



# Persistence File Cache

## User Manual

<b>Description</b>	V 0.2
<b>Date</b>	April 1th, 2014

Copyright © 2014 XS Embedded GmbH

All rights reserved. No part of this document may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher.

## Table of Contents

<b>1. Persistence File Cache .....</b>	<b>3</b>
1.1. Persistence File Cache Component.....	3
1.2. Persistence File Cache License.....	3
1.3. Restrictions.....	3
1.3.1. File Caching .....	3
1.4. Setup of Caching Location.....	3
1.5. Backup and Recovery Strategy.....	4
1.6. Caching Strategy.....	6
1.7. Persistence File Cache Configuration.....	7
1.7.1. Location .....	7
1.7.2. Example.....	7
<b>2. Persistence File Cache Library Interfaces .....</b>	<b>8</b>
2.1. Application Programming Interface Description .....	8
2.2. API Details.....	9
2.2.1. PfcCacheAffinity .....	9
2.2.2. pfcInitCache .....	9
2.2.3. pfcOpenFile .....	10
2.2.4. pfcCloseFile .....	11
2.2.5. pfcReadFile .....	11
2.2.6. pfcWriteFile .....	12
2.2.7. pfcFileSeek.....	13
<b>3. Dynamic Behavior .....</b>	<b>14</b>
<b>4. How to build.....</b>	<b>15</b>
4.1. Dependencies.....	15
<b>5. Testing and Debugging .....</b>	<b>16</b>
5.1. Running the tests .....	16
<b>6. Appendix .....</b>	<b>17</b>

## Table of Figures

Recovery Algorithm 1	5
Dynamic Behavior 2	14

## Table Directory

Table 1 - Related Documents	18
Table 2 - Abbreviations	18
Table 3 - History	18

## **1. Persistence File Cache**

The scope of this document covers detailed interface description, building the library, testing the component and debugging.

### **1.1. Persistence File Cache Component**

The Persistence File Cache (PFC) is used to do caching of files using a RAM disk.

The intended usage of the PFC is in combination with the Persistence Client Library (PCL). The PCL uses the PFC to cache the files accessed via the PCL file interface. Nevertheless it can also be used by any other component which wants to do file caching in a RAM disk.

### **1.2. Persistence File Cache License**

The Source Code Form is subject to the terms of the Mozilla Public License, v. 2.0. If a copy of the MPL was not distributed with this file. You can obtain one at <http://mozilla.org/MPL/2.0/>

### **1.3. Restrictions**

#### **1.3.1. File Caching**

Only complete files are kept in the cache, partial caching of files will not be done.

If a file is bigger than the cache it will not be cached.

If a file is written and the file size after the write operation is bigger than the cache, the file will not be written and an error code is returned.

### **1.4. Setup of Caching Location**

The Persistence File Cache Component does not provide any functionality of setting up a RAM disk. The using project itself is responsible to create and mount the RAM disk.

## 1.5. Backup and Recovery Strategy

A backup copy of the file including the generation of a checksum file is created when the file is modified the first time (Copy-on-write). There is no backup created if the file will only be read.

The checksum file is used during recovery to ensure that the backup copy is not corrupt.

A backup copy is created only once during the lifecycle and is removed during the shutdown phase when an application has successfully written the data using `pfcWriteFile` and calls `pfcCloseFile`.

When the `pfcCloseFile` function is called the backup and checksum file is deleted. This is an indicator that the system has been shut down correctly. If the PFC detects in a new lifecycle that there is a backup and/or a checksum file available this is an indicator that the system hasn't been shut down correctly and the files must be checked for consistency.

The backup functionality can be disabled if the PFC is used within the PCL, as the PCL already implements the same backup and recovery functionality. The backup functionality in the PFC is needed when it is used standalone without the PCL.

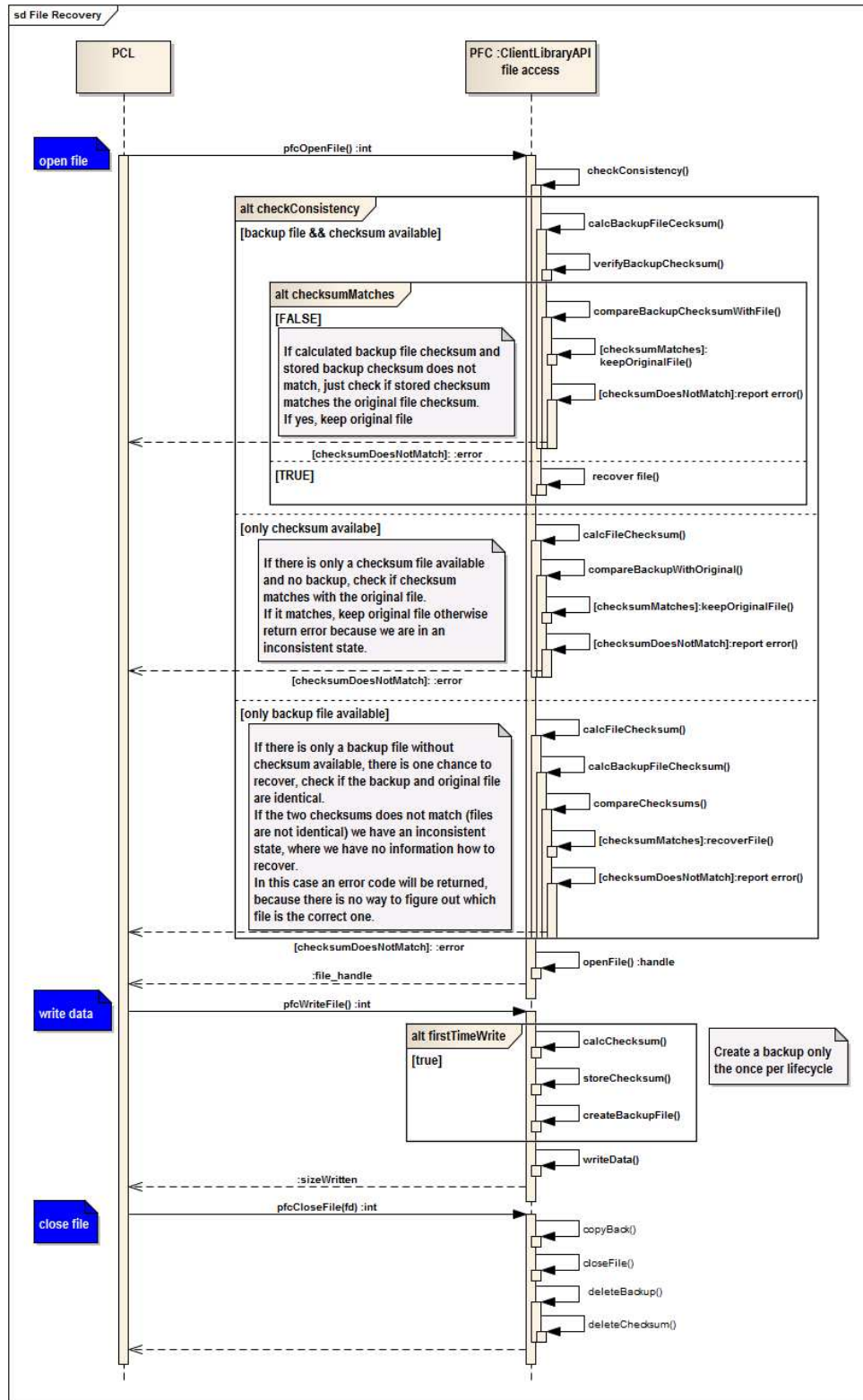
**Note:**

A system failure which ends up in a restart of the system can occur at any time within a lifecycle.

One critical path is during the `pfcWriteFile` function call, where the checksum is calculated and stored and the backup file is created.

The other critical path is during the `pfcCloseFile` function call, where the backup file and the checksum file is deleted and the file is transferred from the RAM disk to the non-volatile storage device.

To cover this "reset" scenarios in the best way, there are some assumptions on the `pfcOpenFile` function, where we try to detect such system failures and do a best guess to react properly. The following sequence diagram explains the recovery algorithm.



## Recovery Mechanism

### 1.6. Caching Strategy

The file is transferred into the cache when it is modified the first time.

All further access to the file (read/write) will only affect the file kept in the cache.

If the PFC is used a standalone component without the PCL, the shutdown scenario is the following.

- Application receive shutdown notification
- Application writes the data for the last time in this lifecycle.
- Application closes the handle.
  - PFC copies back the file from cache to e.g. eMMC
  - PFC closed the POSIX file descriptor to this file
  - PFC deletes backup
  - PFC deletes checksum file

If the PFC is used within the PCL the shutdown scenario is the following.

- Application receives shutdown notification
- Application writes the data for the last time in this lifecycle
- Application sends notification back to lifecycle that shutdown is OK now
- PCL instance of the application receives shutdown notification
- PCL instance of the application write data back from cache to e.g. eMMC
- PCL instance of the application closed the POSIX file descriptor to the files
- PCL instance of the application removes backup and checksum file.

## 1.7. Persistence File Cache Configuration

A configuration file is used to configure the PFC.

The configuration file contains the following:

- Path to the RAM disk or tempfs.
  - Keyword *CachePath* followed by the path
- Path to the backup partition
  - Keyword *BackupPath* followed by the path
- Size of the cache for each application in bytes
  - The application ID followed by the size

### 1.7.1. Location

The configuration file must be located by default under

`/etc/persistence_pfc.conf`

The location can be changed by using the environment variable `PERS_PFC_CONF_PATH`.

### 1.7.2. Example

```
CachePath /tmp/mycache/  
BackupPath /Data/mnt-backup/  
NavigationApp 4096  
MediaApp 1024  
TunerApp 2048
```

**Note:**

The application id (e.g. NavigationApp) must be **unique** in the system and must match the `applID` parameter used for the `pfclnitCache` function.



## 2. Persistence File Cache Library Interfaces

### 2.1. Application Programming Interface Description

The Persistence File Cache Library provides a C-API for applications allowing to access files (open/read/write) which are cached.

The API has the following functions:

- `int pfcInitCache(const char* appID)`
- `int pfcOpenFile(const char* path)`
- `int pfcOpenFile(const char* path, PfcCachePriority_e cachePrio)`
- `int pfcCloseFile(int handle)`
- `int pfcReadFile(int handle, void *buf, size_t count)`
- `int pfcWriteFile(int handle, const void *buf, size_t nbyte)`
- `int pfcFileSeek(int handle, long int offset, int whence)`

## 2.2. API Details

### 2.2.1. PfcCacheAffinity

Enumerator used to pass the cache affinity to the pfcOpenFile.

The cache affinity is a hint to the cache removal algorithm among other indicators (like file size and last access time) which file to keep the longest in the cache and not to remove when the cache space is needed. The cache affinity is an indicator the user can use as input to the removal algorithm. The affinity is used when calling pfcOpenFile.

The highest cache affinity indicates a file to be later removed then others.

```
typedef enum _PfcCacheAffinity_e
{
    CacheAffLow           = 0,
    CacheAffBelowMedium,
    CacheAffMedium,
    CacheAffAboveMedium,
    CacheAffHighest

} PfcCacheAffinity;
```

### 2.2.2. pfcInitCache

This function initializes the cache component

#### Syntax

```
int pfcInitCache
(
    const char* appID
)
```

#### Parameter

appID

The applicationID must be unique in the system and must match at least one entry in the configuration file, see 1.5.

#### Return Value

A positive value (bigger than zero):

On error a negative value is returned with the following error codes:

### 2.2.3. pfcOpenFile

The open function returns a file descriptor; if a file does not exist the file is created with read write permission for the current user.

#### Syntax

```
Int pfcOpenFile
(
    const char* path,
)
int pfcOpenFile
(
    const char* path,
    PfcCachePriority_e cacheAffinity
)
```

#### Parameter

Path

The path to the file

PfcCachePriority\_e cacheAffinity

The cache affinity is used to

#### Return Value

Return positive value (bigger than zero): the handle

On error a negative value is returned with the following error codes:

#### 2.2.4. pfcCloseFile

The function pfcCloseFile closes a file descriptor, so that it no longer refers to any file and may be reused.

**Syntax**

```
int pfcCloseFile
(
    int handle
)
```

**Parameter**

Handle  
The file descriptor

**Return Value**

Return positive value (bigger than zero): the handle  
On error a negative value is returned with the following error codes:

#### 2.2.5. pfcReadFile

Read data from a file

**Syntax**

```
int pfcReadFile
(
    int handle,
    void *buf,
    size_t count
)
```

**Parameter**

int handle  
The file descriptor  
void \*buf  
The buffer to store the data  
size\_t count  
The size of the buffer

**Return Value**

Return positive value (bigger than zero): the handle  
On error a negative value is returned with the following error codes:

### 2.2.6. pfcWriteFile

Write data to the file

#### Syntax

```
int pfcWriteFile  
(  
    int handle,  
    const void *buf,  
    size_t nbyte  
)
```

#### Parameter

```
int handle  
    The file descriptor  
const void *buf  
    The buffer containing the data  
size_t nbyte  
    The number of data to write
```

#### Return Value

Return positive value (bigger than zero): the handle  
On error a negative value is returned with the following error codes:

### 2.2.7. pfcFileSeek

The file pfcFileSeek sets the file position indicator within a file

#### Syntax

```
int pfcFileSeek
(
    int handle
    long int offset
    int whence
)
```

#### Parameter

int handle

The file descriptor

long int offset

The offset in bytes for the file position indicator

int whence

The direction to reposition

SEEK\_SET

The offset is set to offset bytes.

SEEK\_CUR

The offset is set to its current location plus offset bytes.

SEEK\_END

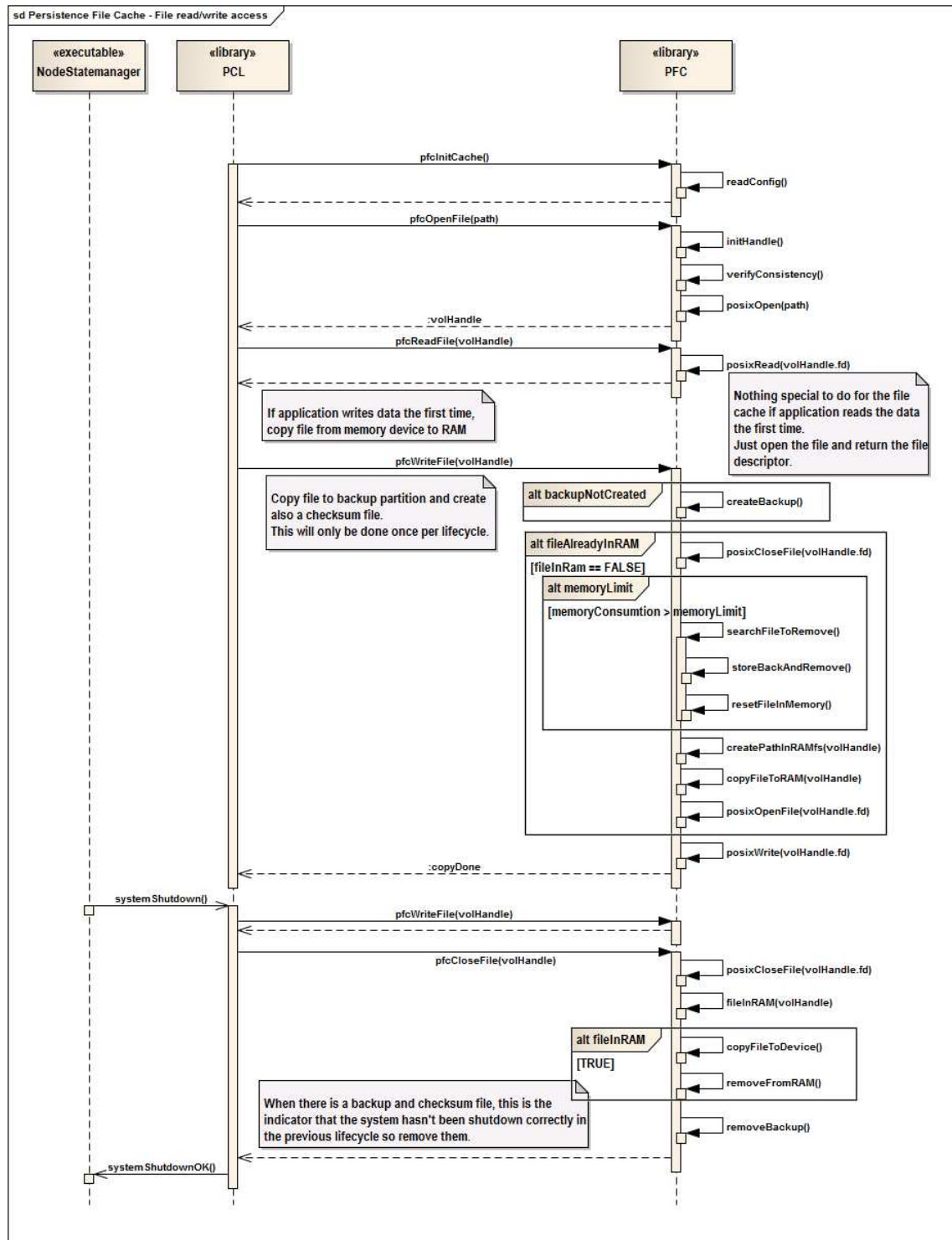
The offset is set to the size of the file plus offset bytes.

#### Return Value

Return positive value (0 or greater): resulting offset location;

On error, the value -1 is returned and errno is set to indicate the error.

## 3. Dynamic Behavior



## 4. How to build

The Persistence File Cache Library component uses automake to build the library. Execute the following steps in order to build the component:

- autoreconf -vi
- configure, with the following options
  - --enable-tests to enable the build of the tests
- Make

### 4.1. Dependencies

The persistence file cache library has the following dependencies

- Libraries
  - automotive-dlt
    - <http://projects.genivi.org/diagnostic-log-trace/>
  - check unit test framework for C
    - used when configured with "--enable-tests"
    - Ubuntu: use Synaptic package manager
    - Or download <http://check.sourceforge.net/>
- autotools
- libtool



## 5. Testing and Debugging

The test framework “check” has been used to write unit tests which are run automatically when the test binary is started. At the end a test report is printed to the console showing first a summary about number of tests that have been executed and how many tests have been passed or failed. After the summary a test report is generated showing the status of each test. When a bug is fixed a test is written to verify the problem has been solved.

DTL (Diagnostic, Log and Trace) is used by the PCL to send status and error. For details about DLT, please refer to the GENIVI DLT project page (<http://projects.genivi.org/diagnostic-log-trace/>).

### 5.1. Running the tests

There are unit tests available for the persistence file cache library component available. The unit tests are used to verify that the component is working correctly and exclude any regressions. Please refer always to the source code to see the available tests.

#### Run test

- run unit test `./persistence_file_cache_test`

#### Expected Result

The expected result is to have 0 failures and 0 errors, see example output below:

```
Running suite(s): Persistency file cache
100%: Checks: 3, Failures: 0, Errors: 0
persistence_file_cache_test.c:215:P:OpenFile:test_OpenFile:0: Passed
persistence_file_cache_test.c:304:P:WriteFile:test_WriteFile:0: Passed
persistence_file_cache_test.c:369:P:RemoveFromCache:test_RemoveFromCache:0:
Passed
```

The output above may vary as the test cases is adopted or extended.

## 6. Appendix

<b>Appendix A: Related Documents .....</b>	<b>18</b>
<b>Appendix B: Abbreviations .....</b>	<b>18</b>
<b>Appendix C: History.....</b>	<b>18</b>

## Appendix A: Related Documents

Document	Reference
Persistence_ArchitectureDocumentation.pdf	<a href="http://projects.genivi.org/persistence-client-library/documentation">http://projects.genivi.org/persistence-client-library/documentation</a>

Table 1 - Related Documents

## Appendix B: Abbreviations

Abbreviation	Description
PFC	Persistence File Cache
API	Application Programming Interface

Table 2 - Abbreviations

## Appendix C: History

Rev.	Status	Change	Date
V 0.1	Draft	Initial creation of the documentation	February 26 <sup>th</sup> , 2014
V 0.2	Draft	Added section "Backup and Recovery" and section "Caching Strategy"	April 1 <sup>th</sup> , 2014

Table 3 - History

# XSe

**XS Embedded GmbH**

Peterzeller Straße 8  
D-78048 Villingen-Schwenningen

Telefon +49 7721 4060-0  
Fax +49 7721 4060-499

[www.xse.de](http://www.xse.de)  
[info@xse.de](mailto:info@xse.de)

based.on.visions