**Ethernet HVVC** –Quick Reference

**NOTE: As of UVVM v3.x, all shared variables have been made protected. This means that any access to shared variables must be done**

**using get- and set-methods. This documentation has not yet been updated with the methods for accessing these variables, but will be very soon.**

**Please refer to section 2 of Avalon\_mm\_vvc\_QuickRef for example usage of protected shared variables**

This Ethernet Hierarchical-VVC is based on IEEE 802.3. It does not support optional fields or EtherType, only length is supported.

HVVCs are different than normal VVCs since they represent a higher protocol level than the physical layer, i.e. they have no physical connections. However due to

**HVVC**

similarities in the core code, the VVC term is used instead.

For general information see UVVM Essential Mechanisms located in uvvm\_vvc\_framework/doc. **CAUTION**: shaded code/description is preliminary

****

|  |
| --- |
| ethernet\_transmit (VVCT, vvc\_instance\_idx, channel, [mac\_destination], [mac\_source], payload, msg, [scope]) |
| Example: ethernet\_transmit(ETHERNET\_VVCT, 0, TX, v\_mac\_dest, v\_mac\_src, v\_payload, “Transmit an ethernet packet”, C\_SCOPE);  Example: ethernet\_transmit(ETHERNET\_VVCT, 0, TX, v\_payload, “Transmit an ethernet packet using default MAC addresses”); |

*ethernet\_vvc.vhd*

|  |
| --- |
| ethernet\_receive (VVCT, vvc\_instance\_idx, channel, [TO\_SB], msg, [scope]) |
| Example: ethernet\_receive(ETHERNET\_VVCT, 1, RX, “Receive an ethernet packet and store it in the VVC. To be fetched later using fetch\_result()”, C\_SCOPE);  Example: ethernet\_receive(ETHERNET\_VVCT, 1, RX, TO\_SB, “Receive an ethernet packet and send to Scoreboard for checking”); |

|  |
| --- |
| ethernet\_expect (VVCT, vvc\_instance\_idx, channel, [mac\_destination], [mac\_source], payload, msg, [alert\_level, [scope]]) |
| Example: ethernet\_expect(ETHERNET\_VVCT, 1, RX, v\_mac\_dest, v\_mac\_src, v\_payload, “Expect an ethernet packet”, ERROR, C\_SCOPE); |

Ethernet VVC Configuration record **´vvc\_config´ --** accessible via **shared\_ethernet\_vvc\_config**

**Common VVC procedures applicable for this VVC**  
- See UVVM Methods QuickRef for details.

**await\_[any]completion**() **enable\_log\_msg**() **disable\_log\_msg**()

**fetch\_result**()

**flush\_command\_queue**()  
**terminate\_current\_command**() **terminate\_all\_commands**() **insert\_delay**()

**get\_last\_received\_cmd\_idx()**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Record element** | | **Type** | | **C\_ETHERNET\_VVC\_CONFIG\_DEFAULT** |
| inter\_bfm\_delay | | t\_inter\_bfm\_delay | | C\_ETHERNET\_INTER\_BFM\_DELAY\_DEFAULT |
| cmd\_queue\_count\_max | | natural | | C\_CMD\_QUEUE\_COUNT\_MAX |
| cmd\_queue\_count\_threshold | | natural | | C\_CMD\_QUEUE\_COUNT\_THRESHOLD |
| cmd\_queue\_count\_threshold\_severity | | t\_alert\_level | | C\_CMD\_QUEUE\_COUNT\_THRESHOLD\_SEVERITY |
| result\_queue\_count\_max | | natural | | C\_RESULT\_QUEUE\_COUNT\_MAX |
| result\_queue\_count\_threshold | | natural | | C\_RESULT\_QUEUE\_COUNT\_THRESHOLD |
| result\_queue\_count\_threshold\_severity | | t\_alert\_level | | C\_RESULT\_QUEUE\_COUNT\_THERSHOLD\_SEVERITY |
| bfm\_config1 | | t\_ethernet\_protocol\_config | | C\_ETHERNET\_PROTOCOL\_CONFIG\_DEFAULT |
| msg\_id\_panel | | t\_msg\_id\_panel | | C\_ETHERNET\_VVC\_MSG\_ID\_PANEL\_DEFAULT |
|  | |  | |  |
| Ethernet Protocol Configuration record ´**t\_ethernet\_protocol\_config´** | | | | | |
| **Record element** | | **Type** | | **C\_ETHERNET\_PROTOCOL\_CONFIG\_DEFAULT** | |
| mac\_destination | | unsigned(47 downto 0) | | x”000000000000” | |
| mac\_source | | unsigned(47 downto 0) | | x”000000000000” | |
| fcs\_error\_severity | | t\_alert\_level | | ERROR | |
| interpacket\_gap\_time2,3 | | time | | 96 ns | |

1 Not strictly a bus functional model (BFM) but holds BFM-like configuration data.

2 Interpacket gap is implemented as a wait statement after the ethernet packet has been

transmitted. Check of interpacket gap on receive is not implemented.

3 If the physical VVC has a timeout, e.g. max\_wait\_cycles, it must be big enough to handle

the interpacket gap and any other delays in the transmission.



|  |  |
| --- | --- |
| Record ´**t\_ethernet\_frame´** | |
| **Record element** | **Type** |
| mac\_destination | unsigned(47 downto 0) |
| mac\_source | unsigned(47 downto 0) |
| payload\_length | integer |
| payload | t\_byte\_array |
| fcs | std\_logic\_vector(31 downto 0) |

Ethernet VVC Status record signal **´vvc\_status´ --** accessible via **shared\_ethernet\_vvc\_status**

|  |  |  |
| --- | --- | --- |
| **Record element** | **Type** |  |
| current\_cmd\_idx | natural |  |
| previous\_cmd\_idx | natural |  |
| pending\_cmd\_cnt | natural |  |

VVC target parameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Example(s)** | **Description** |
| VVCT | t\_vvc\_target\_record | ETHERNET\_VVCT | VVC target type compiled into each VVC in order to differentiate between VVCs. |
| vvc\_instance\_idx | integer | 0 | Instance number of the VVC. |
| channel | t\_channel | TX, RX | The VVC channel of the VVC instance. |

VVC functional parameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Example(s)** | **Description** |
| mac\_destination | unsigned(47 downto 0) | x”00\_00\_00\_00\_00\_02” | The MAC address of destination. |
| mac\_source | unsigned(47 downto 0) | x”00\_00\_00\_00\_00\_01” | The MAC address of source. |
| payload | t\_byte\_array | (x”01”, x”23”, x”45”, x”AB”, x”CD”) | The payload of the packet. |
| alert\_level | t\_alert\_level | ERROR or TB\_WARNING | Set the severity for the alert that may be asserted by the procedure. |
| msg | string | “Send to DUT” | A custom message to be appended in the log/alert. |
| scope | string | “Ethernet\_VVC” | A string describing the scope from which the log/alert originates. |

VVC entity generic constants

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| GC\_INSTANCE\_IDX | natural | - | Instance number to assign the VVC. |
| GC\_PHY\_INTERFACE | t\_interface | - | Physical VVC interface type, e.g. SBI, GMII. (see note below) |
| GC\_PHY\_VVC\_INSTANCE\_IDX | natural | - | Instance number of the physical VVC. |
| GC\_PHY\_MAX\_ACCESS\_TIME | time | 1 us | Maximum time that the physical VVC takes to execute an access, e.g. GMII write 1 byte. It should also account for any margin it needs. |
| GC\_DUT\_IF\_FIELD\_CONFIG | t\_dut\_if\_field\_config\_direction\_array | C\_DUT\_IF\_FIELD\_CONFIG\_DIRECTION\_  ARRAY\_DEFAULT | Array of configurations for address based VVC interfaces. See chapter 0 for details. |
| GC\_ETHERNET\_PROTOCOL\_CONFIG | t\_ethernet\_protocol\_config | C\_ETHERNET\_PROTOCOL\_CONFIG\_DEFAULT | Configuration of the Ethernet protocol. |
| GC\_CMD\_QUEUE\_COUNT\_MAX | natural | 1000 | Absolute maximum number of commands in the VVC command queue |
| GC\_CMD\_QUEUE\_COUNT\_THRESHOLD | natural | 950 | An alert will be generated when reaching this threshold to indicate that the command queue is almost full. The queue will still accept new commands until it reaches C\_CMD\_QUEUE\_COUNT\_MAX. |
| GC\_CMD\_QUEUE\_COUNT\_THRESHOLD\_SEVERITY | t\_alert\_level | WARNING | Alert severity which will be used when command queue reaches GC\_CMD\_QUEUE\_COUNT\_THRESHOLD. |
| GC\_RESULT\_QUEUE\_COUNT\_MAX | natural | 1000 | Maximum number of unfetched results before result\_queue is full. |
| GC\_RESULT\_QUEUE\_COUNT\_THRESHOLD | natural | 950 | An alert with severity 'result\_queue\_count\_threshold\_severity' will be issued if result queue exceeds this count. Used for early warning if result queue is almost full. Will be ignored if set to 0. |
| GC\_RESULT\_QUEUE\_COUNT\_THRESHOLD\_SEVERITY | t\_alert\_level | WARNING | Severity of alert to be initiated if exceeding result\_queue\_count\_threshold |

Note that you can use any of the physical interfaces already implemented just by using the appropriate name in GC\_PHY\_INTERFACE and instantiating the corresponding VVC in the testbench (in addition to the HVVC). For more information see *UVVM Essential Mechanisms* located in uvvm\_vvc\_framework/doc.

If you however want to use an interface type which is not already included, see *HVVC\_to\_VVC\_Bridge\_Implementation\_Guide* located in bitvis\_vip\_hvvc\_to\_vvc\_bridge/doc for more info.

VVC details

All VVC procedures are defined in vvc\_methods\_pkg (dedicated to this VVC), and uvvm\_vvc\_framework.td\_vvc\_framework\_common\_methods\_pkg (common VVC procedures).   
It is also possible to send a multicast to all instances of a VVC with ALL\_INSTANCES as parameter for vvc\_instance\_idx.   
*Note: Every procedure here can be called without the optional parameters enclosed in [ ].*

# VVC procedure details and examples

|  |  |
| --- | --- |
| **Procedure** | **Description** |
| **ethernet\_transmit()** | **ethernet\_transmit (VVCT, vvc\_instance\_idx, channel, [mac\_destination], [mac\_source], payload, msg, [scope])**  The ethernet\_transmit() VVC procedure adds a transmit command to the Ethernet VVC executor queue, which runs as soon as all preceding commands have completed. When the command is scheduled to run, the executor calls the priv\_ethernet\_transmit\_to\_bridge() procedure. This procedure builds an Ethernet packet and transmits each field using the HVVC-to-VVC bridge which then transfers the data to the lower level VVC (physical interface). After it has finished, it waits for the configured interpacket gap time. |
| **ethernet\_receive()** | **ethernet\_receive (VVCT, vvc\_instance\_idx, channel, [TO\_SB], msg, [scope])**  The ethernet\_receive() VVC procedure adds a receive command to the Ethernet VVC executor queue, which runs as soon as all preceding commands have completed. When the command is scheduled to run, the executor calls the priv\_ethernet\_receive\_from\_bridge() procedure. This procedure receives an Ethernet packet by requesting each field from the HVVC-to-VVC bridge which calls the lower level VVC (physical interface) to read the data. When the complete packet is received, it computes the FCS and checks that it corresponds to the one received in the packet.  The received data from the DUT is not to be returned in this procedure call since it is non-blocking for the sequencer/caller, but it will be stored in the VVC for a potential future fetch (see example with *fetch\_result* below).  If the option TO\_SB is applied, the received data will be sent to the Ethernet VVC dedicated scoreboard. There, it is checked against the expected value (provided by the testbench).  **Example with fetch\_result() call**: Result is placed in **v\_result**  variable v\_cmd\_idx : natural; -- Command index for the last receive  variable v\_result : bitvis\_vip\_ethernet.vvc\_cmd\_pkg.t\_vvc\_result; -- Result from receive.  (…)  ethernet\_receive(ETHERNET\_VVCT, 1, RX, “Receive ethernet packet”);  v\_cmd\_idx := get\_last\_received\_cmd\_idx(ETHERNET\_VVCT, 1, RX);  await\_completion(ETHERNET\_VVCT, 1, RX, v\_cmd\_idx, 1 us, "Wait for receive to finish");  fetch\_result(ETHERNET\_VVCT, 1, RX, v\_cmd\_idx, **v\_result**, "Fetching result from receive operation"); |
| **ethernet\_expect()** | **ethernet\_expect (VVCT, vvc\_instance\_idx, channel, [mac\_destination], [mac\_source], payload, msg, [alert\_level, [scope]])**  The ethernet\_expect() VVC procedure adds an expect command to the Ethernet VVC executor queue, which runs as soon as all preceding commands have completed. When the command is scheduled to run, the executor calls the priv\_ethernet\_expect\_from\_bridge() procedure. This procedure performs a receive operation, then checks if the received data is equal to the expected data. The received data is not stored in this procedure. |

# VVC Configuration

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Record element** | **Type** | **C\_ETHERNET\_VVC\_CONFIG\_DEFAULT** | | **Description** |
| inter\_bfm\_delay | t\_inter\_bfm\_delay | C\_ETHERNET\_INTER\_BFM\_DELAY\_DEFAULT | Delay between any requested BFM accesses towards the DUT. - TIME\_START2START: Time from a BFM start to the next BFM start   (A TB\_WARNING will be issued if access  takes longer than TIME\_START2START).  - TIME\_FINISH2START: Time from a BFM end to the next BFM start. Any insert\_delay() command adds to the above minimum delays, giving for instance the ability to skew the BFM starting time. | |
| cmd\_queue\_count\_max | natural | C\_CMD\_QUEUE\_COUNT\_MAX | Maximum pending number in command queue before queue is full. Adding additional commands will result in an ERROR. | |
| cmd\_queue\_count\_threshold | natural | C\_CMD\_QUEUE\_COUNT\_THRESHOLD | An alert with severity “cmd\_queue\_count\_threshold\_severity” will be issued if command queue exceeds this count. Used for early warning if command queue is almost full. Will be ignored if set to 0. | |
| cmd\_queue\_count\_threshold\_severity | t\_alert\_level | C\_CMD\_QUEUE\_COUNT\_THRESHOLD\_SEVERITY | Severity of alert to be initiated if exceeding cmd\_queue\_count\_threshold | |
| result\_queue\_count\_max | natural | C\_RESULT\_QUEUE\_COUNT\_MAX | Maximum number of unfetched results before result\_queue is full. | |
| result\_queue\_count\_threshold | natural | C\_RESULT\_QUEUE\_COUNT\_THRESHOLD | An alert with severity 'result\_queue\_count\_threshold\_severity' will be issued if result queue exceeds this count. Used for early warning if result queue is almost full. Will be ignored if set to 0. | |
| result\_queue\_count\_threshold\_severity | t\_alert\_level | C\_ RESULT\_QUEUE\_COUNT\_THRESHOLD\_SEVERITY | Severity of alert to be initiated if exceeding result\_queue\_count\_threshold | |
| bfm\_config | t\_ethernet\_protocol\_config | C\_ETHERNET\_PROTOCOL\_CONFIG\_DEFAULT | Configuration of the Ethernet protocol. | |
| msg\_id\_panel | t\_msg\_id\_panel | C\_ETHERNET\_VVC\_MSG\_ID\_PANEL\_DEFAULT | VVC dedicated message ID panel. See section 16 of uvvm\_vvc\_framework/doc/UVVM\_VVC\_Framework\_Essential\_Mechanisms.pdf for how to use verbosity control. | |

# The configuration record can be accessed from the Central Testbench Sequencer through the shared variable array, e.g.:

shared\_ethernet\_vvc\_config(1).inter\_bfm\_delay.delay\_in\_time := 50 ns;

shared\_ethernet\_vvc\_config(1).bfm\_config.interpacket\_gap\_time := 96 ns;

# VVC Status

The current status of the VVC can be retrieved during simulation. This is achieved by reading from the shared variable shared\_ethernet\_vvc\_status record from the test sequencer. The record contents can be seen below:

|  |  |  |
| --- | --- | --- |
| **Record element** | **Type** | **Description** |
| current\_cmd\_idx | natural | Command index currently running |
| previous\_cmd\_idx | natural | Previous command index to run |
| pending\_cmd\_cnt | natural | Pending number of commands in the command queue |

# Activity watchdog

The VVCs support a centralized VVC activity register which the activity watchdog uses to monitor the VVC activities. The VVCs will register their presence to the VVC activity register at start-up, and report when ACTIVE and INACTIVE, using dedicated VVC activity register methods, and trigger the global\_trigger\_vvc\_activity\_register signal during simulations. The activity watchdog is continuously monitoring the VVC activity register for VVC inactivity and raises an alert if no VVC activity is registered within the specified timeout period.

Include activity\_watchdog(num\_exp\_vvc, timeout, [alert\_level, [msg]]) in the testbench to start using the activity watchdog.   
Note that setting the exact number of expected VVCs in the VVC activity register can be omitted by setting num\_exp\_vvc = 0.

More information can be found in UVVM Essential Mechanisms PDF in the UVVM VVC Framework doc folder.

# Transaction Info

This VVC supports transaction info, a UVVM concept for distributing transaction information in a controlled manner within the complete testbench environment. The transaction info may be used in many different ways, but the main purpose is to share information directly from the VVC to a DUT model.

Table 5.1 Ethernet transaction info record fields. Transaction type: t\_base\_transaction (BT) **-** accessible via **shared\_ethernet\_vvc\_transaction\_info.bt**.

|  |  |  |  |
| --- | --- | --- | --- |
| **Info field** | **Type** | **Default** | **Description** |
| operation | t\_operation | NO\_OPERATION | Current VVC operation, e.g. INSERT\_DELAY, POLL\_UNTIL, READ, WRITE. |
| ethernet\_frame | t\_ethernet\_frame | C\_ETHERNET\_FRAME\_DEFAULT | Ethernet Frame. |
| vvc\_meta | t\_vvc\_meta | C\_VVC\_META\_DEFAULT | VVC meta data of the executing VVC command. |
| **→** msg | string | “ “ | Message of executing VVC command. |
| **→** cmd\_idx | integer | -1 | Command index of executing VVC command. |
| transaction\_status | t\_transaction\_status | C\_TRANSACTION\_STATUS\_DEFAULT | Set to INACTIVE, IN\_PROGRESS, FAILED or SUCCEEDED during a transaction. |

See UVVM VVC Framework Essential Mechanisms PDF, section 6, for additional information about transaction types and transaction info usage.

# Scoreboard

This VVC has built in Scoreboard functionality where data can be routed by setting the TO\_SB parameter in supported method calls, i.e. ethernet\_receive(). Note that the data is only stored in the scoreboard and not accessible with the fetch\_result() method when the TO\_SB parameter is applied.

See the Generic Scoreboard Quick Reference PDF in the Bitvis VIP Scoreboard document folder for a complete list of available commands and additional information. The Ethernet scoreboard is accessible from the testbench as a shared variable ETHERNET\_VVC\_SB, located in the vvc\_methods\_pkg.vhd. All of the listed Generic Scoreboard commands are available for the Ethernet VVC scoreboard using this shared variable.

# DUT interface field configuration

The table below shows which index in the DUT IF field configuration array the Ethernet fields are associated with. These configurations are only necessary when the lower level VVC is address-based, e.g. SBI. The DUT IF field configuration array is a two-dimensional array (direction and index). If the same configuration is used for all fields, only one configuration per direction is needed. The highest indexed configuration is used for indexes higher than those supplied. E.g. if the array consists of two configurations the first configuration, index 0, is used for the field preamble & SFD and the other fields use the last configuration, index 1. Each index holds an element of type t\_dut\_if\_field\_config, see table below.

|  |  |  |
| --- | --- | --- |
| **Record** ´**t\_dut\_if\_field\_config´** | | |
| **Record element** | **Type** | **Description** |
| dut\_address | unsigned | Address of the DUT IF field. |
| dut\_address\_increment | integer | Incrementation of the address on each access. |
| data\_width | positive | Width of the data per transfer, must be <= bus width. |
| use\_field | boolean | Used by the HVVC to send/request fields to/from the HVVC-to-VVC bridge or ignore them when not applicable. |
| field\_description | string | Description of the DUT IF field. |

**The Ethernet interface fields are associated with the following indexes**

|  |  |  |
| --- | --- | --- |
| **Ethernet field** | **Name** | **Index** |
| Preamble & SFD | C\_FIELD\_IDX\_PREAMBLE\_AND\_SFD | 0 |
| MAC destination | C\_FIELD\_IDX\_MAC\_DESTINATION | 1 |
| MAC source | C\_FIELD\_IDX\_MAC\_SOURCE | 2 |
| Payload length | C\_FIELD\_IDX\_PAYLOAD\_LENGTH | 3 |
| Payload | C\_FIELD\_IDX\_PAYLOAD | 4 |
| FCS | C\_FIELD\_IDX\_FCS | 5 |

# Additional Documentation

Additional documentation about UVVM and its features can be found under “/uvvm\_vvc\_framework/doc/”.

# Compilation

The Ethernet VVC must be compiled with VHDL 2008.   
It is dependent on the following libraries

* ***UVVM Utility Library (UVVM-Util), version 2.15.0 and up***
* ***UVVM VVC Framework, version 2.11.0 and up***
* ***Bitvis VIP Scoreboard***
* ***Library of the physical interface used (e.g. Bitvis VIP GMII)***
* ***HVVC-to-VVC Bridge***

Before compiling the Ethernet VVC, assure that uvvm\_vvc\_framework, uvvm\_util and bitvis\_vip\_scorebord have been compiled.

See UVVM Essential Mechanisms located in uvvm\_vvc\_framework/doc for information about compile scripts.

**Compile order for the Ethernet VVC:**

|  |  |  |
| --- | --- | --- |
| **Compile to library** | **File** | **Comment** |
| bitvis\_vip\_ethernet | support\_pkg.vhd | Ethernet support package |
| bitvis\_vip\_ethernet | transaction\_pkg.vhd | Ethernet transaction package with DTT types, constants, etc. |
| bitvis\_vip\_ethernet | vvc\_cmd\_pkg.vhd | Ethernet VVC command types and operations |
| bitvis\_vip\_ethernet | ethernet\_sb\_pkg.vhd | Ethernet Scoreboard package |
| bitvis\_vip\_ethernet | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_target\_support\_pkg.vhd | UVVM VVC target support package, compiled into bitvis\_vip\_ethernet library. |
| bitvis\_vip\_ethernet | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_vvc\_framework\_common\_methods\_pkg.vhd | UVVM framework common methods compiled into bitvis\_vip\_ethernet library |
| bitvis\_vip\_ethernet | vvc\_methods\_pkg.vhd | Ethernet VVC methods |
| bitvis\_vip\_ethernet | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_queue\_pkg.vhd | UVVM queue package for the VVC |
| bitvis\_vip\_ethernet | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_vvc\_entity\_support\_pkg.vhd | UVVM VVC entity support compiled into bitvis\_vip\_ethernet library |
| bitvis\_vip\_ethernet | ethernet\_rx\_vvc.vhd | Ethernet RX VVC |
| bitvis\_vip\_ethernet | ethernet\_tx\_vvc.vhd | Ethernet TX VVC |
| bitvis\_vip\_ethernet | ethernet\_vvc.vhd | Ethernet VVC |
|  |  |  |

# Simulator compatibility and setup

See README.md for a list of supported simulators.

For required simulator setup see ***UVVM-Util*** Quick reference.

IMPORTANT

This is a simplified Verification IP (VIP) for Ethernet. This Ethernet VVC is based on IEEE 802.3. It does not support optional fields or EtherType, only length is supported. This VIP is not an Ethernet protocol checker. For a more advanced VIP please contact Bitvis AS at [support@bitvis.no](mailto:support@bitvis.no)

Disclaimer: This IP and any part thereof are provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement.  
In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with this IP.

**INTELLECTUAL**

**PROPERTY**