APPENDIX 1 – UVM Recorder API

```
class uvm_recorder extends uvm_object;
  integer tr_handle = 0;
  virtual function void    record_field      (string name, uvm_bitstream_t value, int size,
                                               uvm_radix_enum  radix=UVM_NORADIX);
  virtual function void    record_field_real (string name, real value);
  virtual function void    record_object  (string name, uvm_object value);
  virtual function void    record_string  (string name, string     value);
  virtual function void    record_time    (string name, time       value);
  virtual function void    record_generic(string name, string     value);
  virtual function integer create_stream  (string name, string t, string scope);
  virtual function void    set_attribute  (integer txh, string nm, logic [1023:0] value,
                                          uvm_radix_enum radix, integer numbits=1024);
  virtual function integer begin_tr (string txtype, integer stream, string nm,
                                     string label="", string desc="", time begin_time=0);
  virtual function void    end_tr   (integer handle, time end_time=0);
  virtual function void    link_tr  (integer h1, integer h2, string relation="");
  virtual function void    free_tr  (integer handle);
  virtual function bit     open_file();
  virtual function void    m_set_attribute   (integer txh, string nm, string value);
  virtual function integer check_handle_kind (string htype, integer handle);
endclass
```

APPENDIX 2 – PLI based, vendor specific API example

```
$add_attribute( trHandle tr, <SVType> value, string attribute_name);
  Add the attribute with the name and value to the transaction tr.
$add_relation(trHandle tr1, trHandle tr2, string relationship);
  tr1 is a <relationship> to tr2.
$begin_transaction(streamHandle stream, string transaction_name, time begin_time,
                   trHandle parent_transaction);
  Create a new transaction with name, starting at begin_time on the stream.
  If this is a phase child, then the parent is parent_transaction.
$create_transaction_stream(string stream_name, string stream_kind);
  Create a new stream named stream_name. stream_kind is ignored.
$delete_transaction(trHandle tr);
  Delete a transaction, tr.
$end_transaction(trHandle tr, time end_time, bit free);
  End a transaction tr and time end_time. If free is set, then free the underlying handle.
```

APPENDIX 3 – UVM Transaction API

```
function integer uvm_transaction::begin_tr (time begin_time=0);
  return m_begin_tr(begin_time, 0, 0);
endfunction

function integer uvm_transaction::m_begin_tr (time begin_time=0, integer parent_handle=0, bit has_parent=0);
  if(begin_time != 0)
    this.begin_time = begin_time;
  else
    this.begin_time = $realtime;

  // May want to establish predecessor/successor relation
  // (don't free handle until then)
  if(record_enable) begin
    if(m_recorder.check_handle_kind("Transaction", tr_handle)==1)
      end_tr();
    if(!has_parent)
      tr_handle = m_recorder.begin_tr("Begin_No_Parent, Link",
                  stream_handle, get_type_name(),"","",begin_time);
    else begin
      tr_handle = m_recorder.begin_tr("Begin_End, Link",
                  stream_handle, get_type_name(),"","",begin_time);
      if(parent_handle)
        m_recorder.link_tr(parent_handle, tr_handle, "child");
    end
    m_recorder.tr_handle = tr_handle;
    if(m_recorder.check_handle_kind("Transaction", tr_handle)!=1)
      $display("tr handle %0d not valid!",tr_handle);
  end
  else
    tr_handle = 0;
  do_begin_tr(); //execute callback before event trigger
  begin_event.trigger();
  return tr_handle;
endfunction

function void uvm_transaction::end_tr (time end_time=0, bit free_handle=1);
  if(end_time != 0)
    this.end_time = end_time;
  else
    this.end_time = $realtime;
  do_end_tr(); // Callback prior to actual ending of transaction
  if(is_active()) begin
    m_recorder.tr_handle = tr_handle;
```

```
    record(m_recorder);
    m_recorder.end_tr(tr_handle,end_time);
    if(free_handle && m_recorder.check_handle_kind("Transaction", tr_handle)==1)
    begin
      // once freed, can no longer link to
      m_recorder.free_tr(tr_handle);
    end
    tr_handle = 0;
  end
  end_event.trigger();
endfunction
```

APPENDIX 4 – UVM Component API

```
function integer uvm_component::begin_tr (uvm_transaction tr,
                                           string stream_name ="main",
                                           string label="",
                                           string desc="",
                                           time begin_time=0,
                                           integer parent_handle=0);
  return m_begin_tr(tr, parent_handle, (parent_handle!=0), stream_name, label, desc, begin_time);
endfunction

function integer uvm_component::m_begin_tr (uvm_transaction tr,
                                           integer parent_handle=0,
                                           bit    has_parent=0,
                                           string stream_name="main",
                                           string label="",
                                           string desc="",
                                           time begin_time=0);
  uvm_event e;
  integer stream_h;
  integer tr_h;
  integer link_tr_h;
  string name;
  string kind;
  uvm_recorder recordr;

  if (tr == null)
    return 0;
  recordr = (recorder == null) ? uvm_default_recorder : recorder;
  if (!has_parent) begin
    uvm_sequence_item seq;
    if ($cast(seq,tr)) begin
      uvm_sequence_base parent_seq = seq.get_parent_sequence();
      if (parent_seq != null) begin
        parent_handle = parent_seq.m_tr_handle;
        if (parent_handle!=0)
          has_parent = 1;
      end
    end
  end
  tr_h = 0;
  if(has_parent)
    link_tr_h = tr.begin_child_tr(begin_time, parent_handle);
  else
    link_tr_h = tr.begin_tr(begin_time);
  if (tr.get_name() != "")
    name = tr.get_name();
  else
    name = tr.get_type_name();
  if(stream_name == "") stream_name="main";
  if (uvm_verbosity'(recording_detail) != UVM_NONE) begin
    if(m_stream_handle.exists(stream_name))
        stream_h = m_stream_handle[stream_name];
    if (recordr.check_handle_kind("Fiber", stream_h) != 1) begin
        stream_h = recordr.create_stream(stream_name, "TVM", get_full_name());
        m_stream_handle[stream_name] = stream_h;
    end
    kind = (has_parent == 0) ? "Begin_No_Parent, Link" : "Begin_End, Link";
    tr_h = recordr.begin_tr(kind, stream_h, name, label, desc, begin_time);
    if (has_parent && parent_handle != 0)
        recordr.link_tr(parent_handle, tr_h, "child");
    m_tr_h[tr] = tr_h;
    if (recordr.check_handle_kind("Transaction", link_tr_h) == 1)
      recordr.link_tr(tr_h,link_tr_h);
    do_begin_tr(tr,stream_name,tr_h);
  end
  e = event_pool.get("begin_tr");
  if (e!=null)
    e.trigger(tr);
  return tr_h;
endfunction

function void uvm_component::end_tr (uvm_transaction tr, time end_time=0, bit free_handle=1);
  uvm_event e;
  integer tr_h;
  uvm_recorder recordr;
  if (tr == null)
    return;
  recordr = (recorder == null) ? uvm_default_recorder : recorder;
  tr_h = 0;
  tr.end_tr(end_time,free_handle);
  if (uvm_verbosity'(recording_detail) != UVM_NONE) begin
    if (m_tr_h.exists(tr)) begin
      tr_h = m_tr_h[tr];
      do_end_tr(tr, tr_h); // callback
      m_tr_h.delete(tr);
      if (recordr.check_handle_kind("Transaction", tr_h) == 1) begin
        recordr.tr_handle = tr_h;
        tr.record(recordr);
```

```
        recordr.end_tr(tr_h,end_time);
        if (free_handle)
           recordr.free_tr(tr_h);
      end
    end
    else begin
      do_end_tr(tr, tr_h); // callback
    end
  end
  e = event_pool.get("end_tr");
  if(e!=null)
    e.trigger();
endfunction
```

APPENDIX 5 – UVM Sequence API

```
virtual task start (uvm_sequencer_base sequencer,
                    uvm_sequence_base parent_sequence = null,
                    int this_priority = -1,
                    bit call_pre_post = 1);
  if (m_sequencer != null) begin
    if (m_parent_sequence == null) begin
      m_tr_handle = m_sequencer.begin_tr(this, get_name());
    end else begin
      m_tr_handle = m_sequencer.begin_child_tr(this, m_parent_sequence.m_tr_handle,
                                               get_root_sequence_name());
    end
  end
  pre_start();
  if (call_pre_post == 1) begin
    pre_body();
  end
  if (parent_sequence != null) begin
    parent_sequence.pre_do(0);    // task
    parent_sequence.mid_do(this); // function
  end
  body();
  if (parent_sequence != null) begin
    parent_sequence.post_do(this);
  end

  if (call_pre_post == 1) begin
    post_body();
  end
  post_start();
  if (m_sequencer != null) begin
    m_sequencer.end_tr(this);
  end
endtask
```