

# Making the Transition to IEEE 1666 SystemC

John Aynsley, CTO, Doulos



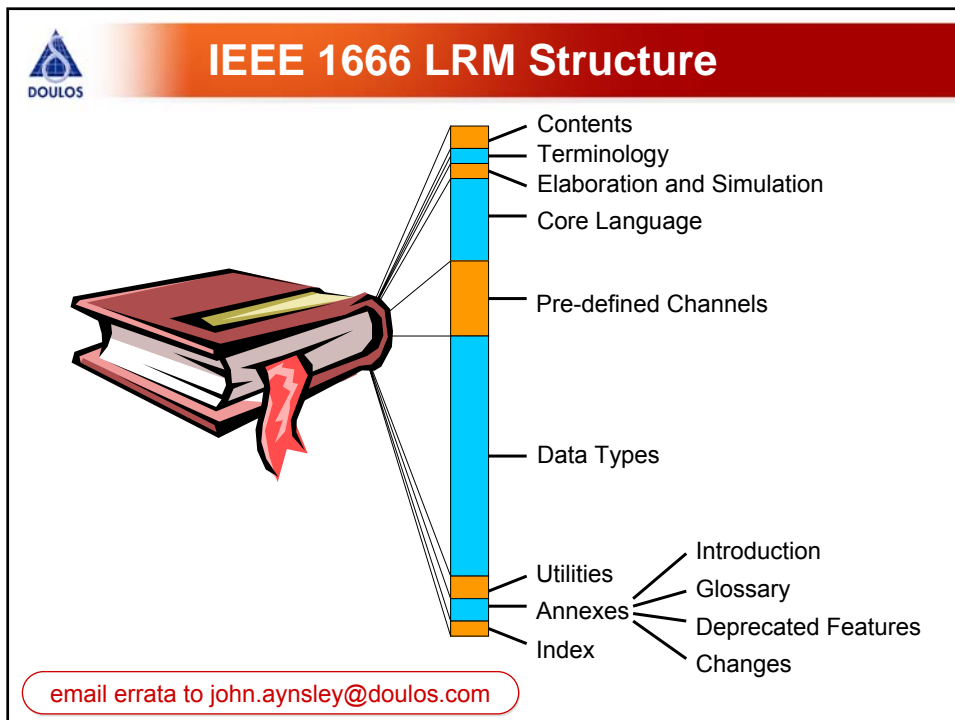
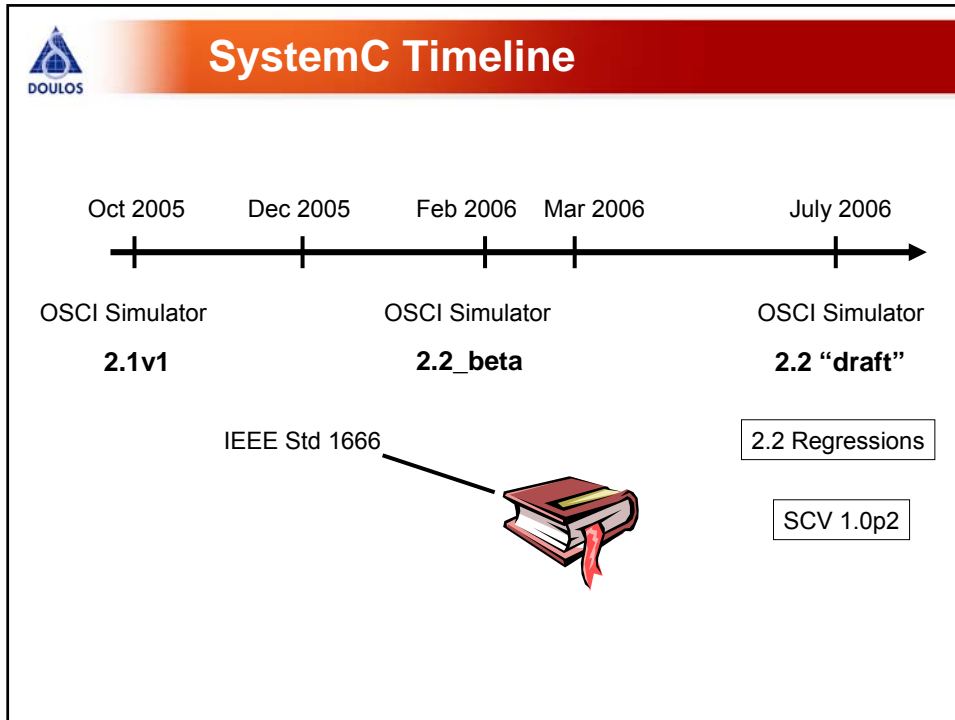
## Making the Transition to IEEE 1666 SystemC

### CONTENTS

---

- SystemC timeline
- New features
- Deprecated features
- SystemC 2.2 support for transition







## 2.1 New Features

- Added in 2.1
  - **Dynamic processes (sc\_spawn)**
  - **Exports (sc\_export)**
  - **New callbacks (before\_end\_of\_elaboration...)**
  - **New sc\_report (using string id)**
  - reset\_signal\_is (async reset for clocked thread)
  - Event queue
  - ...



## 1666 & 2.2 New Features

- Added in 1666 & 2.2
  - Header <systemc>, namespace sc\_core (2.1v1)
  - std::string (2.1v1)
  - std::vector (2.1v1)
  - get\_parent\_object, get\_child\_objects, sc\_get\_top\_level\_objects
  - New sc\_process\_handle
  - Port binding policy
  - ...
- 2.2 supports GNU C++, HP C++, Sun C++, 64-bit Linux, Visual C++ 7.1



## 1666 & 2.2 Deprecated Features

- Deprecated in 1666 & 2.2
  - `sc_cycle`, `sc_initialize`                      use `sc_start`
  - `sc_simcontext`    use global functions
  - `sc_process_b`    use `sc_process_handle`
  - `notify_delayed`    use `notify(sc_time)`
  - `::notify`    use `sc_event::notify`
  - `sc_module::timed_out`
  - `sc_module::operator`,                                      Positional port binding...
  - `sc_module::operator<<`                                      use `operator()`
  - `sc_sensitive::operator()`                                  use `sc_sensitive::operator<<`
  - `sc_sensitive_pos`    use `pos()`
  - `sc_sensitive_neg`    use `neg()`
  - `sc_module::end_module`



## 1666 & 2.2 Deprecated Features

- continued
  - Default time units
  - `trace`, `add_trace`    use `sc_trace`
  - `sc_signal::get_data_ref`    use `read`
  - `sc_signal::get_new_value`
  - `sc_inout_clk`, `sc_out_clk`                                      use `sc_out<bool>`
  - wif and isdb trace file formats
  - `sc_bit`    use `sc_bool`
  - `SC_THREAD( non-event-finder )`
  - Global and local watching,  $\lambda$ -expressions              use `reset_signal_is`
  - `wait(args)` in a clocked process
  - `sc_report` with integer message id
  - `sc_string`, `sc_pvector`, `sc_plist`, `sc_phash`, `sc_ppq`
  - ...



## Deprecated Feature Reports

SystemC 2.2.05jun06\_beta --- Jun 7 2006 09:48:00  
Copyright (c) 1996-2006 by all Contributors  
ALL RIGHTS RESERVED

Info: (I804) /IEEE\_Std\_1666/deprecated: sc\_bit is deprecated, use bool instead

Info: (I804) /IEEE\_Std\_1666/deprecated: use of () to specify sensitivity is deprecated, use << instead

Info: (I804) /IEEE\_Std\_1666/deprecated: You can turn off warnings about IEEE 1666 deprecated features by placing this method call as the first statement in your sc\_main() function:

```
sc_report_handler::set_actions("/IEEE_Std_1666/deprecated", SC_DO_NOTHING);
```

or setenv SC\_DEPRECATED\_WARNINGS DISABLE



## Reporting

- Reports embedded within pre-compiled IP

```
SC_REPORT_INFO("/IEEE_Std_1666/deprecated",  
               "sc_bit is deprecated ...");  
SC_REPORT_INFO("/IEEE_Std_1666/deprecated",  
               "use of () to specify ...");  
SC_REPORT_INFO("/IEEE_Std_1666/deprecated",  
               "You can turn off warnings ...");
```

- Application code

```
sc_report_handler::set_actions("/IEEE_Std_1666/deprecated",  
                               SC_DO_NOTHING);  
  
sc_report_handler::set_actions(SC_WARNING, SC_STOP);
```



## Header File

Old header (not deprecated)

```
#include "systemc.h"

struct Mod: sc_module
{
    sc_signal<sc_logic> sig;
};
cout << endl;
```

New header (recommended)

```
#include "systemc"

using sc_core::sc_module;
using sc_core::sc_signal;
using sc_dt::sc_logic;
using std::cout;
using std::endl;

struct Mod: sc_module
{
    sc_signal<sc_logic> sig;
};
cout << endl;
```



## Standard Classes

sc\_string

replaced by

std::string

sc\_pvector

replaced by

std::vector

sc\_exception

replaced by

std::exception

sc\_plist

sc\_phash

sc\_ppq

} deprecated



## Backward Compatibility for sc\_string

```
namespace sc_dt {  
    class sc_string_old; = sc_string class from 2.0.1  
}
```

```
#ifndef SC_USE_SC_STRING_OLD  
    typedef sc_dt::sc_string_old sc_string;  
#endif  
  
#ifndef SC_USE_STD_STRING  
    typedef ::std::string sc_string;  
#endif
```



## Elaboration and Callbacks 1

- Instantiating modules, ports, exports, primitives
- Binding ports and exports
- Calling event finders
  - Before or during before\_end\_of\_elaboration
  
- Static process macros and static sensitivity
  - Before or during end\_of\_elaboration
  
- Calling sc\_port::operator-> or sc\_port::operator[]
  - end\_of\_elaboration onward



## Elaboration and Callbacks 2

- Instantiating `sc_objects`
- Calling `sc_spawn`
- Calling `sc_prim_channel::request_update` (or `sc_signal::write`)
  - Any time before `end_of_simulation`
- Calling `notify`
  - Between `start_of_simulation` and `end_of_simulation`



## Multiple Writers

```
sc_signal<bool> sig;  
... {  
    SC_THREAD(T1);  
    SC_THREAD(T2);  
}  
void T1() {  
    sig.write(false);  
}  
void T2() {  
    sig.write(true);  
}
```

SystemC 2.2.05jun06\_

Error: (E115) `sc_signal<T>` cannot have more than one driver:  
signal `top.signal\_0' (`sc_signal`)  
first driver `top.T1' (`sc_thread_process`)  
second driver `top.T2' (`sc_thread_process`)  
In file: `c:\systemc-2.2.05jun06_beta\src\sysc\communication\sc_signal.cpp:126`  
In process: `top.T2 @ 0 s`

or `setenv SC_SIGNAL_WRITE_CHECK DISABLE`



## Hierarchy Traversal

```
void recursive_descent(sc_object* obj)
{
    if (s = dynamic_cast<sc_signal<bool>*>(obj))
        ... Print all boolean signals?

    std::vector<sc_object*> children =
        obj->get_child_objects();

    for ( unsigned i = 0; i < children.size(); i++ )
        if ( children[i] )
            recursive_descent( children[i] );
}

std::vector<sc_object*> tops =
    sc_get_top_level_objects();

for ( unsigned i = 0; i < tops.size(); i++ )
    if ( tops[i] )
        recursive_descent( tops[i] );
```



## Process Kind

```
void process()
{
    sc_process_handle h;
    h = sc_get_current_process_handle();
    if ( h.valid() )
        switch ( h.proc_kind() )
        {
            case SC_THREAD_PROC_:
                ... break;
            case SC_CTHREAD_PROC:
                ... break;
            case SC_METHOD_PROC:
                ... break;
        }
}
```



## Process Termination

- fork...join\_none

```
sc_process_handle h1 = sc_spawn(&proc1, "proc1");  
sc_process_handle h2 = sc_spawn(&proc2, "proc2");
```

- fork...join

```
wait(h1.terminated_event() & h2.terminated_event());
```

- fork...join\_any

```
wait(h1.terminated_event() | h2.terminated_event());
```



## Port Policies

```
sc_port<IF>  
sc_port<IF,0>  
sc_port<IF,3>
```

Bound to exactly 1 channel  
Bound to 1 or more channels  
Bound to 1, 2 or 3 channels

```
sc_port<IF,0,SC_ZERO_OR_MORE_BOUND>  
sc_port<IF,3,SC_ZERO_OR_MORE_BOUND>
```

Bound to 0 or more channels  
Bound to 0, 1, 2 or 3 channels

```
sc_port<IF,3,SC_ALL_BOUND>
```

Bound to exactly 3 channels



## For More FREE Information

- IEEE 1666

[standards.ieee.org/getieee/1666/index.html](http://standards.ieee.org/getieee/1666/index.html)

- OSCI SystemC 2.2

[www.systemc.org](http://www.systemc.org)

- Paper on new features

[www.doulos.com/knowhow/systemc/new\\_standard](http://www.doulos.com/knowhow/systemc/new_standard)