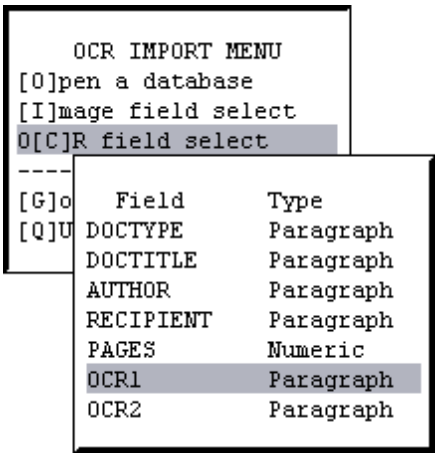
1. On the **File** menu, click **Begin Program**.
2. Locate and open the **ReadOCR1_v[version #].cpl** file.
3. When prompted, press **OK**.



4. Click **[I]mage Field Select**, using the arrow keys, locate the field that contains the full path to the OCR text, and then press Enter.

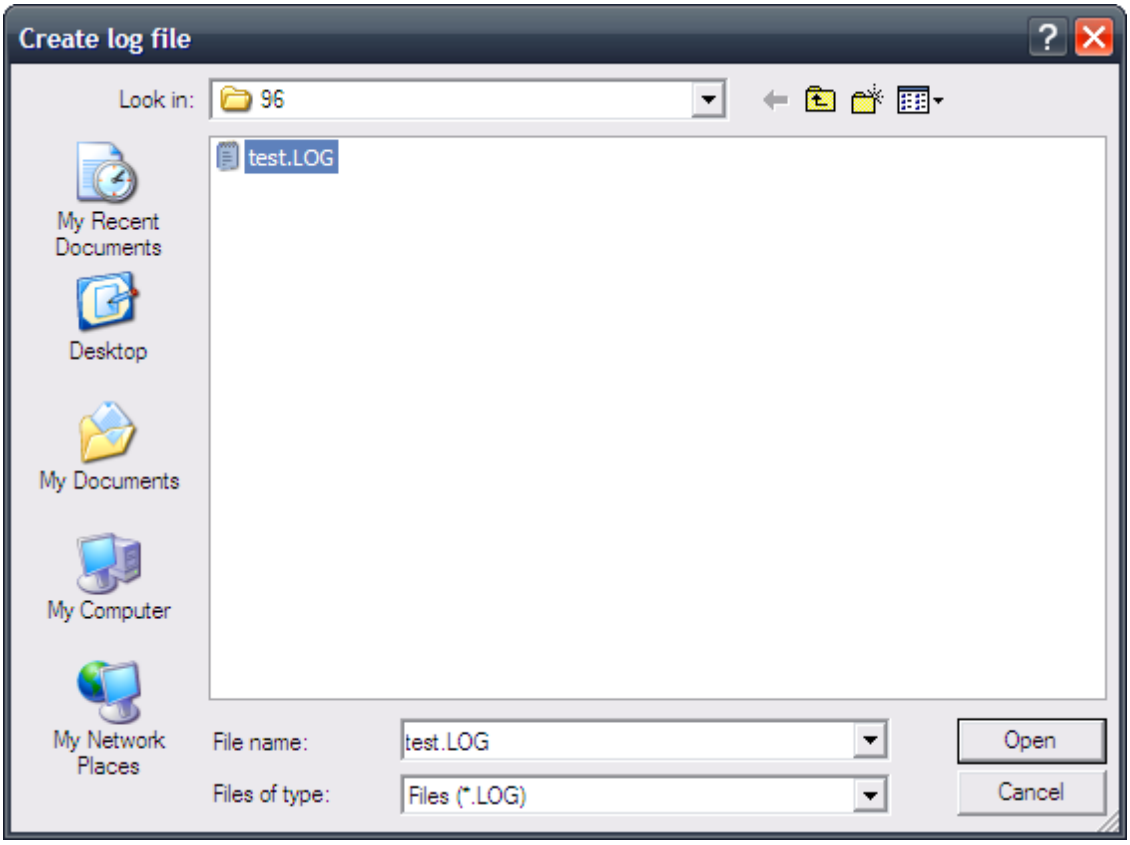


5. Click **O[C]R Field Select**, using the arrow keys, locate the destination field you want to place the OCR text, and then press Enter.



6. Click **[G]o!**.

7. When prompted, designate a location and file name for the log file.



8. When the CPL is finished, click **Quit**, and then verify that the CPL performed correctly. The targeted field should be populated with the data from the text file.

ReadOCR_v10.00

Use the ReadOCR CPL to import multipage text into an existing database records utilizing a unique identifier.

The ReadOCR CPL works with the following versions of Concordance:

- 8.*x*
- 9.5*x*
- 10

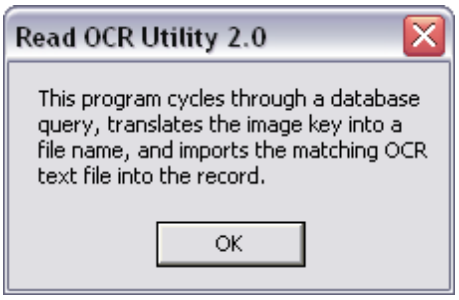
Make sure that you are using the correct CPL for your version of Concordance.

To run the ReadOCR_v[version #].cpl:

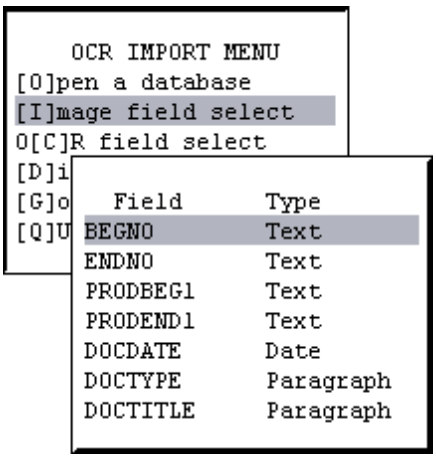
1. On the **File** menu, click **Begin Program**.

2. Locate and open the **ReadOCR_v[version #].cpl** file.

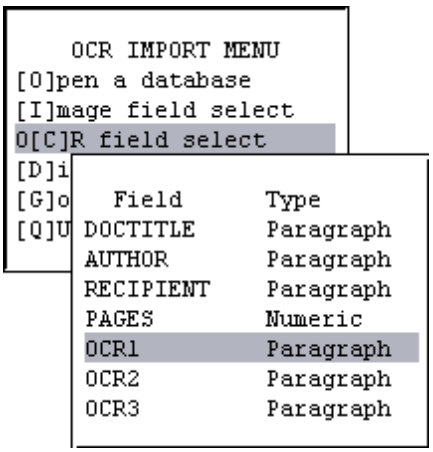
3. When prompted, click **OK** to import the OCR text.



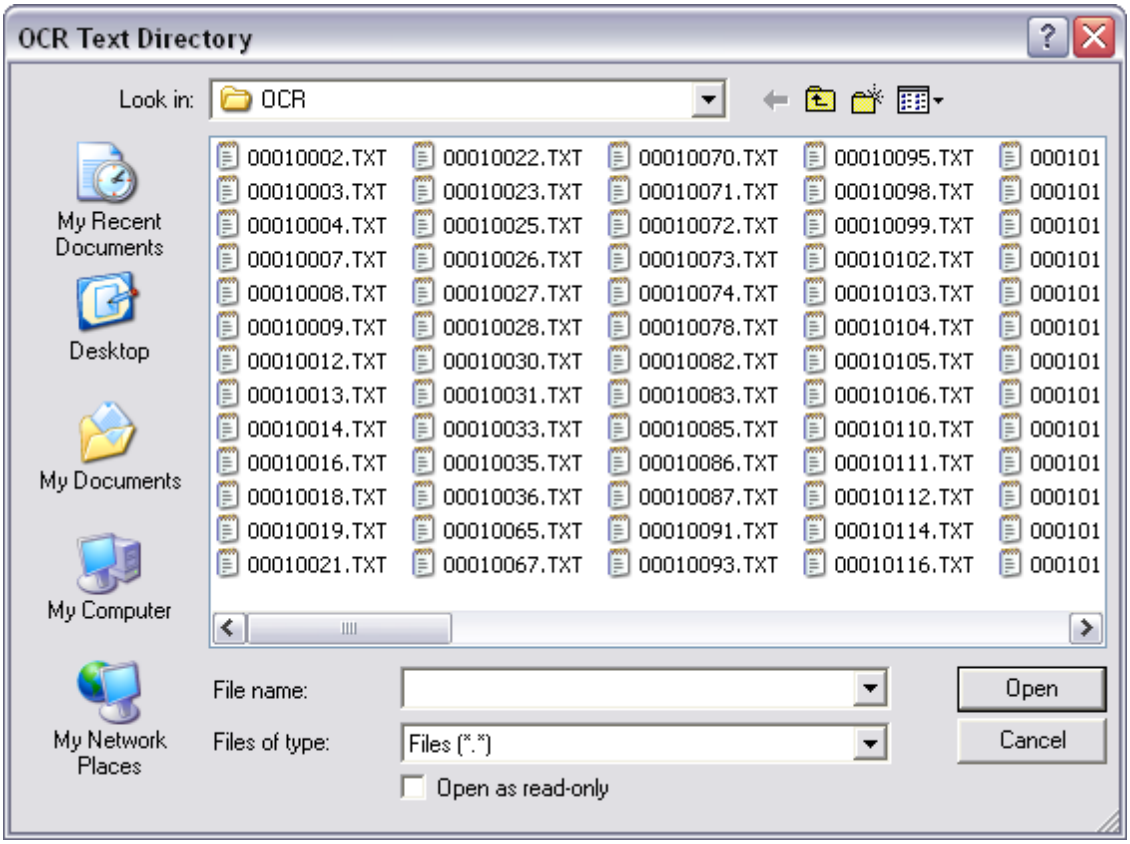
4. Click **[I]mage field select**, using the arrows keys, locate the unique field name identifier that matches both the database and the OCR text, and then press Enter.



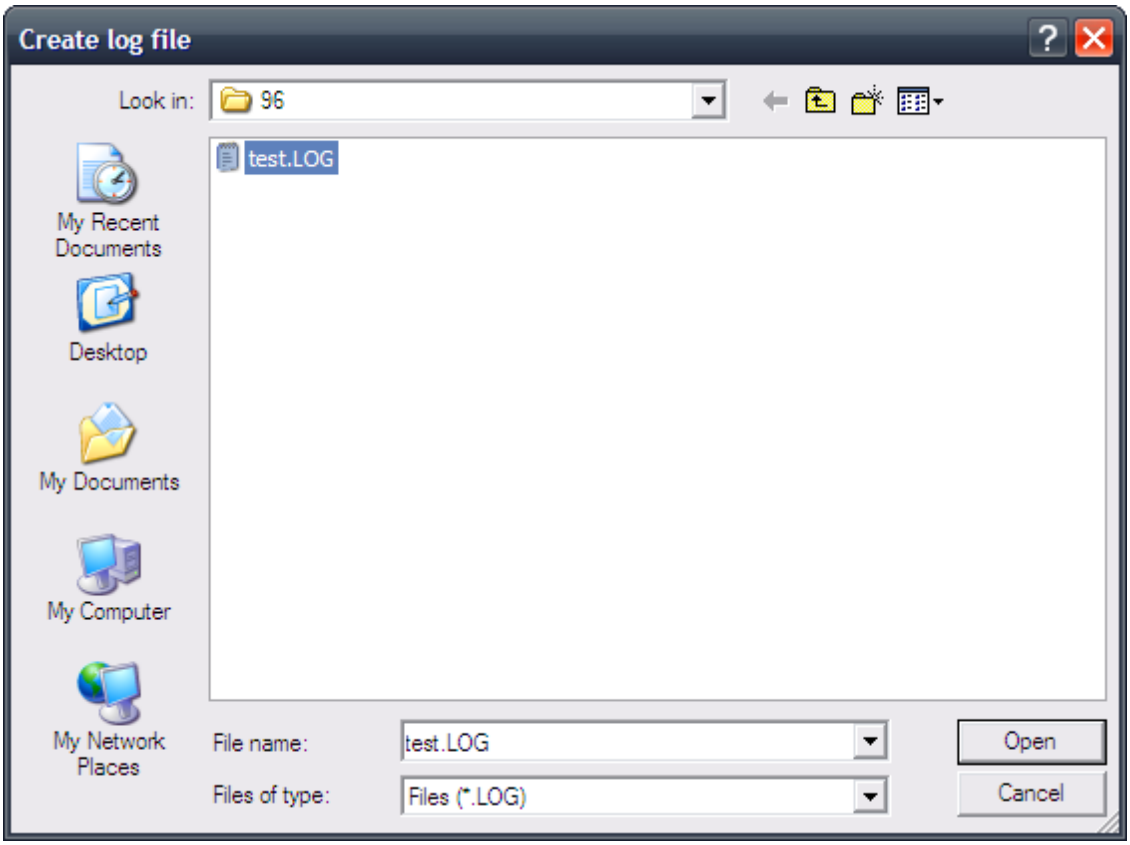
5. Click **O[C]R field select**, using the arrows keys, locate the field name you want to place the OCR text and then press Enter.



6. Click **[D]irectory of OCR Text**, and then locate and open the corresponding OCR text file.



7. When finished, click **Go**.
8. When prompted, designate a location and file name for the .log file.



9. When the CPL is finished, click **Quit**, and then verify that the CPL performed correctly. The targeted OCR text field displays the data from the text file.

If the OCR data is not visible within the text field, try reindexing the database or restarting Concordance.

ReindexingDaemon_v10.00

Use the ReindexingDaemon CPL to reindex all databases within a specified directory.

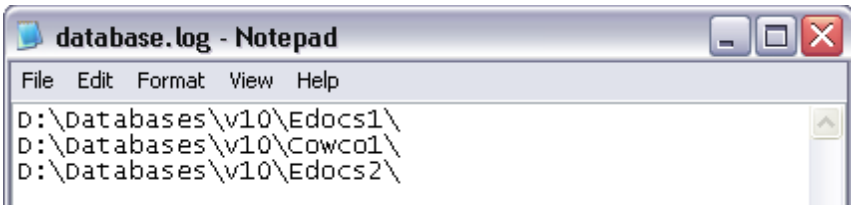
This CPL should be run as a scheduled task within Windows. Also, if you have multiple versions of Concordance, you must prepare this CPL for each version.

The ReindexingDaemon CPL works with the following versions of Concordance:

- 8.*x*
- 9.5*x*
- 10

To run the ReindexingDaemon_v[version #].cpl:

1. Using any text editing application, create a new file, enter the names of all the database directories you want to reindex, and then save the file.



2. In the text editor, open the **ReindexingDaemon_v[version #].cpl**, locate the text **text szUserID = "";** and **text szPassword = "";** type the username and password that matches all the databases that are specified in the text file you created.

```
text szUserID = "username";
text szPassword = "password";
```

If the credentials do not match, you will be prompted for a login when you execute the CPL.

3. Locate the text **text logFile = "";**, and type the full path of the new text file you created in step one, and then save the file.

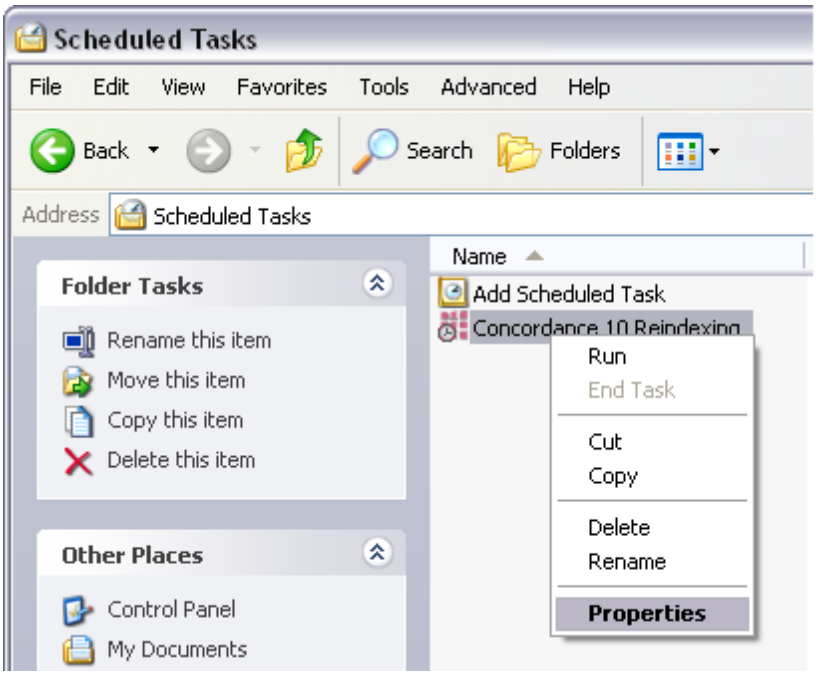
```
text szUserID = "";
text szPassword = "";

text logFile = "C:\database.log";
```

You can run the CPL at this point, and it will work; however, to get the full advantage of the functionality, it is highly recommended that you set this up as a scheduled task.

To set up the ReindexingDaemon CPL as a scheduled Task:

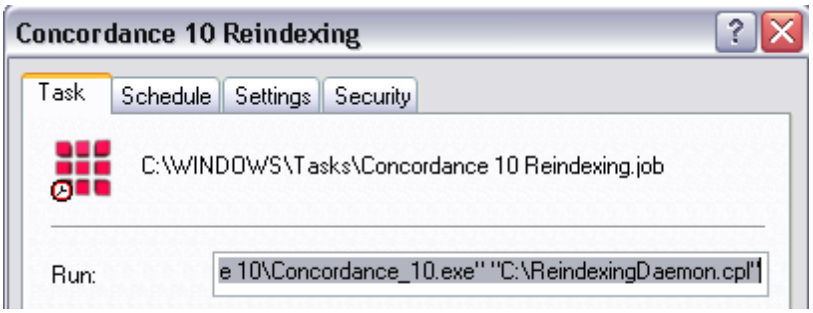
1. From the Control Panel, locate and open the **Scheduled Tasks** manager.
2. Click **Add Scheduled Task** and follow the prompts to setup a new scheduled task for Concordance.
3. When finished locate the newly created task in the Scheduled Tasks manager, and then right-click the task and click **Properties**.



4. In the **Properties** window, modify the **Run** line to be similar to this, and then click **OK**:

   `"C:\Program Files\LexisNexis\Concordance 10\Concordance_10.exe" "C:\`

   The quotes and the space between the quotes are required for this CPL function properly.



5. Type the username and password you used during the Scheduled Task Setup process, and then click **OK**.

6. Right-click the Scheduled Task you just created and click Run. If this CPL ran correctly, you should get a new .txt file in the C:\Logs folder

Renumber_v10.00

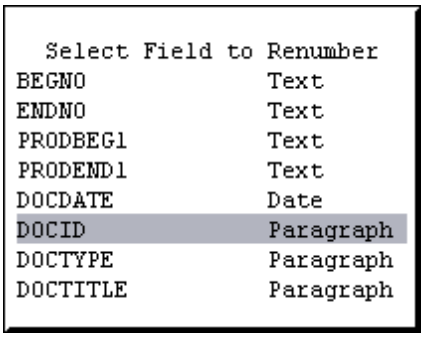
Use the Renumber CPL to assign an alphanumeric serial number to a specified field utilizing user specified prefixes and starting numbers.

The Renumber CPL works with the following versions of Concordance:

- 8.*x*
- 9.5*x*
- 10

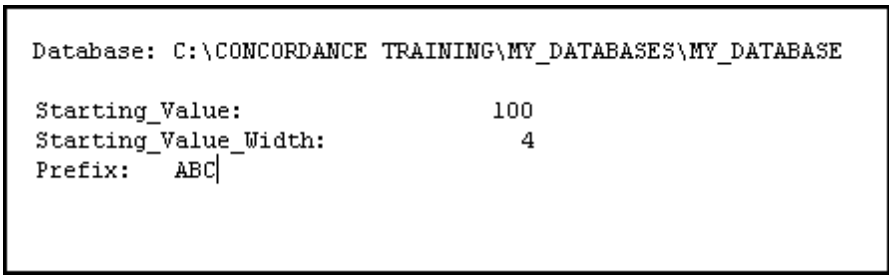
To run the Renumber_v[version #].cpl

1. On the **File** menu, click **Begin Program**.
2. Locate and open the **Renumber_v[version #].cpl** file.
3. Using the arrow keys, select the field you want to renumber, and then press Enter.



4. Do the following:
   - For **Starting_Value** enter a number to begin renumbering..
   - For **Starting_Value_Width** that designates how many digits you wish the starting number to be.
   - Enter an alphanumeric prefix for **Prefix** that you want to start each number with.

Make sure that you specify a large starting value width or the numbering will be replaced with the # symbol after a certain length. For example, if you have 400 documents, a starting value of 1 and width of 2, when the number reaches 100 your renumbering will starting displaying # instead of incrementing the number.



5. When finished, press Enter.
6. When the CPL is finished, verify that it performed correctly. The designated field should now display the number you specified, (i.e. ABC100).

ShowSystemFields_v10.00

Use the ShowSystemFields CPL to display or hide all system fields in a database for the duration of the open session.

The ShowSystemFields CPL works with the following versions of Concordance:

- 8.*x*
- 9.5*x*
- 10

To run the ShowSystemFields_v[version #].cpl

1. On the **File** menu, click **Begin Program**.
2. Locate and open the **ShowSystemFields_v[version #].cpl** file.
3. On the **File** menu, click **Modify** to verify that the CPL ran properly. Any fields that are System fields will have the System check box selected.

Run this CPL a second time to hide all system fields.

Spell_v10.00

Use the Spell CPL to check words in the database against a known spelling list.

This CPL requires that the ***Mark.cpl*** file be in the same location as the ***Spell.cpl*** file, as well

as the ***words.list*** file. By default, the Spell CPL only checks for English words.

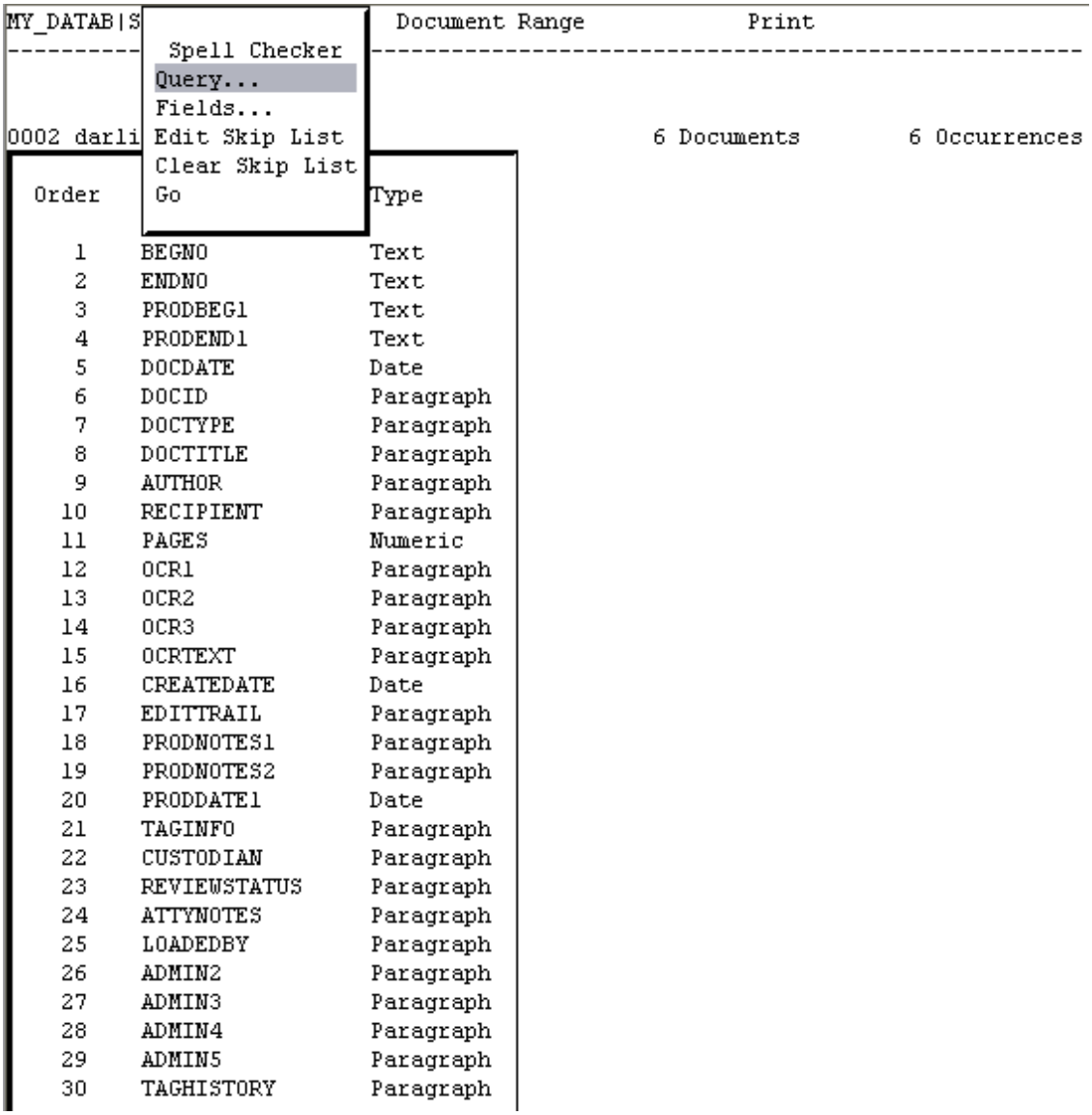
The Spell CPL works with the following versions of Concordance:

- 8.*x*
- 9.5*x*
- 10

⚠ Use only version 10 of this CPL. Due to comment lines, previous versions give Syntax Errors.

To run the Spell_v[version #]cpl:

1. On the **File** menu, click **Begin Program**.
2. Locate and open the **Spell_v[version #].cpl** file.
3. The Spell CPL offers the following functions:
   - **Query** - runs the spell check on an existing query.
   - **Fields** - runs the spell check on the fields you select.
   - **Edit Skip List** - adds words to and edits the Skip Words list. The Skip Words List is created when you select the ***Skip All Occurrences*** during the spell checking process.
   - **Clear Skip List** - clears the entire Skip Words list. The function deletes the ***[database].spl*** file where ***[database]*** is the name of the database on which you are running the Spell CPL.



4. To run the Spell Checker, click **Go**.

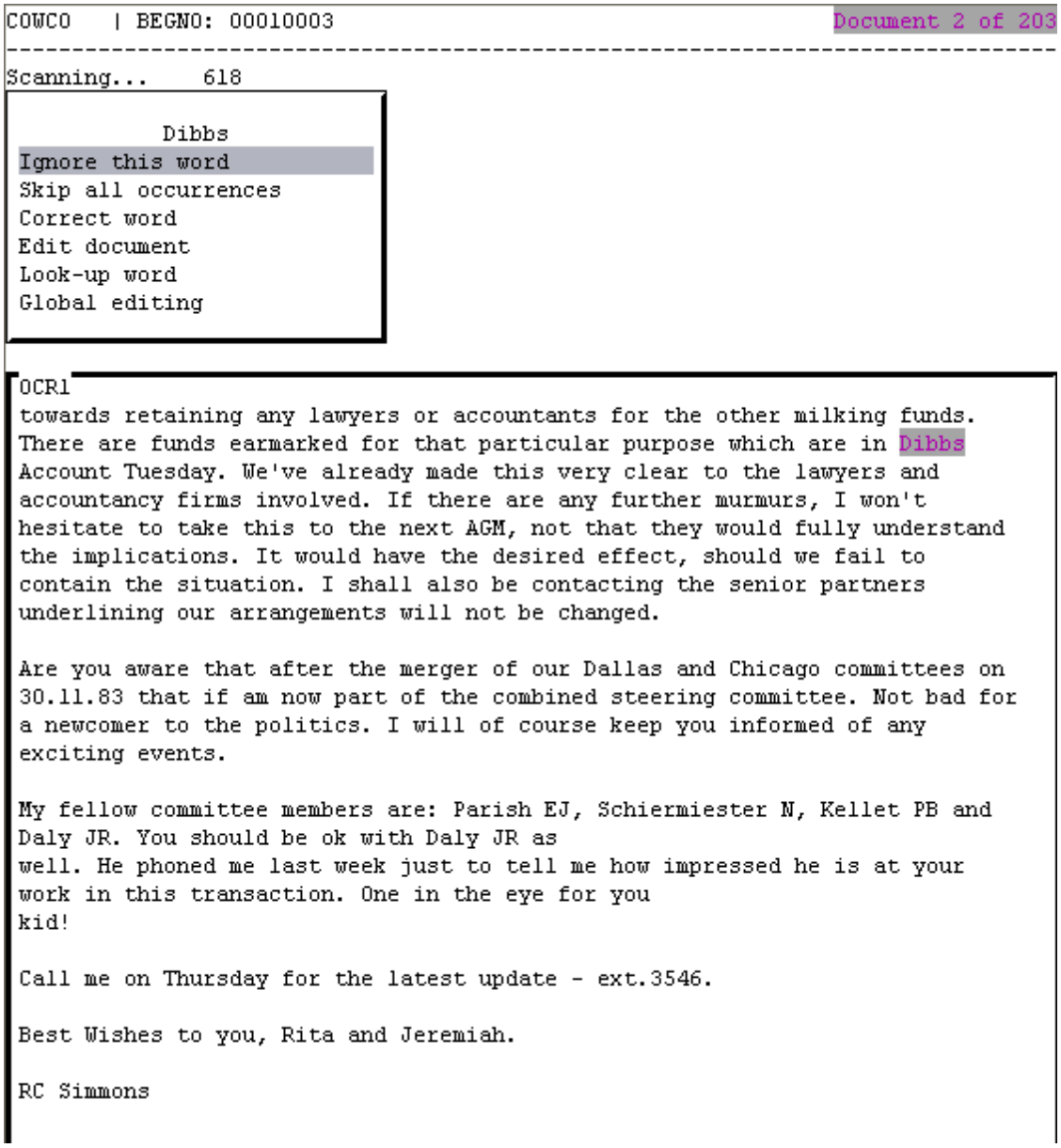
To correct words:

Select any of the following during the spell check process:

- **Ignore this word** - ignores this instance of the highlighted word.
- **Skip all occurrences** - skips all instances of the highlighted word and adds the word to the [database].spl file.
- **Correct word** - allows you to correct the spelling of the word.
- **Edit document** - allows you to edit the data, similar to the Edit Mode in Concordance.
- **Look-up word** - allows you to look up the highlighted word in the words.list file and

correct it.

- **Global editing** - launches the Global Replace function from Concordance.

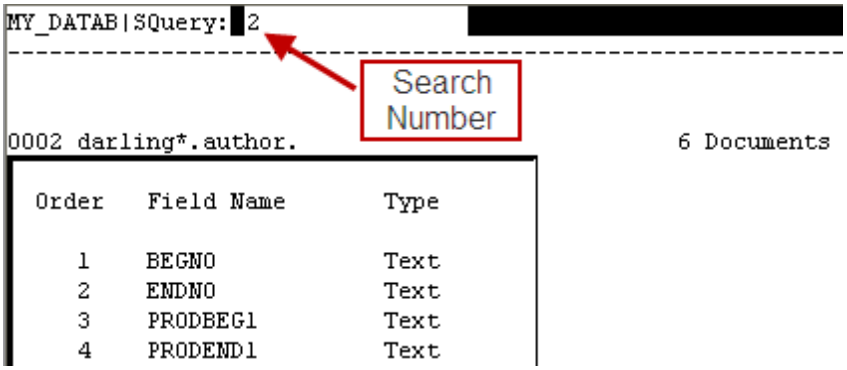


To spell check a query:

1. From the **Spell Checker** dialog box, click **Query**.
2. Enter the Search Number shown in the **Review window** in Concordance, and then press Enter.

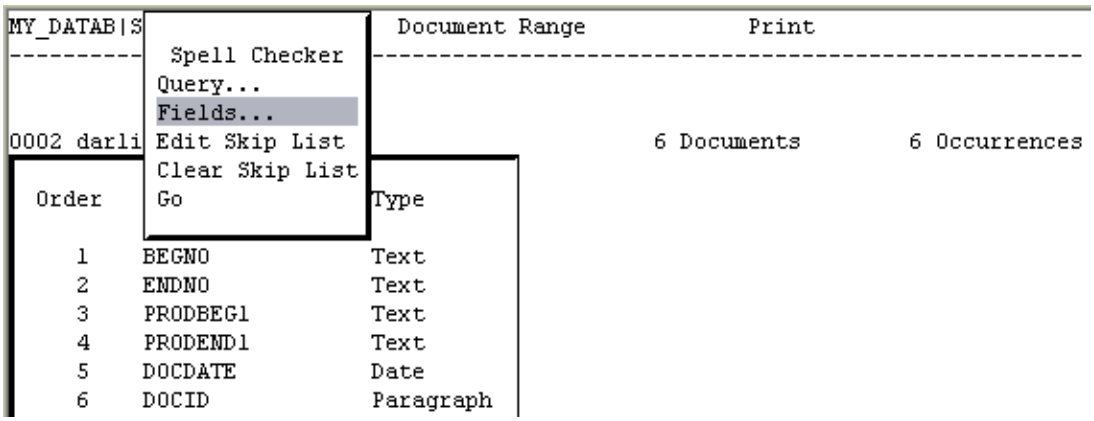
Search Number	Number of Hits	Number of Docs	Search Term(s)
00000	203	203	<Entire Database>
00001	11	11	goniff*.author.
00002	6	6	darling*.author.

Review window



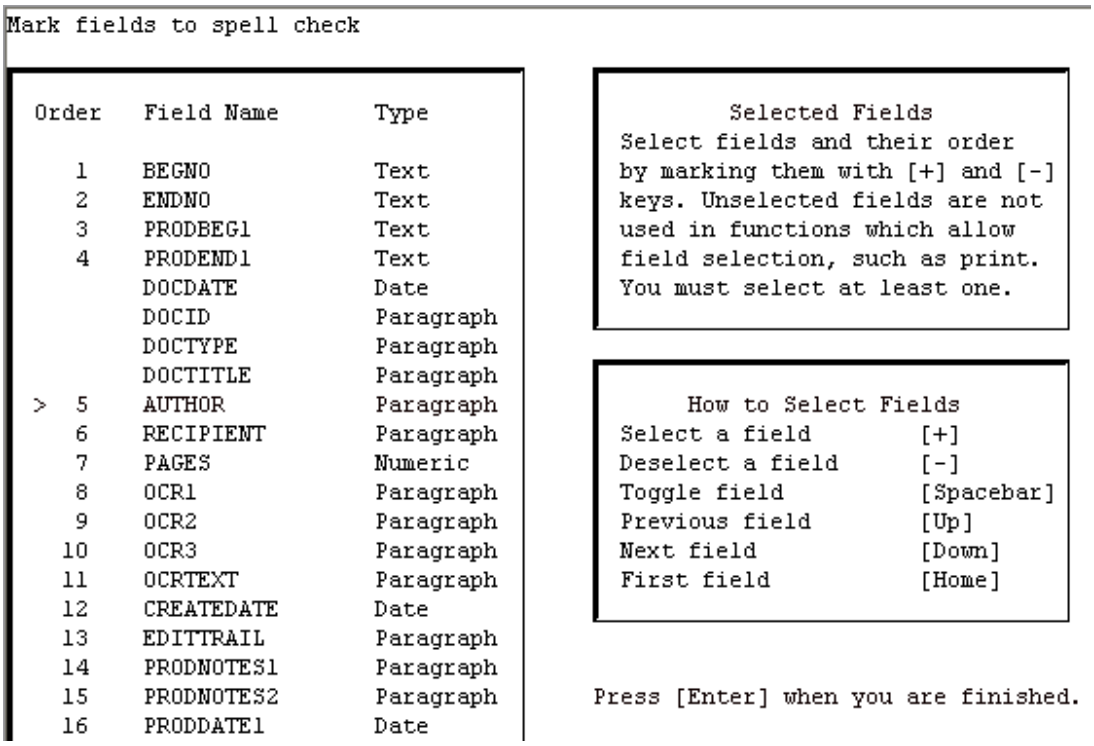
To check the spelling of specific fields:

1. From the **Spell Checker** dialog box, click **Fields**.



   The numbers next to each field indicate the order that the fields are checked with the Spell Checker utility. A blank next to the field indicates that the field is excluded from the spell check.

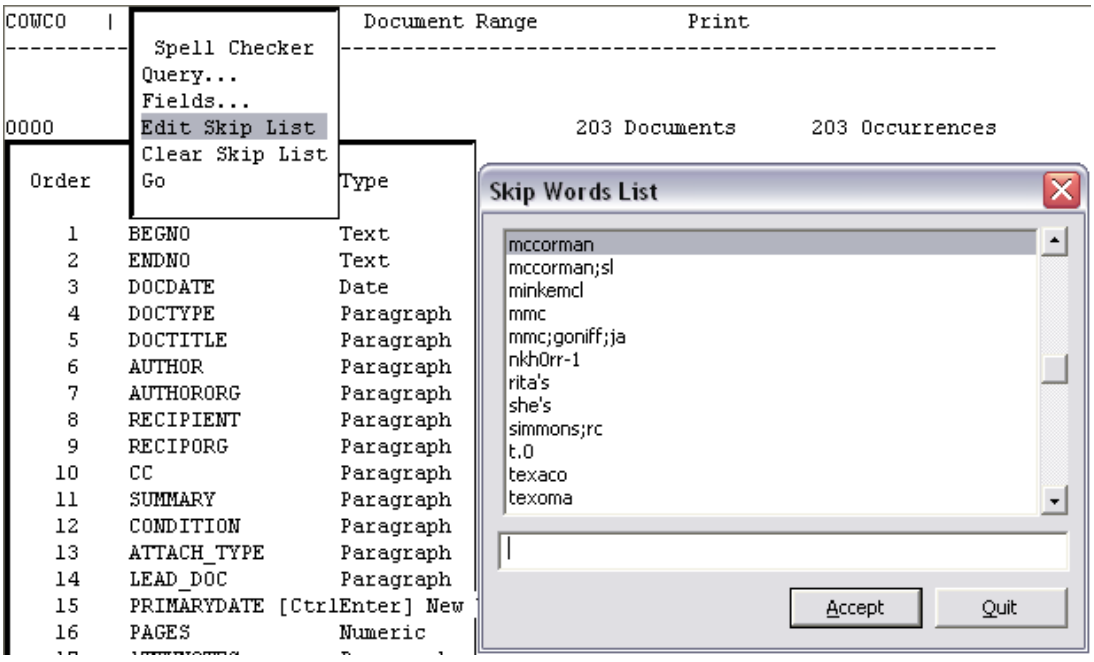
2. Use the following keys to select the fields:
   - [+] Select a field
   - [-] Deselect a field
   - [Spacebar] Toggle the field
   - [Up Arrow] Previous field
   - [Down Arrow] Next field
   - [Home] First field



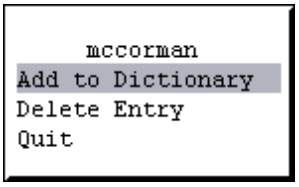
3. When finished, press Enter.

To add words to the dictionary:

1. Click **Edit Skip List**.



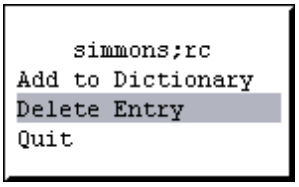
2. Click the word you wish to add to the dictionary, and then click **Accept**.



3. Click **Add to Dictionary**.
4. When finished, click **Ok**.

To delete a single word from the Skip List:

1. Click **Edit Skip List**.
2. Click the word you wish to remove from the list, and then click **Accept**.



3. Click **Delete Entry**.
4. When finished, click **Ok**.

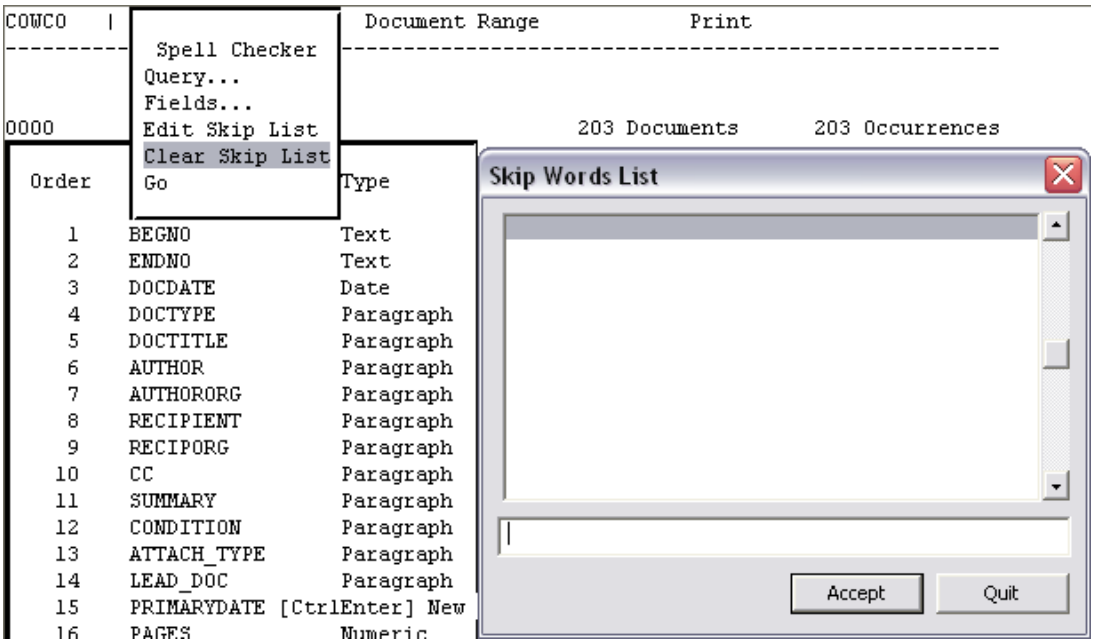
To clear all the words from the Skip List:

Click **Clear Skip List**.

To verify the Skip List is cleared

Do any of the following:

- Click **Clear Skip List**. A dialog box opens indicating that the list is already clear.
- Click **Edit Skip List**. The Skip Word dialog box should not display any entries.



Synonym_v10.00

The Synonym CPL loads a text file containing synonyms into the Concordance .SYN file for searching.

The .SYN file is located in the same directory folder as the primary database.

The Synonym CPL works with the following versions of Concordance:

- 8.*x*
- 9.5*x*
- 10

To create a Synonym load file

When creating a synonyms list, keep the following in mind:

- Group words together to form a synonym group.
- Use a blank line to separate one synonym group from another.
- Precede words with a minus sign to create subcategories of a specified word.
- Use the plus sign to store a word as a synonym and then assign subgroups.

1. Open any text editor, such as Notepad or TextPad to create a new text file.

2. Type the synonyms you wish to add to the Concordance .SYN file, and then save the file to a convenient location.

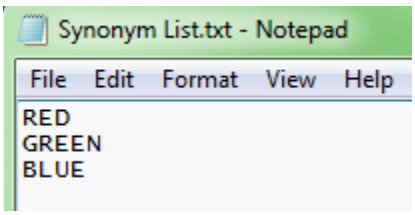
To run the Synonym_v[version #].cpl

1. Open a database in Concordance.
2. In Concordance, on the **File** menu, click **Begin Program**.
3. Locate and open the **Synonym_v[version #].cpl.**
4. In the **Input File** dialog box, locate the Synonym list text file you want to load, and then click **Open**.
5. Verify that the CPL ran correctly by searching for one of the items in your Synonym text file. The CPL should return not only the word you are searching for, but also the words you have designated as synonyms.

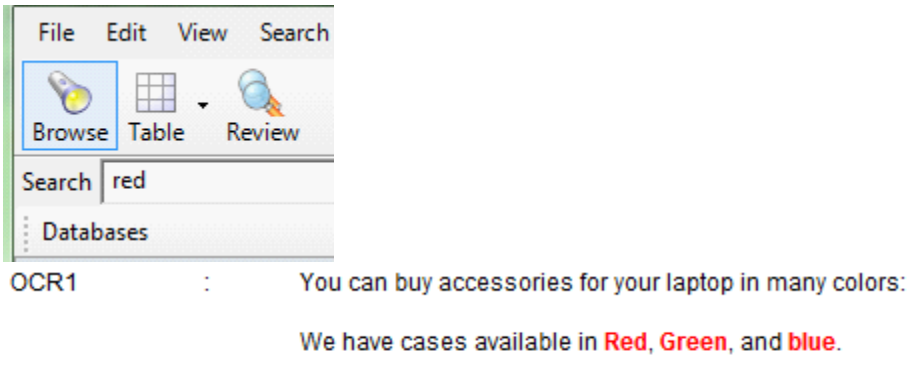
Synonym list examples

To create a synonym group

Words that are grouped together without any symbols form a synonym group.



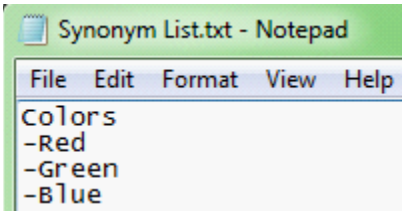
For example, searching for any of the words on this list, the search results returns all of the words on the list. Thus, searching for the word *Red* returns results for the words *Red*, *Green* and *Blue*.



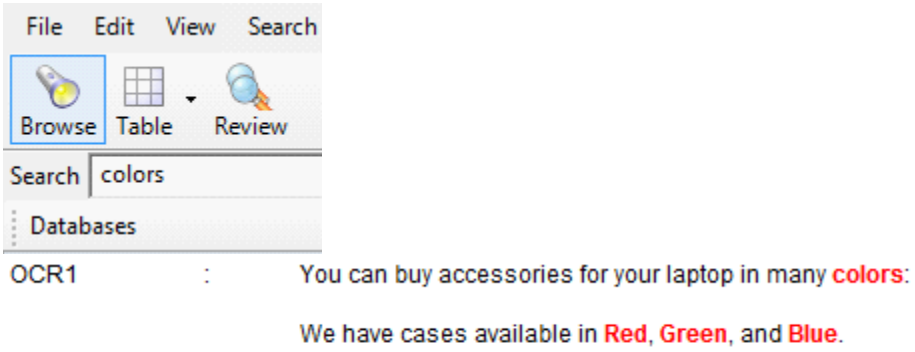
Using the same synonyms list, searching for the word *Green*, returns the words *Green*, *Red* and *Blue.*

To create synonym subcategories

Words preceded with a minus sign create subcategories of the main word.



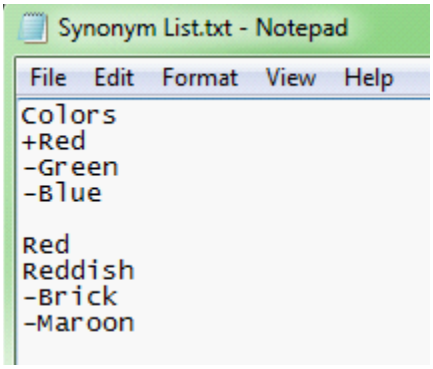
For example, searching for the word *Colors*, the search results would include the words, *Colors*, *Red*, *Green*, and *Blue*.



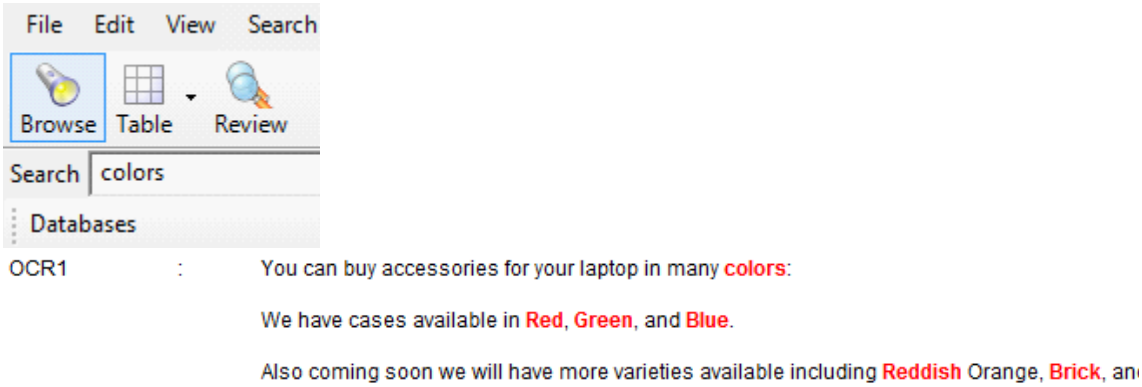
However, searching for the word *Blue*, returns only the word *Blue* and NOT the words *Colors*, *Green* or *Red*.

To create a synonym subgroup

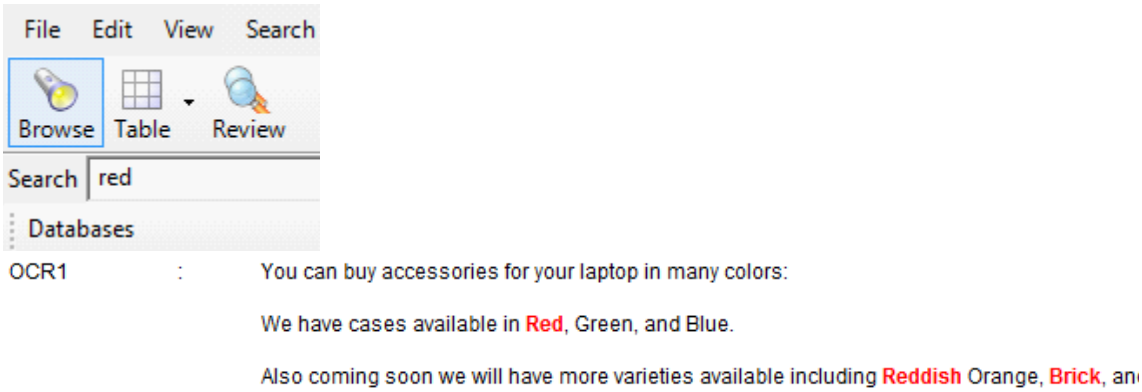
Words preceded by a plus sign store the word as a synonym and then assign a subgroup of words.



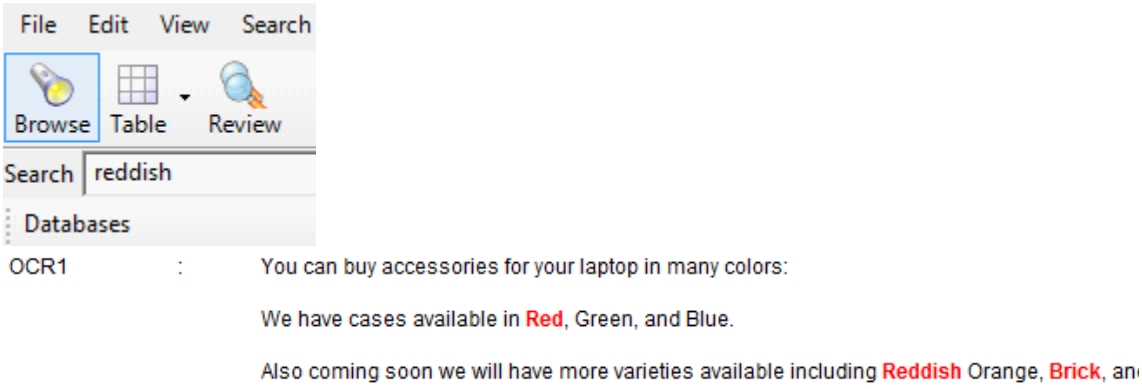
For example, searching for the word *Colors*, returns the words *Colors*, *Red, Green, Blue, Reddish, Brick,* and *Maroon.*



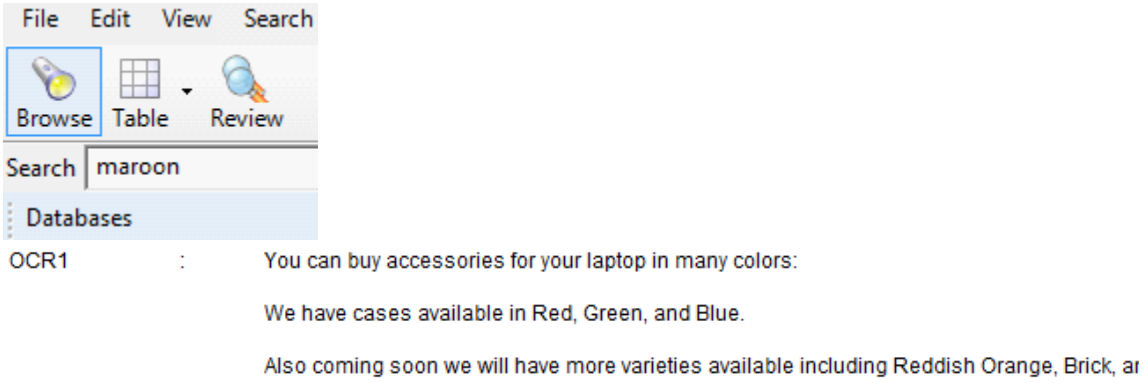
Searching for the word *Red*, returns the words *Red*, *Reddish*, *Brick*, and *Maroon*.



Similarly, searching for the word *Reddish*, returns the words *Red*, *Reddish*, *Brick*, and *Maroon*.



However, searching for the word *Maroon*, returns only the word *Maroon*.



TagHistoryAndStoreIt_v10.00

The TagHistoryAndStoreIt CPL writes the tag history of each record to a specified field in the database.

The TagHistoryAndStoreIT CPL works with the following versions of Concordance:

- 8.*x*
- 9.5*x*
- 10

To run the TagHistoryAndStoreIt_v[version #].cpl:

1. Using any text editing application, locate and open the **TagHistoryandStoreIt_v[version #].cpl** file.
2. Using the text editor's Find and Replace features, replace the name of the current field specified in the CPL (default value "TAGINFO") with the name of the field you want to write the tag history to.

```
23    if (isfield(0, "TAGINFO") == 0)
24        return(messageBox("This program
25                           "This database
26                           "it and try ru
```

Before

```
23    if (isfield(0, "TAGHISTORY") == 0)
24        return(messageBox("This program s
25                           "This database
26                           "it and try rur
```

After

3. Verify that the field name is changed in all location in the text file, and then save the CPL.

4. Open your Concordance database.
5. On the **File** menu, click **Begin Program**, and then locate and open the **TagHistoryAndStoreIt_v[version #].cpl** file.
6. When the CPL is finished, verify that it ran properly.
7. Locate the field specified. If tags exist for the document, the following information should be displayed.

```
TAGHISTORY :    Record number: 2
                Accession number: 2
                Serial number: [GFG711-2] 1261005726 HUMP

                Tags added:
                +[GFG711-2] 1261005726 HUMP   Hot      humphrtj on 12/17/2009 01:42:31 GMT

                Tags deleted:
                -[GFG711-2] 1261005726 HUMP   TEST1    humphrtj on 12/17/2009 01:42:36 GMT
```

TAGSAVER_v10.00

The TagSaver CPL saves or restores tags based on a unique identifier field.

If the database is new and has never been indexed, run an index before running the Tagsaver CPL.

The TagSaver CPL works with the following versions of Concordance:

- 8.*x*
- 9.5*x*
- 10

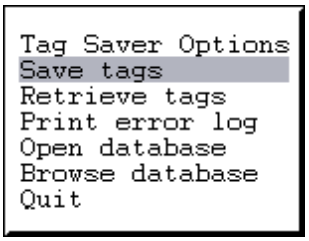
When restoring tags from a .gat file, the maximum number of characters allowed in a tag name is 199.

Make sure that you are using the correct CPL for your version of Concordance.

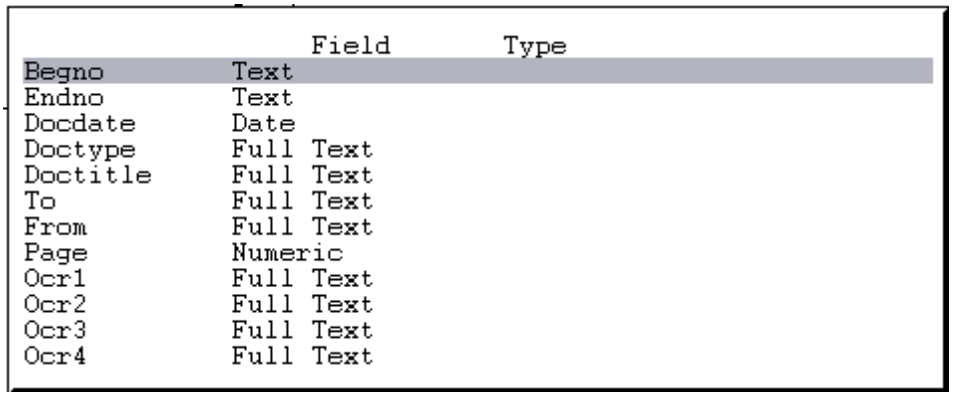
To run the TagSaver_v[version #].cpl:

To save tags:

1. On the **File** menu, click **Begin Program**.
2. Locate and open the **TagSaver_v[version #].cpl** file.
3. In the **Tag Saver Options** dialog box, using the arrow keys, select **Save tags**, and then press Enter.



4. Using the arrow keys, select the field that contains a unique identifier for the database, and then press Enter.



5. When the CPL is finished, click **Quit**.

6. Verify that the CPL executed properly. Locate the new file **[database].gat** in the database directory.

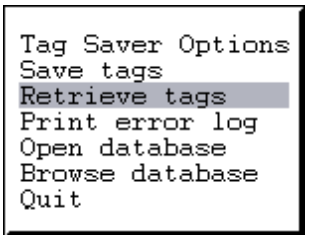
To retrieve tags:

1. Make sure that you have the .gat file in your database directory with the same name as the database you are retrieving tags. For example, Cowco.gat

2. On the **File** menu, click **Begin Program**.

3. Locate and open the **TagSaver_v[version #].cpl** file.

4. In the **Tag Saver Options** dialog box, using the arrow keys, select **Retrieve tags**, and then press Enter.



5. When the CPL is finished, click **Quit**.

6. Verify that the CPL executed properly. Tags are displayed for all the records that were tagged in the .gat file.

TagToField_v10.00

The TagToField CPL writes the tag names to a user specified field in the database separating the tag names with the specified delimiter.

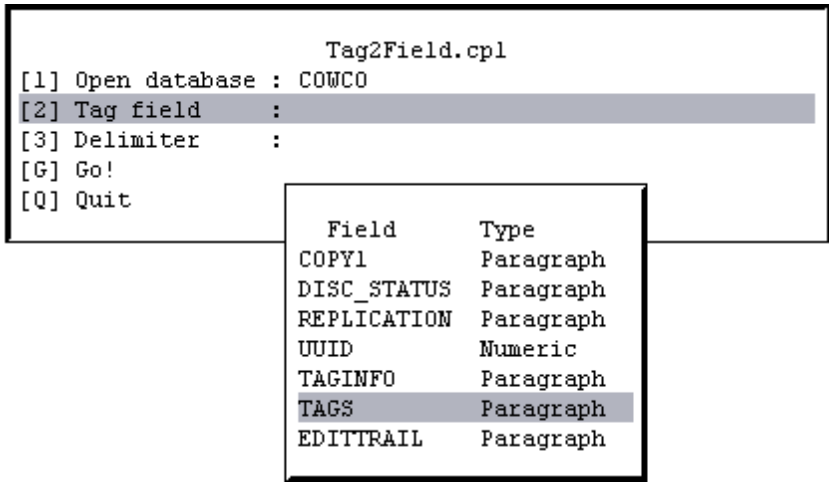
The TagToField CPL works with the following versions of Concordance:

- 8.*x*
- 9.5*x*
- 10

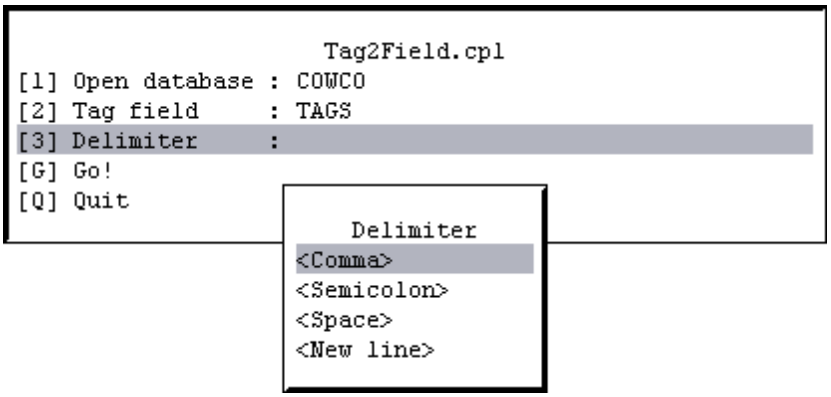
This CPL is also known as TagToField CPL in older versions of Concordance.

To run the TagToField_v[version#].cpl

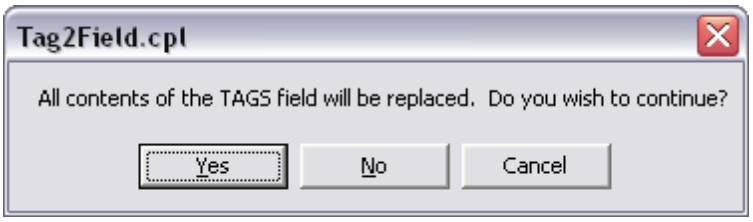
1. On the **File** menu, click **Begin Program**.
2. Locate and open the **TagToField_v[version #].cpl** file.
3. Click **[2] Tag field**, using the arrow keys select the field to write the tag names, and then press Enter.



4. Click **[3] Delimiter**, using the arrow keys select the delimiter you want to use to separate the tags in the field, and then press Enter.



5. Click **[G] Go!**.
6. When prompted, do one of the following:
   - Click **Yes** - confirms that you want to place the contents of the selected field.
   - Click **No** - returns to the main menu allowing you to select a different field.



7. When the CPL is finished, click Quit, and then verify that the CPL executed properly. The selected field should display tags names separated by the delimiter you specified.

   For example, the field TAGS contains the following:

   TAGS : Confidential, Responsive

TextFileToQuery_v10.00

The TextFileToQuery CPL executes queries from an input file (plain ASCII). Each ***hit*** document is tagged, and then all tagged documents are grouped together in a final query.

The TextFileToQuery works with the following versions of Concordance:

- 8.*x*
- 9.5*x*
- 10

To run the TextFileToQuery_v[version #].cpl

1. Using any text editing application, create a new document with each search term that you want to search. Make sure that each term is separated by a hard return.

```
1  complex
2  environment
3  text co technical
```

   These can be either keyword searches or relational searches.

2. Save the document in a .txt format.

   If you are planning on conducting Unicode searches, make sure that you are saving the text file in a Unicode format.

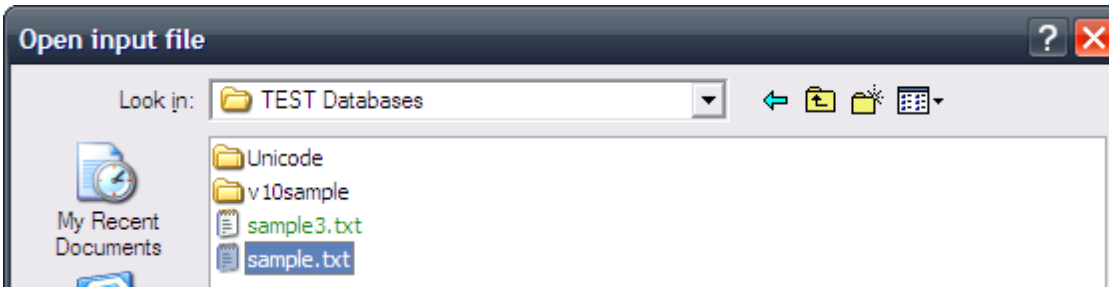
3. In Concordance, on the **File** menu, click **Begin Program**.

4. Locate and open the **TextFileToQuery_v[version #].cpl** file.

5. When prompted, do one of the following:

   - Click **Y** to execute the .cpl file.
   - Click **N** to select a .dcb file to run the .cpl on.

6. In the **Open Input File** dialog box, locate and open the .txt file you created.



7. Locate the folder where you want to store the Error Log file, enter a file name, and then click **Open**.

8. When the CPL is finished, verify that the CPL executed properly.

9. Open the Tags pane and locate the new tag Attachments. The Attachments tag should contain the combined results of all the queries from the text file. You can also view the Review window to display a list of terms in the Search Terms column.

Search Number	Number of Hits	Number of Docs	Search Term(s)
00000	26	26	<Entire Database>
00001	1	1	complex
00002	1	1	environment
00003	1	1	text co technical
00004	1	1	"»Tagging Attachments«"

UpperCase_v10.00

The UpperCase CPL changes all the letters in a paragraph field to uppercase based on an active query. All other fields are not affected by the CPL.

The Uppercase CPL works with the following versions of Concordance:

- 8.*x*
- 9.5*x*
- 10

To run the UpperCase_v[version #].cpl:

1. On the **File** menu, click **Begin Program**.
2. Locate and open the **Uppercase_v[version #].cpl** file.
3. The CPL automatically runs and process the records.
4. When the CPL is finished, verify that the CPL executed properly. The paragraph fields in the query should display the text as all uppercase.