NAME

machine − CHSM machine class

SYNOPSIS

```
#define CHSM_MACHINE_ARGS   /* ... */
#define CHSM_MACHINE_INIT   /* ... */

namespace Concurrent_Hierarchical_State_Machine {

    class machine {
    public:
        typedef state value_type;
        typedef value_type const* const_pointer;
        typedef value_type const& const_reference;

        virtual ~machine();

        bool         active() const;
        virtual bool enter( event const &trigger = prime_ );
        virtual bool exit ( event const &trigger = prime_ );

        class const_iterator {
        public:
            const_iterator();

            const_reference operator* () const;
            const_pointer   operator->() const;

            const_iterator& operator++();
            const_iterator  operator++(int);

            friend bool
            operator==( const_iterator const&, const_iterator const& );
            friend bool
            operator!=( const_iterator const&, const_iterator const& );
        };
        const_iterator begin() const;
        const_iterator end() const;

        enum {
            D_none  = 0x00,
            D_enex  = 0x01,
            D_event = 0x02,
            D_alg   = 0x04,
            D_all   = D_enex | D_event | D_alg
        };

        unsigned debug() const;
        unsigned debug( unsigned );

        void dump_state() const;
    protected:
        machine( CHSM_MACHINE_ARGS );
    };
```

```
}
```

DESCRIPTION

The `machine` class is the base class for user-specified CHSMs.

Functions

`virtual bool enter( event const &trigger = prime_ )`

`virtual bool exit ( event const &trigger = prime_ )`

Returns `true` only if the state was actually entered or exited, respectively. The `trigger` is a reference to the event that is causing the state to be entered or exited. By default, it is an implementation-supplied, transitionless event used solely for this purpose. These functions can be overridden in a derived class to alter the behavior of entrances and exits.

`bool active() const`

Returns `true` only if the `root` cluster, and thus the CHSM as a whole, is currently active. (See **chsm-c++**(4).)

Debugging Functions

`unsigned debug() const`

Returns the current debugging state as an unsigned integer comprised of the bitwise or of the debugging states.

`unsigned debug( unsigned )`

Sets the current debugging state as specified by the argument comprised of the bitwise or of the debugging states. Debugging output goes to standard error. Returns the previous debugging state. The debugging states are:

`D_none`	None.
`D_enex`	Reports state entrances and exits.
`D_event`	Reports event queuing and dequeuing.
`D_alg`	Reports progress during the event-broadcast algorithm.
`D_all`	Reports all debugging information.

`void dump_state() const`

Dumps a printout of the current state to standard error that consists of each state's name, one per line, preceded by an asterisk only if it is active; a space otherwise.

Iterators

The `const_iterator` class is an iterator over the machine's states. It is in STL style. There is no (non-`const`) `iterator` since the only thing that should affect a state's state are events causing transitions. The `machine` member functions `begin()` and `end()` are used to return `const_iterator`s.

Derived Classes

The `machine` class can be derived from to add additional data members and members functions to a CHSM. The macros `CHSM_MACHINE_ARGS` and `CHSM_MACHINE_INIT` are used in the derived class's constructor and shield the user from the ugly arguments lists used in the CHSM implementation.

Additional, trailing constructor arguments may be specified. Such arguments must be repeated in the *parameter-list* in the `chsm` specification.

EXAMPLE

```
#include <chsm.h>

class my_stuff : public CHSM::machine {
public:
    my_stuff( CHSM_MACHINE_ARGS, int arg ) :
        CHSM::machine( CHSM_MACHINE_INIT ), mbr( arg ) { }
protected:              // make accessible to derived class
    int mbr;
```

```
};

%%
chsm<my_stuff> my_machine( int arg ) is {
    state s {
        alpha -> t %{
            mbr = 0;    // note: it's accessible
        %};
    }
    // ...
}

%%
int main() {
    my_machine m( 42 ); // pass only additional argument(s)
}
```

NOTES

All lines of debugging output are preceded by the | character to make them easily distinguishable.

SEE ALSO

CHSM::cluster(3), **CHSM::parent**(3), **CHSM::set**(3), **CHSM::state**(3), **chsm-c++**(4), **iterator**(STL)

AUTHORS

Paul J. Lucas <*paul@lucasmail.org*>
Fabio Riccardi <*fabio.riccardi@mac.com*>