Next Generation File Format: XKEYED Files

By Nick Decker

We've had many requests over the past several years to enhance BBx®'s most advanced file type, multi-keyed MKEYED files. In particular, many Customers were starting to feel the constraints of the format's key definitions. The MKEYED file format allows for up to 16 keys per record, with a maximum of 48 segments for all of the keys. Additionally, the format limited a key to 120 bytes regardless of the number of segments. While practical limitations to key definitions exist, a total of 16 keys could be confining for specific data files. If a particular application uses a file mainly for lookups, for example, it could very well benefit from having numerous keys as the program can use them to more effectively filter the data during queries. However, if an application is constantly modifying a file with record addition or removal, having too many keys can slow down the update process.

We introduced the XKEYED file format in BBj® 1.0 and designed it to remove many of the restrictions of the MKEYED file format. To accomplish our primary objective, we abolished all of the key limitations. As a result, the XKEYED file format allows for an unlimited number of keys, defined in an unlimited number of segments, for an unlimited length. You're now free to define your files exactly as you'd like without having to worry about past restrictions.

We also wanted the XKEYED file format to be extensible. In contrast, an MKEYED file contains all of the file definition data in the first 512 bytes of the file, known as the file header. As with many concepts in the computer industry, 512 bytes may have seemed like a huge amount several years ago, but in today's world, it doesn't afford much space for file definitions. As we were running out of space in the header, we couldn't do much to enhance the MKEYED file type. In order to make dramatic improvements and allow for future growth and capabilities, we had to go about storing the file's definition in a different manner. As a result, XKEYED file definitions are unlimited in length and do not have to occupy a contiguous space at the beginning of the file. Because of this, we have the ability to add features to the format later in time without having to worry about fitting everything into a header with a fixed size. Additionally, it also gives us great freedom when modifying a file's key definition after the file has been created - something else you can't do with MKEYED files.

XKEYED files offer a couple more advantages - size and iteration speed. Although file sizes vary depending on the size and number of keys, on average, XKEYED files tend to be smaller - about 90% of the size of a comparable MKEYED file. While this is not dramatic, it did end up saving us about 100 MB for our internal accounting system data files and it reduced backup times. Record iteration is faster as well. As an example, an SQL SELECT that iterated through 8,625 records of our serial number transaction database executed in 90% of the time when the data files were in XKEYED format compared to MKEYED. XKEYED files perform even better when accessed via the interpreter. Using READ RECORDS to access the file eliminates the overhead of the SQL API, and iteration through XKEYED files is about 65% of the time for MKEYED files.

Converting your existing MKEYED files to XKEYED is a relatively simple task using BBj's object interface. The first step is to instantiate a BBj file system object. Once you have done that, you can make use of the ConvertMKeyedToXKeyed method for the object that does the actual file conversion. Here's a sample:

```
myBBJ!=BBJAPI()
myFS!=myBBJ!.getFilesystem()
myFS!.ConvertMKeyedToXKeyed(Filename$,1)
```

This converts the file that's stored in the Filename$ variable to a highly recoverable XKEYED file. In order to preserve the original file, a backup copy of the MKEYED file is created before the file is converted. As a result, you will have two files in the directory after the conversion. The first is the XKEYED version that uses the original filename. The second is the backup copy of the MKEYED file, which has a bak extension added to the original filename.

With the addition of XKEYED files come a few new verbs and functions that are specific to the new file type. For example, to get extended information about an XKEYED file, you use the XFID() and XFIN() functions instead of the FID() and FIN() functions. To create an XKEYED file, you use the XKEYED or XFILE verb. Also a new SETOPTS bit facilitates the migration to XKEYED files. When byte 8, bit 40 is on, BBj will automatically create XKEYED files even if the application uses the MKEYED verb.