GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	./				
File:	storage/blockdevice/source/HeapBlockDevice.cpp	Lines:	90	92	97.8 %
Date:	2021-05-06 12:39:05	Branches:	48	58	82.8 %

Line	Branch	Exec	Source
1			/* mbed Microcontroller Library
2			* Copyright (c) 2017 ARM Limited
3			* SPDX-License-Identifier: Apache-2.0
4			*
5			* Licensed under the Apache License, Version 2.0 (the "License");
6			* you may not use this file except in compliance with the License.
7			* You may obtain a copy of the License at
8			*
9			* http://www.apache.org/licenses/LICENSE-2.0
10			*
11			* Unless required by applicable law or agreed to in writing, software
12			* distributed under the License is distributed on an "AS IS" BASIS,
13			* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14			* See the License for the specific language governing permissions and
15			* limitations under the License.
16			*/
17			
18			#include "blockdevice/HeapBlockDevice.h"
19			#include "platform/mbed_atomic.h"
20			#include <stdlib.h>
21			#include <string.h>
22			
23			namespace mbed {
24			
25		36	HeapBlockDevice::HeapBlockDevice(bd_size_t size, bd_size_t block)
26			: _read_size(block), _program_size(block), _erase_size(block)
27		37	, _count(size / block), _blocks(0), _init_ref_count(0), _is_initialized(false)
28			{
29	✓✓✗✗	36	MBED_ASSERT(_count * _erase_size == size);
30		35	}

```
31
32      1  HeapBlockDevice::HeapBlockDevice(bd_size_t size, bd_size_t read, bd_size_t program, bd_size_t erase)
33             : _read_size(read), _program_size(program), _erase_size(erase)
34      1      , _count(size / erase), _blocks(0), _init_ref_count(0), _is_initialized(false)
35             {
36  ✗✓✗✗   1      MBED_ASSERT(_count * _erase_size == size);
37      1  }
38
39     72  HeapBlockDevice::~HeapBlockDevice()
40             {
41     ✓✓     36      if (_blocks) {
42     ✓✓   1696          for (size_t i = 0; i < _count; i++) {
43     ✓✓   1662              delete[] _blocks[i];
44                         }
45
46     ✓✗     34          delete[] _blocks;
47             34          _blocks = nullptr;
48                     }
49     36  }
50
51  10257  int HeapBlockDevice::init()
52             {
53  10257      uint32_t val = core_util_atomic_incr_u32(&_init_ref_count, 1);
54
55     ✓✓  10257      if (val != 1) {
56          10120          return BD_ERROR_OK;
57                     }
58
59     ✓✓    137      if (!_blocks) {
60     ✓✗     34          _blocks = new (std::nothrow) uint8_t *[_count];
61     ✗✓     34          if (!_blocks) {
62                             return BD_ERROR_DEVICE_ERROR;
63                         }
64
65     ✓✓   1696          for (size_t i = 0; i < _count; i++) {
66           1662              _blocks[i] = nullptr;
67                         }
68                     }
69
70    137      _is_initialized = true;
```

```
             137      return BD_ERROR_OK;
                  }

           10245  int HeapBlockDevice::deinit()
                  {
    ✓✓     10245      if (!_is_initialized) {
               1          return BD_ERROR_OK;
                      }

           10244      uint32_t val = core_util_atomic_decr_u32(&_init_ref_count, 1);

    ✓✓     10244      if (val) {
           10116          return BD_ERROR_OK;
                      }

    ✗✓       128      MBED_ASSERT(_blocks != NULL);
                      // Memory is lazily cleaned up in destructor to allow
                      // data to live across de/reinitialization
             128      _is_initialized = false;
             128      return BD_ERROR_OK;
                  }

         6317204  bd_size_t HeapBlockDevice::get_read_size() const
                  {
         6317204      return _read_size;
                  }

          143410  bd_size_t HeapBlockDevice::get_program_size() const
                  {
          143410      return _program_size;
                  }

              17  bd_size_t HeapBlockDevice::get_erase_size() const
                  {
              17      return _erase_size;
                  }

         2158453  bd_size_t HeapBlockDevice::get_erase_size(bd_addr_t addr) const
                  {
         2158453      return _erase_size;
```

```
                }
      1849205   bd_size_t HeapBlockDevice::size() const
                {
      1849205       return _count * _erase_size;
                }

      1729227   int HeapBlockDevice::read(void *b, bd_addr_t addr, bd_size_t size)
                {
  ✓✓  1729227       if (!_is_initialized) {
            1           return BD_ERROR_DEVICE_ERROR;
                    }
  ✓✓  1729226       if (!is_valid_read(addr, size)) {
            1           return BD_ERROR_DEVICE_ERROR;
                    }

      1729225       uint8_t *buffer = static_cast<uint8_t *>(b);

  ✓✓  5187675       while (size > 0) {
      1729225           bd_addr_t hi = addr / _erase_size;
      1729225           bd_addr_t lo = addr % _erase_size;

  ✓✓  1729225           if (_blocks[hi]) {
      1728600               memcpy(buffer, &_blocks[hi][lo], _read_size);
                        } else {
          625               memset(buffer, 0, _read_size);
                        }

      1729225           buffer += _read_size;
      1729225           addr += _read_size;
      1729225           size -= _read_size;
                    }

      1729225       return 0;
                }

        61481   int HeapBlockDevice::program(const void *b, bd_addr_t addr, bd_size_t size)
                {
  ✓✓    61481       if (!_is_initialized) {
            1           return BD_ERROR_DEVICE_ERROR;
```

```
                    }
    ✓✓   61480      if (!is_valid_program(addr, size)) {
             1          return BD_ERROR_DEVICE_ERROR;
                    }

         61479      const uint8_t *buffer = static_cast<const uint8_t *>(b);

    ✓✓  184437      while (size > 0) {
         61479          bd_addr_t hi = addr / _erase_size;
         61479          bd_addr_t lo = addr % _erase_size;

    ✓✓   61479          if (!_blocks[hi]) {
         29117              _blocks[hi] = new (std::nothrow) uint8_t[_erase_size];
    ✗✓   29117              if (!_blocks[hi]) {
                                return BD_ERROR_DEVICE_ERROR;
                            }
                        }

         61479          memcpy(&_blocks[hi][lo], buffer, _program_size);

         61479          buffer += _program_size;
         61479          addr += _program_size;
         61479          size -= _program_size;
                    }

         61479      return 0;
                }

         17126  int HeapBlockDevice::erase(bd_addr_t addr, bd_size_t size)
                {
    ✓✓   17126      if (!_is_initialized) {
             1          return BD_ERROR_DEVICE_ERROR;
                    }
    ✓✓   17125      if (!is_valid_erase(addr, size)) {
             1          return BD_ERROR_DEVICE_ERROR;
                    }

    ✓✓   46337      for (size_t i = 0; i < (size / _erase_size); i++) {
         29213          size_t index = addr / _erase_size + i;
    ✓✓   29213          delete[] _blocks[index];
```

Line	Branch	Exec	Source
191		29213	`        _blocks[index] = nullptr;`
192			`    }`
193			
194		17124	`    return 0;`
195			`}`
196			
197		3	`const char *HeapBlockDevice::get_type() const`
198			`{`
199		3	`    return "HEAP";`
200			`}`
201			
202			`} // namespace mbed`

Generated by: GCOVR (Version 4.2)