Debugging with Cmajor Debugger

Seppo Laakko

June 12, 2014

1 Introduction

Cmajor programs can be debugged using the Cmajor source level debugger (cmdb). Currently the program to be debugged and libraries it uses must be first compiled with Cmajor compiler (cmc) using the "C backend" option (-backend=c) and the debug configuration with "generate debug information" option (-g) enabled. In Windows programs can be debugged directly in Cmajor Development Environment (IDE) that talks to cmdb behind the scenes. Cmajor debugger in turn talks to GNU debugger (gdb) behind the scenes.

2 Debugging Commands

Table 1 lists available debugging commands, their abbreviations and IDE equivalents.

Table 1: Debugging Commands

Command	**Abbreviation**	**IDE command**
start		
quit	q	`Stop Debugging`
help	h	`Help | Debugging`
continue	c	`Debug | Continue`
next	n	`Debug | Step Over`
step	s	`Debug | Step Into`
out	o	`Debug | Step Out`
break [FILE:]LINE	b [FILE:]LINE	`Debug | Toggle breakpoint`
clear N	cl N	`Debug | Toggle breakpoint`
callstack	ca	
frame N	f N	
list [FILE:]LINE	l [FILE:]LINE	
list*	l*	
list	l	
inspect EXPR	i EXPR	`Debug | Inspect...`
show breakpoints		
show libraries		
set debug library LIB (on / off)		
set break on throw (on / off)		
empty line	ENTER	

Descriptions of the commands:

- start

  Starts a debugging session.

- quit

  Ends debugging session and exits.

- continue

  Runs program until it stops to a breakpoint or exits.

- next

  Goes to next source line by stepping over function calls.

- step

  Goes to next source line or steps into a function.

- out

  Goes out of function.

- break [FILE:]LINE

  Sets a breakpoint to a source line.

- clear N

  Clears breakpoint number N.

- callstack

  Shows current call stack.

- frame N

  Sets current call frame for inspecting.

- list [FILE:]LINE

  Lists source code starting from line LINE.

- list*

  Lists source code around current position.

- list

  Lists next lines.

- inspect EXPR

  Inspects value of expression EXPR.

- show breakpoints

  Shows list of breakpoints.

- show libraries

  Shows debuggable libraries.

- set debug library LIB on

  Step command goes into functions in library LIB. Initially all libraries are enabled.

- set debug library LIB off

  Step command does not go into functions in library LIB.

- set break on throw on

  Sets implicit breakpoint in each throw statement.

- set break on throw off

  Clears implicit breakpoint in each throw statement.

- empty line

  Repeats last command.

3 Inspect Expressions

Inspect expression is an additive expression.

⟨*inspect-expr*⟩ ::= ⟨*additive-expr*⟩

3.1 Additive Expression

Additive expression consists of one or more prefix expressions separated by '+' or '-' operators.

⟨*additive-expr*⟩ ::= ⟨*prefix-expr*⟩ (`+` | `-`) ⟨*prefix-expr*⟩)*

3.2 Prefix Expression

A prefix expression consists of a prefix operator '*' or '#' followed by another prefix expression, or a postfix expression.

⟨*prefix-expr*⟩ ::= (`*` ⟨*prefix-expr*⟩)
| (`#` ⟨*prefix-expr*⟩)
| ⟨*postfix-expr*⟩

The '*' prefix operator derefences a following pointer expression.

The '#' prefix operator can be applied to a pointer, a class object or a System Library container. When applied to a pointer and the pointer points to a class object, the non-static member variables of that class are listed. When applied to a class object, the non-static member variables of that class are listed. When applied to a container, the objects contained are listed. Currently the container can be a *System.Collections.List*, *System.Collections.Set* or a *System.Collections.Map*.

3.3 Postfix Expression

A postfix expression consists of a primary expression followed by zero or more postfix operations.

⟨*postfix-expr*⟩ ::= ⟨*primary-expr*⟩ (`.` ⟨*identifier*⟩ | `->` ⟨*identifier*⟩ | `[` ⟨*int*⟩ `]`)*

A '.' postfix operation selects a member of a class.

A '− >' postfix operation selects a member of an expression that has a type of a pointer to class.

A '[' ']' postfix operation selects an item of *System.Collections.List*.

3.4 Primary Expression

A primary expression is either a cast expression, this-pointer, a name of a local variable, an integer, a handle expression or a parenthesized inspect expression.

⟨*primary-expr*⟩ ::= ⟨*cast-expr*⟩ | `this` | ⟨*identifier*⟩ | ⟨*int*⟩ | `$` ⟨*int*⟩ | `(` ⟨*inspect-expr*⟩ `)`

⟨*cast-expr*⟩ ::= `cast` `<` ⟨*typename*⟩ `>` '(' ⟨*inspect-expr*⟩ ')'

⟨*typename*⟩ ::= ⟨*qualified-id*⟩ `*`?

⟨*qualified-id*⟩ ::= ⟨*identifier*⟩ ('.' ⟨*identifier*⟩)*

⟨*identifier*⟩ ::= (letter | '_') (letter | digit | '_')*

A handle expression consists of a '$' character followed by a handle returned by GDB. A contents of a list, set or map can be inspected using handle expressions.

4 Usage

```
usage: cmdb [options] program [arguments...]
options:
-------------------------------------------------------
-showgdb:          show conversation with gdb
-singlestep:       debug using single step mode
-devenv:           don't print source lines
-listcount=N:      set list command line count to N
-log <filename>:   write execution log to file <filename>
```

Running program in single step mode is very slow (maybe hundreds of times slower than running in continuous execution mode). It may still be useful in some situations because in single step mode breakpoints can be set while the program is running.