Contents

1

enum **objectID**

Different parsed object type identifiers

2

const int **MAX_LINE_LENGTH**

Maximum allowed line length in input file

3

enum **TCCparserState**

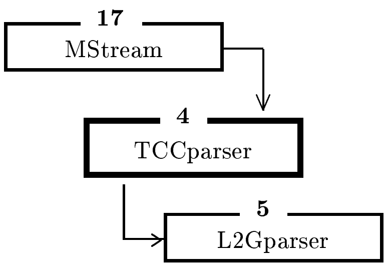
different TCC parser states

4

class **TCCparser** : public MStream

Class which parses the L2 global configuration file

Inheritance



Public Members

4.1		**TCCparser** (char *saddr, int size)	*Constructor to set up the MemoryStream*	6
4.2		**TCCparser** (char *saddr, char *eaddr)	*Constructor to set up the MemoryStream*	6
4.3	void	**getLine** (ParseString &line)	*Reads in a line from the current input file*	7
4.4	bool	**parseFile** (char *fname)	*Opens the given file and parses it*	7
4.5	void	**addParser** (ObjectParser *oparser)	*Adds the code to parse the given object*	7
4.6	string	**errorLine** (void)	*Returns a string containing the line number of the current input file*	8

Class which parses the L2 global configuration file. It reads the low level ASCII format and turns it into a binary file for downloading to the Alpha VME cards. The writing to memory is done through the MemoryStream class.

Author: Roger Moore (moore@pa.msu.edu)

4.1

TCCparser (char *saddr, int size)

Constructor to set up the MemoryStream

Constructor to set up the MemoryStream. The arguments are identical to those of the MStream class constructor which requires the start address and size of the available memory block. This constructor calls inherited MStream class constructor with the given arguments.

Parameters: `saddr` — start address of memory stream
`size` — size in bytes of ememory stream buffer

4.2

TCCparser (char *saddr, char *eaddr)

Constructor to set up the MemoryStream

Constructor to set up the MemoryStream. The arguments are identical to those of the MStream class constructor which requires the start and end addresses of the available memory block. This constructor calls inherited MStream class constructor with the given arguments.

Parameters: `saddr` — start address of memory stream
`eaddr` — end address of memory stream

4.3

void **getLine** (ParseString &line)

Reads in a line from the current input file

Reads in a line from the current input file. The routine checks that the current parser state is open and then reads in the next line using the getline method of the ifstream class. This character array is used to initialize the ParseString class and the lower and tidy methods of this class are called to format the line. Finally the line number of the input file is increased by one.

Parameters: `line` — parse string in which to return the next input line

4.4

bool **parseFile** (char *fname)

Opens the given file and parses it

Opens the given file and parses it. The routine loads the ASCII file line by line. Each line is then scanned for ”new

”$.Whenthispatternisfoundtheparsercallstheobjectparserassociatedwiththegivenstring.Thisobjectparseristhenresponsibleforreadingtheentire ASC$

4.5

void **addParser** (ObjectParser *oparser)

Adds the code to parse the given object

Adds the code to parse the given object. The name of the object and its parser are stored in the internal object map. This map is used to link the object type detected in the data with the correct parsing code for that object type.

Parameters:

`name` — type name of the object associated with this parser
`oparser` — parser class which can read in this type of object

4.6

string **errorLine** (void)

Returns a string containing the line number of the current input file

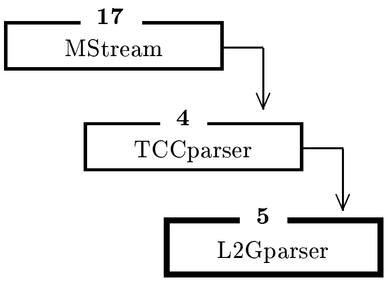
Returns a string containing the line number of the current input file. This routine is intended for use for printing error messages. When called it returns the string " at line XXXX", where XXXX is the current line number of the input file.

Return Value: string containing current input file line number

5 class **L2Gparser** : public TCCparser

Class which parses the L2 global configuration file

Inheritance



Public Members

5.1	**L2Gparser** (char *saddr, int size) *Constructor to set up the class*	9
5.2	**L2Gparser** (char *saddr, char *eaddr) *Constructor to set up the class*	10

Class which parses the L2 global configuration file. This class inherits the parsing functionality from the generic TCCparser class. The new constructor function creates the tool and script parser objects needed to parse the L2 global configuration files and then registers them with the parser.

Author: Roger Moore (moore@pa.msu.edu)

5.1 **L2Gparser** (char *saddr, int size)

Constructor to set up the class

Constructor to set up the class. This function calls the generic TCCparser constructor function and then creates and registers parsers for the tool and script objects in the input file for the L2 global processor.

Parameters: `saddr` — start address of memory stream
`size` — size in bytes of ememory stream buffer

5.2

L2Gparser (char *saddr, char *eaddr)

Constructor to set up the class

Constructor to set up the class. This function calls the generic TCCparser constructor function and then creates and registers parsers for the tool and script objects in the input file for the L2 global processor.

Parameters: `saddr` — start address of memory stream
`eaddr` — end address of memory stream

6

const int **HEADER_SIZE**

Number of bytes in the binary data object header.

7

const string **HEADER_NAME**

Name of the header field (in lower case)

8

const int **TRAILER_SIZE**

Number of bytes in the binary data object trailer.

9

const int **TRAILER_WORD**

Object trailer word which is used to confirm the end of an object

10

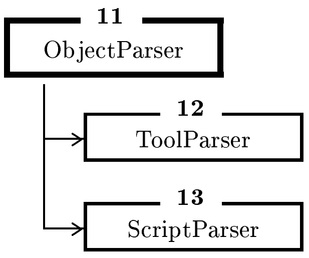
const string **NAME_NAME**

Name of the name field (in lower case)

11 class **ObjectParser**

Class which parses a given object in the L2 ASCII script file

Inheritance



Public Members

11.1		**ObjectParser** (string name, objectID id)	*The constructor function for the class*	17
11.2	bool	**parse** (TCCparser &parser)	*Parses the object from the given input stream*	17
	string	**getName** (void)	*Returns the name of the type of object parsed.*	

Protected Members

11.3	virtual bool	**check** (list<ObjectParam> &op)	*Checks the list of object parameters found*	18

Class which parses a given object in the L2 ASCII script file. It is called by the TCCparse class to read in an object and then right it out to the binary memory stream.

All subclasses which inherit from this class must be copy-safe since the TCCparser class makes a copy of the parser when a new parser is added to the class.

Author: Roger Moore (moore@pa.msu.edu)

11.1

ObjectParser (string name, objectID id)

The constructor function for the class

The constructor function for the class. The routine converts the object type name into lower case and then copies it into internal storage. The ID parameter is an integer saved in the binary output format which is used to identify the object type.

Parameters: `name` — name of object type for this parser
`id` — object type identification number

11.2

bool **parse** (TCCparser &parser)

Parses the object from the given input stream

Parses the object from the given input stream. The TCCparser class calls this function which will then read in the ASCII representation of the object and write it out to the given memory stream. The function reads in a list of parameters and then passes these to the virtual check function to ensure that all is well, before writing it out to the given parser class.

Parameters: `parser` — TCC parser which is reading the input file

11.3

virtual bool **check** (list<ObjectParam> &op)

Checks the list of object parameters found

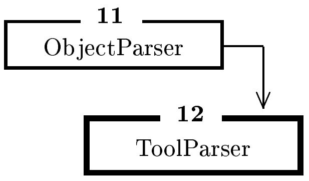
Checks the list of object parameters found. This function varies depending on the type of object being parsed. If it returns false then the ObjectParser will likewise return false to its calling routine.

Return Value:	true if no errors found, false otherwise
Parameters:	`op` — list of object parameters found in file

12 class **ToolParser** : public ObjectParser

Class which parses a tool for the L2G from the TCC script file

Inheritance



Public Members

12.1		**ToolParser** (void)	*The constructor function for the class*	19
12.2	bool	**check** (list<ObjectParam> &op)	*Checks the parameter list for the tool object*	20

Class which parses a tool for the L2G from the TCC script file. The class is called from the TCCparser class when it finds an object type of type "TOOL" in the input file. This object is then parser and the parameters checked by this class.

Author: Roger Moore (moore@pa.msu.edu)

12.1 **ToolParser** (void)

The constructor function for the class

The constructor function for the class. The routine calls the ObjectParser base class constructor function with "tool" as the object name.

12.2

bool **check** (list<ObjectParam> &op)

Checks the parameter list for the tool object

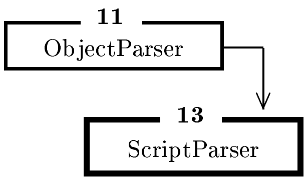
Checks the parameter list for the tool object. All tools must have the TYPE field as the first parameter and the INDEX field as their second paramter. If not this routine will stop the parser and print an error message.

Return Value:	true if no errors found, false otherwise
Parameters:	`op` — list of object parameters

13 class **ScriptParser** : public ObjectParser

Class which parses a script for the L2G from the TCC script file

Inheritance



Public Members

13.1		**ScriptParser** (void)	*The constructor function for the class*	21
13.2	bool	**check** (list<ObjectParam> &op)	*Checks the parameter list for the script object*	22

Class which parses a script for the L2G from the TCC script file. The class is called from the TCCparser class when it finds an object type of type "SCRIPT" in the input file. This object is then parser and the parameters checked by this class.

Author: Roger Moore (moore@pa.msu.edu)

13.1 **ScriptParser** (void)

The constructor function for the class

The constructor function for the class. The routine calls the ObjectParser base class constructor function with "script" as the object name.

13.2

bool **check** (list<ObjectParam> &op)

Checks the parameter list for the script object

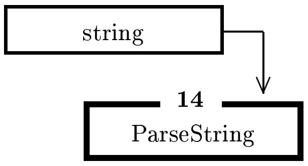
Checks the parameter list for the script object. All scripts must have the BIT field as the first parameter. If not this routine will stop the parser and print an error message.

Return Value:	true if no errors found, false otherwise
Parameters:	`op` — list of object parameters

14 class **ParseString** : public string

Subclass of string with added functionaity for parsing

Inheritance



Public Members

14.1		**ParseString** (char *value)	*Constructor taking a character array to initialize the string*	24
14.2		**ParseString** (void)	*Constructor taking no arguments* 24	
14.3	void	**tidy** (void)	*Removes preceeding and excess spaces*	24
14.4	void	**lower** (void)	*Converts the entire string to lower case letters*	25
14.5	string	**token** (int n, char delim=' ')	*Returns the given token in the string using the delimiter*	25

Subclass of string with added functionaity for parsing. The added functions are to simplify the task of parsing strings to look for specific keywords. The new routines can convert the string to lower case, tidy up the spacing and then split the string according to a given delimiter.

Author: Roger Moore (moore@pa.msu.edu)

14.1

ParseString (char *value)

Constructor taking a character array to initialize the string

Constructor taking a character array to initialize the string. The routine simply calls the base class string constructor function which takes a character array as an argument.

14.2

ParseString (void)

Constructor taking no arguments

Constructor taking no arguments. The routine simply calls the base class string constructor function which requires no arguments.

14.3

void **tidy** (void)

Removes preceeding and excess spaces

Removes preceeding and excess spaces. The routine loops over the string removing any spaces in front of the text. It then reduces spacing in the string to single space characters. For example the string " hello there" would become "hello there" after a call to tidy.

14.4

void **lower** (void)

Converts the entire string to lower case letters

Converts the entire string to lower case letters. Loops over the string elements and if they are capital letters it converts them to the lower case equivalent.

14.5

string **token** (int n, char delim=' ')

Returns the given token in the string using the delimiter

Returns the given token in the string using the delimiter. The routine loops over the string elements looking for the delimiter character. When it is found it will mark the beginning of the token and continue looping until it finds the next delimiter or end of the string which marks the end of the token. It then uses the substr method inherited from string to return the token substring.

Token zero starts at the begining of the string and ends at the first delimiter or end of the string.

Parameters:	`n` — number of token requested
	`delim` — delimiter character used to divide the string

15

enum **objptype**

Different object parameter types

16 class **ObjectParam**

Data for a single object parameter

Public Members

16.1		**ObjectParam** (string name, string value)	*Constructor function which fills the data fields* 27
	string	**name**	*name of the object parameter*
	string	**value**	*value of the paramter*
	objptype	**type**	*type of the parameter*

Data for a single object parameter. This class stores the data for one parameter of an object. The constructor also uses the value field to determine the type of the parameter and fills the type field accordingly.

Author: Roger Moore (moore@pa.msu.edu)

16.1 **ObjectParam** (string name, string value)

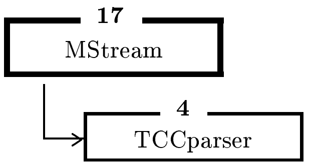
Constructor function which fills the data fields

Constructor function which fills the data fields. The type field is filled by examining the value field passed. If the value starts and ends with a double quote (") the parameter type is set to STRING and the quotes are removed. If a decimal point is found then the type is set to FLOAT, otherwise the type defaults to INT.

17 class **MStream**

Class which streams data into memory

Inheritance



Public Members

17.1		**MStream** (char *start, int size)	*Constructor which requires the maximum memory available*	29
17.2		**MStream** (char *start, char *end)	*Constructor which requires the start and end memory locations*	30
17.3	char*	**start** (void)	*Function which returns the first memory location of the stream*	30
17.4	char*	**end** (void)	*Function which returns the current internal pointer value*	30
17.5	MStream&	**operator<<** (int i)	*Operator to write an integer into the stream*	31
17.6	MStream&	**operator<<** (float f)	*Operator to write a float into the stream*	31
17.7	MStream&	**operator<<** (double d)	*Operator to write a double into the stream*	31
17.8	MStream&	**operator<<** (const char *s)	*Operator to write a constant null terminated string into the stream*	32
17.9	MStream&	**operator>>** (int &i)		

Class which streams data into memory.The class will only take int, float or string (char * only) variables and streams them into memory. Care is taken to ensure that all data is aligned on 4 byte boundaries since otherwise Alpha CPUs will produce unalign exception errors. Padding of strings is taken care of by the class so that the user need not worry about such things.

At initialization time the class will also perform a simple check on the byte order of the host machine. If this differs from the Alpha architecture standard then all integers and floats will be byteswapped to the Alpha standard before being stored in the memory buffer.

The class is written without inheritance from the STL because it must be capable of running online on the Alpha VME hardware. This also precludes use of dynamic memory allocation except.

Author: Roger Moore (moore@pa.msu.edu)

17.1 **MStream** (char *start, int size)

Constructor which requires the maximum memory available

Constructor which requires the maximum memory available. This constructs the stream in cases when the maximum available memory is known. The limits are checked everytime something is read from or written to the stream.

Parameters: `start` — memory location where the stream will start
`size` — available memory size in bytes

17.2

MStream (char *start, char *end)

Constructor which requires the start and end memory locations

Constructor which requires the start and end memory locations. This constructs the stream such that all the data will lie between the two locations given. Each time data is written to or from the stream these limits are checked.

Parameters:	`start` — memory location where the stream will start
	`end` — highest memory location the stream can use

17.3

char* **start** (void)

Function which returns the first memory location of the stream

Function which returns the first memory location of the stream. The function is a simple inlined call returning the value of the 'som' internal pointer.

Return Value:	pointer to the start of the stream's memory block

17.4

char* **end** (void)

Function which returns the current internal pointer value

Function which returns the current internal pointer value. The function is a simple inlined call returning the value of the 'ptr' internal pointer. When writing data to the stream this defines the end of the currently valid data and so can be used to find out which region of memory needs to be written to a file.

Return Value:	pointer to one byte beyond the end of the valid data

17.5

MStream& **operator<<** (int i)

Operator to write an integer into the stream

Operator to write an integer into the stream. If the host's byte order differs from that of an Alpha processor the bytes will be swapped before writing. If the stream size is exceeded then no data will be written.

Return Value:	reference to the memory stream being used
Parameters:	`i` — integer value to write into the stream

17.6

MStream& **operator<<** (float f)

Operator to write a float into the stream

Operator to write a float into the stream. If the host's byte order differs from that of an Alpha processor the bytes will be swapped before writing. If the stream size is exceeded then no data will be written.

Return Value:	reference to the memory stream being used
Parameters:	`f` — floating point value to write into the stream

17.7

MStream& **operator<<** (double d)

Operator to write a double into the stream

Operator to write a double into the stream. The routine casts the double as a float and then writes out the single precision value this will result in a loss of precision but does reduce the variable toa 4 byte quantity.

Return Value: reference to the memory stream being used
Parameters: `d` — variable to write into the stream as a float

17.8

MStream& **operator<<** (const char *s)

Operator to write a constant null terminated string into the stream

Operator to write a constant null terminated string into the stream. This operator will also pad the string out to the next 4-byte boundary. This is to ensure correct alignment for Alpha processors.

Return Value: reference to the memory stream being used
Parameters: `s` — pointer to the character string to write into the stream

17.9

MStream& **operator>>** (int &i)

Operator to read an integer from a memory stream

Operator to read an integer from a memory stream. This routine reads the data at the current pointer to the memory stream and returns it in the integer given. The pointer is also advanced 4 bytes.

Return Value: reference to the memory stream being used
Parameters: `i` — reference to integer variable to be read from the stream

17.10

MStream& **operator>>** (float &f)

Operator to read a float from a memory stream

Operator to read a float from a memory stream. The routine reads data pointed to by the internal stream pointer and returns this as the given floating point variable. The pointer is advanced by 4 bytes.

Return Value:	reference to the memory stream being used
Parameters:	`f` — reference to float variable to be read from the stream

17.11

MStream& **operator>>** (double &d)

Operator to read a double from a memory stream

Operator to read a double from a memory stream. The routine reads data pointed to by the internal stream pointer as a 4 byte floating point value. this 4 byte value is then cast into the given double precision variable and returned. The internal pointer is advanced by 4 bytes by this function.

Return Value:	reference to the memory stream being used
Parameters:	`d` — reference to double variable to be read from the stream

17.12

MStream& **operator>>** (char *s)

Operator to read an integer from a memory stream

Operator to read an integer from a memory stream. The routine reads data pointed to by the internal stream pointer and puts it into the given string array. The pointer is then advanced by the length of the string rounded to the next largest multiple of four. (This is to ensure correct int and float byte alignment for alpha processors.)

Return Value:	reference to the memory stream being used
Parameters:	`s` — pointer to character string to be read from the stream

Class Graph