GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	./				
File:	storage/blockdevice/source/ExhaustibleBlockDevice.cpp	Lines:	80	91	87.9 %
Date:	2021-05-06 12:39:05	Branches:	42	52	80.8 %

```
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/ExhaustibleBlockDevice.h"
  19               #include "platform/mbed_atomic.h"
  20               #include "platform/mbed_assert.h"
  21
  22               namespace mbed {
  23
  24                  5  ExhaustibleBlockDevice::ExhaustibleBlockDevice(BlockDevice *bd, uint32_t erase_cycles)
  25                  5      : _bd(bd), _erase_array(NULL), _erase_cycles(erase_cycles), _init_ref_count(0), _is_initialized(false)
  26               {
  27                  5  }
  28
  29                 10  ExhaustibleBlockDevice::~ExhaustibleBlockDevice()
  30               {
```

```
✓✗    5      delete[] _erase_array;
      5  }

      5  int ExhaustibleBlockDevice::init()
         {
             int err;
      5      uint32_t val = core_util_atomic_incr_u32(&_init_ref_count, 1);

✗✓    5      if (val != 1) {
                 return BD_ERROR_OK;
             }

      5      err = _bd->init();
✗✓    5      if (err) {
                 goto fail;
             }

✓✗    5      if (!_erase_array) {
                 // can only be allocated after initialization
✓✗    5          _erase_array = new uint32_t[_bd->size() / _bd->get_erase_size()];
✓✓   10          for (size_t i = 0; i < _bd->size() / _bd->get_erase_size(); i++) {
      5              _erase_array[i] = _erase_cycles;
                 }
             }

      5      _is_initialized = true;
      5      return BD_ERROR_OK;

         fail:
             _is_initialized = false;
             _init_ref_count = 0;
             return err;
         }

      6  int ExhaustibleBlockDevice::deinit()
         {
✓✓    6      if (!_is_initialized) {
      1          return BD_ERROR_OK;
             }

```

```
        5       core_util_atomic_decr_u32(&_init_ref_count, 1);

   ✗✓   5       if (_init_ref_count) {
                    return BD_ERROR_OK;
                }

                // _erase_array is lazily cleaned up in destructor to allow
                // data to live across de/reinitialization
        5       _is_initialized = false;
        5       return _bd->deinit();
            }

        1   int ExhaustibleBlockDevice::sync()
            {
   ✓✗   1       if (!_is_initialized) {
        1           return BD_ERROR_DEVICE_ERROR;
                }

                return _bd->sync();
            }

        1   int ExhaustibleBlockDevice::read(void *buffer, bd_addr_t addr, bd_size_t size)
            {
   ✓✗   1       if (!_is_initialized) {
        1           return BD_ERROR_DEVICE_ERROR;
                }

                return _bd->read(buffer, addr, size);
            }

        4   int ExhaustibleBlockDevice::program(const void *buffer, bd_addr_t addr, bd_size_t size)
            {
   ✓✓   4       if (!_is_initialized) {
        1           return BD_ERROR_DEVICE_ERROR;
                }

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

```

```
✓✓  2      if (_erase_array[addr / get_erase_size()] == 0) {
    1          return 0;
           }

    1      return _bd->program(buffer, addr, size);
       }

    5  int ExhaustibleBlockDevice::erase(bd_addr_t addr, bd_size_t size)
       {
✓✓  5      if (!_is_initialized) {
    1          return BD_ERROR_DEVICE_ERROR;
           }

✓✓  4      if (!is_valid_erase(addr, size)) {
    1          return BD_ERROR_DEVICE_ERROR;
           }

    3      bd_size_t eu_size = get_erase_size();
✓✓  9      while (size) {
               // use an erase cycle
✓✓  3          if (_erase_array[addr / eu_size] > 0) {
    2              _erase_array[addr / eu_size] -= 1;
               }

✓✓  3          if (_erase_array[addr / eu_size] > 0) {
    1              int  err = _bd->erase(addr, eu_size);
✗✓  1              if (err) {
                       return err;
                   }
               }

    3          addr += eu_size;
    3          size -= eu_size;
           }

    3      return 0;
       }

    2  bd_size_t ExhaustibleBlockDevice::get_read_size() const
       {
```

```
✓✓   2      if (!_is_initialized) {
     1          return 0;
            }

     1      return _bd->get_read_size();
        }

     8  bd_size_t ExhaustibleBlockDevice::get_program_size() const
        {
✓✓   8      if (!_is_initialized) {
     1          return 0;
            }

     7      return _bd->get_program_size();
        }

     7  bd_size_t ExhaustibleBlockDevice::get_erase_size() const
        {
✓✓   7      if (!_is_initialized) {
     1          return 0;
            }

     6      return _bd->get_erase_size();
        }

    10  bd_size_t ExhaustibleBlockDevice::get_erase_size(bd_addr_t addr) const
        {
✓✓  10      if (!_is_initialized) {
     1          return 0;
            }

     9      return _bd->get_erase_size(addr);
        }

     2  int ExhaustibleBlockDevice::get_erase_value() const
        {
✓✓   2      if (!_is_initialized) {
     1          return BD_ERROR_DEVICE_ERROR;
            }

```

```
191         1      return _bd->get_erase_value();
192            }
193
194         7  bd_size_t ExhaustibleBlockDevice::size() const
195            {
196    ✓✓   7      if (!_is_initialized) {
197         1          return 0;
198                }
199
200         6      return _bd->size();
201            }
202
203         1  const char *ExhaustibleBlockDevice::get_type() const
204            {
205    ✓✗   1      if (_bd != NULL) {
206         1          return _bd->get_type();
207                }
208
209                return NULL;
210            }
211
212            } // namespace mbed
```

Generated by: GCOVR (Version 4.2)