Desktop Data Delivered

By Adam Hawthorne

While earlier versions of BBj® allowed access to the client's filesystem via a fat client deployment, access in a thin client deployment was not available. Frequently, developers expressed the desire for file access outside of fat client deployment; they wanted flexible and full access to files and directories anywhere in their client-server deployment.

BASIS offers two solutions. New MODE="CLIENT" options on the FILEOPEN and FILESAVE functions provide easy interaction with the client's filesystem from the traditional standard dialogs. However, when the application requires a customized dialog, or more flexibility choosing files and directories, enhancements in BBj 7.0 expand the user's ability to interact with the client computer. The new BBjFileChooser control and BBjClientFileSystem object allow unprecedented interaction with the client's filesystem. This article focuses on the latter solution and the new freedom developers have to access information, anywhere, regardless of whether it is server- or client-based.

Introduction

The BBjClientFileSystem provides a straightforward interface to handle system-wide file actions. Using BBjClientFileSystem, the developer can obtain a specific file from the root filesystem, the user's home directory, or an absolute path to the desired file. Another new object, the BBjClientFile, provides the API for individual files. Its API models the Java class called **`java.io.File`**, which represents the name of a legal file on the client's filesystem. BBjClientFile objects may represent non-existent files or directories, and the API provides methods to create a non-existent file or directory. Most of the methods on the BBjClientFile match their respective methods on an instance of a **`java.io.File`**. These include self-explanatory methods such as canRead, canWrite, isDirectory, exists, etc. For more information about the other methods, refer to the BBjClientFile in the online documentation at www.basis.com.

Specifically, these four BBjClientFile methods allow data exchange between the client and the server:

RETURN VALUE	METHOD
`string`	`BBjClientFile::copyFromClient()`
`void`	`BBjClientFile::copyToClient(string fileName$)`
`string`	`BBjClientFile::getContents()`
`void`	`BBjClientFile::setContents(string contents$)`

To give these methods context, let us put them to work.

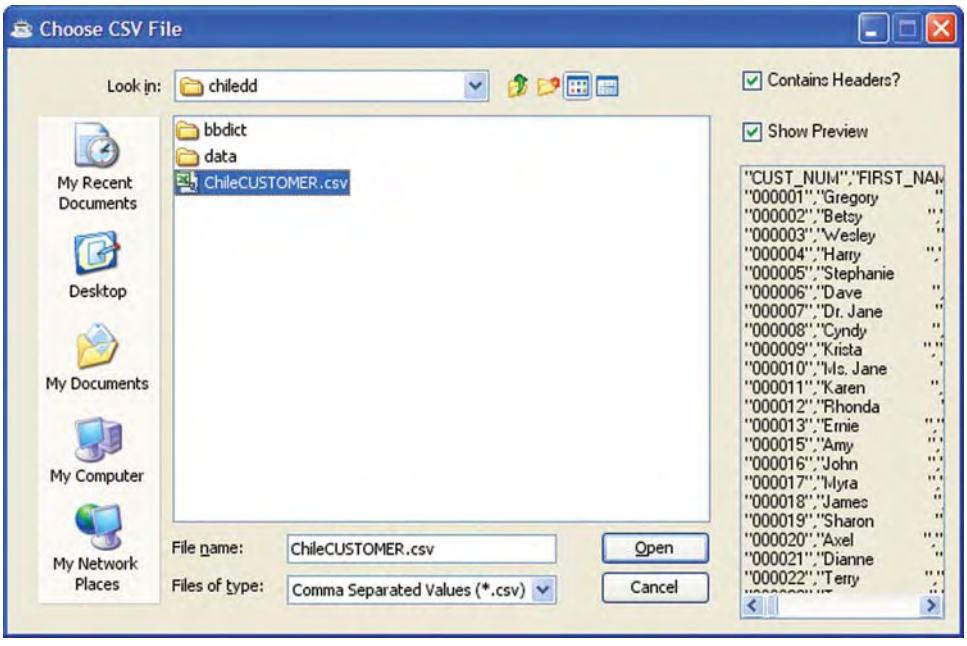

Figure 1. The customized file chooser dialog with file preview capabilities

Sample Program

To demonstrate the BBjClientFileSystem, look at the sample program **`clientFSArticle.src`**, downloadable from the URL noted at the end of this article. It allows users to select a CSV file on their own computer to display in a grid. It uses the aforementioned getContents and copyFromClient methods to obtain the contents of the file in two different contexts. When showing a customized client-side file chooser dialog (see **Figure 1**), the sample uses getContents to show a preview of the CSV file. It retrieves the data from the client and places it into the preview control on the file chooser dialog displayed on the client. This allows the user

continued...

Adam Hawthorne
Software Engineer

to see the contents of the file before selecting it to determine whether the file has a header row. This is a classic example of going above and beyond the standardized file open dialog; it extends the basic dialog with preview capabilities and other user options to provide all of the necessary information to display the CSV file.

Once the user selects a file, the application uses the copyFromClient method to copy the file from the client's machine to the server machine, thus obtaining the data in the file. When the sample calls copyFromClient on the BBjClientFile variable **`file!`**, BBj retrieves the data in the file named by the **`file!`** variable, copies it to a temporary file located on the server, and returns the name of the temporary file. When the copying operation completes, the program parses the file into a **`CSVTable`** object and displays the table in a grid on the client system, as shown in **Figure 2**. When the program terminates, it automatically deletes the temporary files that this method created. To retain a copy of the file on the server after the end of the program, use the RENAME verb to move the file to a different location.

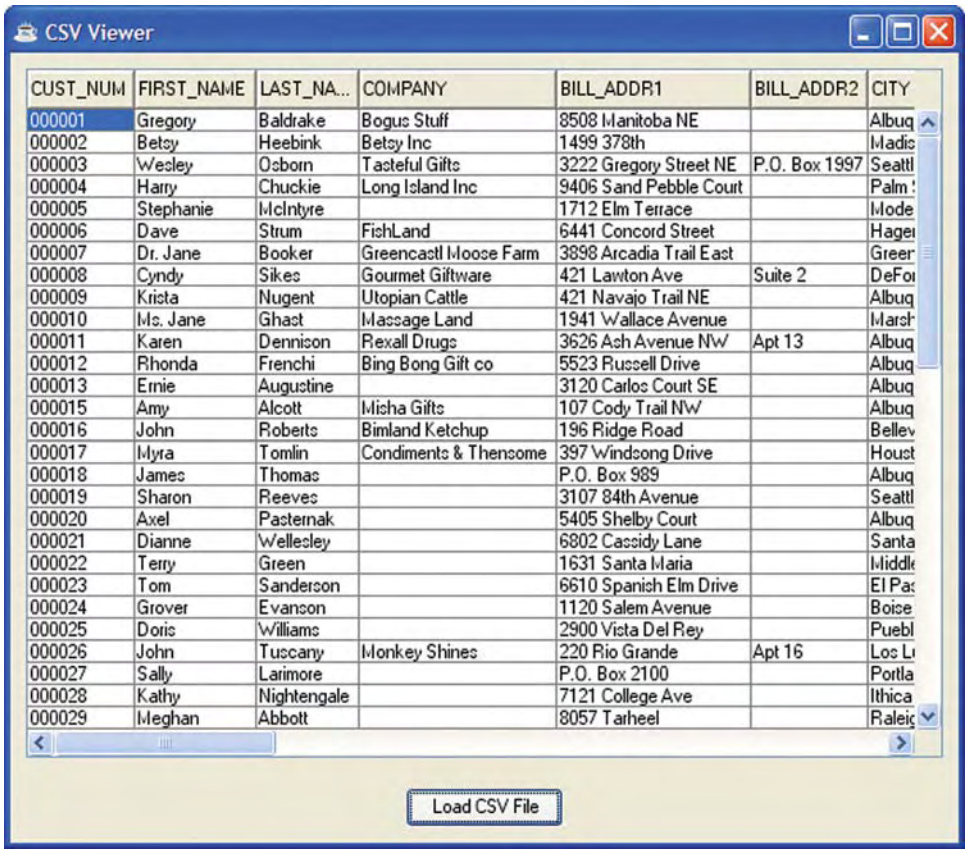

Figure 2. The contents of the CSV file displayed in a grid

The majority of the code in this sample application comes from parsing the CSV file. The source code that shows the file chooser dialog and copies the file from the client, including the code to preview files and respond to events, is relatively small. The supplemental program, **`csvLibrary.src`**, provides an object-oriented model for a simple table and a class with methods to produce the table from a file.

Summary

BBj 7.0 gives developers who leverage these new API tools much greater flexibility in interacting with the client computer. They can store or retrieve configuration, data files, or even customized programs on or from the client computer using BBjClientFileSystem and BBjClientFile. Programs can deliver PDF files, spreadsheets or any other useful files directly to the user's computer at a location the user chooses using the BBjFileChooser in client mode and can retrieve configuration or other files as needed. Now, enjoy the freedom to place files where you need them using the BBjClientFileSystem. Deliver data, right to your desktop!

Download the sample code from
www.basis.com/advantage/mag-v11n1/clientfilesystem.zip