YANG model for management of Network Tester

IETF 121

2–8 November 2024

Dublin, Ireland

Module changes in draft-ietf-bmwg-network-tester-cfg-05

```
module: ietf-traffic-analyzer

  augment /if:interfaces/if:interface:
    +--rw traffic-analyzer!
       +--rw testframe-filter! {testframe-filter}?
       |  +--rw type    identityref
       |  +--rw mask?   string
       |  +--rw data?   string
       +--rw capture {capture}?
       ...
       |  +--rw stop-trigger
       |     +--rw (stop-trigger)?
       |        +--:(when-full)
       |           +--rw when-full?   empty
       +--ro state
       ...
          |  +--ro last-sequence-error
          |     +--ro timestamp?   yang:date-and-time
          |     +--ro expected?    yang:counter64
          |     +--ro received?    yang:counter64
       ...
```

Usecases:

*RFC2544 26.5 System recovery

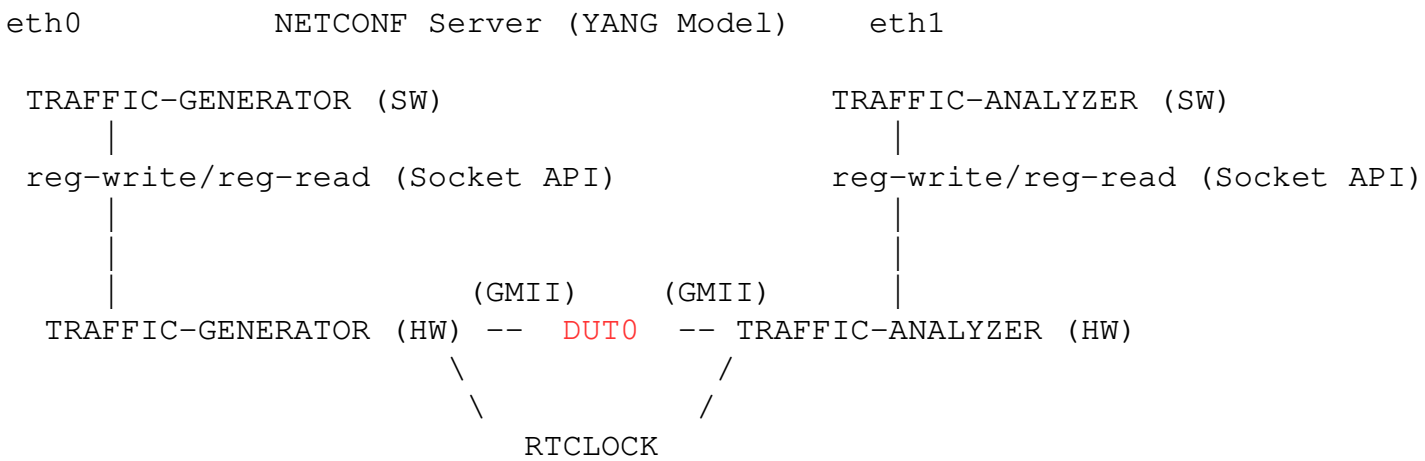
*RFC2544 26.6 Reset

*Others

Hackathon Plan

- validate draft-ietf-bmwg-network-tester-cfg-05 modules
- simulate and target test latest reference implementation

Validation: rfc2544 benchmark ↔ netconfd ↔ cocotb simulation of Verilog HDL



Added **dut0** - trivial packet filter that can be configured to drop packets if the interframe gap is less then certain threshold.

Simulation: rfc2544-benchmark run (1/4)

```
$ sim-devmem32 0x30000008 21
$ rfc2544-benchmark --config=config.xml --dst-node=tester0 --dst-node-interface=eth1 --src-node=tester0 --src-node-interface=eth0
--dst-mac-address="70:B3:D5:EC:20:10" --src-mac-address="70:B3:D5:EC:20:11" --dst-ipv4-address="192.168.1.145" --src-ipv4-udp-
port=49184 --src-ipv4-address="192.168.0.145" --frame-size=64 --trial-time=2 --speed=10000 --simulation | tee report-full.log |
grep ^#
#===Throughput===
#1 14.880952 pps, 20 octets interframe gap, 100.00% ... 15 / 29
#2 7.911392 pps, 94 octets interframe gap, 53.16% ... 15 / 15
#3 11.363636 pps, 46 octets interframe gap, 76.36% ... 22 / 22
#4 13.020833 pps, 32 octets interframe gap, 87.50% ... 26 / 26
#5 13.888889 pps, 26 octets interframe gap, 93.33% ... 27 / 27
#6 14.367816 pps, 23 octets interframe gap, 96.55% ... 28 / 28
#7 14.534884 pps, 22 octets interframe gap, 97.67% ... 29 / 29
#8 14.705882 pps, 21 octets interframe gap, 98.82% ... 29 / 29
#Result: 14.705882 pps
#===Latency===
#Measurement style - bit forwarding
#1 24 ns (min=24 ns, max=24 ns) ... 29 / 29
#===Frame loss rate===
#1 100% rate, 48% loss, (100.000000% rate actual), 14.880952 pps (14.880952 pps actual), 20 octets interframe gap ... 15 / 29
#2 90% rate, 0% loss, (89.361702% rate actual), 13.392857 pps (13.297872 pps actual), 30 octets interframe gap ... 26 / 26
#3 80% rate, 0% loss, (80.000000% rate actual), 11.904762 pps (11.904762 pps actual), 41 octets interframe gap ... 23 / 23
#===Back to back frames===
#1 2 back-to-back frames ... 1 / 2
$ sim-finish
```

Simulation: rfc2544-benchmark run (2/4)

```
  <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
      <interface>
        <name>eth0</name>
        <type
          xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">ianaift:ethernetCsmacd</type>
        <traffic-generator xmlns="urn:ietf:params:xml:ns:yang:ietf-traffic-generator">
          <testframe-type
            xmlns:nttg="urn:ietf:params:xml:ns:yang:ietf-traffic-generator">nttg:dynamic</
testframe-type>
          <frame-size>64</frame-size>
       <frame-data>
70B3D5EC201070B3D5EC201108004500002E000000000A112D4DC0A80091C0A80191C0200007001A000000010203040506
0708090A0B0C0D0E0F10114602DB62
       </frame-data>
          <interframe-gap>20</interframe-gap>
          <total-frames>29</total-frames>
        </traffic-generator>
      </interface>
...
```

Simulation: rfc2544-benchmark run (3/4)

```
<interface>
  <name>eth1</name>
  <type
    xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">ianaift:ethernetCsmacd</type>
  <traffic-analyzer xmlns="urn:ietf:params:xml:ns:yang:ietf-traffic-analyzer">
    <state>
      <pkts>15</pkts>
      <octets>960</octets>
      <octets-idle>5818</octets-idle>
      <bad-crc-octets>0</bad-crc-octets>
      <bad-crc-pkts>0</bad-crc-pkts>
      <bad-preamble-octets>0</bad-preamble-octets>
      <bad-preamble-pkts>0</bad-preamble-pkts>
      <octets-total>6948</octets-total>
      <testframe-stats>
        <pkts>15</pkts>
        <sequence-errors>14</sequence-errors>
        <last-sequence-error>
          <expected>27</expected>
          <received>28</received>
          <timestamp>1970-01-01T00:00:00.000053760Z</timestamp>
        </last-sequence-error>
        <latency>
          <samples>15</samples>
          <min-sec>0</min-sec>
          <min>24</min>
          <max-sec>0</max-sec>
          <max>24</max>
          <last-sec>0</last-sec>
          <last>24</last>
        </latency>
      </testframe-stats>
      <capture>
        <timestamp>1970-01-01T00:00:00.000053760Z</timestamp>
        <sequence-number>15</sequence-number>
      <data>55555555555555D570B3D5EC201070B3D5EC201108004500002E000000000A112D4DC0A80091C0A80191C0200007001A0000000000000000001C0000000000000000D1E8967A2B4B</data>
      </capture>
    </state>
  </traffic-analyzer>
</interface>
```

GTKWave - sim_build/tester_loop.fst

Marker: 0 sec | Cursor: 34340 ns

From: 0 sec To: 1899540001 ps

SST

- tester_loop
  - dut0 (filter_gmii)
  - rtclock0 (rtclock)
  - traffic_analyzer_gmii0 (traffic_a
  - traffic_generator_gmii0 (traffic

Type	Signals
reg	run
wire	sec[47:0]
reg	sequence_errors[63:0]
integer	sequence_errors_delta
reg	sequence_errors_reg[63:0]
wire	sequence_num[63:0]
reg	state[1:0]
wire	testframe_filter_data[319:0]
wire	testframe_filter_mask[319:0]
wire	testframe_match
reg	testframe_pkts[63:0]
integer	testframe_pkts_delta
reg	testframe_pkts_reg[63:0]
reg	timestamp_nsec[31:0]
reg	timestamp_nsec_reg[31:0]
reg	timestamp_sec[63:0]
reg	timestamp_sec_reg[63:0]
wire	timestamp_tx_nsec[31:0]
wire	timestamp_tx_sec[47:0]
reg	update_latency
reg	update_latency_regs
reg	version_reg[31:0]

Append Insert Replace

Signals

Time

gmii_en_in=
gmii_d_in[7:0]=
gmii_en_out=
gmii_d_out[7:0]=
min_interframe_gap_reg[31:0]=
latency_nsec_reg[31:0]=
last_sequence_error_timestamp_sec_reg[63:0]=
last_sequence_error_timestamp_nsec_reg[31:0]=
last_sequence_error_received_reg[63:0]=
last_sequence_error_expected_reg[63:0]=
timestamp_nsec_reg[31:0]=

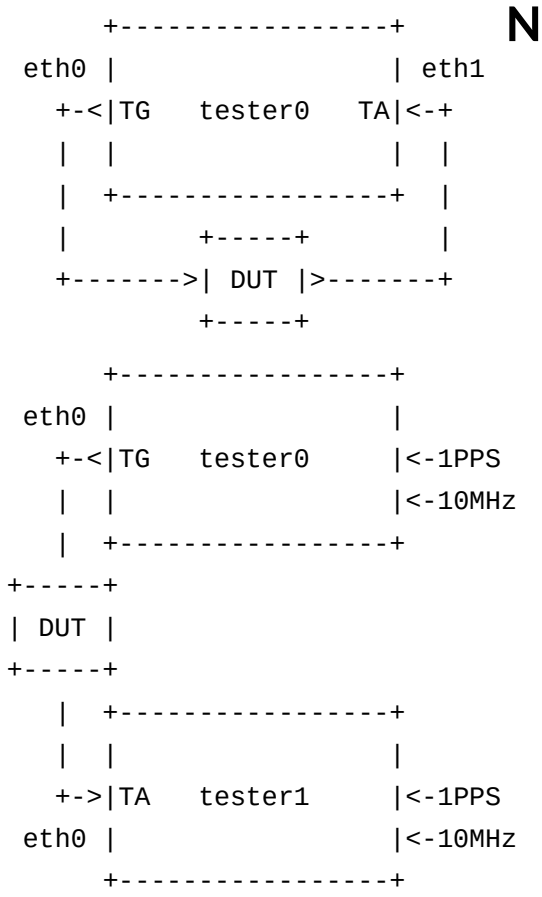
Waves

35 us 36 us 37 us 38 us 39 us 40 us 41 us

00
00 00 00 00 00 00
00000015
00000002 00000018
0000000000000000
00000000 00008DE8 00009328 00009868 00009DA8
0000000000000000 0000000000000002 0000000000000004 0000000000000006 0000000000
0000000000000000 0000000000000001 0000000000000003 0000000000000005 0000000000
00000000 000088A8 00008DE8 00009328 00009868 00009DA8

The End

references, repository links
and some updated slides
presented at earlier sessions
follow



Network Tester Management Solutions

- Command line (SCPI) over TCP/IP or GPIB (IEEE 488)
- Cisco TRex (YAML,JSON-RPC)
- Keysight Open Traffic Generator APIs (REST) & Data Models (2023)
- High Level Test Application Programming Interface (HLTAPI)
- Other

- YANG (RFC7950) model and NETCONF(RFC6241) protocol implementation

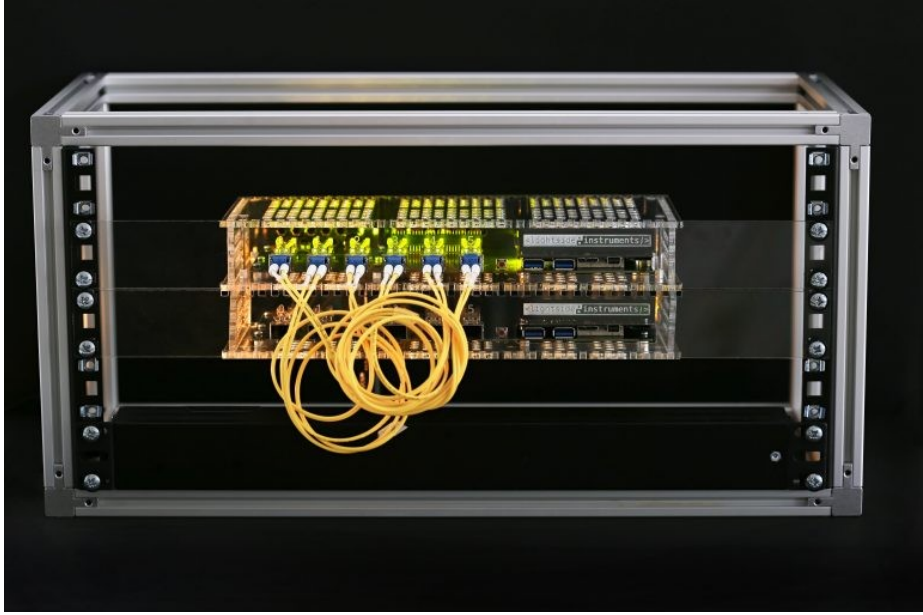
Test & Measurement Instrument Automation Wish-list 1/2

1. have an interchangeable interface - IVI, YANG/NETCONF
2. be scalable instead of complex – combining L2-3 with >L4 devices is unnecessary
3. have transactional management model with hierarchical (tree-like) configuration and state representation - YANG/NETCONF
4. model that does not put constrains on the precision of the implementation (example: traffic spec with Pkt/sec, Bandwidth or interframe-gaps)
5. model that does not limit re-usability of implementations
have model with optional features and a mechanism to announce the set of implemented features – RFC8525 YANG Library)
6. management model has to have a compatible and simple command line manipulation syntax which does not require extra work to be developed – **yangcli**

Test & Measurement Instrument Automation Wish-list 2/2

7. model has to be as deterministic as it can be - avoid RFC 8342 NMDA delayed commit provision on test instruments
8. have model applicable to real devices, discrete event network simulation and gate level simulations
9. should generate report that is both human readable and machine readable – record the NETCONF session
10. bulk recording of state variables – NETCONF <get> with filter
11. be implemented based on open standards if such are available (ietf-network, ietf-interfaces)
12. processing requirements should be in the range of common embedded systems used for management of test instruments and configuration transaction should take less then 10 ms

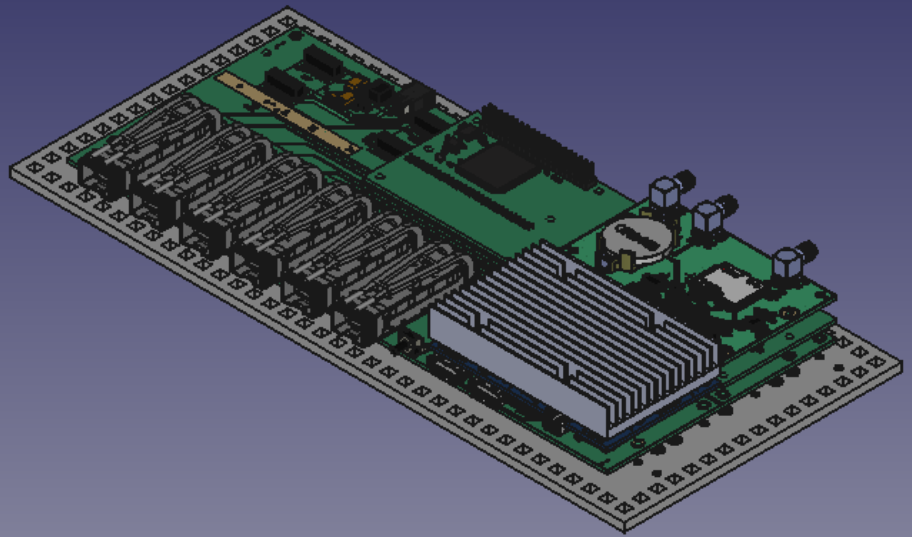
The project



Specification:
* draft-ietf-bmwg-network-tester-cfg-05

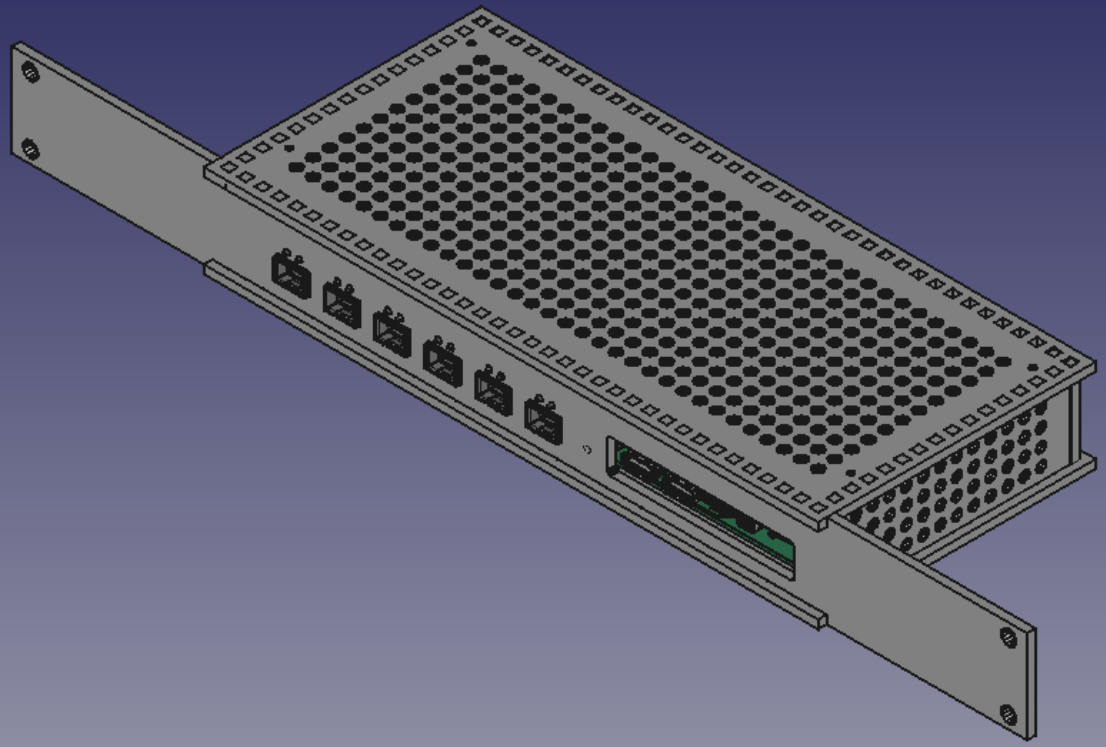
Client side examples:
* rfc2544-benchmark.py (Python)

Device side:
* Software - YANG/NETCONF server instrumentation code (C)
* Firmware - (Verilog)
* Hardware – off-the-shelf FPGA module Ultra96 + 6x SFP+ network programmability kit shield (KiCAD, Walk-through, OSHWA UIDs NO000005, NO000006)
* Pre-silicon gate level simulation with cocotb/iverilog as alternative to target hardware



OSHW
NO000005

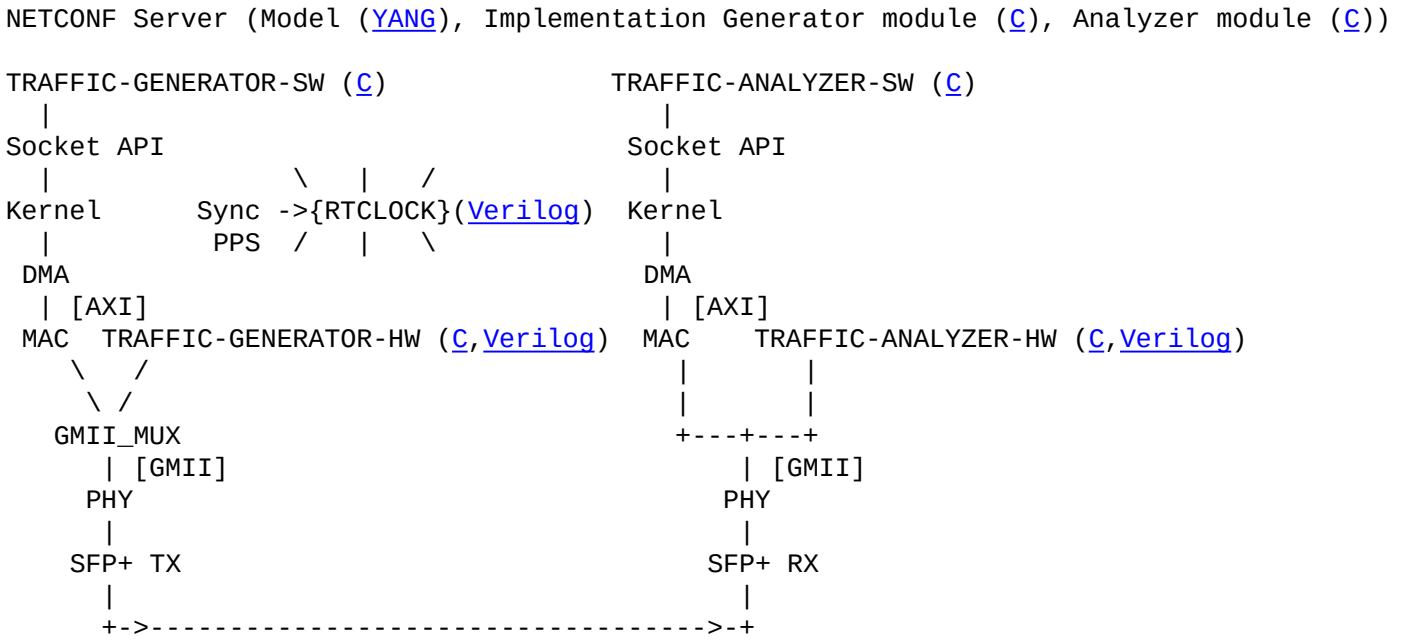
OSHW
NO000006



OSHW
NO000005

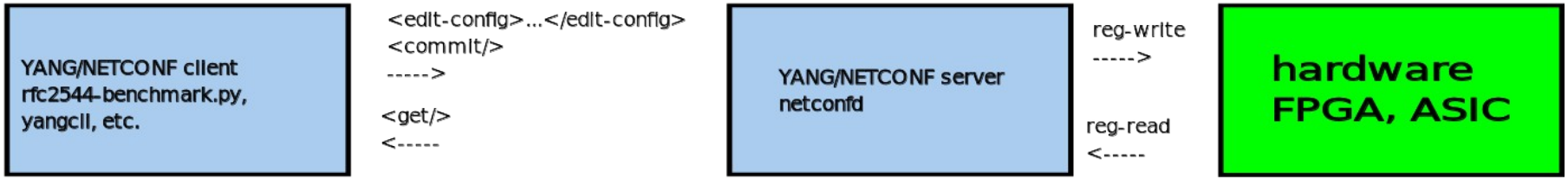
OSHW
NO000006

Design and implementation



* - underlined text has links to repositories

Model defined pre-silicon testing environment



Alternatives:
* UVM, UVVM
*
Cadence/Spirent

...
sim-run 1000
ns
sim-finish

Pre-silicon testing

In the following example a RFC2544 benchmark against a **netconfd** server implementing the model by controling a gate-level simulation of the synthesizble traffic-generator-gmii and traffic-analyzer-gmii cores in **cocotb**
sim_time_ns=1565330 (we used bogus 10 Kb Ethernet speed to actually simulate the dataplane in realtime).

We published the results in a branch :

* Waveform trace (cocotb/iverilog gate-level generated)
* Report (with verbose NETCONF transaction)

Some random screenshots taken during this process complete this presentation.