

FairCom White Paper

# Memory Files

Audience

**Developers**

Subject

**FairCom's high-performance NAV and SQL database technology.**

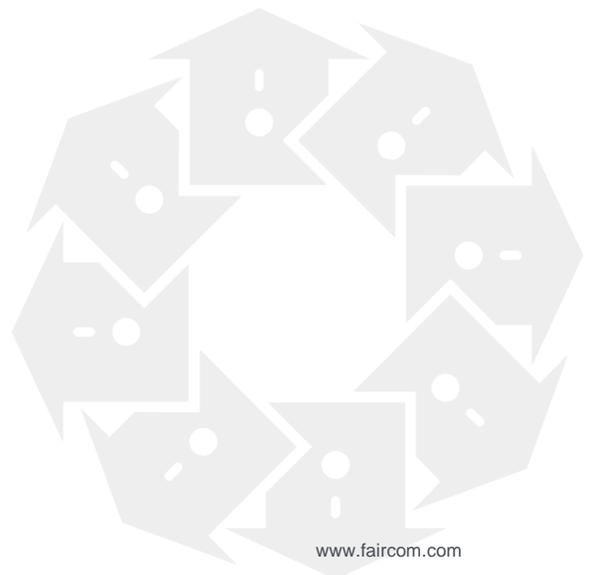
© Copyright 2021, FairCom Corporation. All rights reserved. For full information, see the FairCom Copyright Notice (page vi).



**FairCom**<sup>®</sup>

# Contents

<b>1.</b>	<b>Memory Files Overview .....</b>	<b>1</b>
1.1	HUGE File Support .....	2
1.2	Creating Memory Files Using Server Configuration Keyword .....	2
1.3	Creating Memory Files Programmatically .....	2
1.4	Sharing Memory Files Created Programmatically .....	3
1.5	Collecting Memory File Statistics .....	4
1.6	Memory File Limitations .....	4
1.7	Tip: Faster Server Shutdown with Memory Files .....	5
<b>2.</b>	<b>Document History .....</b>	<b>5</b>
<b>3.</b>	<b>Index .....</b>	<b>8</b>



# 1. Memory Files Overview

The FairCom DB Server supports pure memory-resident data and index files. The distinguishing characteristic of memory files is that they exist solely in memory, occupying their own memory space separate from the data and index caches.

Memory files satisfy the need for the creation and manipulation of temporary data or index files that are always memory resident and never touch disk. Contrast this with non-memory files, whose contents may be paged out of the database cache and written to disk. Memory files are ideal for true temporary files: they can exist in memory as long as the FairCom DB Server or c-tree application process is running.

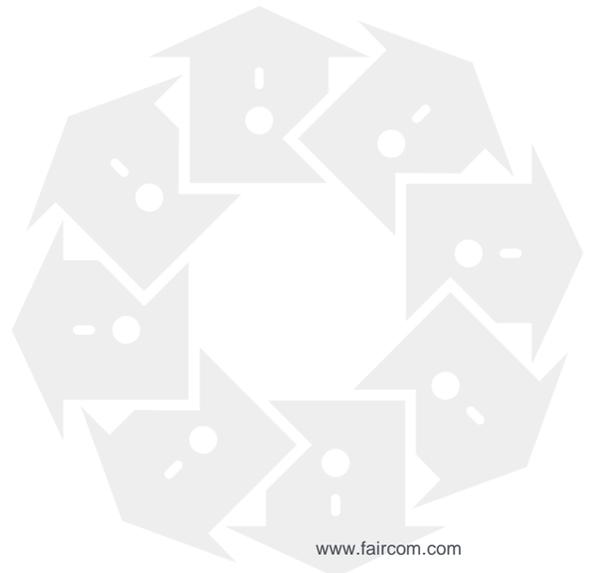
Some applications of memory files include:

- **Temporary files:** Temporary files can be created as memory data or index files. The file contents are always memory resident, and when the memory file is finally closed, it ceases to exist.
- **Storing read-only file contents:** At server or application startup, the contents of a disk file can be read into a memory file and subsequent read requests can be satisfied from the memory file.
- **In-memory list management:** Applications frequently need to maintain in-memory lists. A memory index provides easy creation and manipulation of items in a B+-tree-indexed list using the c-tree API, rather than requiring the developer to implement custom list management routines.

## Using Memory Files

The main operational differences between memory files and non-memory files involve creating and closing the files. After memory files are created, they are accessed using the standard c-tree API functions, just as non-memory files.

Published 4/15/2021





## 1.1 HUGE File Support

As memory files use the native memory address size associated with their underlying platform architecture, they must match the attributes of that platform. Importantly, a memory file on a 64-bit OS requires 64-bit offset support. (Contrast that to filesystem support which is separate and independent of memory architecture support.)

Memory files on 32-bit OS platforms must use 32-bit file offsets (and *cannot* be HUGE).

Memory files on 64-bit OS platforms must use 64-bit file offsets (and *must* have **HUGE** file support).

*When using memory files on a 64-bit server:* If you create any indexes that allow duplicates, be sure to include 8 additional bytes in the total key length (rather than the typical 4 bytes) for the record offset indicator. This will avoid potentially triggering a **582 error**. 115 errors can also be observed as key segment lengths do not match the assigned key length.

## 1.2 Creating Memory Files Using Server Configuration Keyword

The FairCom DB Server supports creating memory files using a server configuration keyword. This feature allows developers to create memory files using their existing application code, provided that the file is created using an *Xtd8* create function such as **CreatelFileXtd8()**, see *Xtd8 File Creation Functions* (<http://docs.faircom.com/doc/ctreeplus/30545.htm>). To create a memory file using the server configuration keyword, specify one or more entries of the form:

```
MEMORY_FILE <file name>#<max size>
```

where the file name may include wild card characters, and the maximum size is optional. If no maximum size is specified, then 4GB is used. If a file is being created and matches one of the `MEMORY_FILE` file name entries, then it will be created as a memory file unless it is a superfile host, superfile member, mirrored, segmented or partitioned file.

To cause all possible files to be created as memory files, add the following configuration entry:

```
MEMORY_FILE *
```

The `MEMORY_FILE` keyword is useful to quickly test how a file or set of files will behave as memory files.

## 1.3 Creating Memory Files Programmatically

To create a memory file programmatically, include the `ctMEMFILE` attribute in the `x8mode` member of the file's `XCREblk` structure (see *Extended File Creation Block Structure* (<http://docs.faircom.com/doc/ctreeplus/30571.htm>)), and specify the maximum file size using the `mxfilzhw` and `mxfilzlw` members of the `XCREblk`.



A temporary, memory-resident index file can be created independent of any data file. Simply call **CreateIndexFileXtd8()** (<https://docs.faircom.com/doc/ctreepus/createindexfilextd8.htm>) with an *XCREblk* set as described above.

See the pseudo-code below to create a memory file:

```
XCREblk xcreblk[] = {
    {ctMEMFILE, 0, 0, 104857600, 0,0,0,0,0,0, ctKEEPOPEN, 0,0,0,0,0},
    {ctMEMFILE, 0, 0, 104857600, 0,0,0,0,0,0, ctKEEPOPEN, 0,0,0,0,0}
};
...
if (CreateIFileXtd8(&vcustomer,NULL,NULL,0,NULL,NULL,xcreblk))
{
    ctrt_printf("\nCould not create file %d with error %d.\n",isam_fil,isam_err);
}
```

**Note:** Memory files can not have *ctTRNLOG* mode applied or **CreateIFileXtd8()** will fail with error 749 (bad parameter). Be sure to adjust any *IFIL* definitions.

## 1.4 Sharing Memory Files Created Programmatically

There is a subtle issue about how to get a memory-resident file to be a shared file. At create, it is exclusive. If it is created and immediately closed, to be reopened for shared access, it would disappear on the close, and the reopen would fail. You can use the **UpdateFileMode()** c-tree API function to change the file mode from exclusive to shared:

```
CreateIFileXtd8(...);
UpdateFileMode(datno,ctSHARED);
UpdateFileMode(keyno_1,ctSHARED);
...
UpdateFileMode(keyno_n,ctSHARED);
```

The **UpdateFileMode()** call will fail if there are transaction-dependent actions pending. For example, if the **CreateIFileXtd8()** call in the above pseudo code were for a transaction-dependent (*ctTRANDEP*) file, the **UpdateFileMode()** call could not be called until the create is committed.

As noted above, the final close of a memory file causes all the existing memory file contents (data records or index nodes) to be lost. A final close refers to a file close that causes the user count of the file to drop to zero. It is possible to make the data file persist even after the final close by including *ctKEEPOPEN* bit in the *splval* member of the data file's *XCREblk*. Then the final close leaves the file open (but without any user attached to the file.) It can then be opened again, and the data still exists. The file will be closed when the server terminates, or when a call is made to the **CloseCtFileByName()** c-tree API function:

```
CloseCtFileByName(pTEXT filnam,pTEXT fileword)
```

It is possible to use the *ctKEEPOPEN* flag on the create, and then close and reopen the file shared (without calling **UpdateFileMode()**), but only if the file's creation is not pending commit. In this case, the sequence would be like:



```
XCREblk xcreblk[] = {
    {ctMEMFILE, 0, 0, 104857600, 0,0,0,0,0,0, ctKEEPOPEN, 0,0,0,0,0},
    {ctMEMFILE, 0, 0, 104857600, 0,0,0,0,0,0, ctKEEPOPEN, 0,0,0,0,0}
};
...
if (CreateIFileXtd8(&vcustomer, NULL, NULL, 0, NULL, NULL, xcreblk))
{
    ctrt_printf("\nCould not create file %d with error %d.\n",
        isam_fil, isam_err);
}
else
{
    CloseIFile(&vcustomer);
    if ((eRet = OpenIFileXtd(&vcustomer, NULL, NULL, NULL)) != 0)
    {
        ctrt_printf("\nUNEXPECTED ERROR %d ON REOPEN.\n", eRet);
    }
    else
    {
        ctrt_printf("\nFile created in memory.");
    }
}
}
```

## 1.5 Collecting Memory File Statistics

Memory file statistics are available through the FairCom DB command-line utility `ctstat` using the `-m` switch, discussed in the topic titled *ctstat - Statistics Utility* (<http://docs.faircom.com/doc/ctreeplus/52856.htm>).

Memory file statistics can also be collected using the **SnapShot()** function call discussed here in the *SnapShot* (<http://docs.faircom.com/doc/ctreeplus/snapshot.htm>) topic. See the details for the *ctGFMS* structure where the following memory file values are maintained:

- *memcnt*: current number of memory records
- *hghcnt*: highest number of memory records
- *phyrec*: current total memory allocated for memory records
- *mghghbyt*: highest amount of memory allocated for memory records

## 1.6 Memory File Limitations

Memory files are subject to the following limitations:

- A memory file cannot be mirrored, partitioned, or segmented.
- A memory file cannot be backed up by a dynamic dump.
- The ISAM rebuild function **RebuildFile()** does not support rebuilding indexes associated with a memory data file; instead use the low-level **RebuildIndex()** function to rebuild individual indexes.



- The restorable delete (*ctRSTRDEL*) attribute is not supported for memory files.
- The atomic, recoverable transaction control attribute (*ctTRNLOG*) is not supported for memory files. However, the atomic, non-recoverable transaction control (*ctPREIMG*) and the transaction-dependent (*ctTRANDEP*) attributes are supported for memory files.
- When traversing memory files in “physical order” there is no guarantee of any particular ordering. The sequence of **FirstRecord()**, **NextRecord()** operations will traverse records ordered by their internal hash table positioning, which can be quite different from a typical physical ordering on disk.
- Be sure to see *HUGE File Support (page 2)* for additional requirements.

## 1.7 Tip: Faster Server Shutdown with Memory Files

When the FairCom DB Server shuts down, it frees individual memory records for memory files that are still open. If large memory files are open, the freeing of the memory records can prolong the server shutdown time. Adding the `COMPATIBILITY MEMORY_FILE_SKIP_FREE` keyword to the server configuration file causes the server to skip the freeing of individual memory records. When the server process terminates, the operating system will automatically reclaim the memory allocated to memory file records.

## 2. Document History

03/20/2019	Added note about HUGE files. Updated for readability.
------------	---

Last published Thursday, April 15, 2021.

# Copyright Notice

Copyright © 1992-2021 FairCom USA Corporation. All rights reserved.

No part of this publication may be stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without the prior written permission of FairCom USA Corporation. Printed in the United States of America.

Information in this document is subject to change without notice.

## Trademarks

FairCom DB, FairCom EDGE, c-treeRTG, c-treeACE, c-treeAMS, c-treeEDGE, c-tree Plus, c-tree, r-tree, FairCom, and FairCom's circular disc logo are trademarks of FairCom USA, registered in the United States and other countries.

The following are third-party trademarks: Btrieve is a registered trademark of Actian Corporation. Amazon Web Services, the "Powered by AWS" logo, and AWS are trademarks of Amazon.com, Inc. or its affiliates in the United States and/or other countries. AMD and AMD Opteron are trademarks of Advanced Micro Devices, Inc. Macintosh, Mac, Mac OS, and Xcode are trademarks of Apple Inc., registered in the U.S. and other countries. Embarcadero, the Embarcadero Technologies logos and all other Embarcadero Technologies product or service names are trademarks, service marks, and/or registered trademarks of Embarcadero Technologies, Inc. and are protected by the laws of the United States and other countries. HP and HP-UX are registered trademarks of the Hewlett-Packard Company. AIX, IBM, POWER6, POWER7, POWER8, POWER9, POWER10 and pSeries are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. Intel, Intel Core, Itanium, Pentium and Xeon are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries. ACUCOBOL-GT, Micro Focus, RM/COBOL, and Visual COBOL are trademarks or registered trademarks of Micro Focus (IP) Limited or its subsidiaries in the United Kingdom, United States and other countries. Microsoft, the .NET logo, the Windows logo, Access, Excel, SQL Server, Visual Basic, Visual C++, Visual C#, Visual Studio, Windows, Windows Server, and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Oracle and Java are registered trademarks of Oracle and/or its affiliates. QNX and Neutrino are registered trademarks of QNX Software Systems Ltd. in certain jurisdictions. CentOS, Red Hat, and the Shadow Man logo are registered trademarks of Red Hat, Inc. in the United States and other countries, used with permission. SAP® Business Objects, SAP® Crystal Reports and SAP® BusinessObjects™ Web Intelligence® as well as their respective logos are trademarks or registered trademarks of SAP. SUSE" and the SUSE logo are trademarks of SUSE LLC or its subsidiaries or affiliates. UNIX and UNIXWARE are registered trademarks of The Open Group in the United States and other countries. Linux is a trademark of Linus Torvalds in the United States, other countries, or both. Python and PyCon are trademarks or registered trademarks of the Python Software Foundation. isCOBOL and Veryant are trademarks or registered trademarks of Veryant in the United States and other countries. OpenServer is a trademark or registered trademark of Xinuos, Inc. in the U.S.A. and other countries. Unicode and the Unicode Logo are registered trademarks of Unicode, Inc. in the United States and other countries.

All other trademarks, trade names, company names, product names, and registered trademarks are the property of their respective holders.

Portions Copyright © 1991-2016 Unicode, Inc. All rights reserved.

Portions Copyright © 1998-2016 The OpenSSL Project. All rights reserved. This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (<http://www.openssl.org/>).

Portions Copyright © 1995-1998 Eric Young (eay@cryptsoft.com). All rights reserved. This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com).

Portions © 1987-2020 Dharma Systems, Inc. All rights reserved.

This software or web site utilizes or contains material that is © 1994-2007 DUNDAS DATA VISUALIZATION, INC. and its licensors, all rights reserved.

Portions Copyright © 1995-2013 Jean-loup Gailly and Mark Adler.

Portions Copyright © 2009-2012 Eric Haszlkiewicz.

Portions Copyright © 2004, 2005 Metaparadigm Pte Ltd.

Portions Copyright © 2008-2020, Hazelcast, Inc. All Rights Reserved.

Portions Copyright © 2013, 2014 EclipseSource.

Portions Copyright © 1999-2003 The OpenLDAP Foundation.

## Open Source Components

Like most software development companies, FairCom uses third-party components to provide some functionality within our technology. Often those third-party components are selected because they are a standard in the industry, they offer specific functionality that is easier to license than to develop and maintain in the long run, or they provide a proven and inexpensive solution to a particular business need. Examples of third-party software FairCom uses are the OpenSSL toolkit that provides Transport Layer Security (TLS) for secure communications and the ICU Unicode libraries to provide wide character support (think international characters and emojis).

Some of these third-party components are the subject to commercial licenses and others are subject to open source licenses. For open source solutions that we incorporate into our technology, we include the package name and associated license in a notice.txt file found in the same directory as the server.

The notice.txt file should always stay in the same directory as the server. This is particularly important in instances where your company has redistribution rights, such as an ISV who duplicates server binaries and (re)distributes those to an eventual end-user at a third-party company. Ensuring that the notice.txt file "travels with" the server binary is important to maintain third-party and FairCom license compliance.

4/15/2021

# 3. Index

**C**  
CloseCtFileByName .....3  
Collecting Memory File Statistics.....4  
Copyright Notice ..... vi  
CreateFileXtd8.....3  
Creating Memory Files Programmatically.....2  
Creating Memory Files Using Server  
    Configuration Keyword .....2  
c-tree Server  
    memory files.....1

**D**  
Document History .....5

**F**  
Features  
    memory files.....1  
Files  
    memory .....1  
    storing read-only contents .....1  
    temporary.....1  
Functions  
    CloseCtFileByName .....3  
    CreateFileXtd8 .....3  
    UpdateFileMode .....3

**H**  
HUGE File Support .....2

**K**  
Keyword  
    MEMORY\_FILE .....2  
    MEMORY\_FILE\_SKIP\_FREE .....5

**L**  
Limitations  
    memory file .....4

**M**  
Manage  
    in-memory list .....1  
Memory file .....1  
    collect statistic.....4  
    create programmatically .....2  
    create with keyword .....2  
    faster shutdown .....5  
    limitation .....4  
    share programmatically .....3  
Memory File Limitations.....4  
Memory Files Overview .....1

**S**  
Sharing Memory Files Created Programmatically ....3

**T**  
Tip  
    Faster Server Shutdown with Memory Files .....5

**U**  
UpdateFileMode.....3

