Appendix A Framework Class Declarations

This appendix contains the framework's C++ class declarations. Anyone attempting to write a simulator should look them over.

BasicCPU Class Declaration

```
 1  ////////////////////////////////////////////////////////////////////////////
 2  // $Id: BasicCPU.hxx,v 1.1 1994/02/18 19:47:58 bmott Exp $
 3  ////////////////////////////////////////////////////////////////////////////
 4  // BasicCPU.hxx
 5  //
 6  // This is the abstract base class for all microprocessors (CPU)
 7  //
 8  //
 9  // BSVC "A Microprocessor Simulation Framework"
10  // Copyright (c) 1993
11  // By: Bradford W. Mott
12  // June 27,1993
13  //
14  ////////////////////////////////////////////////////////////////////////////
15  // $Log: BasicCPU.hxx,v $
16  // Revision 1.1  1994/02/18  19:47:58  bmott
17  // Initial revision
18  //
19  ////////////////////////////////////////////////////////////////////////////
20
21  #ifndef BASICCPU_HXX
22  #define BASICCPU_HXX
23
24  #include "String.h"
25
26  class BasicCPU;
27
28  #include "AddressSpace.hxx"
29  #include "Event.hxx"
30  #include "RegInfo.hxx"
```

```
 31  #include "StatInfo.hxx"
 32
 33  class RegisterInformationList;
 34  class StatisticalInformationList;
 35
 36  ////////////////////////////////////////////////////////////////////////////
 37  // BasicCPU class declaration
 38  ////////////////////////////////////////////////////////////////////////////
 39  class BasicCPU {
 40    private:
 41      // Name of the CPU
 42      const char *name;
 43
 44      // Number of address spaces in the CPU
 45      const int number_of_address_spaces;
 46
 47      // Granularity of the CPU in bytes
 48      const int granularity;
 49
 50      // Record format returned from the ExecuteInstruction function
 51      const char *execution_trace_record;
 52
 53      // Default fields of the trace record that should be displayed by UI
 54      const char *default_execution_trace_entries;
 55
 56    public:
 57      // CPU's Event Handler
 58      EventHandler  events;
 59
 60      // Pointer to the array of address space objects
 61      AddressSpace  *address_space;
 62
 63      BasicCPU(const char* n, const int g, const int a,
 64               AddressSpace *addr, const char* trace,
 65               const char* default_trace)
 66          : name(n),
 67            granularity(g),
 68            number_of_address_spaces(a),
 69            address_space(addr),
 70            execution_trace_record(trace),
 71            default_execution_trace_entries(default_trace)
 72      {};
 73
 74      // Return the name of the microprocessor
 75      inline const char *Name()
 76      { return(name); }
 77
 78      // Return the granularity of the microprocessor
 79      inline int Granularity()
 80      { return(granularity); }
 81
 82      // Return the number of address spaces used by the processor
 83      inline const int NumberOfAddressSpaces()
 84      { return(number_of_address_spaces); }
 85
 86      // Return the execution trace record
 87      inline const char* ExecutionTraceRecord()
 88      { return(execution_trace_record); }
 89
 90      // Return the default execution trace entries
 91      inline const char* DefaultExecutionTraceEntries()
 92      { return(default_execution_trace_entries); }
 93
 94
 95      // Execute the next instruction (NULL or pointer to error message)
 96      virtual const char* ExecuteInstruction(String& trace_record, int
                    trace_flag)=0;
 97
 98      // Handle an interrupt request from a device
 99      virtual void InterruptRequest(BasicDevice* device, int level)=0;
100
101      // Perform a system reset
102      virtual void Reset()=0;
103
```

```
104         // Return the name of the program counter register (usually "PC")
105         virtual char* const NameOfProgramCounter()=0;
106
107         // Return the value of the program counter register
108         virtual unsigned long ValueOfProgramCounter()=0;
109
110         // Set the named register to the given hexidecimal value
111         virtual void SetRegister(String name, String hex_value)=0;
112
113         // Clear the CPU's Statistics
114         virtual void ClearStatistics()=0;
115
116         // Append all of the CPU's registers to the RegisterInformationList object
117         virtual void BuildRegisterInformationList(RegisterInformationList*)=0;
118
119         // Append all of the CPU's stats to the StatisticalInformationList object
120         virtual void BuildStatisticalInformationList(StatisticalInformationList*)=0;
121     };
122     #endif
123
```

BasicDevice Class Declaration

```
 1  ////////////////////////////////////////////////////////////////////////////
 2  // $Id: BasicDevice.hxx,v 1.1 1994/02/18 19:48:13 bmott Exp $
 3  ////////////////////////////////////////////////////////////////////////////
 4  // BasicDevice.hxx - Device base class
 5  //
 6  // This is the abstract base class for all derived devices
 7  //
 8  //
 9  // BSVC "A Microprocessor Simulation Framework"
10  // Copyright (c) 1993
11  // By: Bradford W. Mott
12  // July 26,1993
13  //
14  ////////////////////////////////////////////////////////////////////////////
15  // $Log: BasicDevice.hxx,v $
16  // Revision 1.1  1994/02/18  19:48:13  bmott
17  // Initial revision
18  //
19  ////////////////////////////////////////////////////////////////////////////
20
21  #ifndef BASICDEVICE_HXX
22  #define BASICDEVICE_HXX
23
24  #include "String.h"
25  #include "Event.hxx"
26  #include "BasicCPU.hxx"
27
28  #define AUTOVECTOR_INTERRUPT  -1
29  #define SPURIOUS_INTERRUPT    -2
30
31  class BasicCPU;
32
33  ////////////////////////////////////////////////////////////////////////////
34  // BasicDevice class declaration
35  ////////////////////////////////////////////////////////////////////////////
36  class BasicDevice : public EventBase {
37    protected:
38      BasicCPU*   cpu;                    // CPU that owns the device
39      const char* name;                   // Name of the device
40      String      initilization_arguments; // Args used to setup device
41      String      error_message;          // Startup error message
42      int         interrupt_pending;      // Interrupt pending flag
43
44    public:
45      BasicDevice(const char* n, char* args, BasicCPU* c)
46          : EventBase(&c->events),
47            name(n),
48            initilization_arguments(args),
49            cpu(c),
50            interrupt_pending(0)
51      {};
52
53      virtual ~BasicDevice()
54      {};
55
56      // Set the device's startup error message
57      void SetErrorMessage(const char *message)
58      { error_message=message; }
59
60      // Get the device's startup error message
61      String GetErrorMessage()
62      { return(error_message); }
63
64      // Return the name of the device
65      inline const char* Name()
66      { return(name); }
67
68      // Return the device's CPU
69      inline BasicCPU* CPU()
70      { return(cpu); }
71
```

```
 72        // Return the initialization arguments
 73        inline String InitializationArguments()
 74        { return (initilization_arguments); }
 75
 76        // Tells if the address maps into the device (1=Yes,0=No)
 77        virtual char CheckMapped(unsigned long)=0;
 78
 79        // Return the lowest address used by the device
 80        virtual unsigned long LowestAddress()=0;
 81
 82        // Return the highest address used by the device
 83        virtual unsigned long HighestAddress()=0;
 84
 85        // Get a byte from the device
 86        virtual unsigned char Peek(unsigned long)=0;
 87
 88        // Put a byte into the device
 89        virtual void Poke(unsigned long,unsigned char)=0;
 90
 91        // Reset the device
 92        virtual void Reset();
 93
 94        // This routine sends an interrupt request (IRQ) to the CPU
 95        virtual void InterruptRequest(int level);
 96
 97        // This routine is called by the CPU when it processes the interrupt
 98        virtual long InterruptAcknowledge(int level);
 99    };
100    #endif
```

BasicDeviceRegistry Class Declaration

```
 1  ////////////////////////////////////////////////////////////////////////////// //
 2  // $Id: BasicDeviceRegistry.hxx,v 1.1 1994/02/18 19:48:54 bmott Exp $
 3  ////////////////////////////////////////////////////////////////////////////// //
 4  // BasicDeviceRegistry.hxx
 5  //
 6  //   This abstract base class is used to derive a class that maintains a list
 7  // of al the device in the simulator and allows them to be created.
 8  //
 9  //
10  // BSVC "A Microprocessor Simulation Framework"
11  // Copyright (c) 1993
12  // By: Bradford W. Mott
13  // October 30,1993
14  //
15  ////////////////////////////////////////////////////////////////////////////
16  // $Log: BasicDeviceRegistry.hxx,v $
17  // Revision 1.1  1994/02/18  19:48:54  bmott
18  // Initial revision
19  //
20  ////////////////////////////////////////////////////////////////////////////
21
22  #ifndef BASICDEVICEREGISTRY_HXX
23  #define BASICDEVICEREGISTRY_HXX
24
25  #include "String.h"
26
27  class BasicCPU;
28  class BasicDevice;
29
30  ////////////////////////////////////////////////////////////////////////////
31  // Device Information Type
32  ////////////////////////////////////////////////////////////////////////////
33  typedef struct {
34    const char *name;           // The name of the device ("RAM","m6850",etc)
35    const char *description;    // A short description of the device
36    const char *script;         // UI script to get the device attachment args
37  } DeviceInformation;
38
39  ////////////////////////////////////////////////////////////////////////////
40  // BasicDeviceRegistry class declaration
41  ////////////////////////////////////////////////////////////////////////////
42  class BasicDeviceRegistry {
43    private:
44      // List of devices in the simulator
45      const DeviceInformation *devices;
46
47      // Number of devices in the simulator
48      const int number_of_devices;
49
50    public:
51      BasicDeviceRegistry(const DeviceInformation *devs, int num)
52                        : devices(devs),
53                          number_of_devices(num)
54      {};
55
56      // Return the number of devices
57      inline int NumberOfDevices()
58      { return(number_of_devices); }
59
60      // Get the device information with the given index (return 1=OK,0=ERROR)
61      int Information(int index, DeviceInformation& information);
62
63      // Create a device with the given name (return 1=OK,0=ERROR)
64      virtual int Create(String& name, String& args, BasicCPU* cpu,
65                         BasicDevice* &device)=0;
66  };
67  #endif
68
```

AddressSpace Class Declaration

```
 1  ////////////////////////////////////////////////////////////////////////////
 2  // $Id: AddressSpace.hxx,v 1.1 1994/02/18 19:47:48 bmott Exp $
 3  ////////////////////////////////////////////////////////////////////////////
 4  // AddressSpace.hxx
 5  //
 6  //   This class maintains a list of devices and provides methods to peek and
 7  // poke into them.
 8  //
 9  //
10  // BSVC "A Microprocessor Simulation Framework"
11  // Copyright (c) 1993
12  // By: Bradford W. Mott
13  // June 27,1993
14  //
15  ////////////////////////////////////////////////////////////////////////////
16  // $Log: AddressSpace.hxx,v $
17  // Revision 1.1  1994/02/18  19:47:48  bmott
18  // Initial revision
19  //
20  ////////////////////////////////////////////////////////////////////////////
21
22  #ifndef ADDRESSSPACE_HXX
23  #define ADDRESSSPACE_HXX
24
25  #include "String.h"
26
27  class BasicDevice;
28
29  ////////////////////////////////////////////////////////////////////////////
30  // Used to retrieve information on devices attached to the address space
31  ////////////////////////////////////////////////////////////////////////////
32  struct AddressSpaceDeviceInformation {
33    String name;
34    String initialization_arguments;
35    unsigned int index;
36  };
37
38  ////////////////////////////////////////////////////////////////////////////
39  // AddressSpace class declaration
40  ////////////////////////////////////////////////////////////////////////////
41  class AddressSpace {
42    private:
43      // Structure for linked list of devices attached to the address space
44      struct DeviceNode {
45        BasicDevice *device;
46        DeviceNode  *next;
47      };
48
49      DeviceNode *head;                // Head of the linked list
50      DeviceNode *tail;                // Tail of the linked list
51
52      // Maximum address for this address space (In CPU words not bytes!!)
53      const unsigned long maximum_address;
54
55    public:
56      AddressSpace(unsigned long maximum_address);
57      virtual ~AddressSpace();
58
59      // Return the maximum address of the address space
60      inline unsigned long MaximumAddress()
61      { return(maximum_address); }
62
63      // Attach a device to the address space (1=OK,0=ERROR)
64      int AttachDevice(BasicDevice*);
65
66      // Detach and destory a device from the address space (1=OK,0=ERROR)
67      int DetachDevice(unsigned int index);
68
69      // Reset all of the attached devices
70      void Reset();
71
```

```
72        // Return the number of attached devices
73        int NumberOfAttachedDevices();
74
75        // Get information about the device with the given index (1=OK,0=ERROR)
76        int GetDeviceInformation(int index, AddressSpaceDeviceInformation& info);
77
78        // Peek a location in the address space (1=OK,0=Bus Error)
79        virtual int Peek(unsigned long addr, unsigned char &c);
80
81        // Poke a location in the address space (1=OK,0=Bus Error)
82        virtual int Poke(unsigned long addr, unsigned char c);
83    };
84    #endif
```

BasicLoader Class Declaration

```
 1  ////////////////////////////////////////////////////////////////////////////
 2  // $Id: BasicLoader.hxx,v 1.1 1994/02/18 19:49:06 bmott Exp $
 3  ////////////////////////////////////////////////////////////////////////////
 4  // BasicLoader.hxx
 5  //
 6  //   This abstract base class provides methods to load object files into the
 7  // the simulator.
 8  //
 9  //
10  // BSVC "A Microprocessor Simulation Framework"
11  // Copyright (c) 1993
12  // By: Bradford W. Mott
13  // November 5,1993
14  //
15  ////////////////////////////////////////////////////////////////////////////
16  // $Log: BasicLoader.hxx,v $
17  // Revision 1.1  1994/02/18  19:49:06  bmott
18  // Initial revision
19  //
20  ////////////////////////////////////////////////////////////////////////////
21
22  #ifndef BASICLOADER_HXX
23  #define BASICLOADER_HXX
24
25  #include "String.h"
26
27  class BasicCPU;
28
29  ////////////////////////////////////////////////////////////////////////////
30  // BasicLoader class declaration
31  ////////////////////////////////////////////////////////////////////////////
32  class BasicLoader {
33    protected:
34      BasicCPU*   cpu;
35
36    public:
37      BasicLoader(BasicCPU* c)
38          : cpu(c)
39      {};
40
41      virtual ~BasicLoader()
42      {};
43
44      // Return the loader's CPU
45      inline BasicCPU* CPU()
46      { return(cpu); }
47
48      // Load the file (Error Message or "" is returned)
49      virtual String Load(const char *filename, int space)=0;
50  };
51  #endif
52
```

EventHandler Class Declaration

```
 1  ////////////////////////////////////////////////////////////////////////////
 2  // $Id: Event.hxx,v 1.1 1994/02/18 19:49:44 bmott Exp $
 3  ////////////////////////////////////////////////////////////////////////////
 4  // Event.hxx
 5  //
 6  //   This class maintains a queue of events requested by EventBase derived
 7  // objects.
 8  //
 9  //
10  // BSVC "A Microprocessor Simulation Framework"
11  // Copyright (c) 1993
12  // By: Bradford W. Mott
13  // August 11,1993
14  //
15  ////////////////////////////////////////////////////////////////////////////
16  // $Log: Event.hxx,v $
17  // Revision 1.1  1994/02/18  19:49:44  bmott
18  // Initial revision
19  //
20  ////////////////////////////////////////////////////////////////////////////
21
22  #ifndef EVENT_HXX
23  #define EVENT_HXX
24
25  class EventHandler;
26
27  ////////////////////////////////////////////////////////////////////////////
28  // This should be the base class for any class that is going to register
29  // events with the event handler.
30  ////////////////////////////////////////////////////////////////////////////
31  class EventBase {
32    private:
33      EventHandler* event_handler;
34
35    public:
36      EventBase(EventHandler* h)
37          : event_handler(h)
38      {}
39
40      virtual ~EventBase();
41
42      virtual void EventCallback(long data, void* pointer)=0;
43  };
44
45  ////////////////////////////////////////////////////////////////////////////
46  // This class manages a list of time events.  When the time expires for
47  // an event the EventCallback() for the associate object is called
48  ////////////////////////////////////////////////////////////////////////////
49  class EventHandler {
50    private:
51
52      // The event class
53      class Event {
54        private:
55          // The object that owns this event
56          EventBase* object;
57
58          // Data passed to the callback routine
59          void* pointer;
60          long  data;
61
62        public:
63          Event(EventBase* o, long d, void* p, unsigned long t)
64              : object(o), data(d), pointer(p),
65                total_time(t),
66                next((void *)0)
67          {};
68
69          // Dispatch the event by calling the object's callback routine
70          inline void Dispatch()
71          { object->EventCallback(data, pointer); }
```

```
 72
 73          // Return the owning object
 74          inline EventBase* Owner()
 75          { return(object); }
 76
 77          // Total amount of time to elapse before the event
 78          const long total_time;
 79
 80          // Time left before the event occurs
 81          long delta_time;
 82
 83          // Pointer to the next event
 84          Event *next;
 85      };
 86
 87      // Linked list of events
 88      Event *list;
 89
 90      // Number of calls since last time update
 91      long iterations;
 92
 93      // Last usec_per_check update time
 94      long old_time;
 95
 96      // Average micro-seconds per call to Check
 97      long usec_per_check;
 98
 99    public:
100      EventHandler();
101
102      // Check for any expired events
103      void Check();
104
105      // Add an event to the event list
106      void Add(EventBase* object, long data, void* pointer, long time);
107
108      // Remove events for the given object
109      void Remove(EventBase* object);
110  };
111
112  #endif
```

RegisterInformationList Class Declaration

```
 1  ////////////////////////////////////////////////////////////////////////////// //
 2  // $Id: RegInfo.hxx,v 1.1 1994/02/18 19:50:09 bmott Exp $
 3  ////////////////////////////////////////////////////////////////////////////// //
 4  // RegInfo.hxx
 5  //
 6  //   This class is used by BasicCPU (and derived classes) to manage a list of
 7  // of register structures.
 8  //
 9  //
10  // BSVC "A Microprocessor Simulation Framework"
11  // Copyright (c) 1993
12  // By: Bradford W. Mott
13  // October 25,1993
14  //
15  ////////////////////////////////////////////////////////////////////////////////
16  // $Log: RegInfo.hxx,v $
17  // Revision 1.1  1994/02/18  19:50:09  bmott
18  // Initial revision
19  //
20  ////////////////////////////////////////////////////////////////////////////////
21
22  #ifndef REGINFO_HXX
23  #define REGINFO_HXX
24
25  #include "BasicCPU.hxx"
26
27  ////////////////////////////////////////////////////////////////////////////////
28  // RegisterInformation class declaration
29  ////////////////////////////////////////////////////////////////////////////////
30  class RegisterInformation {
31    private:
32      char *name;         // The name given to the register ("D0","PC",etc)
33      char *hex_value;    // The value of the register in hexidecimal
34      char *description;  // A short description of the register
35
36    public:
37      RegisterInformation(const char* name, const char* hex_value,
38                          const char* description);
39      RegisterInformation();
40      ~RegisterInformation();
41
42      // Set the name, hex_value, and the description fields
43      void Set(const char* name, const char* hex_value, const char* description);
44
45      inline const char* Name()
46      { return(name); }
47
48      inline const char* HexValue()
49      { return(hex_value); }
50
51      inline const char* Description()
52      { return(description); }
53  };
54
55
56  ////////////////////////////////////////////////////////////////////////////////
57  // RegisterInformationList class declaration
58  ////////////////////////////////////////////////////////////////////////////////
59  class RegisterInformationList {
60    private:
61      // Class for a linked list of RegisterInformation structures
62      class RegisterInformationNode : RegisterInformation {
63        public:
64          RegisterInformationNode* next;
65
66          RegisterInformationNode(const char* name, const char* hex_value,
67                                  const char* description)
68              : RegisterInformation(name,hex_value,description),
69                next((RegisterInformationNode*)0)
70          { };
71      };
```

```
72
73        RegisterInformationNode* head;  // Head of the linked list
74        RegisterInformationNode* tail;  // Tail of the linked list
75        int number_of_elements;         // Number of elements in the list
76
77      public:
78        RegisterInformationList(BasicCPU*);
79        ~RegisterInformationList();
80
81        // Append an element to the end of the list
82        void Append(const char* name,const char* hex_value,const char* description);
83
84        // Return the number of elements in the list
85        inline int NumberOfElements()
86        { return(number_of_elements); }
87
88        // Get the element with the given index (return 1=OK,0=ERROR)
89        int Element(int,RegisterInformation&);
90    };
91    #endif
```

StatisticalInformationList Class Declaration

```
 1  ////////////////////////////////////////////////////////////////////////////// //
 2  // $Id: StatInfo.hxx,v 1.1 1994/02/18 19:53:49 bmott Exp $
 3  ////////////////////////////////////////////////////////////////////////////// //
 4  //
 5  // StatInfo.hxx
 6  //
 7  //   This class is used by BasicCPU (and derived classes) to manage a list of
 8  // of statistics objects.
 9  //
10  //
11  // BSVC "A Microprocessor Simulation Framework"
12  // Copyright (c) 1993
13  // By: Bradford W. Mott
14  // December 5,1993
15  //
16  ////////////////////////////////////////////////////////////////////////////////
17  // $Log: StatInfo.hxx,v $
18  // Revision 1.1  1994/02/18  19:53:49  bmott
19  // Initial revision
20  //
21  ////////////////////////////////////////////////////////////////////////////////
22
23  #ifndef STATINFO_HXX
24  #define STATINFO_HXX
25
26  #include "BasicCPU.hxx"
27
28  ////////////////////////////////////////////////////////////////////////////////
29  // The Statistic Information Class
30  ////////////////////////////////////////////////////////////////////////////////
31  class StatisticInformation {
32    private:
33      char *statistic;    // The statistic (i.e. "Number of reads:  100")
34
35    public:
36      StatisticInformation(const char* statistic);
37      StatisticInformation();
38      ~StatisticInformation();
39
40      // Set the statistic fields
41      void Set(const char* statistic);
42
43      inline const char* Statistic()
44      { return(statistic); }
45  };
46
47
48  ////////////////////////////////////////////////////////////////////////////////
49  // The Statistical Information List Class
50  ////////////////////////////////////////////////////////////////////////////////
51  class StatisticalInformationList {
52    private:
53      // Class for a linked list of StatisticInformation structures
54      class StatisticInformationNode : StatisticInformation {
55        public:
56          StatisticInformationNode* next;
57
58          StatisticInformationNode(const char* statistic)
59              : StatisticInformation(statistic),
60                next((void*)0)
61          { };
62      };
63
64      StatisticInformationNode* head;  // Head of the linked list
65      StatisticInformationNode* tail;  // Tail of the linked list
66      int number_of_elements;          // Number of elements in the list
67
68    public:
69      StatisticalInformationList(BasicCPU*);
70      ~StatisticalInformationList();
71
```

```
72        // Append an element to the end of the list
73        void Append(const char* statistic);
74
75        // Return the number of elements in the list
76        inline int NumberOfElements()
77        { return(number_of_elements); }
78
79        // Get the element with the given index (return 1=OK,0=ERROR)
80        int Element(int, StatisticInformation&);
81    };
82    #endif
```

Interface Class Declaration

```
 1  ////////////////////////////////////////////////////////////////////////////
 2  // $Id: Interface.hxx,v 1.1 1994/08/03 22:54:26 bmott Exp $
 3  ////////////////////////////////////////////////////////////////////////////
 4  // Interface.hxx
 5  //
 6  // This is the user interface command class.  It handles all of the
 7  // command's issue by the user interface.
 8  //
 9  //
10  // BSVC "A Microprocessor Simulation Framework"
11  // Copyright (c) 1993
12  // By: Bradford W. Mott
13  // October 21,1993
14  //
15  ////////////////////////////////////////////////////////////////////////////
16  // $Log: Interface.hxx,v $
17  // Revision 1.1  1994/08/03  22:54:26  bmott
18  // Initial revision
19  //
20  //
21  ////////////////////////////////////////////////////////////////////////////
22
23  #ifndef INTERFACE_HXX
24  #define INTERFACE_HXX
25
26  #include "BasicCPU.hxx"
27  #include "BasicDeviceRegistry.hxx"
28  #include "BasicLoader.hxx"
29  #include "BreakpointList.hxx"
30
31  class Interface;
32
33  typedef struct {
34    const char *name;
35    void (Interface::*mfp)(char*);
36  } UICommandTable;
37
38  class Interface {
39    private:
40      const int number_of_commands;      // Number of commands in command table
41      static int interrupt_signal_flag;
42      static void InterruptSignalHandler(int);
43
44      BasicCPU* const cpu;
45      BasicDeviceRegistry* const device_registry;
46      BasicLoader* const loader;
47      BreakpointList breakpoint_list;
48
49      char *Get();                       // Get a string from the UI
50      void Put(const char *);            // Send a string to the UI
51      int  ExecuteCommand(const char*);  // Execute the command
52
53
54      static UICommandTable command_table[];
55
56      void AddBreakpoint(char *);
57      void AttachDevice(char *);
58      void ClearStatistics(char *);
59      void DeleteBreakpoint(char *);
60      void DetachDevice(char *);
61      void ListAttachedDevices(char *);
62      void ListBreakpoints(char *);
63      void ListDevices(char *);
64      void ListDeviceScript(char *);
65      void ListExecutionTraceRecord(char *);
66      void ListDefaultExecutionTraceEntries(char *);
67      void ListGranularity(char *);
68      void ListMemory(char *);
69      void ListMaximumAddress(char *);
70      void ListNumberOfAddressSpaces(char *);
71      void ListPCRegisterName(char *);
```

```
72        void ListRegisters(char *);
73        void ListRegisterValue(char *);
74        void ListRegisterDescription(char *);
75        void ListStatistics(char *);
76        void LoadProgram(char *);
77        void Reset(char *);
78        void Run(char *);
79        void SetRegister(char *);
80        void SetMemory(char *);
81        void Step(char *);
82
83      public:
84        Interface(BasicCPU*, BasicDeviceRegistry*, BasicLoader*);
85
86        void CommandLoop();
87    };
88
89    #endif
```