| LCOV - code coverage report |
| --- |
|  |
| | Current view: | [top level](http://docs.google.com/index.html) - [OfflineApp/src](http://docs.google.com/index.html) - App.cpp (source / [functions](http://docs.google.com/App.cpp.func.html)) |  |  | Hit | Total | Coverage | | --- | --- | --- | --- | --- | --- | --- | | Test: | EBM9K Converter Coverage Report |  | Lines: | 522 | 573 | 91.1 % | | Date: | 2012-03-23 |  | Functions: | 24 | 24 | 100.0 % | | Legend: | Lines: hit not hit | Branches: + taken - not taken # not executed |  | Branches: | 103 | 152 | 67.8 % | |  | |
|  |

|  |
| --- |
| Branch data Line data Source code  1 : : #include <iostream>  2 : : #include <cstdlib>  3 : : #include <cstring>  4 : : #include <strings.h>  5 : :   6 : : #include "print.h"  7 : : #include "AppParams.h"  8 : : #include "App.h"  9 : : #include "BinaryDataGenerator.h"  10 : : #include "FlagField.h"  11 : : #include "OptionsParser.h"  12 : : #include "ConverterParams.h"  13 : : #include "AppParams.h"  14 : : #include "ConfigFileDumper.h"  15 : : #include "PatternEdgeVectors.h"  16 : : #include "MemInfo.h"  17 : : #include "ConversionEngine.h"  18 : : #include "vchip.h"  19 : : #include "ShotDecompiler.h"  20 : : #include "Memory.h"  21 : : #include "LayerCalculator.h"  22 : : #include "Version.h"  23 : : #include "SChipCnfCreator.h"  24 : : #include "Logger.h"  25 : : #include "RetryManager.h"  26 : : #include "AreaAccumulator.h"  27 : : #include "WarningHandler.h"  28 : : #include "ErrorHandler.h"  29 : :   30 : : namespace {  31 : : void  32 : 2 : printHelp() {  33 : 2 : std::cout<<"Offline mode converter "  34 : : <<nft::gVERSION<<std::endl;  35 : 2 : std::cout<<" -f <filename> - Name of config file. Optional."<<std::endl;  36 : 2 : std::cout<<" DPManager reads converter.conf in present directory"  37 : : <<std::endl;  38 : 2 : std::cout<<" -single - Run only the specified DPB. Optional." <<std::endl;  39 : 2 : std::cout<<" Otherwise DPM runs from (sub)frames 1"<<std::endl  40 : : << " to the specified number"  41 : : <<std::endl;  42 : 2 : std::cout<<" -dpm <name> - Name of the DPManager process. Mandatory"  43 : : <<std::endl;  44 : 2 : std::cout<<" -conv <name> - Name of the Converter process. Mandatory"  45 : : <<std::endl;  46 : 2 : std::cout<<" -conv\_host <hostname> - Name of the Converter host machine name. Mandatory"  47 : : <<std::endl;  48 : 2 : std::cout<<" -log\_server <name> - Log Server Process Name. Mandatory"  49 : : <<std::endl;  50 : 2 : std::cout<<" -log\_level <1-3> - Log Level 1-3 (Most verbose). Mandatory"  51 : : <<std::endl;  52 : 2 : std::cout<<" -log\_file <filename> - Name of the log file. Mandatory"  53 : : <<std::endl;  54 : 2 : std::cout<<" -dump\_dir <directory> - Name of the directory for output data. Mandatory"  55 : : <<std::endl;  56 : 2 : std::cout<<" -last\_dpb\_in\_stripe - Set Stripe end flag in SDF-12i file. Optional"  57 : : <<std::endl;  58 : 2 : std::cout<<" -h - Print this message. Optional"  59 : : <<std::endl;  60 : 2 : std::cout<<" -help - Print this message. Optional"  61 : : <<std::endl;  62 : 2 : std::cout<<" -v - Print version number. Optional"  63 : : <<std::endl;  64 : 2 : std::cout<<" --version - Print version number. Optional"  65 : : <<std::endl;  66 : : return;  67 : : }  68 : :   69 : : void  70 : 1 : printVersion() {  71 : 1 : std::cout<<nft::gVERSION<<std::endl;  72 : : }  73 : :   74 : : }  75 : :   76 : : int main(int argc, char \*argv[])  77 : 1215 : {  78 : 2430 : nft::App obj;  79 : 1215 : obj.run(argc,argv);   80 : 1215 : return 0;  81 : 1215 : }  82 : :   83 : : return\_t  84 : 1215 : App::run(int argc, char \*argv[]){  85 : :   86 : 1215 : ConverterParams::Cleaner c;  87 : : //Will automatically cleanup params when function returns  88 : :   89 : 1215 : OptionsParser parser;  90 : 1215 : bool constructLayerFlag = false;  91 : :   92 [ + + ]: 1215 : if( !parser.parseCommandLine( argc, argv ) )  93 : : {  94 : 5 : e9k\_error("Error parsing command line\n");  95 : 5 : return eRet\_failure;  96 : : }  97 : : CommandLineParams \*pCmdlParams  98 : 1210 : = AppParams::instance()->getCmdlParams();  99 : :   100 : 1210 : if( pCmdlParams->getShortHelp() )  101 : : {  102 : 2 : printHelp();  103 : 2 : return eRet\_noaction;  104 : : }  105 : 1208 : if( pCmdlParams->getVersion() )  106 : : {  107 : 1 : printVersion();  108 : 1 : return eRet\_noaction;  109 : : }  110 : :   111 [ + + ]: 1207 : if( eRet\_failure == pCmdlParams->validate())  112 : : {  113 : 5 : e9k\_error("Incorrect Command Line options\n");  114 : 5 : return eRet\_failure;  115 : : }  116 : :   117 : 1202 : Logger::instance()->setCurrentLogLevel(  118 : : static\_cast<DPL::LOG\_MODE>(pCmdlParams->getLogLevel()));  119 : :   120 : 1202 : if( !parser.parseConverterConf(  121 : : pCmdlParams->getConfigFile()) )  122 : : {  123 : 77 : e9k\_error("Error parsing config file\n");  124 : 77 : return eRet\_failure;  125 : : }  126 : :   127 : 1125 : char \*pE9KPath = getenv("E9K\_PATH");  128 : :   129 [ - + ]: 1125 : if( NULL == pE9KPath )  130 : : {  131 : 0 : e9k\_error("The environment variable E9K\_PATH needs to be \n set to the path containing the E9K executables\n");  132 : 0 : return eRet\_failure;  133 : : }  134 : : else  135 : : {  136 : 1125 : setE9KPath( getenv("E9K\_PATH") );  137 : 1125 : e9k\_debug("Will pickup binaries from %s\n", getE9KPath().c\_str());  138 : : }  139 : :   140 : 1125 : uint32\_t maxStripes = (AppParams::instance()->getHeader()->uiStripeNo);  141 : 1125 : uint32\_t maxDpbNo = (AppParams::instance()->getHeader()->uiDpbNo);  142 : : bool processSingleOnly =   143 : 1125 : AppParams::instance()->getCmdlParams()->getSingleDPBOnly();  144 [ - + ]: 1125 : if( processSingleOnly  145 : : && !AppParams::instance()->getObjectCountInLayerPresent() )  146 : : {  147 : 0 : e9k\_error("SF and TF counts must be given when -single is provided\n");  148 : 0 : return eRet\_failure;  149 : : }  150 : 1125 : if( ( !AppParams::instance()->getObjectCountInLayerPresent()  151 : : || maxStripes > 1 )  152 : : && !processSingleOnly )  153 : 0 : constructLayerFlag = true;  154 : 1125 : ChipCnfFileParams \*chipCnf = ChipCnfFileParams::instance();  155 : 1125 : ConverterCommandLineOptions objConvCML;  156 : 1125 : size\_t numParams = 0;  157 : 1125 : char \*\*argV = constructConverterCommandLine(numParams);  158 : 1125 : objConvCML.parseCommandLine(numParams, argV );  159 : : const DPMANAGER\_MESSAGE\_CONVERTER \*pStartMsg  160 : 1125 : = AppParams::instance()->getFirstParam();  161 : : //Initialize RetryManager  162 : 1125 : RetryManager::instance()->init(pStartMsg->uiRetryConstantIntervalTime,  163 : : pStartMsg->uiRetryConstantCount,  164 : : pStartMsg->uiRetryMaxTime);  165 : :   166 : : //Deallocate the converter command line param.  167 [ + - ][ + + ]: 22500 : for(unsigned int i=0; i<numParams; i++){  168 : 21375 : nftFree(argV[i], true);  169 : : }  170 : 1125 : nftFree(argV, true);  171 : :   172 : 1125 : Logger::instance()->setLocalLogFileName(  173 : : pCmdlParams->getOfflineLogFile().c\_str());  174 : 1125 : try  175 : : {  176 : 1125 : Logger::instance()->openOfflineLogFile();  177 : : }  178 [ # # ]: 0 : catch(DPL::Error \*pChild)  179 : : {  180 : 0 : e9k\_error("Error open offline Log file\n");  181 : : }  182 : 1125 : Logger::instance()->setIsLocalLogging(true);  183 : :   184 : : std::string chipcnfFileName  185 : 1125 : = (AppParams::instance()->getSecondParamPaths()->getVsbDir()) + "/chip.cnf";  186 : 1125 : try  187 : : {  188 : 1131 : ChipCnfFileParams::instance()->parseChipCnf(chipcnfFileName);  189 : : }  190 [ + - ]: 6 : catch(DPL::Error \*pChild)  191 : : {  192 : 6 : e9k\_error("Error parsing chipcnf file\n");  193 : 6 : return eRet\_failure;  194 : : }  195 : 1119 : DPMANAGER\_MESSAGE\_CONVERTER\* msgConv = AppParams::instance()->getFirstParam();  196 : 1119 : msgConv->ullVchipSizeX = chipCnf->get\_llChipSizeX\_();  197 : 1119 : msgConv->ullVchipSizeY = chipCnf->get\_llChipSizeY\_();  198 : 1119 : msgConv->ullVchipBSizeX = chipCnf->get\_llChipBlkSizeX\_();  199 : 1119 : msgConv->ullVchipBSizeY = chipCnf->get\_llChipBlkSizeY\_();  200 : 1119 : e9k\_debug( "Frame Orientation: %s\n", chipCnf->get\_sFrmOrntn\_().c\_str() );  201 : 1119 : if( 0 == strcasecmp( chipCnf->get\_sFrmOrntn\_().c\_str(), "horizontal" ))  202 : : {  203 : 1118 : msgConv->usFrameOrientation = 1;  204 : : }  205 : : else  206 : : {  207 : 1 : msgConv->usFrameOrientation = 0;  208 : : }  209 : :   210 : 1119 : msgConv->ullCellMaxSizeX = chipCnf->get\_llCellMaxSizeX\_();  211 : 1119 : msgConv->ullCellMaxSizeY = chipCnf->get\_llCellMaxSizeY\_();  212 : 1119 : msgConv->ullArrCellMaxSizeX = chipCnf->get\_llCellArrMaxSizeX\_();   213 : 1119 : msgConv->ullArrCellMaxSizeY = chipCnf->get\_llCellArrMaxSizeY\_();  214 : 1119 : msgConv->ullDpbWidth = chipCnf->get\_llChipBlkSizeX\_()  215 : : \* chipCnf->get\_uiFrmWidth\_();  216 : 1119 : msgConv->ullDpbHeight = chipCnf->get\_llChipBlkSizeY\_()  217 : : \* chipCnf->get\_uiFrmHeight\_();  218 : 1119 : msgConv->uiMaxSFLID = msgConv->uiSTLCountInColumn  219 : : \* msgConv->uiSFLCountInColumn;  220 : 1119 : setParams();  221 : : //multiDPB support  222 : 1119 : e9k\_debug("Number of stripes= %u \n", maxStripes);  223 : : LayerCalculator objLayerCalculator(AppParams::instance()->getFirstParam(),  224 : 1119 : AppParams::instance()->getLayerParamList());  225 : :   226 : :   227 : : //Contructall DPB's and Populate all params to it  228 : 1119 : objLayerCalculator.calculateLayers(chipCnf);  229 : :   230 [ - + ][ + - ]: 2214 : for( uint32\_t inFrameNo = (processSingleOnly) ? maxStripes : 1;  [ - + ]  231 : : inFrameNo <= maxStripes; inFrameNo++ )  232 : : {  233 : : uint32\_t subFrameCount  234 : : = (processSingleOnly) ? maxDpbNo  235 [ - + ]: 1119 : : chipCnf->get\_frameSubframeCount( inFrameNo - 1);  236 : : //subFrameCount == 0 indicates no subframe  237 : : DPMANAGER\_MESSAGE\_CONVERTER \*pConvMsg  238 : : = const\_cast<DPMANAGER\_MESSAGE\_CONVERTER \*>(  239 : 1119 : ConverterParams::instance()->getStartupMessage());  240 : 1119 : uint32\_t subFrameNum = (subFrameCount > 0);  241 [ - + ]: 1119 : subFrameNum = (processSingleOnly) ? maxDpbNo : subFrameNum;  242 [ + - - + ]: 2214 : for( ; subFrameNum <= subFrameCount; subFrameNum++)  243 : : {  244 [ - + ]: 1119 : uint32\_t subFrameIdx = ( subFrameCount > 0 ) ? subFrameNum - 1 : 0;  245 [ + - ]: 1119 : if( !processSingleOnly )  246 : : {  247 : 1119 : pConvMsg->ullDpbWidth = chipCnf->get\_llChipBlkSizeX\_()  248 : : \* chipCnf->get\_frameWidth(  249 : : subFrameIdx, inFrameNo - 1 );  250 : 1119 : pConvMsg->ullDpbHeight = chipCnf->get\_llChipBlkSizeY\_()  251 : : \* chipCnf->get\_frameHeight(  252 : : subFrameIdx, inFrameNo - 1 );  253 : 1119 : pConvMsg->llDpbX = chipCnf->get\_frameLocX(  254 : : subFrameIdx, inFrameNo - 1 );  255 : 1119 : pConvMsg->llDpbY = chipCnf->get\_frameLocY(  256 : : subFrameIdx, inFrameNo - 1 );  257 : : }  258 : : else  259 : : {  260 : : //Already populated  261 : : }  262 : :   263 [ - + ]: 1119 : if(constructLayerFlag)  264 : 0 : objLayerCalculator.getNextLayerSet(  265 : : pConvMsg->llDpbX, pConvMsg->llDpbY,  266 : : AppParams::instance()->getLayerParamList());  267 : 1119 : setLayerParams();  268 : :   269 : : DPMANAGER\_MESSAGE\_HEADER \*pHeader  270 : 1119 : = ConverterParams::instance()->getHeader();  271 : :   272 : 1119 : pHeader->uiStripeNo = inFrameNo;  273 [ + - ]: 1119 : pHeader->uiDpbNo  274 : : = (0 == subFrameNum)?1:subFrameNum;  275 : 1119 : pHeader->uiColumnNo = AppParams::instance()->getHeader()->uiColumnNo;  276 [ + - ]: 1119 : pHeader->uiColumnNo = (0 == pHeader->uiColumnNo ) ? 1 : pHeader->uiColumnNo;  277 : 1119 : pHeader->ullDrawNo = AppParams::instance()->getHeader()->ullDrawNo;  278 [ + - ]: 1119 : pHeader->ullDrawNo = (0 == pHeader->ullDrawNo ) ? 1 : pHeader->ullDrawNo;  279 : :   280 : :   281 : 1119 : e9k\_debug("inFrameNo = %u, subFrameNum = %u", inFrameNo, subFrameNum);  282 : 1119 : e9k\_debug("Creating necessary Ref data\n");  283 [ + - ]: 1119 : if(eRet\_failure == createRefData(inFrameNo, subFrameNum)){  284 : 0 : e9k\_error("Failed in creating Ref data\n");  285 : 1215 : return eRet\_failure;  286 : : }  287 : :   288 : 1119 : e9k\_debug("Creating necessary Link data\n");  289 [ + - ]: 1119 : if(eRet\_failure == createLinkData(inFrameNo, subFrameNum)){  290 : 0 : e9k\_error("Failed in creating Link data\n");  291 : 0 : return eRet\_failure;  292 : : }  293 : :   294 : 1119 : e9k\_debug("Creating necessary Cell data\n");  295 [ + - ]: 1119 : if(eRet\_failure == createCellData(inFrameNo, subFrameNum)){  296 : 0 : e9k\_error("Failed in creating Cell data\n");  297 : 0 : return eRet\_failure;  298 : : }  299 : :   300 : : //Read Slope, dose, kbr gmc etc data  301 : : // Read div, doc table  302 : 1119 : sgs\_column\_t \*columnParams = AppParams::instance()->getSecondParams();  303 [ + + ]: 1119 : if( columnParams->prox.mode & 0x00000001 )  304 : : {  305 : 206 : e9k\_debug("Creating necessary Dose data\n");  306 [ + + ]: 206 : if(eRet\_failure == createDoseTbl()){  307 : 6 : e9k\_error("Failed in creating Dose Table\n");  308 : 6 : return eRet\_failure;  309 : : }  310 : :   311 : 200 : e9k\_debug("Creating necessary Slope data\n");  312 [ + + ]: 200 : if(eRet\_failure == createSlopeTbl()){  313 : 3 : e9k\_error("Failed in creating Slope Table\n");  314 : 3 : return eRet\_failure;  315 : : }  316 : : }  317 [ + + ]: 1110 : if( 1 == columnParams->kbr.mode )  318 : : {  319 : 1092 : e9k\_debug("Creating necessary KBR data\n");  320 [ + + ]: 1092 : if(eRet\_failure == createKBRGDCMap()){  321 : 3 : e9k\_error("Failed in creating KBRMap\n");  322 : 3 : return eRet\_failure;  323 : : }  324 : :   325 [ + + ]: 1089 : if(eRet\_failure == createKBRGDCTbl()){  326 : 3 : e9k\_error("Failed in creating KBRGMCTbl\n");  327 : 3 : return eRet\_failure;  328 : : }  329 : 1086 : e9k\_debug("KBR data populated in memory\n");  330 : : }   331 : :   332 [ + + ][ + + ]: 1104 : if( 0 != columnParams->gmc.mode   333 : : && (columnParams->gmc.mode & 0x00000002 ))  334 : : {  335 : 38 : e9k\_debug("Creating necessary Gmc data\n");  336 [ + + ]: 38 : if(eRet\_failure == createGMCMap()){  337 : 3 : e9k\_error("Failed in creating GMCMap\n");  338 : 3 : return eRet\_failure;  339 : : }  340 : 35 : e9k\_debug("GMC map data populated in memory\n");  341 : : }  342 : :   343 : 1101 : e9k\_debug("Creating necessary Div Table\n");  344 [ + + ]: 1101 : if(eRet\_failure == createDivTbl()){  345 : 3 : e9k\_error("ERR: Failed in creating Div Table\n");  346 : 3 : return eRet\_failure;  347 : : }  348 : 1098 : e9k\_debug("Div Table populated in memory\n");  349 : :   350 : 1098 : e9k\_debug("Creating necessary Doc Table\n");  351 [ + + ]: 1098 : if(eRet\_failure == createDocTbl()){  352 : 3 : e9k\_error("Failed in creating Doc Table\n");  353 : 3 : return eRet\_failure;  354 : : }  355 : 1095 : e9k\_debug("Doc Table populated in memory\n");  356 : :   357 [ + + ]: 1095 : if( columnParams->prox.mode & 0x00000008 )  358 : : {  359 : 26 : e9k\_debug("Creating necessary AI Table\n");  360 [ + - ]: 26 : if(eRet\_failure == createAITbl()){  361 : 0 : e9k\_error("Failed in creating AI Table\n");  362 : 0 : return eRet\_failure;  363 : : }  364 : 26 : e9k\_debug("AI Table populated in memory\n");  365 : : }  366 : :   367 : : uint16\_t flag = ConverterParams::instance()  368 : 1095 : ->getStartupMessage()->usFlag;  369 : 1095 : if( flag & getFlagMask( FLAG\_SETL\_ENABLE\_MAIN ) )  370 : : {  371 : 1094 : e9k\_debug("Creating necessary MainSetl Table\n");  372 [ + - ]: 1094 : if(eRet\_failure == createMainSetlTbl()){  373 : 0 : e9k\_error("ERR: Failed in creating MainSetl Table\n");  374 : 0 : return eRet\_failure;  375 : : }  376 : 1094 : e9k\_debug("MainSetl Table populated in memory\n");  377 : :   378 : : }  379 : : else  380 : : {  381 : 1 : Logger::instance()->log(LogMessageTable::eUTIL\_GENERAL\_MESSAGE,  382 : : "Main Settling Time map is OFF");  383 : : }  384 : 1095 : if( flag & getFlagMask( FLAG\_SETL\_ENABLE\_SUB ) )  385 : : {  386 : 1094 : e9k\_debug("Creating necessary SubSetl Table\n");  387 [ + - ]: 1094 : if(eRet\_failure == createSubSetlTbl()){  388 : 0 : e9k\_error("ERR: Failed in creating SubSetl Table\n");  389 : 0 : return eRet\_failure;  390 : : }  391 : 1094 : e9k\_debug("SubSetl Table populated in memory\n");  392 : : }  393 : : else  394 : : {  395 : 1 : Logger::instance()->log(LogMessageTable::eUTIL\_GENERAL\_MESSAGE,  396 : : "Sub Settling Time map is OFF");  397 : : }  398 : 1095 : if( flag & getFlagMask( FLAG\_SETL\_ENABLE\_TD ) )  399 : : {  400 : 1094 : e9k\_debug("Creating necessary TDSetl Table\n");  401 [ + - ]: 1094 : if(eRet\_failure == createTDSetlTbl()){  402 : 0 : e9k\_error("ERR: Failed in creating TDSetl Table\n");  403 : 0 : return eRet\_failure;  404 : : }  405 : 1094 : e9k\_debug("TDSetl Table populated in memory\n");  406 : : }  407 : : else  408 : : {  409 : 1 : Logger::instance()->log(LogMessageTable::eUTIL\_GENERAL\_MESSAGE,  410 : : "TD Settling Time map is OFF");  411 : : }  412 : :   413 : 1095 : if( flag & getFlagMask( FLAG\_SETL\_ENABLE\_SIZER ) )  414 : : {  415 : 1094 : e9k\_debug("Creating necessary SizerSetl Table\n");  416 [ + - ]: 1094 : if(eRet\_failure == createSizerSetlTbl()){  417 : 0 : e9k\_error("ERR: Failed in creating SizerSetl Table\n");  418 : 0 : return eRet\_failure;  419 : : }  420 : 1094 : e9k\_debug("SizerSetl Table populated in memory\n");  421 : : }  422 : : else  423 : : {  424 : 1 : Logger::instance()->log(LogMessageTable::eUTIL\_GENERAL\_MESSAGE,  425 : : "Sizer Settling Time map is OFF");  426 : : }  427 : :   428 : 1095 : ConversionEngine engine;  429 : 1095 : try{  430 [ + - ]: 1095 : e9k\_info("Now processing data for DPB ( %u, %u )\n",  431 : : inFrameNo, ( 0 == subFrameNum) ? 1 : subFrameNum);  432 : : //Eupa >>  433 : : //engine.convert();  434 : 1095 : engine.convert(  435 : : static\_cast<DPL::LOG\_MODE>(pCmdlParams->getLogLevel()));  436 : : //<< Eupa  437 : 1031 : if( AppParams::instance()->getCmdlParams()->getSetLastStripe() )  438 : : {  439 : 7 : engine.setLastStripe();  440 : : }  441 [ - + ]: 1031 : if( WarningHandler::instance()->getNumInstantiatedWarnings() )  442 : : {  443 : 0 : e9k\_info("Warnings detected .. see log file for details\n");  444 : : }  445 : 1031 : e9k\_debug("Processing completed.. now dumping data\n");  446 : 1031 : e9k\_debug("Pattern buffer memory usage %llu\n",  447 : : SimplePatternBuffer::MemSize);  448 : 1031 : MemInfo outputMem;  449 : 1031 : engine.getOutputMemory( outputMem );  450 : 1031 : if(outputMem.getMemSize())  451 : : {  452 : : {  453 : 1001 : std::ostringstream sout;  454 : 1001 : sout<<"mkdir -p "  455 : : <<AppParams::instance()->getCmdlParams()->getOutDir();  456 : 1001 : system( sout.str().c\_str() );  457 : : }  458 : :   459 : 1001 : std::ostringstream fileName;  460 : 1001 : fileName  461 : : <<AppParams::instance()->getCmdlParams()->getOutDir()  462 : : <<"/sdf."  463 : : <<ConverterParams::instance()->getHeader()->uiStripeNo;  464 : 1001 : if(0!=ConverterParams::instance()->getHeader()->uiDpbNo){  465 : 1001 : fileName<<"."  466 : : <<ConverterParams::instance()->getHeader()->uiDpbNo;  467 : : }  468 : : else  469 : : {  470 : 0 : fileName<<".1";  471 : : }  472 : 1001 : BinaryDataGenerator dataGen;  473 : : #ifdef DECOMPILE  474 : : ShotDecompiler decompiler;  475 : : decompiler.decompile( outputMem, std::cout, false);  476 : : #endif  477 : 1001 : dataGen.dump( outputMem, fileName.str(), true,  478 : : ConverterParams::instance()->getStartupMessage()  479 : : ->usFlag & getFlagMask( FLAG\_DUMP\_OUTPUT));  480 : 1001 : nftFree( outputMem.getMem(),true );  481 : :   482 : : {  483 : 1001 : SChipCnfCreator schipCnf;  484 : 1001 : std::ostringstream fileName;  485 : 1001 : fileName  486 : : <<AppParams::instance()->getCmdlParams()->getOutDir()  487 : : <<"/schip.cnf";  488 : 1001 : schipCnf.dump(AppParams::instance()->getFirstParam(),   489 : : AppParams::instance()->getSecondParams(),  490 : : fileName.str());  491 : 1001 : ConfigFileDumper dumper;  492 : 1001 : ConverterParams \*pParams = ConverterParams::instance();  493 : 1001 : dumper.dump(AppParams::instance()->getCmdlParams()->getOutDir(),   494 : : ConverterParams::instance()->getHeader(),  495 : : pParams->getStartupMessage(),  496 : : pParams->getLayerParams(),  497 : : pParams->getColumnParams());  498 : : }  499 : : }  500 : 1031 : int64\_t originX = pStartMsg->llDrawTimeCalcOriginX;  501 : 1031 : int64\_t pitch = pStartMsg->llDrawTimeCalcPitch;  502 : 1031 : uint32\_t numColumns = pStartMsg->uiNumRowDrawTimeCalc;  503 : 1031 : bool drawingTimeCalc = pStartMsg->uiDrawTimeCalcMode;  504 : :   505 : 1031 : size\_t timeDataSize = 0;  506 : 1031 : uint64\_t uiShiftOrigin = 0, uiEraseMargin = 0;  507 : 1031 : int i;  508 : : const DPMANAGER\_MESSAGE\_SHOT\_TIMEDATA \*pTimeData  509 : 1031 : = engine.getOutputDrawingTime();  510 : :   511 [ + + ][ + - ]: 1030 : if( pTimeData && drawingTimeCalc ){  512 : :   513 [ + + ][ + + ]: 8763 : for(i=0; i<static\_cast<int>(numColumns); i++){  514 [ + + ]: 8693 : if(pTimeData[i].ullSFTotalTime != 0)  515 : 7735 : break;  516 : 7735 : uiShiftOrigin++;  517 : : }  518 : :   519 [ + + ][ + + ]: 94055 : for(i=numColumns-1; i>=0; i--){  520 [ + + ]: 93985 : if(pTimeData[i].ullSFTotalTime != 0)  521 : 93027 : break;   522 : 93027 : uiEraseMargin++;  523 : : }  524 : :   525 : 1028 : originX += pitch \* uiShiftOrigin;  526 : 1028 : numColumns -= (uiShiftOrigin + uiEraseMargin);  527 : 1031 : timeDataSize = sizeof(DPMANAGER\_MESSAGE\_SHOT\_TIMEDATA)  528 : : \* numColumns;  529 : : //TODO::LOG origin shift  530 : : }  531 : 1031 : e9k\_debug("Drawing Time data -> Rows=%u Origin=%lld\n",  532 : : numColumns, originX );  533 : 1031 : AreaAccumulator \*pAreaAcc = AreaAccumulator::instance();  534 : 1031 : e9k\_info("SUMMARY: Shot Count=%llu TF Count=%llu SF Count=%llu\n",  535 : : pAreaAcc->getShotCount(),  536 : : pAreaAcc->getTFCount(),  537 : : pAreaAcc->getSFCount());  538 : 1031 : pAreaAcc->reset();  539 [ - + ]: 1031 : if( processSingleOnly )  540 : 1031 : return eRet\_success;  541 : : }  542 [ + - ]: 64 : catch(DPL::Error \*pErr)  543 : : {  544 : 64 : e9k\_error("Error occurred in E9K: %s\n", pErr->getAnyMessege());  545 : : }  546 : : }  547 : : }  548 : :   549 : : //Close Logfile  550 : 1095 : Logger::instance()->closeOfflineLogFile();  551 : :   552 : 1095 : return eRet\_success;  553 : : }  554 : :   555 : : void   556 : 1119 : App::setParams(){  557 : 1119 : AppParams \*appParam = AppParams::instance();  558 : 1119 : ConverterParams \*convParam = ConverterParams::instance();  559 : :   560 : : //AppParams::instance()->getSecondParamPaths()  561 : : DPMANAGER\_MESSAGE\_HEADER \*pHeader  562 : : = new DPMANAGER\_MESSAGE\_HEADER(  563 : 1119 : \*(appParam->getHeader()));  564 : 1119 : convParam->setHeader(pHeader);  565 : : DPMANAGER\_MESSAGE\_CONVERTER \*pMsg  566 : 1119 : = new DPMANAGER\_MESSAGE\_CONVERTER( \*(appParam->getFirstParam()));  567 : 1119 : convParam->setStartupMessage(pMsg);  568 : 1119 : convParam->setColumnParams(\*(appParam->getSecondParams()));  569 : : }  570 : : void  571 : 1119 : App::setLayerParams(){  572 : 1119 : AppParams \*appParam = AppParams::instance();  573 : 1119 : ConverterParams \*convParam = ConverterParams::instance();  574 : 1119 : uint32\_t numLayers = convParam->getStartupMessage()->uiSFLCountInDpb;  575 : : DPMANAGER\_MESSAGE\_SFL\_INFO \*p  576 : 1119 : = new DPMANAGER\_MESSAGE\_SFL\_INFO[ numLayers ];  577 : 1119 : const LAYERPARAMLIST &pList = \*(appParam->getLayerParamList());  578 [ + - ][ + + ]: 1188 : for( uint32\_t i = 0; i < numLayers; i++ )  579 : : {  580 : 1188 : \*(p + i) = \*(pList[i]);  581 : : //e9k\_debug("p.llSFLOriginX %ld, p.llSFLOriginY %ld, p.uiSFCountX %u, p.uiSFCountY %u, p.llTFOriginX %ld, p.llTFOriginY %ld, p.uiTFCountX %u, p.uiTFCountY %u", p[i].llSFLOriginX, p[i].llSFLOriginY, p[i].uiSFCountX, p[i].uiSFCountY, p[i].llTFOriginX, p[i].llTFOriginY, p[i].uiTFCountX, p[i].uiTFCountY);  582 : : }  583 : 1119 : convParam->setLayerParams( p, numLayers );  584 [ + - ]: 1119 : delete[] p;  585 : : }  586 : :   587 : : char \*\*  588 : : App::constructConverterCommandLine(size\_t &outNumParams)  589 : 1125 : {  590 : : CommandLineParams \*pCmdlParams  591 : 1125 : = AppParams::instance()->getCmdlParams();  592 : :   593 : 0 : AppParams \*appParam = AppParams::instance();  594 : 1125 : char tmpBuf[BUFSIZ];  595 : 1125 : char \*\*myArgv;  596 : 1125 : myArgv = (char\*\*)malloc(sizeof(char\*)\*20);  597 : 1125 : std::string sConvCLP;  598 : 1125 : size\_t numParams = 0;  599 : 1125 : myArgv[numParams] = strdup("converter-offline");  600 : 1125 : numParams++;  601 : 1125 : myArgv[numParams] = strdup(" -log\_level ");  602 : 1125 : numParams++;  603 : 1125 : sprintf( tmpBuf, "%u", pCmdlParams->getLogLevel());  604 : 1125 : myArgv[numParams] = strdup(tmpBuf);  605 : 1125 : numParams++;  606 : :   607 : 1125 : if(pCmdlParams->getOfflineLogFileFlag()){  608 : 1125 : myArgv[numParams] = strdup(" -log\_file ");  609 : 1125 : numParams++;  610 : 1125 : myArgv[numParams] = strdup(pCmdlParams->getOfflineLogFile().c\_str());  611 : 1125 : numParams++;  612 : : }  613 : 1125 : myArgv[numParams] = strdup(" -error\_file ");  614 : 1125 : numParams++;  615 : 1125 : myArgv[numParams] = strdup(" ./Error.txt ");  616 : 1125 : numParams++;  617 : 1125 : myArgv[numParams] = strdup(" -dump\_dir ");  618 : 1125 : numParams++;  619 : 1125 : myArgv[numParams] = strdup(pCmdlParams->getOutDir().c\_str());  620 : 1125 : numParams++;  621 : :   622 : 1125 : if(pCmdlParams->getDpmFlag()){  623 : 1125 : myArgv[numParams] = strdup(" -mng\_proc ");  624 : 1125 : numParams++;  625 : 1125 : myArgv[numParams] = strdup(pCmdlParams->getDpm().c\_str());  626 : 1125 : numParams++;  627 : 1125 : myArgv[numParams] = strdup(" -proc\_name ");  628 : 1125 : numParams++;  629 : 1125 : myArgv[numParams] = strdup(pCmdlParams->getConverter().c\_str());  630 : 1125 : numParams++;  631 : 1125 : myArgv[numParams] = strdup(" -distributor ");  632 : 1125 : numParams++;  633 : 1125 : myArgv[numParams] = strdup(pCmdlParams->getDpm().c\_str());  634 : 1125 : numParams++;  635 : 1125 : myArgv[numParams] = strdup(" -dosereconst ");  636 : 1125 : numParams++;  637 : 1125 : myArgv[numParams] = strdup(pCmdlParams->getDpm().c\_str());  638 : 1125 : numParams++;  639 : : }  640 : :   641 : 1125 : if(pCmdlParams->getLogServerFlag()){  642 : 1125 : myArgv[numParams] = strdup("-log\_server");  643 : 1125 : numParams++;  644 : 1125 : myArgv[numParams] = strdup(pCmdlParams->getLogServer().c\_str());  645 : 1125 : numParams++;  646 : : }  647 : 1125 : myArgv[numParams]= 0;  648 : 1125 : outNumParams = numParams;  649 : :   650 : 1125 : return myArgv;  651 : : }  652 : :   653 : : return\_t   654 : : App::createRefData( uint32\_t inFrameNo, uint32\_t inSubFrameNo )  655 : 1119 : {  656 : 1119 : char \*pRefData = NULL;  657 : : ConverterSecondParamPaths \*convSecondParamPath  658 : 1119 : = AppParams::instance()->getSecondParamPaths();  659 : 1119 : std::ostringstream fileNameOut;  660 : 1119 : fileNameOut<<convSecondParamPath->getVsbDir()  661 : : <<"/ref."<<inFrameNo;  662 [ - + ]: 1119 : if( 0 != inSubFrameNo )  663 : : {  664 : 0 : fileNameOut<<"."<<inSubFrameNo;  665 : : }  666 : 1119 : e9k\_debug("Loading ref file from %s\n",  667 : : fileNameOut.str().c\_str() );  668 : 1119 : try  669 : : {  670 : 1119 : pRefData = reinterpret\_cast<char \*>(  671 : : mRefFile.mapFileToMemory( fileNameOut.str().c\_str() ) );  672 : : }  673 [ # # ]: 0 : catch(DPL::Error \*pChild)  674 : : {  675 : 0 : pRefData = NULL;  676 : : }  677 [ - + ]: 1119 : if(!pRefData){  678 : 0 : e9k\_error("Could not load Ref Table ....\n");  679 : 0 : return eRet\_failure;  680 : : }  681 : :   682 : 1119 : size\_t uiRefMapSize = mRefFile.getMappedFileSize();   683 : 1119 : char \*pMem = new char[uiRefMapSize];  684 : 1119 : ::memcpy(pMem,pRefData,uiRefMapSize);  685 : :   686 : 1119 : MemInfo RefData(uiRefMapSize, pMem);  687 : 1119 : ConverterParams::instance()->setRefData(RefData);  688 : :   689 : 1119 : return eRet\_success;  690 : : }  691 : :   692 : : return\_t   693 : : App::createLinkData( uint32\_t inFrameNo, uint32\_t inSubFrameNo )  694 : 1119 : {  695 : 1119 : char \*pLinkData = NULL;  696 : : ConverterSecondParamPaths \*convSecondParamPath  697 : 1119 : = AppParams::instance()->getSecondParamPaths();  698 : 1119 : std::ostringstream fileNameOut;  699 : 1119 : fileNameOut<<convSecondParamPath->getVsbDir()  700 : : <<"/link."<<inFrameNo;  701 [ - + ]: 1119 : if( 0 != inSubFrameNo )  702 : : {  703 : 0 : fileNameOut<<"."<<inSubFrameNo;  704 : : }  705 : 1119 : e9k\_debug("Loading link file from %s\n",  706 : : fileNameOut.str().c\_str() );  707 : :   708 : 1119 : pLinkData = reinterpret\_cast<char \*>(  709 : : mLinkFile.mapFileToMemory( fileNameOut.str().c\_str() ) );  710 : :   711 [ - + ]: 1119 : if(!pLinkData){  712 : 0 : e9k\_error("Could not load Link Table ....\n");  713 : 0 : return eRet\_failure;  714 : : }  715 : 1119 : size\_t uiLinkMapSize = mLinkFile.getMappedFileSize();   716 : 1119 : char \*pMem = new char[uiLinkMapSize];  717 : 1119 : ::memcpy(pMem,pLinkData,uiLinkMapSize);  718 : :   719 : 1119 : MemInfo linkData(uiLinkMapSize, pMem);  720 : 1119 : ConverterParams::instance()->setLinkData(linkData);  721 : 1119 : return eRet\_success;  722 : : }  723 : :   724 : : return\_t   725 : : App::createCellData( uint32\_t inFrameNo, uint32\_t inSubFrameNo )  726 : 1119 : {  727 : 1119 : char \*pCellData = NULL;  728 : : ConverterSecondParamPaths \*convSecondParamPath  729 : 1119 : = AppParams::instance()->getSecondParamPaths();  730 : 1119 : std::ostringstream fileNameOut;  731 : 1119 : fileNameOut<<convSecondParamPath->getVsbDir()  732 : : <<"/cell."<<inFrameNo;  733 [ - + ]: 1119 : if( 0 != inSubFrameNo )  734 : : {  735 : 0 : fileNameOut<<"."<<inSubFrameNo;  736 : : }  737 : 1119 : e9k\_debug("Loading cell file from %s\n",  738 : : fileNameOut.str().c\_str() );  739 : :   740 : 1119 : pCellData = reinterpret\_cast<char \*>(  741 : : mCellFile.mapFileToMemory( fileNameOut.str().c\_str() ) );  742 : :   743 [ - + ]: 1119 : if(!pCellData){  744 : 0 : e9k\_error("Could not load Cell Table ....\n");  745 : 0 : return eRet\_failure;  746 : : }  747 : :   748 : 1119 : size\_t uiCellMapSize = mCellFile.getMappedFileSize();   749 : 1119 : char \*pMem = new char[uiCellMapSize];  750 : 1119 : ::memcpy(pMem,pCellData,uiCellMapSize);  751 : :   752 : 1119 : MemInfo cellData(uiCellMapSize, pMem);  753 : 1119 : ConverterParams::instance()->setCellData(cellData);  754 : :   755 : 1119 : return eRet\_success;  756 : : }  757 : :   758 : : return\_t   759 : : App::createAITbl()  760 : 26 : {  761 : 26 : char \*pAIData = NULL;  762 : : ConverterSecondParamPaths \*convSecondParamPath   763 : 26 : = AppParams::instance()->getSecondParamPaths();  764 : :   765 : 26 : pAIData = reinterpret\_cast<char \*>(mAIFile.mapFileToMemory(  766 : : convSecondParamPath->getAITableDir().c\_str()));  767 : :   768 [ - + ]: 26 : if(!pAIData){  769 : 0 : e9k\_error("Could not load AI Table ....\n");  770 : 0 : return eRet\_failure;  771 : : }  772 : :   773 : 26 : size\_t uiAIMapSize = mAIFile.getMappedFileSize();   774 : 26 : char \*pMem = new char[uiAIMapSize];  775 : 26 : ::memcpy(pMem,pAIData,uiAIMapSize);  776 : :   777 : 26 : MemInfo AIMap(uiAIMapSize, pMem);  778 : 26 : ConverterParams::instance()->setAITable(AIMap, false, NULL);  779 : :   780 : 26 : return eRet\_success;  781 : : }  782 : :   783 : : return\_t   784 : : App::createDoseTbl()  785 : 206 : {  786 : 206 : char \*pDoseData = NULL;  787 : : ConverterSecondParamPaths \*convSecondParamPath   788 : 206 : = AppParams::instance()->getSecondParamPaths();  789 : :   790 : 206 : pDoseData = reinterpret\_cast<char \*>(mDoseFile.mapFileToMemory(  791 : : convSecondParamPath->getDoseMapDir().c\_str()));  792 : :   793 [ + + ]: 206 : if(!pDoseData){  794 : 6 : e9k\_error("Could not load Dose Table ....\n");  795 : 6 : return eRet\_failure;  796 : : }  797 : :   798 : 200 : size\_t uiDoseMapSize = mDoseFile.getMappedFileSize();   799 : 200 : char \*pMem = new char[uiDoseMapSize];  800 : 200 : ::memcpy(pMem,pDoseData,uiDoseMapSize);  801 : :   802 : 200 : MemInfo doseMap(uiDoseMapSize, pMem);  803 : 200 : ConverterParams::instance()->setDoseTable(doseMap, false, NULL);  804 : :   805 : 206 : return eRet\_success;  806 : : }  807 : :   808 : : return\_t   809 : : App::createSlopeTbl()  810 : 200 : {  811 : 200 : char \*pSlope = NULL;  812 : 200 : ConverterSecondParamPaths \*convSecondParamPath = AppParams::instance()->getSecondParamPaths();  813 : 200 : pSlope = reinterpret\_cast<char \*>(mSlopeFile.mapFileToMemory(  814 : : convSecondParamPath->getSlopeMapDir().c\_str()) );  815 : :   816 [ + + ]: 200 : if(!pSlope){  817 : 3 : e9k\_error("Could not load Slope Table ....\n");  818 : 3 : return eRet\_failure;  819 : : }  820 : :   821 : 197 : size\_t uiSlopeMapSize = mSlopeFile.getMappedFileSize();   822 : 197 : char \*pMem = new char[uiSlopeMapSize];  823 : 197 : ::memcpy(pMem,pSlope,uiSlopeMapSize);  824 : :   825 : 197 : MemInfo slopeMap(uiSlopeMapSize, pMem);  826 : 197 : ConverterParams::instance()->setSlopeData(slopeMap, false, NULL);  827 : :   828 : 200 : return eRet\_success;  829 : : }  830 : :   831 : : return\_t   832 : : App::createKBRGDCMap()   833 : 1092 : {  834 : 1092 : char \*pKbrMap = NULL;  835 : 1092 : ConverterSecondParamPaths \*convSecondParamPath = AppParams::instance()->getSecondParamPaths();  836 : 0 : sgs\_column\_t \*sgsColParams = AppParams::instance()->getSecondParams() ;  837 : 1092 : sgs\_kbr\_param\_t &kbrParams = sgsColParams->kbr;  838 : :   839 : 1092 : pKbrMap = reinterpret\_cast<char \*>(mKbrMapFile.mapFileToMemory(  840 : : convSecondParamPath->getKbrMapDir().c\_str()) );  841 [ + + ]: 1092 : if(!pKbrMap){  842 : 3 : e9k\_error("Could not load KBRGDCMap ....\n");  843 : 3 : return eRet\_failure;  844 : : }  845 : 1089 : size\_t uiKBRMapSize = mKbrMapFile.getMappedFileSize();   846 : 1089 : char \*pMem = new char[uiKBRMapSize];  847 : 1089 : ::memcpy(pMem,pKbrMap,uiKBRMapSize);  848 : :   849 : 1089 : MemInfo kbrMap(uiKBRMapSize, pMem);  850 : : //TODO check the rows/cols  851 : 1089 : try{  852 : 1089 : ConverterParams::instance()->setKBRMap(kbrMap  853 : : , kbrParams.map.mesh\_x, kbrParams.map.mesh\_y, false, NULL);  854 : : }  855 [ # # ]: 0 : catch( DPL::Error \*pChild )  856 : : {  857 : : // ErrorHandler \*pErr = ErrorHandler::instance();  858 : : // pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  859 : : // ErrorHandler::eErrFunction,  860 : : // pErr->getErrorMsg(ErrorHandler::eErrFunction,  861 : : // "App::readKBRMap"));  862 : 0 : return eRet\_failure;  863 : : }  864 : :   865 : 1092 : return eRet\_success;  866 : : }  867 : :   868 : : return\_t   869 : : App::createKBRGDCTbl()  870 : 1089 : {  871 : 1089 : char \*pKbrTbl = NULL;  872 : 1089 : ConverterSecondParamPaths \*convSecondParamPath = AppParams::instance()->getSecondParamPaths();  873 : 0 : sgs\_column\_t \*sgsColParams = AppParams::instance()->getSecondParams() ;  874 : 1089 : sgs\_kbr\_param\_t &kbrParams = sgsColParams->kbr;  875 : :   876 : 1089 : pKbrTbl = reinterpret\_cast<char \*>(mKbrTblFile.mapFileToMemory(  877 : : convSecondParamPath->getKbrTblDir().c\_str()) );  878 [ + + ]: 1089 : if(!pKbrTbl){  879 : 3 : e9k\_error("Could not load KBRTbl ....\n");  880 : 3 : return eRet\_failure;  881 : : }  882 : :   883 : 1086 : size\_t uiKBRTblSize = mKbrTblFile.getMappedFileSize();  884 : 1086 : char \*pMem = new char[uiKBRTblSize];  885 : 1086 : ::memcpy(pMem,pKbrTbl,uiKBRTblSize);  886 : :   887 : 1086 : MemInfo kbrTbl(uiKBRTblSize, pMem);  888 : : //TODO check the rows/cols  889 : 1086 : ConverterParams::instance()->setKBRTable(kbrTbl  890 : : , kbrParams.map.mesh\_x, kbrParams.map.mesh\_y, false, NULL);  891 : :   892 : 1089 : return eRet\_success;  893 : : }  894 : :   895 : : return\_t   896 : : App::createGMCMap()  897 : 38 : {  898 : 38 : char \*pGMCMap = NULL;  899 : : const sgs\_column\_t \*columnParams  900 : 38 : = ConverterParams::instance()->getColumnParams();  901 : 38 : const sgs\_gmc\_param\_t &gmcParams = columnParams->gmc;  902 : :   903 : 38 : ConverterSecondParamPaths \*convSecondParamPath = AppParams::instance()->getSecondParamPaths();  904 : :   905 : 38 : pGMCMap = reinterpret\_cast<char \*>(mGmcMapFile.mapFileToMemory(  906 : : convSecondParamPath->getGmcMapDir().c\_str()) );  907 : :   908 [ + + ]: 38 : if(!pGMCMap){  909 : 3 : e9k\_error("Could not load KBRGMCMap ....\n");  910 : 3 : return eRet\_failure;  911 : : }  912 : 35 : size\_t uiGmcMapSize = mGmcMapFile.getMappedFileSize();  913 : 35 : char \*pMem = new char[uiGmcMapSize];  914 : 35 : ::memcpy(pMem,pGMCMap,uiGmcMapSize);  915 : :   916 : 35 : MemInfo gmcMap(uiGmcMapSize, pMem);  917 : : //TODO check the rows/cols  918 : 35 : ConverterParams::instance()->setGMCMap(gmcMap  919 : : , gmcParams.map.mesh\_x, gmcParams.map.mesh\_y, false, NULL);  920 : :   921 : 38 : return eRet\_success;  922 : : }   923 : :   924 : : return\_t   925 : : App::createDivTbl()  926 : 1101 : {  927 : 1101 : char \*pDivTbl = NULL;  928 : 1101 : ConverterSecondParamPaths \*convSecondParamPath = AppParams::instance()->getSecondParamPaths();  929 : :   930 : 1101 : pDivTbl = reinterpret\_cast<char \*>(mDivTblFile.mapFileToMemory(  931 : : convSecondParamPath->getDivTblDir().c\_str()));  932 : :   933 [ + + ]: 1101 : if(!pDivTbl){  934 : 3 : e9k\_error("Could not load Div Table to memory ....\n");  935 : 3 : return eRet\_failure;  936 : : }  937 : :   938 : 1098 : size\_t uiDivTblSize = mDivTblFile.getMappedFileSize();  939 : 1098 : char \*pMem = new char[uiDivTblSize];  940 : 1098 : ::memcpy(pMem,pDivTbl,uiDivTblSize);  941 : :   942 : 1098 : MemInfo divTbl(uiDivTblSize, pMem);  943 : 1098 : ConverterParams::instance()->setDIVTable(divTbl, false, NULL);  944 : :   945 : 1101 : return eRet\_success;  946 : : }  947 : :   948 : : return\_t   949 : : App::createDocTbl()  950 : 1098 : {  951 : 1098 : char \*pDocTbl = NULL;  952 : 1098 : ConverterSecondParamPaths \*convSecondParamPath = AppParams::instance()->getSecondParamPaths();  953 : :   954 : 1098 : pDocTbl = reinterpret\_cast<char \*>(mDocTblFile.mapFileToMemory(  955 : : convSecondParamPath->getDocTblDir().c\_str()));  956 : :   957 [ + + ]: 1098 : if(!pDocTbl){  958 : 3 : e9k\_error("Could not load DocTbl ....\n");  959 : 3 : return eRet\_failure;  960 : : }  961 : :   962 : 1095 : size\_t uiDocTblSize = mDocTblFile.getMappedFileSize();  963 : 1095 : char \*pMem = new char[uiDocTblSize];  964 : 1095 : ::memcpy(pMem,pDocTbl,uiDocTblSize);  965 : 1095 : MemInfo docTbl(uiDocTblSize, pMem);  966 : 1095 : ConverterParams::instance()->setDOCTable(docTbl, false, NULL);  967 : :   968 : 1098 : return eRet\_success;  969 : : }  970 : :   971 : : return\_t   972 : : App::createMainSetlTbl()  973 : 1094 : {  974 : 1094 : char \*pSetlTbl = NULL;  975 : : const sgs\_column\_t \*columnParams  976 : 1094 : = ConverterParams::instance()->getColumnParams();  977 : 1094 : const MAIN\_SETL &setlParams = columnParams->def.setl\_tbl.main;  978 : :   979 : : ConverterSecondParamPaths \*convSecondParamPath  980 : 1094 : = AppParams::instance()->getSecondParamPaths();  981 : :   982 : 1094 : pSetlTbl = reinterpret\_cast<char \*>(mMainSetlTblFile.mapFileToMemory(  983 : : convSecondParamPath->getMainSetlTbl().c\_str()) );  984 : :   985 [ - + ]: 1094 : if(!pSetlTbl){  986 : 0 : e9k\_error("Could not load Main Settling Table ....\n");  987 : 0 : return eRet\_failure;  988 : : }  989 : 1094 : size\_t uiSize = mMainSetlTblFile.getMappedFileSize();  990 : 1094 : char \*pMem = new char[uiSize];  991 : 1094 : ::memcpy(pMem,pSetlTbl,uiSize);  992 : :   993 : 1094 : MemInfo map(uiSize, pMem);  994 : : //TODO check the rows/cols  995 : 1094 : ConverterParams::instance()->setMainSetlTbl( map,  996 : : setlParams.setl\_main\_table.mesh\_x,  997 : : setlParams.setl\_main\_table.mesh\_y,  998 : : false, NULL);  999 : 1094 : return eRet\_success;  1000 : : }   1001 : :   1002 : : return\_t   1003 : : App::createSubSetlTbl()  1004 : 1094 : {  1005 : 1094 : char \*pSetlTbl = NULL;  1006 : : const sgs\_column\_t \*columnParams  1007 : 1094 : = ConverterParams::instance()->getColumnParams();  1008 : 1094 : const SUB\_SETL &setlParams = columnParams->def.setl\_tbl.sub;  1009 : :   1010 : : ConverterSecondParamPaths \*convSecondParamPath  1011 : 1094 : = AppParams::instance()->getSecondParamPaths();  1012 : :   1013 : 1094 : pSetlTbl = reinterpret\_cast<char \*>(mSubSetlTblFile.mapFileToMemory(  1014 : : convSecondParamPath->getSubSetlTbl().c\_str()) );  1015 : :   1016 [ - + ]: 1094 : if(!pSetlTbl){  1017 : 0 : e9k\_error("Could not load Sub Settling Table ....\n");  1018 : 0 : return eRet\_failure;  1019 : : }  1020 : 1094 : size\_t uiSize = mSubSetlTblFile.getMappedFileSize();  1021 : 1094 : char \*pMem = new char[uiSize];  1022 : 1094 : ::memcpy(pMem,pSetlTbl,uiSize);  1023 : :   1024 : 1094 : MemInfo map(uiSize, pMem);  1025 : : //TODO check the rows/cols  1026 : 1094 : ConverterParams::instance()->setSubSetlTbl( map,  1027 : : setlParams.setl\_sub\_table.mesh\_x,  1028 : : setlParams.setl\_sub\_table.mesh\_y,  1029 : : false, NULL);  1030 : 1094 : return eRet\_success;  1031 : : }  1032 : :   1033 : : return\_t   1034 : : App::createTDSetlTbl()  1035 : 1094 : {  1036 : 1094 : char \*pSetlTbl = NULL;  1037 : : const sgs\_column\_t \*columnParams  1038 : 1094 : = ConverterParams::instance()->getColumnParams();  1039 : 1094 : const TD\_SETL &setlParams = columnParams->def.setl\_tbl.td;  1040 : :   1041 : : ConverterSecondParamPaths \*convSecondParamPath  1042 : 1094 : = AppParams::instance()->getSecondParamPaths();  1043 : :   1044 : 1094 : pSetlTbl = reinterpret\_cast<char \*>(mTDSetlTblFile.mapFileToMemory(  1045 : : convSecondParamPath->getTDSetlTbl().c\_str()) );  1046 : :   1047 [ - + ]: 1094 : if(!pSetlTbl){  1048 : 0 : e9k\_error("Could not load TD Settling Table ....\n");  1049 : 0 : return eRet\_failure;  1050 : : }  1051 : 1094 : size\_t uiSize = mTDSetlTblFile.getMappedFileSize();  1052 : 1094 : char \*pMem = new char[uiSize];  1053 : 1094 : ::memcpy(pMem,pSetlTbl,uiSize);  1054 : :   1055 : 1094 : MemInfo map(uiSize, pMem);  1056 : : //TODO check the rows/cols  1057 : 1094 : ConverterParams::instance()->setTDSetlTbl( map,  1058 : : setlParams.setl\_td\_table.mesh\_x,  1059 : : setlParams.setl\_td\_table.mesh\_y,  1060 : : false, NULL);  1061 : 1094 : return eRet\_success;  1062 : : }  1063 : :   1064 : : return\_t   1065 : : App::createSizerSetlTbl()  1066 : 1094 : {  1067 : 1094 : char \*pSetlTbl = NULL;  1068 : : const sgs\_column\_t \*columnParams  1069 : 1094 : = ConverterParams::instance()->getColumnParams();  1070 : 1094 : const SIZER\_SETL &setlParams = columnParams->def.setl\_tbl.sizer;  1071 : :   1072 : : ConverterSecondParamPaths \*convSecondParamPath  1073 : 1094 : = AppParams::instance()->getSecondParamPaths();  1074 : :   1075 : 1094 : pSetlTbl = reinterpret\_cast<char \*>(mSizerSetlTblFile.mapFileToMemory(  1076 : : convSecondParamPath->getSizerSetlTbl().c\_str()) );  1077 : :   1078 [ - + ]: 1094 : if(!pSetlTbl){  1079 : 0 : e9k\_error("Could not load Sizer Settling Table ....\n");  1080 : 0 : return eRet\_failure;  1081 : : }  1082 : 1094 : size\_t uiSize = mSizerSetlTblFile.getMappedFileSize();  1083 : 1094 : char \*pMem = new char[uiSize];  1084 : 1094 : ::memcpy(pMem,pSetlTbl,uiSize);  1085 : :   1086 : 1094 : MemInfo map(uiSize, pMem);  1087 : : //TODO check the rows/cols  1088 : 1094 : ConverterParams::instance()->setSizerSetlTbl( map,  1089 : : setlParams.setl\_sizer\_table.mesh\_x,  1090 : : setlParams.setl\_sizer\_table.mesh\_y,  1091 : : false, NULL);  1092 : 1094 : return eRet\_success;  1093 : : }  1094 : :   1095 : : void   1096 : : App::setE9KPath( const std::string &value)  1097 : 1125 : {  1098 : 1125 : mE9KPath = value;  1099 : 1215 : }  1100 : 2430 : |

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