



New and Improved Locking Detection

You may be familiar with the earlier *BASIS International Advantage* article [Solving the Locked Record 'Whodunit'](#) about working with locked records. The information below provides a valuable update to that article with examples and additional tips using tools now available from version 14.20 of BBj®.

BBj provides developers with several ways to access data files using direct file access as well as SQL. When using direct file access, developers use the OPEN, READ RECORD, and WRITE RECORD verbs. In addition, the developer can control access to the file or even individual records in the file using LOCK and EXTRACT. While these calls are very useful and often necessary, the potential arises where a program EXTRACTs a record or LOCKs a file on behalf of a user who might get distracted or leave for lunch. As a result, other users cannot access the data they need. This article provides some ways to resolve such circumstances and make it easier to track down the offending user.

Locked Files

When a program makes a LOCK call on a file channel, the file is locked from access by any other process. If a program attempts to open a locked file, the OPEN call throws an 'error 0' but it does not give any indication as to who has the file locked. Knowing who has a file locked is very useful information, especially when a file that has been locked for some time prevents other users from accessing it. The Enterprise Manager (EM) displays the list of open files, so an administrator can check this list, find out which user has the file locked, and force close that file if necessary. However, sometimes it is beneficial for the program to indicate who has a file locked in its lock error message. That way, an administrator does not need to be brought in every time there is a lock left open on a file. The following code samples demonstrate how to easily access this information from within a BBj program.

Run the **SetLock.bbj** program in **Figure 1** to create a sample file and lock that file.

```
filename$="sample.dat"
erase filename$,err=*next
mkeyed filename$, 1, 0, 16
open (1) filename$
lock (1)
escape
```



By Jeff Ash
Software Engineer

Figure 1. Sample SetLock.bbj program that creates and locks a file



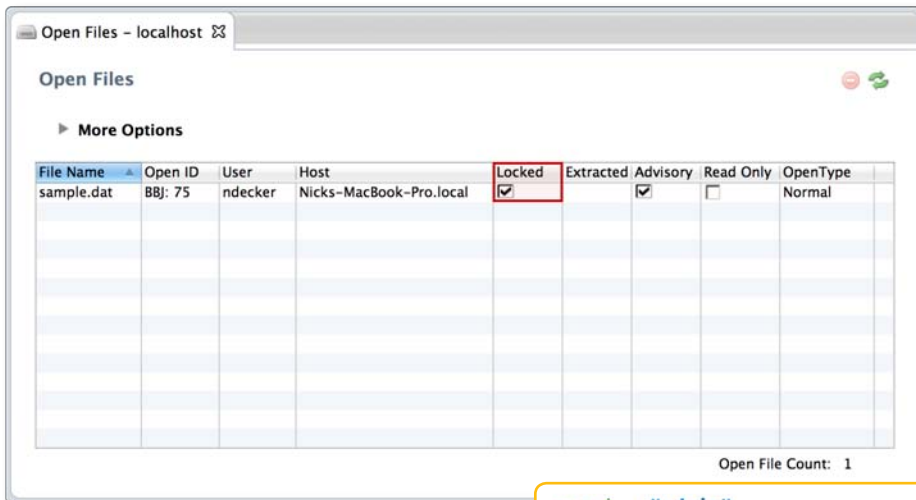


Figure 2. The Open Files list showing a locked file

After running **SetLock.bbj**, log in to the EM and locate the "Open Files" node under "File System" in EM's navigator. Note the file appears in the list in **Figure 2**, and indicates the user who has the file locked.

At this point, an administrator could force close this file to free up the lock. However, an alternative is to throw a new error with a more complete error message indicating who has the file locked as well as their hostname. While **SetLock.bbj** is still sitting at a ready prompt, execute the **EncounterLock.bbj** program in **Figure 3**.

Running the code in **Figure 3** returns the more detailed error message shown in **Figure 4** that indicates who has the file locked and their machine name.

Extracted Records In Files

When a program executes an **EXTRACT** call, it locks a single record in the file and prevents other processes from updating that particular record. If the program is using advisory locking, other processes can read an extracted record, but still cannot update that record. As with locked files, it would be very nice for a program to indicate who has a particular record extracted (locked) in a file. Reporting this information is more difficult and time consuming than locating a locked file because when a program encounters an extracted record, the only information it has immediately is called a 'lock byte'. This value is simply an integer indicating the portion of the file that is currently extracted. By itself, this information is not very useful. However, using the lock byte value in EM, an administrator can locate the open file instance that contains a matching lock byte value in the "Extracted" column. Run the **SetExtract.bbj** program in **Figure 5** that simulates extracting 100 different records on a file at a given point in time.



```

user$ = "admin"
password$ = "admin123"
filename$ = "sample.dat"
open (1, err=errtrap) filename$
print "File successfully opened!"
stop

errtrap:
if err <> 0 then escape
api! = BBjAdminFactory.getBBjAdmin(user$, password$)
fullPath$ = BBjAPI().getFileSystem().resolvePath(filename$)

bbjFile! = api!.getBBjFile(fullPath$)
openFile! = bbjFile!.getOpenFileLocked()
if openFile! <> NULL() then
    lockUser$ = openFile!.getString(BBjAdminOpenFile.USER)
    host$ = openFile!.getString(BBjAdminOpenFile.HOST)
    errMsg$ = lockUser$ + "@" + host$ + " has the file locked."
    throw errMsg$, 0

    REM Optionally force close the locked file.
    REM openFile!.forceClose()
else
    REM If we get here the file was unlocked while
    REM processing this error.
    retry
endif

```

Figure 3. A sample EncounterLock.bbj program that returns a more detailed error message

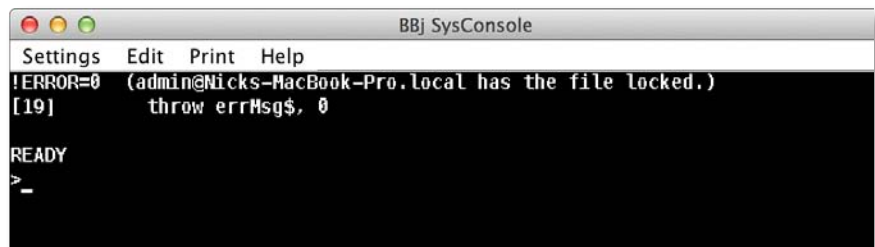


Figure 4. The more detailed error resulting from the code in Figure 3

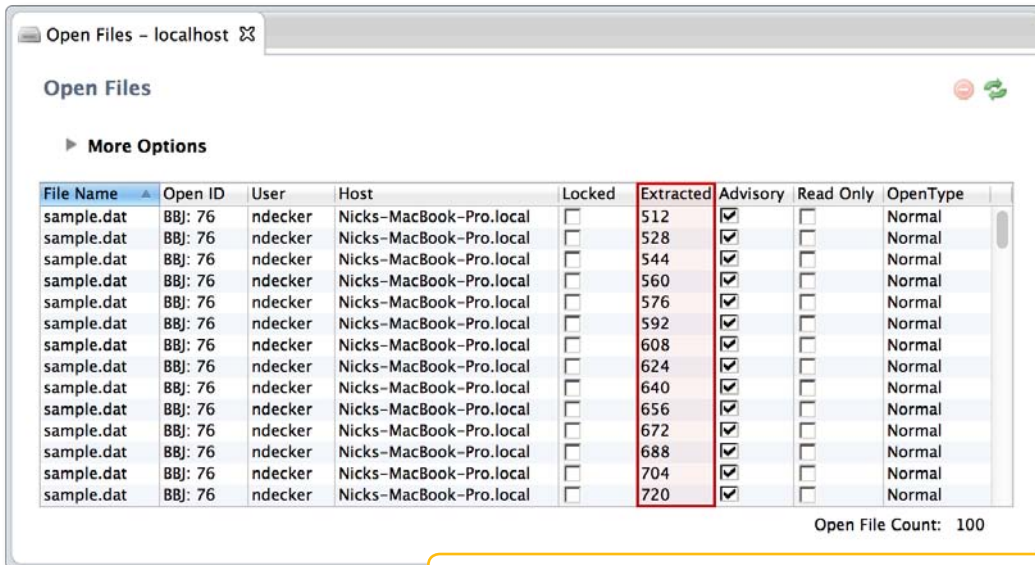
```

filename$="sample.dat"
erase filename$, err=*next
mkeyed filename$, 3, 0, 16
for i = 1 to 100
    open(i) filename$
    val$ = str(i)
    write (i, key=val$) val$
    extract (i, key=val$)
next i
escape

```

Figure 5. The SetExtract.bbj program that simulated extracting 100 records





File Name	Open ID	User	Host	Locked	Extracted	Advisory	Read Only	OpenType
sample.dat	BBJ: 76	ndecker	Nicks-MacBook-Pro.local	<input type="checkbox"/>	512	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Normal
sample.dat	BBJ: 76	ndecker	Nicks-MacBook-Pro.local	<input type="checkbox"/>	528	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Normal
sample.dat	BBJ: 76	ndecker	Nicks-MacBook-Pro.local	<input type="checkbox"/>	544	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Normal
sample.dat	BBJ: 76	ndecker	Nicks-MacBook-Pro.local	<input type="checkbox"/>	560	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Normal
sample.dat	BBJ: 76	ndecker	Nicks-MacBook-Pro.local	<input type="checkbox"/>	576	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Normal
sample.dat	BBJ: 76	ndecker	Nicks-MacBook-Pro.local	<input type="checkbox"/>	592	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Normal
sample.dat	BBJ: 76	ndecker	Nicks-MacBook-Pro.local	<input type="checkbox"/>	608	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Normal
sample.dat	BBJ: 76	ndecker	Nicks-MacBook-Pro.local	<input type="checkbox"/>	624	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Normal
sample.dat	BBJ: 76	ndecker	Nicks-MacBook-Pro.local	<input type="checkbox"/>	640	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Normal
sample.dat	BBJ: 76	ndecker	Nicks-MacBook-Pro.local	<input type="checkbox"/>	656	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Normal
sample.dat	BBJ: 76	ndecker	Nicks-MacBook-Pro.local	<input type="checkbox"/>	672	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Normal
sample.dat	BBJ: 76	ndecker	Nicks-MacBook-Pro.local	<input type="checkbox"/>	688	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Normal
sample.dat	BBJ: 76	ndecker	Nicks-MacBook-Pro.local	<input type="checkbox"/>	704	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Normal
sample.dat	BBJ: 76	ndecker	Nicks-MacBook-Pro.local	<input type="checkbox"/>	720	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Normal

Open File Count: 100

Figure 6. Large number of open file channels

Knowing the lock byte value of the extracted record (acquired by getting the TCB(10) value after an error 0 occurs), an administrator can manually use the EM to locate the entry in the open files list that has that matching lock byte value in the “Extracted” column. However, this could be an arduous task if there is a large number of open file channels on a particular file as shown in Figure 6.

Admin API to the Rescue

Using the Admin API, a program can locate the information about the open file channel that has the particular extracted record of interest. The **EncounterExtract.bbj** program shown in Figure 7 demonstrates how to throw a new error that includes the user and host for the open file channel with the extracted record using the lock byte value, resulting in the message that appears in Figure 8.

Summary

Discovering who has a file locked or a particular record extracted is no longer a complicated process requiring an administrator. Using some simple methods available in the Admin API, developers can extend the application to provide more informative error messages, making it possible for users to potentially resolve lock issues without involving an administrator. Grab the latest version of BBJ and improve your users’ experience today! ■

```

user$ = "admin"
password$ = "admin123"
filename$ = "sample.dat"
open (1) filename$
extract (1, key="15", err=errtrap)
print "Record successfully extracted!"
stop

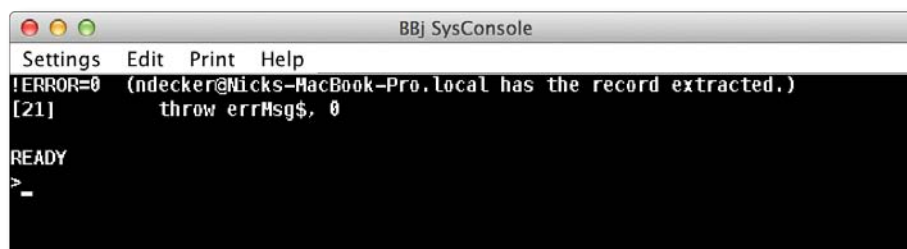
errtrap:
  if err <> 0 then escape
  lockbyte = tcb(10)
  api! = BBJAdminFactory.getBBJAdmin(user$, password$)
  fullPath$ = BBJAPI().getFileSystem().resolvePath(filename$)

  bbjFile! = api!.getBBJFile(fullPath$)
  openFile! = bbjFile!.getOpenFileExtracted(lockbyte)
  if openFile! <> NULL() then
    extractUser$ = openFile!.getString(BBJAdminOpenFile.USER)
    host$ = openFile!.getString(BBJAdminOpenFile.HOST)
    errMsg$ = extractUser$ + "@" + host$ + " has the record extracted."
    throw errMsg$, 0

  REM Optionally force the file closed automatically
  REM openFile!.forceClose()
  else
    REM If it gets here file is no longer locked.
    retry
  endif

```

Figure 7. The EncounterExtract.bbj code that returns complete information about the locked record



```

BBJ SysConsole
Settings Edit Print Help
!ERROR=0 (ndecker@Nicks-MacBook-Pro.local has the record extracted.)
[21]      throw errMsg$, 0

READY
>_

```

Figure 8. The more detailed error message resulting from running EncounterExtract.bbj



Download the examples at links.basis.com/lockextractcode