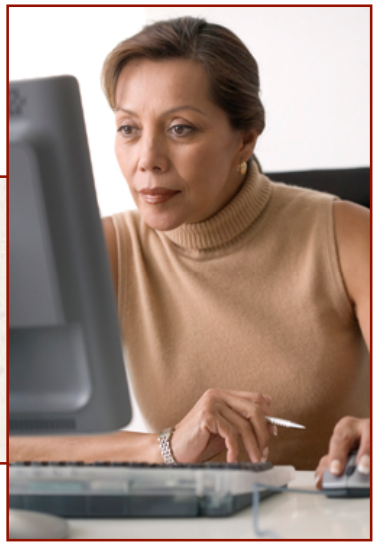
Save User Input In File - Application



Sample code for the system demonstrating this "How To" can be run from the "Save User Input to a File - Application" section of:

http://www.exsys.com/support/howto

Since this is requires running as an Application, it cannot run on-line but can be run on the local machine.

In some systems, it is desirable to allow the end user to answer some questions and quit mid-session, with the ability to return to the same session later. When running as an applet or servlet and connected on-line to a server, this is done by saving the data in a database. (This is explained other "How To" documents.) However, if the expert system will be run as an application with no connection to the Internet and no server, then the user's data cannot be stored in a database on a server. Instead, the expert system can store the data on the local machine in a file. The sample system shows how to add a push button that saves the user's answers to a file and recovers that file the next time the user runs the system.

Must Run as a Java Application

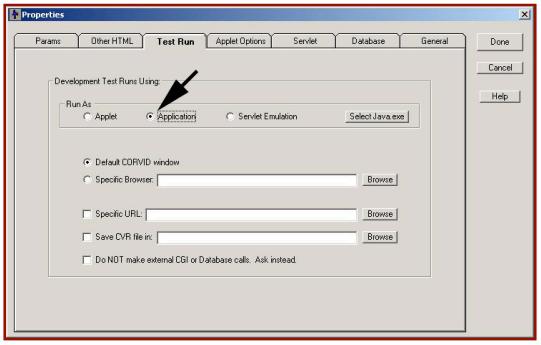
To save data by writing a local file, the system must run as an Application instead of as an Applet because the Java Security Manager will not allow an Applet to write to the local hard drive.

To run as an Application, Java must be installed. Search the hard drive for java.exe. Usually it is in C:\Program Files\Java\... or C:\jdk... or C:\jre.... If it is not installed, it can be downloaded for free from:

http://java.sun.com/

(This site changes frequently, but look for download of the download for the JRE for MS Windows.)

In the Corvid editor, click the "Properties" button and under the "Test Run" tab, select the "Application" radio button. Click the "Select Java.exe" button and browse to the location of java.exe.
You can now run the system as an Application. You must run the sample as an application or it will not work correctly.



Running the Demo

To run the sample system, open it in Corvid editor and select to run as an Application with java.exe selected as described above. Click the “Run” button (the blue triangle). The first question asks your User ID. This will be used as the identifier for the file that will eventually hold the input data. Any text string can be used, but when the system is run a second time, the identical string must be entered to recover the data. The string entered will be part of a file name, so use only characters that can appear in a file name: a-z and 0-9 are safe.

Since the saved data files are only created by users on this machine, there is less of a chance for users to overwrite each others data than for an on-line system. If the machine will be used by multiple users, unique ID strings should be used.

The system will now ask some other questions. There is no “decision logic” in the system, so it does not matter what input is provided. The purpose of the demo is to show how input can be saved and recovered. This can be added to any system that would have decision-making logic in it.

The first question is to select a color. Enter any color, but remember what color was selected. The second question is to select a pet. At this point, click the “Save” button. The system will terminate.

Run the system again. The first question will again be the User ID string. Reenter exactly the same string as on the first run. If they match, the system will ask if you wish to recover the earlier data, or start a new session. Select to recover the earlier data. The system will then ask the “pet” question. Select any pet.

The system will display the Results. The results screen will show the color that was selected on the first run and recovered from the file, along with the pet just selected.

How it Works

The Save User Input feature saves all the user's answers for the current run (session) in a file and reads that data back in next time the user starts a new run. The data is stored in a text file. Since several users could use the system at different times, each user’s data must be stored separately. This demo accomplishes that by asking for a UserID, which is used as part of the file name to make it unique. (In a more complex system, some more automated UserID could be used, especially if there would be many users running the system on the same computer. The ID could also be used to identify different “projects” that had data stored for the same user.)

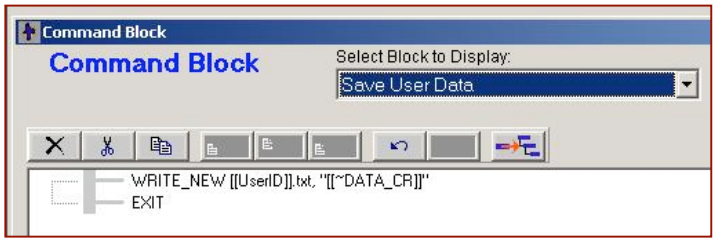
The main command block starts with the command:

```
EXEC BLOCK=Recover User Data
```

This calls the command block that is responsible for recovering the previous run, if any.

The command:

```
SPECIAL_BUTTON "Save", "Save User Data"
```



causes Corvid to add an extra button labeled “Save” to the screens. If the end user clicks on this button, the Command Block named “Save User Data” will be called to write the user data out to a file and exit. (The Command Block that is specified in any SPECIAL_BUTTON command should end or terminate the system. See the Corvid manual's discussion of the SPECIAL_BUTTON command for limitations on what the command block can do and where it should be used.)

Now, examine the Save User Data command block. The command:

```
WRITE_NEW [[UserID]].txt, "[[~DATA_CR]]"
```

writes the file of data, overwriting it if it already exists.

The name of the file is defined by an embedded variable [[UserID]] followed by the ".txt" extension. This simple system does not check that [UserID] has only characters that are legal in filenames. This could be added in a more complex system.

Since Corvid will need the value of this embedded variable [UserID] to execute the command, it will ask the user for the value.

The data written to the file is the string:

"[[~DATA_CR]]"

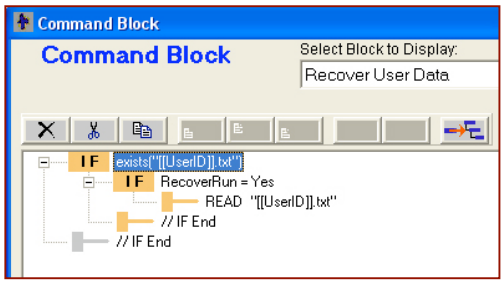
which is a quoted string that has the value of the special Corvid pseudo variable [~DATA_CR] embedded in it. [~DATA_CR] is not an actual system variable, but when embedded, is replaced by the values of all variables that have values, each separated by a carriage return.

Notice that the WRITE command did not specify the folder where the file was to be written. Corvid will write the file in the current working directory. The folder to store the data could be specified in the WRITE_NEW command or input by the user as part of the ID string. To specify precisely where the file will be stored, include the full path for the file, such as:

WRITE_NEW C:\KB_data\user_inputs\[[UserID]].txt, "[[~DATA_CR]]"

where you must replace "C:\KB_data\user_inputs" with the drive and folders where you want the file to be written. Corvid will create the new file. *Note: Corvid will create the specified file, but this command will NOT create the folder "C:\KB_data\user_inputs", so it must already exist before the system is run.*

Examine the Recover User Data Command Block. It recovers data from a previous run, if any.



The system first checks:

exists("[[UserID]].txt")

This will check to see if a file exists under the UserID entered. Since this expression has an embedded variable, Corvid will automatically ask the user for the value of [UserID], allowing them to specify their file of data. If there is a file of data, the system asks the user if they wish to use it with the command:

IF RecoverRun = Yes

This uses a static List variable to ask the user if they wish to recover data. They may either wish to ignore the old data and start a new run, they may want to recover the data. If they select to read the old data, this is done with the command:

READ "[[UserID]].txt"

If you modified the WRITE command to specify the path, you must include the paths here too, like this:

IF exists("C:\KB_data\user_inputs\[[UserID]].txt")

and

READ "C:\KB_data\user_inputs\[[UserID]].txt"

Open a saved data file and look at the contents, the file has a variable name followed by its value. The name/value pairs are separated by a tab. The READ command will load the data into the system and assign the values to their associated variables. The system now starts a normal run, which leads to it asking for the value for variables. For those questions where the value is already assigned from the READ command, Corvid does not need to ask the user for input and can proceed with the logic. Since the same values input will lead to the same questions, the set of input read from the file will take the user to exactly the same point in the system where they left off, with the same rules firing etc.

Note: If the logic or rules in a system is modified, old data files should not be used since they may not match the variables in the new system.

Merging the Sample System to Add Functionality

This sample system can be merged with another system to add the functionality of saving input. This should be done with care and only after fully understanding how the system works. To merge the system with another:

- Make a backup copies of your system and the sample system
- Open the copy of the sample system in Corvid editor
- Delete “Logic Block 1”
- Delete the variables [color], [pet], and [conf]
- Save the copy of the sample system and exit Corvid
- Open your system in Corvid editor
- Under the “File” menu and select “Merge”
- Browse to the modified copy of the sample system and select it to merge

After they are merged, move the first 3 commands from “Example Main” to the starting command block in the merged system. You can open both blocks simultaneously and Copy and Paste using the toolbar buttons. You may want to enhance the system by using a different approach to set the UserID.