| LCOV - code coverage report |
| --- |
|  |
| | Current view: | [top level](http://docs.google.com/index.html) - [LibVsb/src](http://docs.google.com/index.html) - parse.cpp (source / [functions](http://docs.google.com/parse.cpp.func.html)) |  |  | Hit | Total | Coverage | | --- | --- | --- | --- | --- | --- | --- | | Test: | EBM9K Converter Coverage Report |  | Lines: | 43 | 234 | 18.4 % | | Date: | 2012-03-23 |  | Functions: | 4 | 15 | 26.7 % | | Legend: | Lines: hit not hit | Branches: + taken - not taken # not executed |  | Branches: | 36 | 156 | 23.1 % | |  | |
|  |

|  |
| --- |
| Branch data Line data Source code  1 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   2 : : \* Copyrighted 2007 by Interra Systems, USA  3 : : \* All rights reserved   4 : : \*  5 : : \* Date of Creation: Fri Jul 20 15:56:04 IST 2007  6 : : \* File Name: parse.cpp  7 : : \* Purpose:   8 : : \* Author:   9 : : \*  10 : : \* Revision History:   11 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/   12 : :   13 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  14 : : \* Header Include section   15 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  16 : : #include <stdio.h>  17 : : #include <string.h>  18 : : #include <stdlib.h>  19 : : #include <sys/types.h>  20 : : #include <ctype.h>  21 : : #include "parse.h"  22 : :   23 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  24 : :   25 : : \* Macro Definition section   26 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/   27 : : #define SUCCESS 0  28 : : #define FAILURE -1  29 : :   30 : :   31 : : static int checkEOS(\_docCont\* const ptrDocCont) ;  32 : :   33 : : /\*------------------------------------------------------------------------------  34 : : \*\* Open the input imaf12 script file and set the default values to  35 : : \*\* docCont structure.  36 : : \*\* <return>  37 : : \*\* 0 on sucess, -1 otherwise  38 : : \*\*----------------------------------------------------------------------------\*/  39 : : int openDoc(char const \* const fname, \_docCont\* const cont, char eos)  40 : 0 : {  41 : 0 : strcpy(cont->filename, fname) ;  42 : : //cont->fp = ext\_fopen(fname, "r") ;  43 : 0 : cont->fp = fopen(fname, "r") ;  44 [ # # ]: 0 : if(NULL == cont->fp)  45 : 0 : return FAILURE ;  46 : :   47 : 0 : cont->eos\_code = eos ;  48 : 0 : cont->offset = 0 ;  49 : 0 : cont->linenum = 0 ;  50 : 0 : strcpy(cont->line, "") ;  51 : 0 : return SUCCESS ;  52 : : }  53 : :   54 : : /\*------------------------------------------------------------------------------  55 : : \*\* Close the document  56 : : \*\*----------------------------------------------------------------------------\*/  57 : : int closeDoc(\_docCont \* const cont)  58 : 0 : {  59 [ # # ]: 0 : if(cont->fp)  60 : : {  61 : : //ext\_fclose(cont->fp) ;  62 : 0 : fclose(cont->fp) ;  63 : 0 : return SUCCESS ;  64 : : }  65 : 0 : return FAILURE ;  66 : : }  67 : :   68 : : /\*------------------------------------------------------------------------------  69 : : \*\* nextLine  70 : : \*\* read next uncommented line  71 : : \*\* <arg>  72 : : \*\* \_docCont \*cont ; document control block  73 : : \*\* <return>  74 : : \*\* size of the statement (byte)  75 : : \*\* = 0 end of the document  76 : : \*\* = -1 illegal statement/error reading  77 : : \*\* else number of chars in the line (after treaming)  78 : : \*\*----------------------------------------------------------------------------\*/  79 : : int nextLine( \_docCont\* const ptrDocCont, int const dbgLvl )  80 : 0 : {  81 : 0 : char \*nextln = NULL ;  82 : 0 : char \*comment = NULL ;  83 : 0 : int len = 0;  84 : :   85 : : //erase the contents of the ptrDocCont->line  86 : 0 : ptrDocCont->line[0] = '\0' ;  87 : :   88 : 0 : for(;;)  89 : : {  90 [ # # ]: 0 : len = strlen( ptrDocCont->line ) ;  91 : 0 : nextln = fgets( &(ptrDocCont->line[len]), LEN\_LINE, ptrDocCont->fp ) ;  92 : 0 : ptrDocCont->linenum++ ;  93 : :   94 : : #ifdef DEBUG   95 : : //fprintf( stdout, " %03d : %s", ptrDocCont->linenum, &(ptrDocCont->line[len])) ;  96 : : //fflush(stdout);  97 : : #endif   98 : :   99 : : // check for any error in file reading  100 [ # # ]: 0 : if( NULL == nextln )  101 : : {  102 [ # # ]: 0 : if ( feof( ptrDocCont->fp ))  103 : 0 : return 0 ;  104 [ # # ]: 0 : else if ( ferror( ptrDocCont->fp ))  105 : : {  106 : 0 : fprintf( stdout, "ERROR : nextLine:%s:%d: reading %s Line No = %d \n",  107 : : \_\_FILE\_\_, \_\_LINE\_\_, ptrDocCont->filename, ptrDocCont->linenum ) ;   108 : :   109 : 0 : return -1 ;  110 : : }  111 : : }  112 : :   113 : : // remove comments  114 [ # # ]: 0 : ( comment = strstr( &(ptrDocCont->line[len]), "//")) && (comment[0] = '\0' ) ;  115 [ # # ]: 0 : ( comment = strstr( &(ptrDocCont->line[len]), "#" )) && (comment[0] = '\0' ) ;  116 : :   117 : : // trim the recently read line  118 : 0 : trim( &( ptrDocCont->line[len] )) ;  119 : :   120 [ # # ]: 0 : len = strlen( ptrDocCont->line ) ;  121 : :   122 : : //--------------------------------------------------------------  123 : : // if there is a 'line joining' character at the end then  124 : : // we need to read the next line and join it with current line.  125 : : //--------------------------------------------------------------  126 [ # # ]: 0 : if ( ptrDocCont->line[len-1] == '\\' )  127 : : {  128 : 0 : ptrDocCont->line[len-1] = '\0' ; // remove the line joining character  129 : :   130 : 0 : trim( ptrDocCont->line ) ; // trim the line after removing line joining character  131 : :   132 : 0 : continue ;  133 : : }  134 [ # # ]: 0 : else if( len <= 1 ) // the line is blank line or only comments  135 : : {  136 : 0 : ptrDocCont->line[0] = '\0' ; // Erase ptrDocCont->line and read next line  137 : 0 : continue ;  138 : : }  139 : :   140 : : // check for End of Statement  141 : 0 : len = checkEOS(ptrDocCont) ;  142 : 0 : return len ;  143 : : }  144 : : }  145 : : /\*------------------------------------------------------------------------------  146 : : \*\* get\_tokens  147 : : \*\* takes a string and returns all the token delimited by the delimiter list  148 : : \*\* <return>  149 : : \*\* = Count of tokens On success  150 : : \*\* = -1 if error  151 : : \*\*----------------------------------------------------------------------------\*/  152 : : int get\_tokens( char \*statement, char\* dellst, char (\*value)[LEN\_LINE] )  153 : 541197 : {  154 : 541197 : int count = 1 ;  155 : 541197 : char \*tmpBuf = NULL;  156 : 541197 : int retVal = 0 ;  157 : :   158 : 541197 : value[0][0] = '\0' ;  159 : 541197 : value[1][0] = '\0' ;  160 : : // PR 13595  161 : 541197 : tmpBuf = new char[strlen(statement) + 1];  162 : 541197 : char \*pAlloc = tmpBuf;  163 : :   164 : 541197 : strcpy( tmpBuf, statement ) ;  165 : 541197 : trim( tmpBuf ) ;  166 : :   167 : : //PR 13560::Do not parse comment/blank line  168 : 541197 : size\_t lineLength = strlen(tmpBuf);  169 : :   170 [ + - ][ + - ]: 541197 : if ( lineLength &&   [ + - ][ + - ]  171 : : !(lineLength == 1 // Blank line  172 : : || (lineLength >= 2 && !strncmp(tmpBuf, "//", 2)) // Comment  173 : : )  174 : : )  175 : : {  176 : 541197 : char \*ti = NULL ;  177 : :   178 [ + - ]: 541197 : ( ti = strsep( (char \*\*)&tmpBuf, dellst)) && strcpy( value[0], ti ) ;  179 : 541197 : trim(value[0]) ;  180 : 541197 : strcpy(value[count], tmpBuf);  181 : 541197 : count++;  182 : 541197 : value[count][0] = '\0' ;   183 : 541197 : retVal = count ;  184 : :   185 : : }  186 : :   187 : : //strcpy( statement, tmpBuf ) ;  188 [ + - ]: 541197 : delete[] pAlloc;  189 : :   190 : 541197 : return retVal;  191 : : }  192 : :   193 : : /\*-----------------------------------------------------------------------------\*  194 : : \*\* checkEOS  195 : : \*\* The function checks for EOS.  196 : : \*\* The function return number of characters in the line. If the EOS statement  197 : : \*\* is required and the EOS is not found then -1 is returned to indicated   198 : : \*\* FAILURE condition.  199 : : \*\*----------------------------------------------------------------------------\*/  200 : : int checkEOS(\_docCont\* const ptrDocCont)  201 : 0 : {  202 [ # # ]: 0 : int strLen = strlen( ptrDocCont->line ) ;  203 : :   204 : : // No EOS check required  205 [ # # ]: 0 : if( '\0' == ptrDocCont->eos\_code )  206 : 0 : return strLen ;  207 : :   208 : : // EOS check required  209 [ # # ]: 0 : if ( ptrDocCont->eos\_code == ptrDocCont->line[strLen-1] )  210 : : {  211 : 0 : ptrDocCont->line[strLen-1] = '\0' ;  212 : 0 : trim(ptrDocCont->line) ;  213 [ # # ]: 0 : strLen = strlen( ptrDocCont->line ) ;  214 : : }  215 : : else  216 : : {  217 : 0 : fprintf( stdout, " Error : checkEOS:%s:%d: End of Statement delimiter '%c' is missing at line = %d \n",  218 : : \_\_FILE\_\_, \_\_LINE\_\_, ptrDocCont->eos\_code, ptrDocCont->linenum ) ;  219 : 0 : strLen = -1 ;  220 : 0 : }  221 : :   222 : : return strLen ;  223 : : }  224 : :   225 : : /\*-----------------------------------------------------------------------------\*  226 : : \*\* trim  227 : : \*\* <arg>  228 : : \*\* char \*s; sting to be trimmed  229 : : \*\* <output>  230 : : \*\* trimmed string  231 : : \*\*----------------------------------------------------------------------------\*/  232 : :   233 : : void trim( char\* s )  234 : 1092465 : {  235 : 2174859 : bool status ;  236 : 2174859 : for (;;)  237 : : {  238 : :   239 : 2174859 : status = false ;  240 : :   241 : : // Remove leading and trailing newlines  242 : 2174859 : status = status || removechar(s, '\n') ;  243 : :   244 : : // Remove leading and trailing blanks  245 [ + + ]: 2174859 : status = status || removechar(s, ' ') ;  246 : :   247 : : // Remove leading and trailing tabs  248 [ + + ]: 2174859 : status = status || removechar(s, '\t') ;  249 : :   250 [ + + ]: 2174859 : if ( !status ) break ;  251 : : }  252 : : }  253 : : /\*------------------------------------------------------------------------------  254 : : \*\* removechar  255 : : \*\* RETURN : True if one or more char has been removed else false  256 : : \*\*----------------------------------------------------------------------------\*/  257 : : bool removechar( char \*s, char c )  258 [ + + ][ + + ]: 6524577 : {  [ - + ]  259 : 4911057 : uint i=0;  260 : 4911057 : uint len = strlen(s) ;  261 : 4911057 : char buf[BUFSIZ];  262 : 4911057 : bool rem = false;  263 : 4911057 : int j=0;  264 [ + - ][ + + ]: 108914279 : for(i=0;i<len; i++)  [ + - ][ + + ]  [ + - ][ + + ]  [ + - ][ + + ]  [ # # ][ # # ]  265 : : {  266 [ + + ][ + + ]: 104003222 : if(c == s[i])  [ + + ][ - + ]  [ # # ]  267 : 1710927 : rem = true;  268 : : else  269 : : {  270 : 102292295 : buf[j] = s[i];  271 : 102292295 : j++;  272 : : }   273 : : }  274 : 4911057 : buf[j]='\0';  275 : 4911057 : strcpy(s,buf);  276 : 10071 : return rem;  277 : :   278 : : }  279 : : /\*------------------------------------------------------------------------------  280 : : \*\* goToLine  281 : : \*\* seeks to the required line  282 : : \*\* <arg>  283 : : \*\* n\_docCont\* const ptrDocCont : document control block  284 : : \*\* int const lineNo : input line number  285 : : \*\* <return>  286 : : \*\* 0 if success or  287 : : \*\* -1 if failure  288 : : \*\*----------------------------------------------------------------------------\*/  289 : : int goToLine( \_docCont\* const ptrDocCont, int const lineNo )  290 : 0 : {  291 : 0 : int numChar = 0;  292 : 0 : int retVal = 0 ;  293 : :   294 : : // first rewind to the beginning of the file  295 : : //ext\_fseek( ptrDocCont->fp, 0L, 0) ;  296 : 0 : fseek( ptrDocCont->fp, 0L, 0) ;  297 : :   298 : : // reset the line number  299 : 0 : ptrDocCont->linenum = 0 ;  300 : :   301 : : // go to the requierd line number  302 [ # # ][ # # ]: 0 : for( ; ptrDocCont->linenum != lineNo ; )  303 : : {  304 : 0 : numChar = nextLine( ptrDocCont, 0 ) ; // debug level = 0 (silent)  305 [ # # ]: 0 : if ( 1 >= numChar )  306 : : {  307 : 0 : fprintf( stdout, " ERROR : goToLine:%s:%d: Could not seek to the line %d.\n",  308 : : \_\_FILE\_\_, \_\_LINE\_\_, lineNo ) ;   309 : 0 : retVal = -1 ;  310 : 0 : break ;  311 : : }  312 : : }  313 : :   314 : 0 : return retVal ;  315 : : }  316 : : /\*------------------------------------------------------------------------------  317 : : \*\* populateItem  318 : : \*\* RETURN : 0 for success else failure  319 : : \*\*----------------------------------------------------------------------------\*/  320 : : int populateItem(\_parseRec \* parseTbl, char (\*value)[LEN\_LINE], int totTokens, bool bNoCheckForMaxEntries )  321 : 0 : {  322 : 0 : int retVal = -1 ;  323 : 0 : int i = 0 ;  324 : 0 : void\* ptrData = NULL;  325 : :   326 [ # # ]: 0 : while ( NULL != parseTbl[i].keyword)  327 : : {  328 [ # # ]: 0 : if( 0 == strcmp(value[0], parseTbl[i].keyword))  329 : : {  330 : 0 : retVal = 0 ;  331 : : break ;  332 : : }  333 [ # # ]: 0 : i++ ;  334 : : }  335 : :   336 : 0 : if (0 != retVal){  337 : 0 : fprintf( stdout, "WRN : populateItem:%s:%d: Unknown keyword \"%s\" found.\n",  338 : : \_\_FILE\_\_, \_\_LINE\_\_, value[0]) ;  339 : 0 : return SUCCESS ; // skip the unknown keyword  340 : : }  341 : :   342 [ # # ]: 0 : if (totTokens != parseTbl[i].reqTokens){  343 : 0 : fprintf( stdout, "ERROR : populateItem:%s:%d: Incorrect number of tokens found for keyword \"%s\" found.\n",  344 : : \_\_FILE\_\_, \_\_LINE\_\_, parseTbl[i].keyword) ;  345 : 0 : return FAILURE ; ;  346 : : }  347 : :   348 [ # # ][ # # ]: 0 : if (( bNoCheckForMaxEntries == false ) && parseTbl[i].numEntries >= parseTbl[i].reqEntries)  349 : : {  350 : 0 : fprintf( stdout, "ERROR : populateItem:%s:%d: Number of entries for keyword \"%s\" exceeded MAX limit.\n",  351 : : \_\_FILE\_\_, \_\_LINE\_\_, parseTbl[i].keyword) ;  352 : 0 : return FAILURE ; ;  353 : : }  354 : :   355 : 0 : ptrData = parseTbl[i].ptrData ;  356 : 0 : retVal = parseTbl[i].funcPtr(value[1], ( void\* ) &parseTbl[i]) ;  357 : 0 : parseTbl[i].numEntries += 1 ;  358 : :   359 : 0 : return (retVal) ;  360 : : }  361 : :   362 : : /\*------------------------------------------------------------------------------  363 : : \*\* strToInt8  364 : : \*\* RETURN : 0 for success else failure  365 : : \*\*----------------------------------------------------------------------------\*/  366 : : int strToInt8(char\* value, void\* rec)  367 : 0 : {  368 : 0 : int64\_t intVal = 0;  369 : 0 : int retVal = 0;  370 : 0 : \_parseRec\* parseRec = NULL;  371 : :   372 [ # # ]: 0 : if (NULL == value || NULL == rec )  373 : : {  374 : 0 : retVal = -1 ;  375 : : }  376 : : else  377 : : {  378 : 0 : parseRec = (\_parseRec \*) rec ;  379 : 0 : intVal = atoll(value) ;  380 : :   381 [ # # ]: 0 : if( parseRec->minVal > parseRec->maxVal)   382 : : {  383 : : // range check not required  384 : 0 : \*(int64\_t \*)parseRec->ptrData = intVal ;  385 : 0 : retVal = 0 ;  386 : : }  387 [ # # ][ # # ]: 0 : else if((intVal >= parseRec->minVal) && (intVal <= parseRec->maxVal))  388 : : {  389 : 0 : \*(int64\_t \*)parseRec->ptrData = intVal ;  390 : 0 : retVal = 0 ;  391 : : }  392 : : else  393 : : {  394 : 0 : fprintf( stdout, "ERROR : strToInt:%s:%d: Value out of range for \"%s\".\n",  395 : : \_\_FILE\_\_, \_\_LINE\_\_, parseRec->keyword) ;  396 : 0 : retVal = -1 ;  397 : : }  398 : : }  399 : 0 : return (retVal) ;  400 : : }  401 : :   402 : : /\*------------------------------------------------------------------------------  403 : : \*\* strToInt  404 : : \*\* RETURN : 0 for success else failure  405 : : \*\*----------------------------------------------------------------------------\*/  406 : : int strToInt(char\* value, void\* rec)  407 : 0 : {  408 : 0 : int intVal = 0;  409 : 0 : int retVal = 0;  410 : 0 : \_parseRec\* parseRec = NULL;  411 : :   412 [ # # ]: 0 : if (NULL == value || NULL == rec )  413 : : {  414 : 0 : retVal = -1 ;  415 : : }  416 : : else  417 : : {  418 : 0 : parseRec = (\_parseRec \*) rec ;  419 : 0 : intVal = atoll(value) ;  420 : :   421 [ # # ]: 0 : if( parseRec->minVal > parseRec->maxVal)   422 : : {  423 : : // range check not required  424 : 0 : \*(int \*)parseRec->ptrData = intVal ;  425 : 0 : retVal = 0 ;  426 : : }  427 [ # # ][ # # ]: 0 : else if((intVal >= parseRec->minVal) && (intVal <= parseRec->maxVal))  428 : : {  429 : 0 : \*(int \*)parseRec->ptrData = intVal ;  430 : 0 : retVal = 0 ;  431 : : }  432 : : else  433 : : {  434 : 0 : fprintf( stdout, "ERROR : strToInt:%s:%d: Value out of range for \"%s\".\n",  435 : : \_\_FILE\_\_, \_\_LINE\_\_, parseRec->keyword) ;  436 : 0 : retVal = -1 ;  437 : : }  438 : : }  439 : 0 : return (retVal) ;  440 : : }  441 : :   442 : : /\*------------------------------------------------------------------------------  443 : : \*\* strToStr  444 : : \*\* RETURN : 0 for success else failure  445 : : \*\*----------------------------------------------------------------------------\*/  446 : : int strToStr(char\* value, void\* rec)  447 : 0 : {  448 : 0 : int retVal = 0;  449 : 0 : \_parseRec\* parseRec = NULL;  450 : :   451 [ # # ]: 0 : if (NULL == value || NULL == rec )  452 : : {  453 : 0 : retVal = -1 ;  454 : : }  455 : : else  456 : : {  457 : : // removed quotes if present  458 : 0 : removechar(value, '\"') ;  459 : 0 : trim(value) ;  460 : 0 : parseRec = (\_parseRec \*) rec ;  461 : 0 : strncpy((char\*)parseRec->ptrData, value, LEN\_NAME) ;  462 : 0 : retVal = 0 ;  463 : : }  464 : 0 : return (retVal) ;  465 : : }  466 : : // Fix for PR 12536  467 : : /\*------------------------------------------------------------------------------  468 : : \*\* strHexToInt  469 : : \*\* RETURN : 0 for success else failure  470 : : \*\*----------------------------------------------------------------------------\*/  471 : : int strHexToInt(char\* value, void\* rec)  472 : 0 : {  473 : 0 : int intVal = 0;  474 : 0 : int retVal = 0;  475 : 0 : \_parseRec\* parseRec = NULL;  476 : :   477 [ # # ]: 0 : if (NULL == value || NULL == rec )  478 : : {  479 : : // TODO - error message  480 : 0 : retVal = -1 ;  481 : : }  482 : : else  483 : : {  484 : 0 : parseRec = (\_parseRec \*) rec ;  485 : 0 : intVal = (int )strtol(value, (char \*\*)NULL, 16) ;  486 : :   487 [ # # ]: 0 : if( parseRec->minVal > parseRec->maxVal)   488 : : {  489 : : // range check not required  490 : 0 : \*(int \*)parseRec->ptrData = intVal ;  491 : 0 : retVal = 0 ;  492 : : }  493 [ # # ][ # # ]: 0 : else if(intVal >= parseRec->minVal && intVal <= parseRec->maxVal)  494 : : {  495 : 0 : \*(int \*)parseRec->ptrData = intVal ;  496 : 0 : retVal = 0 ;  497 : : }  498 : : else  499 : : {  500 : 0 : fprintf( stdout, "ERROR : strHexToInt:%s:%d: Value out of range for \"%s\".\n",  501 : : \_\_FILE\_\_, \_\_LINE\_\_, parseRec->keyword) ;  502 : 0 : retVal = -1 ;  503 : : }  504 : : }  505 : 0 : return (retVal) ;  506 : : }  507 : :   508 : : /\*------------------------------------------------------------------------------  509 : : \*\* popOnOff  510 : : \*\* The function populates items whose value can be either "On" or "Off"  511 : : \*\*----------------------------------------------------------------------------\*/  512 : : int popOnOff(char\* value, void\* rec)  513 : 0 : {  514 : 0 : int retVal = 0;  515 : 0 : \_parseRec\* parseRec = NULL;  516 : :   517 [ # # ]: 0 : if (NULL == value || NULL == rec )  518 : : {  519 : : // TODO - error message  520 : 0 : retVal = -1 ;  521 : : }  522 : : else  523 : : {  524 : 0 : parseRec = (\_parseRec \*) rec ;  525 : :   526 [ # # ]: 0 : if (!strcmp(value, "on"))  527 : 0 : strncpy((char\*)parseRec->ptrData, value, LEN\_NAME) ;  528 [ # # ]: 0 : else if (!strcmp(value, "off"))  529 : 0 : strncpy((char\*)parseRec->ptrData, value, LEN\_NAME) ;  530 : : else  531 : : {  532 : 0 : fprintf( stdout, "ERROR : popOnOff:%s:%d: %s is not settable value for \"%s\".\n",  533 : : \_\_FILE\_\_, \_\_LINE\_\_, value, parseRec->keyword) ;  534 : 0 : retVal = -1 ;  535 : : }   536 : : }  537 : 0 : return (retVal) ;  538 : : }  539 : : /\*------------------------------------------------------------------------------  540 : : \*\* checkItems()  541 : : \*\* The function checks that every non-optional items must be populated from  542 : : \*\* input data (text file). It also generates warning for optional items if they  543 : : \*\* are using default values.  544 : : \*\* RETURN : 0 for success else failure  545 : : \*\*----------------------------------------------------------------------------\*/  546 : : int checkItems(\_parseRec \* parseTbl)  547 : 0 : {  548 : 0 : int retVal = 0 ;  549 : 0 : int i = 0 ;  550 : :   551 [ # # ]: 0 : while ( NULL != parseTbl[i].keyword)  552 : : {  553 [ # # ][ # # ]: 0 : if((1 == parseTbl[i].optional) && (0 == parseTbl[i].numEntries))  554 : : {  555 : : ;  556 : : // fprintf( stdout, "WRN : checkItems:%s:%d: Default value used for optional item \"%s\".\n",  557 : : // \_\_FILE\_\_, \_\_LINE\_\_, parseTbl[i].keyword) ;  558 : : }  559 [ # # ][ # # ]: 0 : else if((0 == parseTbl[i].optional) && (0 == parseTbl[i].numEntries))  560 : : {  561 : 0 : fprintf( stdout, " ERROR : checkItems:%s:%d: Value not defined for non-optional item \"%s\".\n",  562 : : \_\_FILE\_\_, \_\_LINE\_\_, parseTbl[i].keyword) ;  563 : 0 : retVal = -1 ;  564 : 0 : break ;  565 : : }  566 [ # # ]: 0 : i++ ;  567 : : }  568 : 0 : return retVal ;  569 : : }  570 : :   571 : : /\*------------------------------------------------------------------------------  572 : : \*\* parse()  573 : : \*\* Simple parser  574 : : \*\*----------------------------------------------------------------------------\*/  575 : :   576 : : int parse(char \*statement, char\* item, char (\*value)[256])  577 : 0 : {  578 : 0 : char\* comment = NULL ;  579 : 0 : int count = 0 ;  580 : :   581 : 0 : \*item = '\0' ;  582 : 0 : value[0][0] = '\0' ;  583 : :   584 : : //eat ..eol comments  585 [ # # ]: 0 : (comment = strstr(statement, "//")) && (comment[0] = '\0');  586 [ # # ]: 0 : (comment = strstr(statement, "#")) && (comment[0] = '\0');  587 : :   588 [ # # ]: 0 : if (strlen(statement))  589 : : {  590 : 0 : char \*ti;   591 : 0 : char \*tv;  592 : :   593 [ # # ]: 0 : (ti = strtok(statement, ":;=")) && strcpy(item, ti) ;  594 : 0 : trim(item) ;  595 : :   596 : : // while (tv=strtok(0, ":,=;"))  597 : 0 : while ((tv=strtok(0, ",;")))  598 : : {  599 : 0 : strcpy(value[count], tv) ;  600 : 0 : trim(value[count]) ;  601 [ # # ]: 0 : count++ ;  602 : : }  603 : :   604 : 0 : return 1 ;  605 : : }  606 : 0 : return 0 ; //pure comment line  607 : : } |

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