| LCOV - code coverage report |
| --- |
|  |
| | Current view: | [top level](http://docs.google.com/index.html) - [ArbitraryModeFlow/src](http://docs.google.com/index.html) - ArbitraryModeCellProcessor.cpp (source / [functions](http://docs.google.com/ArbitraryModeCellProcessor.cpp.func.html)) |  |  | Hit | Total | Coverage | | --- | --- | --- | --- | --- | --- | --- | | Test: | EBM9K Converter Coverage Report |  | Lines: | 41 | 54 | 75.9 % | | Date: | 2012-03-23 |  | Functions: | 11 | 11 | 100.0 % | | Legend: | Lines: hit not hit | Branches: + taken - not taken # not executed |  | Branches: | 9 | 24 | 37.5 % | |  | |
|  |

|  |
| --- |
| Branch data Line data Source code  1 : : #include <boost/bind.hpp>  2 : : #include "SDDB.h"  3 : : #include "ArbitraryModeCellProcessor.h"  4 : : #include "ArbitraryModeProcessor.h"  5 : : #include "print.h"  6 : :   7 : : namespace nft {  8 : :   9 : : void  10 : : ArbitraryModeCellProcessor::scheduleCell(  11 : 63 : ArbitraryModeCellContainer::Pointer inCell) {  12 : 63 : mCells.scheduleCell( inCell );  13 : 63 : e9k\_debug("Cell added to thread\n");  14 : : }  15 : :   16 : : void  17 : 46 : ArbitraryModeCellProcessor::operator()() throw() {  18 : 109 : while(true)  19 : : {  20 : : {  21 : 109 : boost::mutex::scoped\_lock l(mMutex);  22 [ + - ][ + + ]: 109 : if( !mStop && mCells.isEmpty() )  23 : : {  24 : 90 : mEventOccured.wait(l);  25 : : }  26 [ + + ]: 109 : if( mStop )  27 : : {  28 : 46 : try  29 : : {  30 : 46 : SDDB::instance()->allCellProcessingCompleted();  31 : : }  32 [ # # ]: 0 : catch( DPL::Error \*pChild ) {  33 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  34 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  35 : : ErrorHandler::eErrFunction,  36 : : pErr->getErrorMsg(  37 : : ErrorHandler::eErrFunction,  38 : : "SDDB::allCellProcessingCompleted"),  39 : : getThreadManager());  40 : 0 : return;  41 : : }  42 : :   43 : 46 : return;  44 : : }  45 [ - + ]: 63 : if( mCells.isEmpty() )  46 : : {  47 : 0 : continue; //For handling spurious wakeups  48 : : }  49 : 63 : ArbitraryModeCellContainer::Pointer cell = mCells.getNextCell();  50 : 63 : try  51 : : {  52 : 63 : mShotDistributor->reset();  53 : 63 : cell->process( \*mPatternProcessor, \*mShotDistributor );  54 : 63 : mOutputGenerator->notifyCellProcessingCompletion( cell );   55 : 63 : try  56 : : {  57 : 63 : SDDB::instance()->cellProcessingCompleted();  58 : : }  59 [ # # ]: 0 : catch(DPL::Error \*pChild)  60 : : {  61 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  62 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  63 : : ErrorHandler::eErrFunction,  64 : : pErr->getErrorMsg( ErrorHandler::eErrFunction,  65 : : "SDDB::cellProcessingCompleted"),  66 : : getThreadManager());  67 : 0 : return;  68 : : }  69 : : }  70 [ # # ]: 0 : catch( DPL::Error \*pChild )  71 : : {  72 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  73 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  74 : : ErrorHandler::eErrFunction,  75 : : pErr->getErrorMsg(  76 : : ErrorHandler::eErrFunction,  77 : : "ArbitraryModeCellProcessor::process"),  78 : : getThreadManager());  79 : 0 : return;  80 : : }  81 : : }  82 [ + + ]: 63 : if( mCells.isEmpty() )  83 : : {  84 : 44 : e9k\_debug("All scheduled cells to thread %d processed .. notifying processor\n", getThreadID() );  85 [ # # ]: 90 : mScheduler->notifyProcessingCompletion();  86 : : }  87 : : }  88 : : }  89 : :   90 : : void  91 : 46 : ArbitraryModeCellProcessor::stop() {  92 : 46 : boost::mutex::scoped\_lock l(mMutex);  93 : 46 : mStop = true;  94 : 46 : mEventOccured.notify\_one();  95 : : }  96 : :   97 : : //This is used by scheduler to notify the cell processor, that all cells have been scheduled.  98 : : void  99 : 44 : ArbitraryModeCellProcessor::notifyCellSchedulingCompletion() {  100 : 44 : boost::mutex::scoped\_lock l(mMutex);  101 : 44 : e9k\_debug("Cell scheduling completed for this phase\n");  102 : 44 : mEventOccured.notify\_one();  103 : : }  104 : :   105 : : ArbitraryModeCellProcessor::ArbitraryModeCellProcessor(  106 : : ThreadID inThreadID, OutputGenerator::Pointer inOutputGenerator,  107 : : PatternProcessor \*inPatternProcessor,  108 : : ArbitraryModeShotDistributor \*inShotDistributor)  109 : 46 : : Thread(),  110 : : mMutex(),  111 : : mEventOccured(),  112 : : mStop(),  113 : : mPatternProcessor(inPatternProcessor),  114 : : mOutputGenerator(inOutputGenerator),  115 : : mShotDistributor(inShotDistributor),  116 : : mCells(),  117 : 92 : mScheduler() {  118 : 46 : setThreadID( inThreadID );  119 : : }  120 : :   121 : :   122 : 92 : ArbitraryModeCellProcessor::~ArbitraryModeCellProcessor() {  123 [ + - ][ # # ]: 46 : delete mPatternProcessor;  [ # # ]  124 : 46 : }  125 : :   126 : : void  127 : 46 : ArbitraryModeCellProcessor::setScheduler(ArbitraryModeProcessor \* value) {  128 : 46 : mScheduler = value;  129 : : }  130 : :   131 : :   132 : 1215 : } // namespace nft |

|  |
| --- |
| Generated by: [LCOV version 1.9](http://ltp.sourceforge.net/coverage/lcov.php) |