

## AlliedWare Plus™ OS

# Overview of | Virtual Chassis Stacking (VCStack™)

## Introduction

Virtual Chassis Stacking (VCStack™) is the name given to two or more Allied Telesis switches that are configured to operate as a single switch. From a configuration and management point of view, it is as though the switches are just one device with a seamless transition from the ports of one stack member to the ports of the next.

When configuring a VCStack, there are no limitations on how the ports of one stack member can interact with the ports of another stack member—they can belong to VLANs together, they can belong to port aggregations together, they can mirror to each other, and port-range configuration can span multiple stack members. The stack member ports truly operate as though they all belong to one virtual switch.

The same applies with Layer 2 and Layer 3 switching (both unicast and multicast). The stack operates as a single device and is not perceived by end users, or the traffic itself, to be any more than a single network node.

There are some limitations to the seamlessness of virtual chassis stacking; for example, the file systems on the individual stack members remain discrete.

This Overview explains the physical creation of a VCStack, the configuration required on stack members, and how to monitor the operation of the VCStack. It also provides an understanding of how the stack behaves when a stack member stops responding.



### List of terms:

#### **Stack member**

An individual switch that is part of a VCStack.

#### **VCStack or stack**

A group of two or more switches operating as a single switch.

#### **Stack master or active master**

The switch that manages the stack.

## Contents

Introduction .....	1
Contents .....	2
Which products and software versions does this Overview apply to? .....	3
Connecting switches into a stack .....	3
Front-port stacking using XEM-STKs on x900 Series switches .....	4
Back-port stacking on SwitchBlade x908 switches .....	5
AT-StackXG slide-in modules on x600 Series switches .....	6
Connecting the cables to the switches .....	7
Restrictions on stacking combinations, connections, and sizes .....	8
How the stack communicates .....	9
Roles of each switch in a stack .....	9
Selecting the active master .....	10
Identifying each switch with stack IDs .....	11
Displaying the stack IDs .....	11
Assigning stack IDs .....	12
Caution with stack ID renumbering .....	15
Steps to set up a VCStack .....	15
Steps to replace a stack member .....	19
Provisioning .....	20
Provisioning a bay .....	21
Provisioning a switch .....	24
Re-provisioning .....	26
Configuring the stack .....	26
Port numbering .....	27
VLAN and IP subnet restrictions .....	27
Quality of Service (QoS) restriction .....	27
Stacking triggers .....	28
Software and configuration file synchronisation .....	28
Software release auto-synchronisation .....	29
Shared running configuration .....	30
Shared startup configuration .....	31
Scripts .....	31
Rolling Reboot .....	32
Failover and recovery .....	33
The resiliency link feature .....	33
Virtual MAC .....	35
Repairing a broken stack .....	35
Executing commands and managing files on a specific stack member .....	35
Executing commands .....	36
Managing files .....	38

Remote Login .....	38
Monitoring and troubleshooting .....	41
Checking stack status .....	41
Stack debug output .....	44
Counters .....	51

## Which products and software versions does this Overview apply to?

This Overview applies to the following Allied Telesis switches running the AlliedWare Plus™ operating system, software version 5.3.1 or later.

---

**Note:** The Provisioning, Rolling Reboot, and Remote Login features are available on software version 5.3.4 and above.

---

- SwitchBlade x908 switches
- x900 Series switches
- x600 Series switches

## Connecting switches into a stack

---

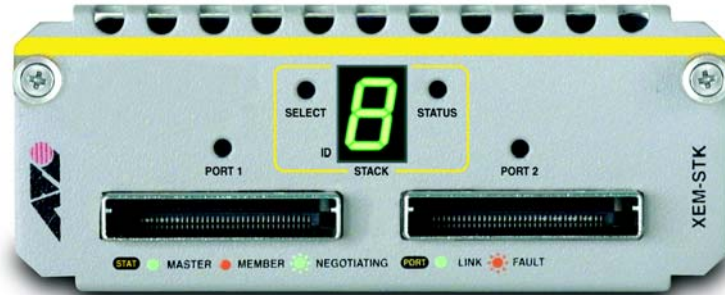
The proprietary high-speed communication protocol that is used over the stacking links requires multiple twisted pairs and a high level of shielding. This means that to stack x-Series switches, specialised cables and connections are required.

The types of cables and connections available are dependant on the type of x-Series switches you are stacking. The stacking options are:

- **Front-port stacking using XEM-STKs on x900 Series switches**
- **Back-port stacking on SwitchBlade x908 switches**
- **AT-StackXG slide-in modules on x600 Series switches**

## Front-port stacking using XEM-STKs on x900 Series switches

You can fit the XEM bays on x900 Series switches with a specialised stacking XEM called the XEM-STK.



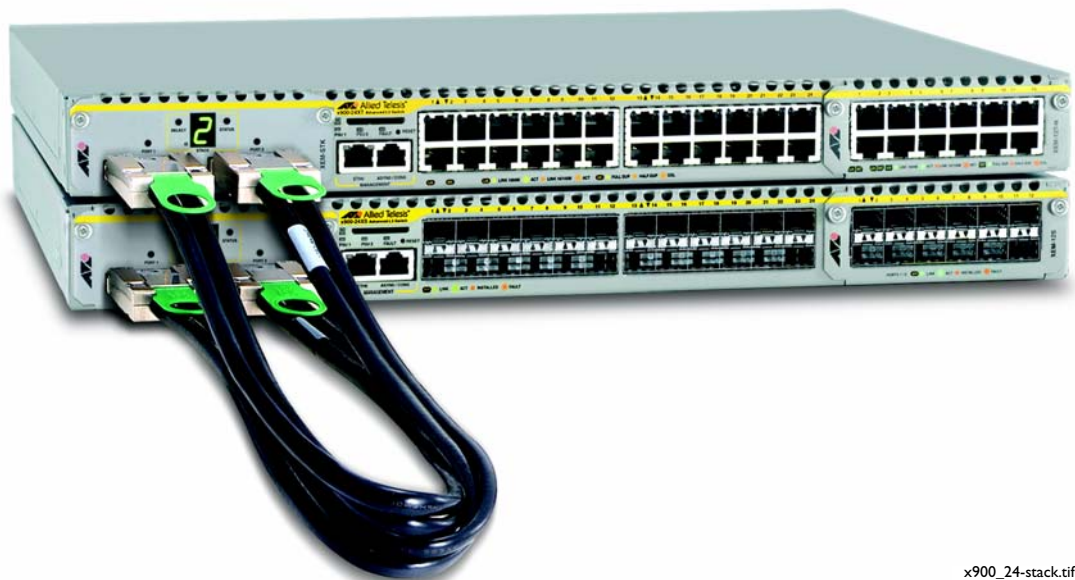
XEM-STK\_front.tif

The LED number on the front of the XEM-STK shows the stack ID of the switch that the XEM-STK is installed in. See the section ["Identifying each switch with stack IDs"](#) on page 11 for more information about stack IDs.

The specific cable type that connects these XEMs are purchased individually as either 0.5 or 2 metre long cables. The product codes are:

- AT-XEM-STK-CBL0.5
- AT-XEM-STK-CBL2.0

With these XEMs and cables, you can create a stack of up to two x900 Series switches.

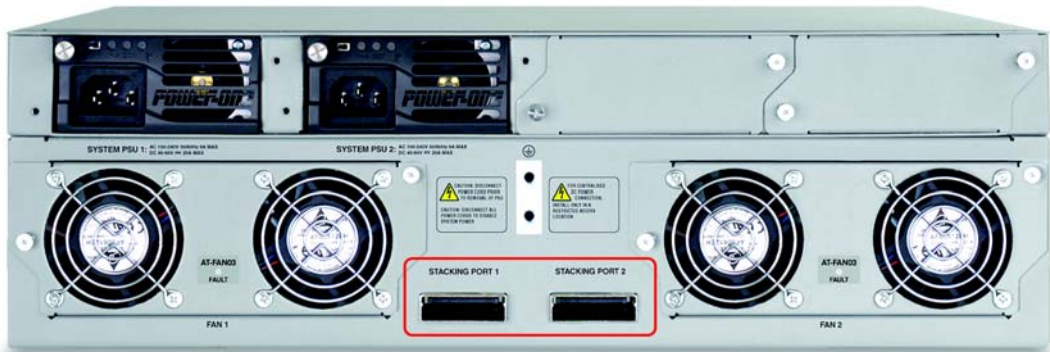


x900\_24-stack.tif

Although you can use XEM-STKs with x908 switches, we do not recommend doing so, as back-port stacking performs significantly better on SwitchBlade x908 switches. Front-port stacking also uses 2 XEM slots which you could use for other XEM modules.

## Back-port stacking on SwitchBlade x908 switches

On the rear of the SwitchBlade x908 chassis, there is a pair of fixed stacking ports.



SBx908rear\_ports.tif

Back port stacking requires a specific cable type, different than the cable required for the XEM-STK. The product code is AT-HS-STK-CBL1.0.

You can stack two SwitchBlade x908 switches together using these ports and cables.



SBx908rearstacked.tif

Note that the cables are crossed over—port 1 of the top switch is connected to port 2 of the bottom switch, and vice versa.

## AT-StackXG slide-in modules on x600 Series switches

On the rear of the x600 chassis you can insert a slide-in module called the AT-StackXG.



ATstackXG\_front.tif

The specific cable type that connects the AT-StackXG are purchased as either 0.5 or 1 metre long cables. The product codes are:

- AT-STACKXG/0.5
- AT-STACKXG/1

You can stack up to 4 x600 switches using the AT-StackXG.



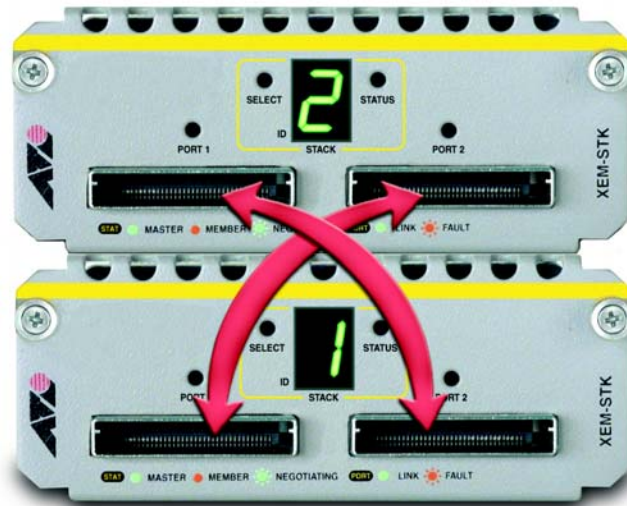
x600-4stackedrear.tif

Like the other stacking methods, the connections are crossed-over—port 1 on one switch is connected to port 2 on its neighbour and the switches are connected in a ring.

## Connecting the cables to the switches

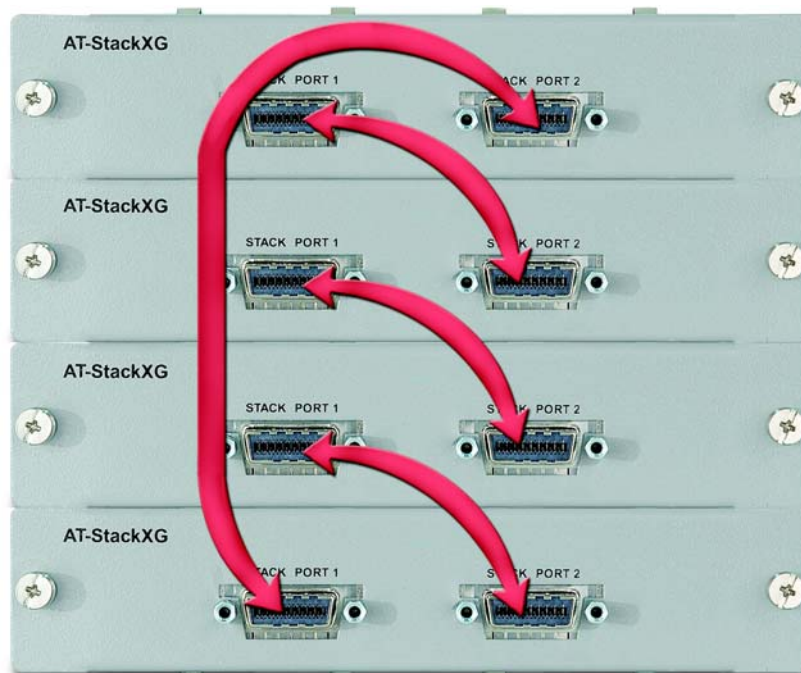
Stacked switches are connected in a ring, with port 1 of one switch connected to port 2 of the next switch. The last switch in the stack then connects to the first switch using port 2 of the first switch and port 1 of the last switch.

In two switch stacks, this means that the connection consists of just a crossed pair of cables. You may also use only one cable to connect the switches, but you will halve the bandwidth between stack members.



xem\_crossover.tif

The following figure shows how to connect a 4 switch stack of x600 Series switches. Once again, you could connect the switches without one of the cables, but you would halve the bandwidth between stack members.



ATstackXG\_4stack.tif

## Restrictions on stacking combinations, connections, and sizes

There are some restrictions to what products and connections you can combine in a single stack. In general:

- different switch families cannot be stacked together
- you cannot combine different stacking methods or cables

### x900 Series restrictions

<b>compatible products</b>	x900 Series switches can only stack with other x900 switches. Within the x900 Series: <ul style="list-style-type: none"> <li>• x900-12XT/S switches can only stack with x900-12XT/S switches</li> <li>• x900-24XS and x900-24XT can stack together</li> </ul>
<b>maximum stack size</b>	2 switches.
<b>cable type</b>	AT-XEM-STK-CBL0.5 (0.5 metre length) or AT-XEM-STK-CBL2.0 (2 metre length).

### SwitchBlade x908 restrictions

<b>compatible products</b>	A SwitchBlade x908 switch can only stack with another SwitchBlade x908 switch.
<b>maximum stack size</b>	2 switches.
<b>cable type</b>	For back-port stacking: AT-HS-STK-CBL1.0.

It does not matter what type of XEMs are installed in each SwitchBlade x908, nor where each XEM is installed in the switch. The stack cannot contain a mixture of back-port stacking and XEM stacking. This means that it is not possible to stack two x908 switches together with back-port stacking, and then connect one of those x908 switches to another x908 using XEM stacking.

### x600 Series restrictions

<b>compatible products</b>	x600 Series switches can only stack with other x600 switches. Within the x600 Series, you can stack any model with another model. This means that the x600-24Ts, x600-24Ts/XP, x600-24Ts/XP-POE, x600-48Ts, and x600-48Ts/XP switches can all stack together without restriction.
<b>maximum stack size</b>	4 switches.
<b>cable type</b>	AT-STACKXG/0.5 (0.5 metre length) or AT-STACKXG/1 (1 metre length).

## How the stack communicates

---

The stack communicates through the stacking cables. You can also configure a resiliency link between stack members, which is used when a failover event occurs on the stack. See the section "[Failover and recovery](#)" on [page 33](#) for more information.

One switch controls all switch management on a stack. Which stack member does this is discussed in "[Roles of each switch in a stack](#)" on [page 9](#).

The software version, startup configuration, and running configuration are exactly the same on each member of a stack. For information on how the stack synchronises the files, see the section "[Software and configuration file synchronisation](#)" on [page 28](#).

The internal communication between stack members is carried out using IP packets sent over the stacking links. This stack management traffic is tagged with a specific VLAN ID and uses IP addresses in a specified subnet. The default is:

- VLAN 4094
- Subnet 192.168.255.0/28

The management traffic is queued to egress queue 7 on the stack link. The section "[Configuring the stack](#)" on [page 26](#) discusses the minor restrictions necessary when configuring VLANs, IP subnets, and QoS on the stack.

## Roles of each switch in a stack

---

Each switch in a stack acts in one role at any time. This is either as a backup member (also called stack member) or a stack master (normally as the active master). The stack members are controlled by the stack master. All configuration, including updating the switch software, is set on the stack members by the stack master.

The stack master performs a number of tasks that a stack member does not perform:

- It controls all switch management activity.
- It synchronises boot release and configuration files with stack members.
- All routing protocol packets are processed by the stack master. The stack master then transfers any requisite table updates to the stack members.

The stack master also handles communication on behalf of the stack.

- When you telnet or SSH to the stack, you connect to a process running on the stack master.
- When you connect to the asyn port on a stack member, this automatically creates an SSH session to the master. This connection will behave as if you were connected to the asyn port on the master.
- The stack master handles SNMP interactions. It gathers MIB statistics from the stack members, and delivers these statistics in response to SNMP Get requests.

You can still execute some specific commands and manage files on an individual switch in the stack. See the section ["Executing commands and managing files on a specific stack member" on page 36](#) for more information.

When a VCSStack is operating normally, the stack master acts as the active master. However during some failover and resiliency situations, when a stack member cannot contact the active master, it may act as either a disabled master, or fallback master. See the section ["Failover and recovery" on page 33](#) for more information about the differences between these stack master roles.

## Selecting the active master

The stack members negotiate among themselves as to which switch will become the active master. The election of the active master is based on two criteria:

- each stack member's priority setting
- each stack member's MAC address

For each of these criteria, a lower number indicates a higher priority. The active master is the switch with the lowest priority value. If multiple switches share the lowest priority value, then the switch with the lowest MAC address becomes the active master.

### Manually changing the active master

You can choose a specific switch to be active master by changing its priority value. By default, a switch has a stack priority value of 128. The command to change a switch's priority value for stacking is:

```
stack <switch stack ID> priority <0-255>
```

The stack ID is a unique identifier that is assigned to each switch in the stack. See the section ["Identifying each switch with stack IDs" on page 11](#) for more information.

If a switch is already part of a stack, you can still set the priority value on each switch in the stack through the active master. However, if you set the priority value on a stack member lower than the active master's priority value, the active master does not immediately relinquish its role as master. The stack member with the lowest priority will take over as active master only if the current active master leaves the stack (this includes any reboot by the stack master).

If a switch has not yet joined a stack, you can still use the **stack priority** command to change the priority value before connecting it to the stack. However, even if the new stack member has a lower priority value than the active master, it will not take over as active master unless the current active master is removed or rebooted.

## Identifying each switch with stack IDs

---

Each switch in a stack has an ID number, which can be an integer between 1 and 8. The default on each switch is a stack ID of 1.

The ID number of a stack member is an important identifier. All commands that are port or switch specific need to use the stack ID to identify which stack member the commands apply to.

The stack ID is also used to manage specific stack members. When you telnet or SSH to the stack, your login session terminates on the active master. Similarly, if you connect to the console port of any stack member, your console login session is sent through to the active master. So, the active master switch is the automatic point of contact for any management session on the stack. If you want to examine something that physically resides on one of the other stack members, such as files in a stack member's flash, or voltage levels on a stack member's power supply, then you can do this by specifying the stack ID of the stack member in commands.

The stack IDs are used frequently in the stack configuration scripts. For example, any command in the configuration script that refers to a physical port will include a stack ID in the port number. The section "[Port numbering](#)" on [page 27](#) explains the port numbering scheme used in stacks.

For these reasons, the stack IDs on each switch within a stack are unique and a switch's stack ID normally does not change without user intervention. The only exception to this is when a new switch is connected to a stack and the switch has the same stack ID number as another stack member—the new switch's ID will be automatically renumbered.

There is no connection between stack ID and active master status. The active master switch does not have to be the switch with a Stack ID of 1.

## Displaying the stack IDs

To see the stack ID on a switch before it is connected to a stack, use the command:

```
show stack
```

The output will show the current stack ID and any pending change to the stack ID.

```
x900#sh stack
Virtual Chassis Stacking summary information

ID   Pending ID   MAC address           Priority   Status   Role
1    -             0000.cd27.c4bf       128      Ready   Active Master

Operational Status           Standalone unit
Stack MAC address            0000.cd27.c4bf
```

You can also use this command to see the stack numbers on a two-switch stack. To match the physical switch with the right stack ID number, look for the active master LED on the front panel. Then use the **show stack** command to show all members of the stack. You can match the LED status to the **show** command output.

```
x900#sh stack
Virtual Chassis Stacking summary information

ID   Pending ID   MAC address           Priority   Status   Role
1    -             0000.cd27.c4bf       128      Ready   Active Master
2    -             0000.cd28.0801       128      Syncing Backup Member

Operational Status           Normal operation
Stack MAC address            0000.cd27.c4bf
```

The **show stack** command is available on all stacking switches running the AlliedWare Plus operating system.

On the front of the XEM-STK there is an indicator that displays the stack ID of the switch that the XEM is installed in.

On x600 Series switches, you can use the command:

```
show stack indicator <1-8>|all [time <1-500>]
```

This causes the master LED on the switch to flash in a sequence which indicates the stack ID number. For example, on a four-switch stack with the stack IDs 1, 2, 3, and 4, the switch with stack ID:

- 1 will flash on and off without pausing      \*\*\*\*\*
- 2 will flash twice then pause                \*\* \*\* \*\* \*\* \*\*
- 3 will flash three times then pause        \*\*\*    \*\*\*    \*\*\*
- 4 will flash four times then pause        \*\*\*\*   \*\*\*\*   \*\*\*\*

You can extend the time that the master LEDs will flash by up to 500 seconds to give you more time to reach the stack's location, by using the optional time parameter with the command. By default the LEDs will flash for 5 seconds.

## Assigning stack IDs

Stack IDs can be assigned in a number of ways. We recommend only assigning stack IDs when you set up the stack, as a change to stack ID numbers is not automatically reflected in configuration scripts.

You can assign stack IDs to switches before they are part of a stack:

- **Manual assignment on a switch before stacking**

To assign stack IDs once the stack is created, you can use the following methods:

- **Automatic assignment as a switch joins the stack**

- **Manual renumbering of a switch after stacking**
- **Cascade renumbering of the stack**
- **Renumbering the whole stack using the XEM Select button**

#### Manual assignment on a switch before stacking

You can manually assign the stack IDs on switches before you stack them together.

If your switch has never had a stack ID assigned to it, it will have the stack ID of 1. You can assign it a new stack ID with the command:

```
stack (config)#stack 1 renumber <1-8>
```

If the switch has previously been in a stack and was assigned a stack ID, it keeps that stack ID with it, even if it is removed from the stack. The stack ID has become an inherent property of the switch. If you wish to renumber it you will need to specify the current stack ID. For example to renumber the stack ID from 3 to 5, use the command:

```
stack (config)#stack 3 renumber 5
```

The stack ID renumbering does not take effect until the switch is rebooted, so you will receive the following warning as shown in the next figure.

```
awplus(config)#stack 1 renumber 2
Warning: the new ID will not become effective until the stack-member
reboots.
Warning: the boot configuration may now be invalid.
```

Until the reboot occurs, the new stack ID value is noted as the 'pending' stack ID, as shown in the next figure.

```
x900#sh stack
Virtual Chassis Stacking summary information

ID   Pending ID   MAC address           Priority   Status   Role
1    2             0000.cd27.c4bf       128      Ready   Active Master

Operational Status           Standalone unit
Stack MAC address           0000.cd27.c4bf
```

## Automatic assignment as a switch joins the stack

When a stack is established, the stack will automatically assign a new stack ID to a switch if it has the same stack ID as another member. This means that you can create a stack without previously changing the stack ID numbers from the default of 1. The stack master will be assigned stack ID 1, and the other switches will be automatically assigned other IDs.

## Manual renumbering of a switch after stacking

If you want to manually renumber the switches when they are already part of a stack, use the command:

```
stack <1-8> renumber <1-8>
```

You can issue this command from the active master to renumber any switch in the stack. To avoid duplicate IDs, a warning message appears if you assign to a stack member the same ID that is currently assigned to another stack member. However, you can continue to renumber the stack member IDs and fix potential ID duplications. Once you have removed any duplicate IDs, you can reboot the switches with ID changes to implement your changes.

If you do not remove the duplications, then when the stack reboots, the switch with the highest stack priority (the lowest priority value) is allocated this ID, and the competing switch is automatically assigned another ID.

## Cascade renumbering of the stack

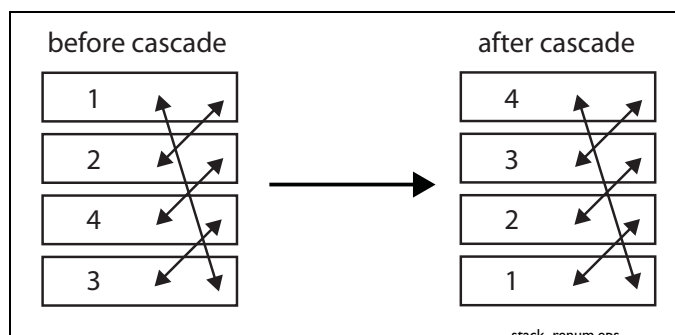
For larger stacks, it is useful to have all the switches numbered in sequence, based on the order of their stack connections. The command that can achieve this is:

```
stack <1-8> renumber cascade [<1-8>]
```

For example, you can enter the command:

```
stack 3 renumber cascade 1
```

This starts the stack numbering on the switch that currently has Stack ID 3 and gives that switch Stack ID 1. The other switches in the stack are renumbered sequentially based on their connection to the switch that now has the stack ID 1. The following figure shows the change from this command.



If the second stack ID parameter is not supplied in the command, then the numbering begins from 1. When the numbering process hits the maximum existing stack ID value, it assigns the value 1 to the next switch in the stack.

We recommend using this command to ensure the switches are sequentially numbered, if you did not manually assign the switches with stack IDs before connecting them to the stack. Use the command straight after the stack is first connected.

### Renumbering the whole stack using the XEM Select button

On switches connected with XEM-STKs, you can use the Select button on the front of the XEM-STK. This achieves the same effect as the renumber cascade command. When you press the Select button on a XEM-STK, the switch with that XEM-STK is assigned stack ID 1, and the other members of the stack are numbered sequentially from there.

## Caution with stack ID renumbering

Stack IDs are critical in identifying each physical switch in the port numbering and configuration commands. However, the stack's configuration script is not altered when stack IDs are renumbered. This means that you may lose the connection between configuration commands and the physical switches, and commands could end up being applied to a different switch.

For example, if switch A in the stack currently has stack ID 2, then any commands in the configuration script that refer to stack ID 2 are applied to switch A. If the stack IDs are renumbered, so that stack ID 2 is now allocated to switch B, then the commands in the configuration script that refer to stack ID 2 will now be applied to switch B. If switch A is given a stack ID which is not in the configuration script, then it will not have any configuration applied.

We recommend that you only renumber the stack when it is initially configured, or during a time of major stack reconfiguration.

## Steps to set up a VCStack

---

There are no set rules regarding the order in which stack configuration tasks need to be carried out. However, these steps provide a guideline to help ensure that the stack creation process goes smoothly.

### 1. Prepare the switches

Before connecting any of the switches together:

- Ensure that all the switches have the same feature licences installed. If you have purchased feature licences to enable certain features to operate on the stack, then all stack members need to have the licences installed. If some stack members have feature licences installed for features that will not be used on the stack, and other switches do not have those licences installed, then remove those unnecessary licences.

- If you are using XEM stacking, you must install the XEMs into the switches before you can enter any stacking commands. An x900 Series switch will reject any stacking commands until a XEM-STK is installed.

```
x900(config)#stack 1 priority 0
% The unit has no XEM-STK installed, stacking commands are not allowed
```

- Decide which stack member will be the active master and set its priority to 0, to ensure it becomes the active master. The command is:

```
stack 1 priority 0
```

(The switch, if it has not been stacked before, should have the default stack ID of 1.)

## 2. Install and power the stack master

Install and power up the master switch. It will detect that there are no other members in the stack, so it will elect itself master. At bootup, it will output the following messages:

```
13:35:10 awplus-1 VCS[1164]: Member 1 (00-00-cd-28-07-c0) has become the
Active Master
```

Then, after the switch has booted, the **show stack** command will show a stack with only one member:

```
x900#sh stack
Virtual Chassis Stacking summary information

ID   Pending ID   MAC address           Priority   Status   Role
1    -             0000.cd27.c4bf        128      Ready   Active Master

Operational Status           Standalone unit
Stack MAC address            0000.cd27.c4bf
```

## 3. Install and power the backup member

Install the next switch, connecting the stacking cable from that switch to the master.

**Note:** Make sure the stacking cables are crossed over between the stack members. This means that stack port 1 on switch 1 should connect to stack port 2 on switch 2. If this is not done, the stack links will not come up and the stack will not form.

Power up the switch. It will detect that there is already an active master, and so will come up as a backup member. The active master will assign it the first available stack ID.

At bootup, the new stack member outputs the following messages:

```
Notice: Possible stack, please wait whilst searching for members.
12:26:57 awplus-2 VCS[1043]: Member 1 (0000.cd27.c4bf) has joined stack
12:26:57 awplus-2 VCS[1043]: Please configure 'stack virtual-mac' to
minimize network disruption from failovers
12:26:57 awplus-2 VCS[1043]: Member 1 (0000.cd27.c4bf) has become the
Active Master
```

After bootup, the **show stack** command will show that there are 2 switches in the stack:

```
x900#sh stack
Virtual Chassis Stacking summary information

ID Pending ID MAC address Priority Status Role
1 - 0000.cd27.c4bf 128 Ready Active Master
2 - 0000.cd28.0801 128 Syncing Backup Member

Operational Status Normal operation
Stack MAC address 0000.cd27.c4bf
```

The active master will check that the new stack member has the same software version as itself. If the software versions are different, the active master will use the software auto-synchronization mechanism to force the new stack member to run the same software version.

#### 4. Install and power the next backup member

Repeat [step 3](#) for each of the other switches in the stack, remembering to connect port 2 of each new switch to port 1 of its neighbour. For last switch added to the stack, connect port 1 of this switch to port 2 of the first installed switch.

#### 5. Confirm that the stack is operating

Check that the stack links have all come up successfully. This can be done by:

- checking the LEDs on the switches or XEMs  
The port LEDs for all stack members should be green. Port LEDs that are off or flashing amber indicate that the stack is not operating correctly. The master or status LED will be green on the switch that is the stack master.
- using the **show stack detail** command  
This command provides a snapshot of the stack status.

See the section "[Checking stack status](#)" on [page 41](#) for more information about LEDs and the **show stack detail** command.

## 6. Check stack numbering

If you are not satisfied with the stack IDs that the switches have been automatically assigned, then this is the right moment to remedy that. To change a stack ID, use the command:

```
stack <1-8> renumber <1-8>
```

It is usually a good idea to give the stack ID 1 to the active master, as it is quite natural to associate the lowest ID with the active master switch.

To sequentially renumber all stack members, you can:

- use the select button on the XEM-STK
- use the **stack renumber cascade** command  
To renumber the stack members and give stack ID 1 to the active master, use the command:

```
stack <current ID on master switch> renumber cascade 1
```

## 7. Reboot the switches

Reboot all the stack members, and check that they all come up with the stack IDs that you are expecting. You can use the command **show stack detail** to check the stack IDs and the status of the neighbour connections.

## 8. Configure any priority changes

If there is another switch that you want to be the one that takes over as active master, if the active master goes down, then set its priority to a value lower than the other switches:

```
stack <1-8> priority 10
```

## 9. Configure the stack as one switch

You are now ready to configure the stack with port channels, VLANs, IP addresses, and so on. For example, to create a port channel that aggregates ports from two different stack members, you would configure as follows:

```
awplus#configure terminal
awplus (config)#interface port1.0.1
awplus (config-if)#channel-group 1 mode active
awplus (config-if)#interface port2.0.1
awplus (config-if)#channel-group 1 mode active
```

Once you are happy with the stack configuration, make a backup copy of the configuration file.

## Steps to replace a stack member

---

If you need to replace a stack member, use the following steps to achieve a smooth transition.

### 1. Configure the Stack ID on the replacement switch

Prepare the replacement switch by configuring it with the same stack ID as the switch that you are replacing.

```
stack (config)#stack <current stack ID> renumber <1-8>
```

### 2. Configure the feature licences

Ensure that the replacement switch is configured with the same set of feature licences as the existing stack members.

### 3. Remove the failed switch

Unplug the failed switch from the stack.

### 4. Install the replacement switch

Connect the stacking cables to the replacement switch. Power up the replacement switch. It will detect that there is already an active master, and so will come up as a stack member.

The active master will check that the new stack member has the same software version as itself. If the software versions are different, the active master will use the software auto-synchronization mechanism to force the new stack member to run the same software version.

The new switch will also receive the startup configuration from the active master. As the replacement switch has been configured with the same stack ID as the replaced switch, it will receive exactly the same configuration as the replaced switch, and will operate exactly as that switch had.

## Provisioning

---

Provisioning provides the ability to pre-configure ports that are not yet present in a switch or in a stack. If a switch can have XEMs hot-swapped into it, then provisioning allows the ports of those yet-to-be-inserted XEMs to be preconfigured ready for the arrival of the XEMs. Similarly, if you know that a unit is going to be added to a stack, you can pre-configure that new switch in anticipation of its addition to the stack.

With provisioning you can configure stack members and their ports, even though they are not currently physically present, and configure them ready for future addition. This means that you can either pre-configure ports belonging to a bay or switch that has not yet been installed, or load a configuration that references these ports.

Provisioning also automatically keeps track of the configuration that was present on XEMs that have been hot-swapped out of a switch, or on units that have been removed from a stack. Provisioning keeps a 'placeholder' for a XEM or switch which has been hot-swapped out.

If you provision a switch or bay, then decide later to change the stack member ID or bay number before it has been installed, you must unprovision (no switch <stack ID> bay/switch) the switch or bay first.

## Provisioning a bay

With the **show sys** command we can see that the stack member 2 x900-24XT switch does not have a XEM in bay 2:

```
awplus#show sys
Stack System Status                                     Wed May 05 00:04:16 2010

Stack member 1:

Board      ID  Bay  Board Name          Rev  Serial number
-----
Base       271      x900-24XS          B-0  P1HF7801H
Expansion  272  Bay1 XEM-1XP             B-0  41AR67008
Expansion  285  Bay2 XEM-STK             A-0  M1L18400R
PSU        212  PSU1 AT-PWR01-AC       F-1  73173269
Fan module 214  PSU2 AT-FAN01           F-1  73169578
-----

RAM: Total: 513372 kB Free: 396680 kB
Flash: 31.0MB Used: 15.9MB Available: 15.1MB
-----

Environment Status : Normal
Uptime              : 0 days 00:55:48
Bootloader version : 1.0.9

Stack member 2:

Board      ID  Bay  Board Name          Rev  Serial number
-----
Base       270      x900-24XT          A-0  M1QH78003
Expansion  285  Bay1 XEM-STK             A-0  M1L17400G
PSU        212  PSU2 AT-PWR01-AC       B-1  61410709
-----

RAM: Total: 513372 kB Free: 410648 kB
Flash: 63.0MB Used: 30.9MB Available: 32.1MB
-----

Environment Status : Normal
Uptime              : 0 days 00:25:34
Bootloader version : 1.0.9

We can see that Stack member 1 is the Master, and we are connected to the
console port on this switch:

awplus#show stack
Virtual Chassis Stacking summary information

ID  Pending ID  MAC address          Priority  Status  Role
1   -           0000.cd27.c4bf       128     Ready   Active Master
2   -           0000.cd28.0801       128     Ready   Backup Member

Operational Status          Normal operation
Stack MAC address           0000.cd27.c4bf
```

On the Stack Master (stack member 1) we can provision a XEM-12 for Stack member 2 in bay 2 (which is currently empty):

Note that the switch automatically provisions all currently installed switches and XEMs as it boots up (it doesn't provision the actual stacking XEMs).

```
awplus(config)#switch 2 bay 2 provision xem-12

switch 1 provision x900-24
switch 1 bay 1 provision xem-1
switch 2 provision x900-24
switch 2 bay 2 provision xem-12
!
interface port2.0.1-2.0.24
  switchport
  switchport mode access
!
interface port2.2.1-2.2.12
  switchport
  switchport mode access
!
```

We can see above that we now have ports 2.2.1-2.2.12 available for configuration in the running-config, even though stack member 2 does not actually have a 12 port XEM (XEM-12) physically installed in bay 2 yet.

This means that these ports can now be configured, ready for when the XEM-12 is installed. Commands can refer to ports on that provisioned XEM as though it were already present:

```
awplus(config)#int port2.2.1
awplus(config-if)#switchport access vlan 2
```

Once a XEM is hot-swapped into bay 2 the "switch 2 bay 2 provision xem-12" will still show in the running configuration, along with the other installed switches and XEMs.

If the XEM is removed, the provisioning for it will remain along with the configuration for the ports associated with it.

### What happens when a provisioned XEM gets hot-swapped out?

In the example below, stack member 1 has a XEM-IXP installed in bay 1 and its port (port1.1.1) has been configured as a trunk:

```
switch 1 provision x900-24
switch 1 bay 1 provision xem-1
switch 2 provision x900-24
!
interface port1.1.1
  switchport
  switchport mode trunk
  switchport trunk allowed vlan all
  switchport trunk native vlan none
!
```

If the XEM-1XP is hot-swapped out of bay 1:

```
awplus#08:23:05 awplus HPI: HOTSWAP Pluggable 1.1.1 hotswapped out: FTRX-1411-3
08:23:05 awplus HPI: HOTSWAP XEM 1 hotswapped out: XEM-1XP
08:23:05 awplus EXFX[1268]: Handle event: bay 1 hsState 4 bt 272 br 0
08:23:05 awplus NSM[1121]: Removal event on bay 1.1 has been completed
```

We can see that the configuration associated with this port is still in the running configuration:

```
interface port1.1.1
  switchport
  switchport mode trunk
  switchport trunk allowed vlan all
  switchport trunk native vlan none
!
```

**What happens when the XEM is hot-swapped back in?**

```
awplus#08:25:18 awplus HPI: HOTSWAP XEM 1 hotswapped in: XEM-1XP
08:25:18 awplus HPI: HOTSWAP Pluggable 1.1.1 hotswapped in: FTRX-1411-3
08:25:18 awplus EXFX[1268]: Handle event: bay 1 hsState 2 bt 272 br 1
08:25:22 awplus EXFX[1268]: Board XEM-1XP inserted into bay 1
08:25:22 awplus EXFX[1268]: Please wait until configuration update is completed
08:25:22 awplus IMI[1123]: All users returned to config mode while switch synchronization is in progress.
08:25:22 awplus VCS[1118]: XEM-1XP has been inserted into bay 1.1
08:25:22 awplus NSM[1121]: Insertion event on bay 1.1 has been completed
08:25:23 awplus IMI[1123]: Configuration update completed for port1.1.1
```

We can see above that port1.1.1 has had its configuration updated from the running configuration.

## What happens if a different type of XEM is hot-swapped in?

If the XEM-1XP is hot-swapped out and a different type of XEM (in this case a XEM-12T) is hot-swapped into bay 1 instead:

```
awplus#08:28:48 awplus HPI: HOTSWAP Pluggable 1.1.1 hotswapped out: FTRX-1411-3
08:28:48 awplus HPI: HOTSWAP XEM 1 hotswapped out: XEM-1XP
08:28:48 awplus EXFX[1268]: Handle event: bay 1 hsState 4 bt 272 br 0
08:28:48 awplus NSM[1121]: Removal event on bay 1.1 has been completed

awplus#08:29:05 awplus HPI: HOTSWAP XEM 1 hotswapped in: XEM-12T
08:29:05 awplus EXFX[1268]: Handle event: bay 1 hsState 2 bt 274 br 2
08:29:08 awplus EXFX[1268]: Board XEM-12T inserted into bay 1
08:29:08 awplus EXFX[1268]: Please wait until configuration update is completed
08:29:08 awplus IMI[1123]: All users returned to config mode while switch synchronization is in progress.
08:29:08 awplus VCS[1118]: XEM-12T has been inserted into bay 1.1
08:29:09 awplus NSM[1121]: Insertion event on bay 1.1 has been completed
08:29:11 awplus IMI[1123]: Configuration update completed for port1.1.1-1.1.12
```

We can see that the provisioning has been modified to reflect the actual hardware installed, and the running configuration now has ports 1.1.1-1.1.12, which are the 12 ports belonging to the XEM-12T in bay 1.

```
switch 1 provision x900-24
switch 1 bay 1 provision xem-12
switch 2 provision x900-24
!
interface port1.1.1-1.1.12
 switchport
 switchport mode access
!
```

## Provisioning a switch

We have a standalone switch that we will be adding another switch to in the future to form a stack:

```
awplus#sh sys
Switch System Status                               Wed May 05 14:34:12 2010

Board      ID  Bay  Board Name                      Rev  Serial number
-----
Base       287      x900-12XT/S                    A-0  M1NB7C023
Expansion  285  Bay1 XEM-STK                        A-0  A1L18305D
-----

RAM: Total: 513372 kB Free: 422964 kB
Flash: 63.0MB Used: 46.0MB Available: 17.0MB
```

The current switch has an ID (stack member) of 2

```
awplus#show stack
Virtual Chassis Stacking summary information

ID Pending ID MAC address          Priority Status Role
2 -          0000.cd28.bff7          128    Ready  Active Master

Operational Status          Standalone unit
Stack MAC address          0000.cd28.bff7
```

We can provision stack member 1 so that we can configure the future stack member's ports before we actually have the second switch connected:

```
awplus(config)#switch 1 provision ?
x600-24 Provision an x600-24 switch
x600-48 Provision an x600-48 switch
x900-12 Provision an x900-12 switch
x900-24 Provision an x900-24 switch
x908 Provision an x908 switch
```

Select the model of switch that will be connected in the future (note that you can only stack, and therefore provision, switches of the same basic model).

If we try to provision an x900-24 switch for stack member 1, and the existing switch (stack member 2) is an x900-12, we get the following error message:

```
awplus(config)#switch 1 provision x900-24
% Board class x900-24 is incompatible with existing stack members
```

We can successfully provision an x900-12 as follows:

```
awplus(config)#switch 1 provision x900-12
```

The running-config shows that we can now configure the ports (1.0.1-1.0.12) on provisioned stack member 1:

```
switch 1 provision x900-12
switch 2 provision x900-12
!
interface port1.0.1-1.0.12
 switchport
 switchport mode access
!
```

Note that the configuration applied to ports 1.0.1-1.0.12 is just the default port configuration. The port trunk configuration that had been provisioned for the XEM-IXP is completely discarded when the XEM-I2S is hot-swapped in instead.

## Re-provisioning

If you want to change the provisioning (for example, change a provisioned x600-24 to an x600-48), you would first have to use the command **no switch x provision**, followed by **switch x provision x600-48**, as 'switch x provision' will fail if there is existing provisioning. However this process means you will lose all of the configuration for portx.0.1-24.

Using **switch x reprovision x600-48** allows you to change the provisioning without losing any existing configuration (within the limits of the respective port counts of the two device types). In short, it allows you to change existing provisioning, provided no actual hardware is present.

We can also reprovision a XEM in a bay. Below we have provisioned a XEM-12 in bay 2 on switch member 2:

```
awplus(config)#switch 2 bay 2 provision xem-12
```

We could then configure port2.2.1 (the first port on the XEM-12) as follows:

```
awplus(config)#int port2.2.1
awplus(config-if)#swi access vlan 2
```

If we decided to use a XEM-IXP instead of the XEM-12, we can reprovision this change and keep the configuration for any ports that overlap - in this case only port2.1.1:

```
awplus(config)#switch 2 bay 2 reprovision xem-1
```

If we had used the **no provision** command, followed by the **provision** command for the new XEM, the overlapping port (port2.2.1) would have been deleted and any configuration on it lost.

## Configuring the stack

---

The configuration script on a VCStack is shared between all stack members. When configuring the stack, you will need to be aware that the stack uses a specific port numbering scheme and that there are some minor restrictions to VLAN, IP subnet, and QoS configuration. There are also specific triggers available for stacking that you may wish to use. See each section for more detail:

- ["Port numbering" on page 27](#)
- ["VLAN and IP subnet restrictions" on page 27](#)
- ["Quality of Service \(QoS\) restriction" on page 27](#)
- ["Stacking triggers" on page 28](#)

For information about how the stack synchronises the startup configuration and running configuration between stack members, see the section ["Software and configuration file synchronisation" on page 28](#).

## Port numbering

In the AlliedWare Plus™ operating system, switchport interfaces are always referenced as portx.y.z. The first number, or 'x', in the three number port identifier is the Stack ID.

For a stand-alone switch that has never been in a stack, the ports are always numbered 1.y.z. When configuring a stack, however, there will be stack members with other stack ID values. To configure ports on these switches, use the stack ID to refer to each physical switch, for example 2.0.1, 2.0.2, 2.0.3, for the first 3 ports on a switch with stack ID 2.

Please note that when a switch is removed from a stack, it maintains its stack ID number. If it is configured as a stand-alone switch, you will need to either change the stack ID back to 1, or use the current stack ID when configuring its ports.

## VLAN and IP subnet restrictions

Internal communication between the stack members is carried out using IP packets sent over the stacking links. This stack management traffic is tagged with a specific ID, and uses IP addresses in a specified subnet.

By default, the VLAN and subnet used are:

- VLAN 4094
- Subnet 192.168.255.0/28

You may need to change these values if they clash with a VLAN ID or subnet that is already in use in the network. Use the commands:

```
stack management subnet <ip-address>
stack management vlan <2-4094>
```

It is important that the settings for **management subnet** and **management vlan** are the same for all the switches in a stack. If a switch is added to a stack, and its setting for **management vlan** and/or **management subnet** differ from those on the other stack members, the new switch will not be joined to the stack, and a log message similar to the following will be created:

```
06:51:41 awplus-vcs VCS[1066]: Member 2 (0009.41fd.dd0b) cannot join stack
- incompatible VCS management subnet
```

## Quality of Service (QoS) restriction

The management communication that the stack members exchange over the stacking link is vital for the successful operation of the stack. Be careful to avoid interrupting that communication.

This communication traffic is transmitted on egress queue 7 on the stacking ports.

We recommend that you avoid sending any user traffic into queue 7 on a VCStack. In any QoS configuration on a stack, prioritize traffic only into queues 0-6. Moreover, you should ensure that the CoS-to-Queue map does not automatically use queue 7 for the transmission of packets that are received with a CoS value of 7. To ensure this, change the cos-to-queue map to put packets with a CoS value of 7 into queue 6. Use the command:

```
mls qos map cos-queue 7 to 6
```

## Stacking triggers

There are five trigger types defined for particular stack events. Any scripts that you configure for triggers are copied from the active master to all stack members. The triggers are:

```
type master-fail
type stack member join
type stack member leave
type stack xem-stk up
type stack xem-stk down
```

## Software and configuration file synchronisation

---

A VCStack requires that the software version and the configuration files on all stack members are the same. If these files are not the same, then the stack cannot form or then operate correctly. To make stack formation easy, all these files are synchronised automatically on the stack by the stack master. The following are synchronised:

- **Software release auto-synchronisation**
- **Shared running configuration**
- **Shared startup configuration**
- **Scripts**

## Software release auto-synchronisation

The VCSStack implementation supports a feature called software auto-synchronization. The effect of this feature is that when a new member joins a stack and has a software release that is different to the active master, then the active master's software release is copied onto the new member. The new member then reboots and comes up on that release.

The sequence of events that occurs at startup is shown in the next figure.

```

Notice: Possible stack, please wait whilst searching for members.
12:37:30 awplus-2 VCS[1057]: Member 1 (0000.cd27.c4bf) has joined stack
12:37:30 awplus-2 VCS[1057]: Please configure 'stack virtual-mac' to
minimize network disruption from failovers
12:37:30 awplus-2 VCS[1057]: Member 1 (0000.cd27.c4bf) has become the
Active Master
12:37:34 awplus-2 VCS[1057]: Software incompatibility detected.
12:37:34 awplus-2 VCS[1057]: Auto synchronization starts. Unit will reboot
once
that finishes. Please wait..
12:39:35 awplus-2 VCS: The new software r1-5.3.4.rel is set as preferred
softwar
e and used at the next reboot.
12:39:35 awplus-2 VCS: The current software r1-main-20100507-1.rel is set
as bac
kup software.
12:39:35 awplus-2 VCS: VCS SW version auto synchronization successfully
completed.

Restarting system

```

The software auto-synchronization feature is enabled on all switches by default. You can enable or disable it using the command:

```
(no) stack <1-8> software-auto-synchronization
```

If you disable software auto-synchronisation, you may disrupt a stack forming if the stack members have different software releases. If a new member joins the stack, and is running a software version that is different to the active master, the active master will reject the new member from the stack if it cannot synchronise the software releases.

Note that this feature can result in the new stack member downgrading its software release if the active master is running an older software version.

Occasionally you will find that the software version running on the existing stack members and the software version on the new stack member are sufficiently incompatible that the stack will not be able to auto synchronise the software version on the new member.

On these occasions, the log on the stack master will contain a message:

```
user.alert awplus VCS[994]: Neighbour on link 1.0.2 cannot join stack-
incompatible stack S/W version
```

Also, the internal stacking log on the stack master will contain similar messages, that can be seen by using the command **show stack full-debug | include version**.

```
awplus#show stack full-debug|include version
2009 Jul 3 02:04:12 user.debug awplus-2 VCS[992]: STK TRACE: Topology
discovery version mismatch - received v23.4
2009 Jul 3 02:04:12 user.debug awplus-2 VCS[992]:Neighbour on link 2.0.1
cannot join stack - incompatible stack S/W version
```

When this occurs you will need to:

- remove the new member from the stack
- manually upgrade the software on that switch to the same version as is running on the existing stack members
- re-connect the new member to the stack

## Shared running configuration

A single, unified running configuration is shared by all the switches in a stack.

This means that the running configuration on each stack member is exactly the same, and contains the configuration information for all stack members. For example, on a two-switch stack, switch A with stack ID 1 will show the configuration for switch B with stack ID 2, as well as its own configuration. If the running configuration on switch B with stack ID 2 is examined, the output will be exactly the same as that produced by switch A.

## Shared startup configuration

Once a stack has formed, the startup configuration stored on each stack member is overwritten by the active master's startup configuration. This means all the switches in a stack have exactly the same startup configuration script.

Every time the startup configuration script on the active master is changed, for example when the running config is copied to the startup config, the updated startup script is copied to all the other stack members.

```
stack(config)#boot config-file interop.cfg
VCS synchronizing file across the stack, please wait..
File synchronization with stack member-2 successfully completed
[DONE]
```

It is important that you take this behaviour into account when you first create a stack. If you have carefully crafted a startup configuration on the switch that you intend to be the active master, but some mistake results in a different switch becoming the active master when the stack forms, then the intended startup configuration will be overwritten by something else (assuming the same filename was used for both configurations).

We recommend that you:

- take care when first installing the VCStack to ensure that you configure the stack priority values correctly. If you do not, the wrong stack member may become the active master on first boot, and overwrite the stack with the wrong startup configuration.
- backup the startup configuration of the intended active master before connecting the switch to the stack. You can make another copy on the switch's flash file system with the command:

```
copy startup_filename.cfg backup_filename.cfg
```

As well as the running configuration and startup configuration, the stack synchronises the boot backup configuration file, and the fallback configuration file. Like the startup configuration file, these are synchronised from the active master's file system, so the same recommendations apply.

## Scripts

Script files stored on the active master's file system are copied to the stack members.

## Rolling Reboot

A major benefit of Virtual Chassis Stacking is that it provides unit resiliency - if one unit in the stack goes down, the other members of the stack will continue to forward data. It is highly desirable for this continuity of service to persist even when the stack is being rebooted. The purpose of the **reboot rolling** command is to reboot a stack in a manner that maintains continuity of service.

This command allows a stack to be rebooted in a rolling sequence so that no more than one unit of the stack is in reboot at any given time. First, the stack master is rebooted causing the remaining stack members to failover and elect a new master.

Here we have a stack of 3 x600 switches:

```
awplus#sh stack
Virtual Chassis Stacking summary information

ID   Pending ID   MAC address           Priority   Status   Role
1    -             0015.77c2.4b7d       128      Ready   Backup Member
3    -             0015.77e8.a892       128      Ready   Backup Member
4  -             0015.77c9.73cb     128    Ready  Active Master

Operational Status           Normal operation
Stack MAC address            0015.77c9.73cb
```

We can see that stack member 4 is the Active Master. Executing the **reboot rolling** command gives the following:

```
awplus#reboot rolling
The stack master will reboot immediately and boot up with the configuration
file settings. The remaining stack members will then reboot once the master
has finished re-configuring.

Continue the rolling reboot of the stack? (y/n):y

awplus#22:11:07 awplus VCS[995]: Automatically rebooting stack member-4
(MAC: 00.15.77.c9.73.cb) due to Rolling reboot

URGENT: broadcast message:
System going down IMMEDIATELY!

... Rebooting at user request ...
```

As soon as the rebooted Active Master has reloaded, it becomes the Active Master again. While it is rebooting, another switch in the stack will assume the Active Master role. Immediately after the Active Master has reloaded and assumed its role again, all of the other switches in the stack are rebooted at the same time.

```

Active Master booting up:

Loading default configuration
.

done!
Received event network.configured

Rolling reboot, rebooting all other stack members, please wait for stack
to reform.

```

You can see in the Active Master's log that the other stack members (1 and 3) have rebooted:

```

2010 May 10 22:12:11 user.crit awplus-4 VCS[995]: Member 4 (0015.77c9.73cb) has become
the Active Master
2010 May 10 22:12:37 local6.notice awplus VCS[995]: Link down event on stack link 4.0.2
2010 May 10 22:12:37 local6.notice awplus VCS[995]: Link down event on stack link 4.0.1
2010 May 10 22:13:32 local6.notice awplus VCS[995]: Link up event on stack link 4.0.1
2010 May 10 22:13:32 local6.notice awplus VCS[995]: Link down event on stack link 4.0.1
2010 May 10 22:13:32 local6.notice awplus VCS[995]: Link up event on stack link 4.0.2
2010 May 10 22:13:33 local6.notice awplus VCS[995]: Link down event on stack link 4.0.2
2010 May 10 22:13:36 local6.notice awplus VCS[995]: Link up event on stack link 4.0.1
2010 May 10 22:13:36 user.crit awplus VCS[995]: Member 3 (0015.77e8.a892) has joined stack
2010 May 10 22:13:36 user.notice awplus VCS[995]: Link between members 4 and 3 is up
2010 May 10 22:13:37 local6.notice awplus VCS[995]: Link up event on stack link 4.0.2
2010 May 10 22:13:37 user.crit awplus VCS[995]: Member 1 (0015.77c2.4b7d) has joined stack
2010 May 10 22:13:37 user.notice awplus VCS[995]: Link between members 4 and 1 is up
2010 May 10 22:13:37 user.notice awplus VCS[995]: Link between members 3 and 1 is up

```

The **reload rolling** command is equivalent to the **reboot rolling** command.

## Failover and recovery

---

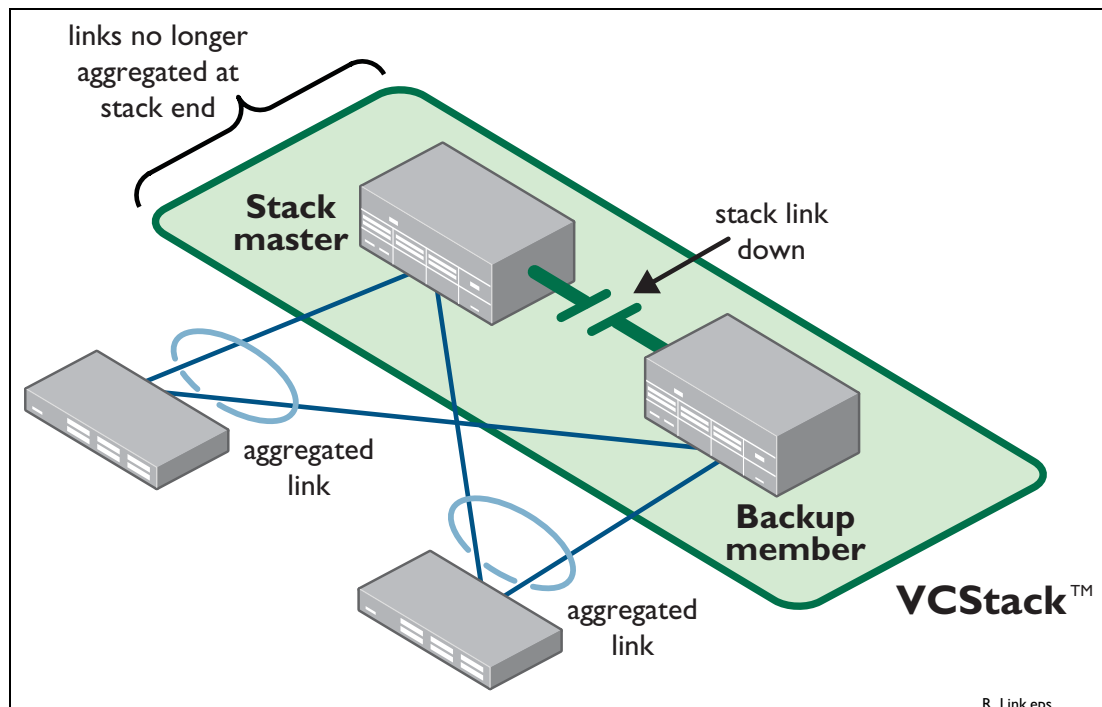
A VCStack behaves in a reliable and consistent manner if any component fails, whether the problem is with a stacking link or stack member. However, before looking at how the stack reacts to each of those scenarios, we first need to understand a generic mechanism that is available in Allied Telesis VCStack to help deal with failure scenarios—namely the resiliency link.

### The resiliency link feature

If one or more stacking links fail, so that some stack members are no longer in contact with the active master switch, then the other stack members are left with a dilemma. Should they assume that the active master switch has gone down, and re-elect a new active master, or should they assume that the active master is still operating?

This problem is particularly important when a stack has multiple connections to edge switches in the network. In this network scenario, if a stack splits into 2 active sections that are operating independently, then the edge switches will continue to treat their uplink ports

as aggregated and share traffic across the links. However, the broken stack link could mean that traffic arriving at some of the stack members cannot reach the intended destination.



So it is necessary to have a backup mechanism through which stack members can check if the active master is still active in case the stack link communication to the active master is lost.

The mechanism provided in Allied Telesis VCStack is called the **resiliency link**, which may be either the eth0 port (only on SwitchBlade x908 or x900 series switches) or a dedicated VLAN (resiliencylink VLAN) to which switch ports may become members. These resiliency ports are connected together, and the stack members all listen for periodic (one per second) Health Check messages from the master. As long as the active master sends Health Check messages, the other stack members know that the active master is still active.

This means that the stack members can know whether the active master is still operating if they lose contact with the active master through the stacking links.

The out-of-band Ethernet port is configured as a resiliency port with the command:

```
awplus(config)# stack resiliencylink eth0
```

Note that even if you configure the eth0 port as a resiliency port, you can still use it for out-of-band management.

A VLAN, and switch port are configured for resiliency link connection with the commands:

```
awplus(config)# stack resiliencylink vlan1000
awplus(config)# interface port1.0.1
awplus(config-if)# switchport resiliencylink
```

Note that this VLAN is dedicated to the resiliency link function and must not be the stack management VLAN or a customer data VLAN.

In the situation where a stack member loses contact through the stacking links, but continues to receive health check messages via the resiliency link, the stack member becomes a disabled master. It runs the same configuration as the active master, but it has its links shutdown.

There is a trigger that can be configured when a switch becomes a disabled master with the commands:

```
awplus(config)# trigger 82
awplus(config-trigger)# type stack disabled master
awplus(config-trigger)# script 1 flash:/disabled.scp
```

Although this command could activate any trigger script, the intention here is that the script will re-activate the links from their previously shutdown state, to enable the user to manage the switch.

## Virtual MAC

As part of a virtual chassis stacking network design, the VCStack™ uses a virtual MAC address for communication with other devices. As this single virtual MAC address is used for the complete VCStack, there is no change of MAC address if a new stack member is required to become master. In conjunction with Fast Failover, this ensures maximum continuity of network service, as there is no need for other devices in the network to learn a new MAC address into their MAC or ARP tables.

The virtual MAC address can be manually configured by specifying a VCStack virtual chassis ID. The ID selected will determine which virtual MAC address the stack will use. The MAC address assigned to a stack must be unique within its network.

The virtual chassis ID entered will form the last 12 bits of a pre selected MAC prefix component; that is, 0000.cd370xxx. For example:

```
awplus(config)# stack virtual-mac
awplus(config)# stack virtual-chassis-id 63
```

This will result in a virtual MAC address of: 0000.cd37.003f

## Repairing a broken stack

When two stub stacks are reconnected, the stack will detect if there is more than one master (active, disabled, or fallback). You will see console messages and logs that report a duplicate master was detected. The stack will carry out an election to choose the true active master, based on priorities and MAC addresses. Any losing master will reboot and become a backup member. The switches that have been running on their fallback configuration (most likely those in the same stub as the losing active master) will reboot and come up on the shared configuration pushed to them by the winning active master.

When a stack detects a duplicate master, a debug snapshot file is created and stored in flash. The file is called debug-duplicate-master-*<time & date>*.tgz. The debug snapshot does not contain a .core file, as is produced when a process unexpectedly terminates on a switch.

Instead, it contains a snapshot of various pieces of information within the software at the time when the stack detected the duplicate master.

## Executing commands and managing files on a specific stack member

---

There are some limited actions that you can do on an individual switch member:

- **Executing commands**  
You can use the **reload** command to reboot an individual switch, or use **show** commands to see the individual physical state of a switch and its file system. **Show** commands that relate to the ports, counters, or configuration are applicable for the stack only.
- **Managing files**  
You can manage files on an individual switch. Note that some files are synchronised between switches. See the "[Software and configuration file synchronisation](#)" section for more information.

### Executing commands

If you want to run a command that displays information from a specific stack member, you can prefix the command with the syntax:

```
remote-command <stack ID>
```

For example, to see the full state of all the file systems on the stack member with stack ID 3, use the command:

```
remote-command 3 show file systems
```

To see the processes running on the stack member with stack ID 2, use the command:

```
remote-command 2 show process
```

To use the **show system environment** command on the stack member with stack ID 2, use the command:

```
remote-command 2 show system environment
```

This will display the following output:

```
x900#remote-command 2 show system environment
Stack Environment Monitoring Status

Stack member 2:

Overall Status: Normal

ID Sensor (Units)                Reading Low Limit High Limit Status
1  Device Present                 Yes    -      -      Ok
2  PSU Overtemp                   No     -      -      Ok
3  PSU Fan Fail                    No     -      -      Ok
4  PSU Power Output                No     -      -      Ok

Resource ID: 2 Name: PSU bay 2 (AT-PWR01-AC)
ID Sensor (Units)                Reading Low Limit High Limit Status
1  Device Present                 Yes    -      -      Ok
2  PSU Overtemp                   No     -      -      Ok
3  PSU Fan Fail                    No     -      -      Ok
4  PSU Power Output                Yes    -      -      Ok

Resource ID: 3 Name: x900-24XT
ID Sensor (Units)                Reading Low Limit High Limit Status
1  Voltage: 2.5V (Volts)          2.578  2.344  2.865  Ok
2  Voltage: 1.65V (Volts)         1.629  1.488  1.816  Ok
3  Voltage: 3.3V (Volts)          3.369  2.973  3.627  Ok
4  Voltage: 1.8V (Volts)          1.797  1.615  1.979  Ok
5  Voltage: 12V (Volts)           11.938 10.813 13.188  Ok
6  Temp: Ambient (Degrees C)       23     -126   55     Ok
7  Temp: Mid Internal (Degrees C)  41     -126   85     Ok
8  Temp: Bk Internal (Degrees C)   31     -126   75     Ok

Resource ID: 4 Name: XEM-STK Bay: 2
ID Sensor (Units)                Reading Low Limit High Limit Status
1  Voltage: 2.5V (Volts)          2.435  2.344  2.865  Ok
2  Voltage: 1.65V (Volts)         1.617  1.491  1.814  Ok
3  Voltage: 3.3V (Volts)          3.248  2.973  3.627  Ok
4  Voltage: 5V (Volts)            5.026  4.505  5.495  Ok
5  Voltage: 12V (Volts)           11.563 10.813 13.188  Ok
6  Voltage: 1.8V (Volts)          1.772  1.617  1.983  Ok
7  Temp: Internal (Degrees C)      40     78(Hyst) 80     Ok
```

Additionally, the **reload** command can take a stack ID as an extra parameter. So, to reboot just the stack member with stack ID 4, use the command:

```
reload stack-member 4
```

You will get the following question from the stack:

```
stack#reload stack-member 4
reboot stack member 4 system? (y/n):
```

## Managing files

If you wish to perform an action on another stack member's file system, the syntax is:

```
<stack-member-name>/flash:[/]<file name>
```

The stack-member-name is not the stack ID, but is an extended hostname formed as:

```
<hostname>-<stack ID>
```

So, if the hostname of the stack is BlueCore, then the stack-member-name for switch 2 in the stack is:

```
BlueCore-2
```

If you do not use the *stack-member-name* prefix, then the command refers to a file that resides on the stack master.

You can also use the *stack-member-name* syntax for the directory command. To view the contents of the flash file system on a specific stack member, you can use the syntax:

```
dir <stack-member-name>/flash:/
```

## Remote Login

Occasionally it can be desirable to login to a specific member of the stack (for example, to manage feature licences per individual unit). Remote login facilitates this by allowing a user on the master switch to log into the CLI of another stack member.

In most respects, once you are logged into the other stack member, the result of entering commands will be as if you were logged into the stack master, i.e. the **show ip interface** command will show all IP interfaces configured on all switches in the stack - not just those on the stack member that you have connected to with the **remote-login** command. Configuration commands are still broadcast to all stack members.

Some **show** commands that display physical attributes of the switch, and commands that access the file system, are executed locally. Also, commands related to feature licences are executed locally.

To login from the Stack master (stack member 1 in this case) to stack member 2:

```
awplus#remote-login ?
  <1-8>  A specific stack member ID

awplus#remote-login 2
Type 'exit' to return to awplus.

AlliedWare Plus (TM) 5.3.4 05/04/10 11:59:17

awplus-2>en
awplus-2#
```

Notice that the prompt has changed to reflect the stack member (2) that we are currently connected to. A directory listing will now show the files on stack member 2 only:

```
awplus-2#dir *.cfg
  948 -rw- May  4 2010 20:59:48  flash:/default.cfg
  677 -rw- May  3 2010 18:39:04  flash:/zz.cfg
 2944 -rw- Mar 23 2010 12:55:40  flash:/ospfv3.cfg
```

We can delete a file from stack member 2 as if we were directly connected to it:

```
awplus-2#del zz.cfg
Delete flash:/zz.cfg? (y/n) [n]:y
Deleting..
Successful operation
awplus-2#
```

To return to the stack master, use the **exit** command.

```
awplus-2#exit
awplus#
```

## Managing licenses on a stack member

Remote Login makes managing feature licenses on the stack members easy. We can simply connect to the stack member:

```
awplus#remote-login 2
Type 'exit' to return to awplus.

AlliedWare Plus (TM) 5.3.4 05/04/10 11:59:17

awplus-2>en
awplus-2#
```

The **show license** command displays the current feature licenses on stack member 2:

```
awplus-2#show license
Software Feature Licenses
-----
Index                : 0
License name         : Base License
Customer name        : Base License
Quantity of licenses : 1
Type of license      : Full
License issue date   : 10-May-2010
License expiry date  : N/A
Features include     : VRRP OSPF-64 RADIUS-100 Virtual-MAC

Index                : 1
License name         : csg
Customer name        : ATL-NZ (Internal Use Only)
Quantity of licenses : 1
Type of license      : Full
License issue date   : 11-Aug-2009
License expiry date  : N/A
Features include     : BGP-64 PIM RIPNG VRRP OSPF-FULL VlanDT OSPF-64
                    : BGP-FULL IPv6Basic MLDSnoop BGP-5K RADIUS-100
                    : RADIUS-FULL PIM-100 ACCESS LAG-128 Virtual-MAC
```

To add a new license we can paste in the **license** command generated by the AlliedWare Plus Licensing web site:

```
awplus-2#license AT-FL-RAD-FULL
4pDI724ugtNcqlf8BmZMti2YEX6MSlSOGxDGCSlaf8aAYVDz
DtpZeg==

% Warning: license was only installed on member-2. Use the 'remote-login'
command to install it on all other stack members.

awplus-2#
awplus-2#show license

Software Feature Licenses
-----
Index                : 0
License name         : Base License
Customer name        : Base License
Quantity of licenses : 1
Type of license      : Full
License issue date   : 10-May-2010
License expiry date  : N/A
Features include     : VRRP OSPF-64 RADIUS-100 Virtual-MAC

Index                : 1
License name         : csg
Customer name        : ATL-NZ (Internal Use Only)
Quantity of licenses : 1
Type of license      : Full
License issue date   : 11-Aug-2009
License expiry date  : N/A
```

[continued on next page...]

```

Features include          : BGP-64 PIM RIPNG VRRP OSPF-FULL VlanDT OSPF-64
                          BGP-FULL IPv6Basic MLDSnoop BGP-5K RADIUS-100
                          RADIUS-FULL PIM-100 ACCESS LAG-128 Virtual-MAC

Index                    : 2
License name             : AT-FL-RAD-FULL
Customer name           : ATL-NZ L3 CSG
Quantity of licenses    : 1
Type of license         : Full
License issue date      : 09-May-2010
License expiry date     : N/A
Features include        : RADIUS-FULL

```

## Monitoring and troubleshooting

---

The physical connection and the signalling communications between stack members are vital to stacking. Because of this, the **show** commands and debugging facilities for VCStack are oriented around stack port status and stack signalling communications.

### Checking stack status

You can check that the stack links have come up successfully by:

- checking the LEDs on the switch or XEM
- using the **show stack detail** command

The tables on the following page describe the LED state and functions for the XEM-STK, SwitchBlade x908, and x600 Series switch.

## LEDs

The LEDs on the XEM-STK show the following:

LED	State	Meaning
Port 1 and Port 2	Green	A stacking link is established.
	Amber (flashing slowly)	The link has transmission fault.
Status	Off	The stacking link is down.
	Green	The switch is the stack master.
	Amber	The switch is a backup member.
	Green (flashing)	The stack is selecting a stack master.
Numeric ID	Off	The switch is not a stack member.
	1 to 8	The stack member ID.

The front panel of the SwitchBlade x908 has the following LEDs for monitoring back-port stacking:

LED	State	Meaning
Port 1 and Port 2	Green	A stacking link is established.
	Amber (flashing slowly)	The link has transmission fault.
Master	Off	The stacking link is down.
	Green	The switch is the stack master.
	Amber	The switch is a backup member.
	Green (flashing)	The stack is selecting a stack master.
	Off	The switch is not a stack member.

The front panel of the x600 Series switch has the following LEDs for monitoring stacking:

LED	State	Meaning
MSTR	Green	The switch is the stack master.
	Off	The switch is a backup member.
1 L/A and 2 L/A	Green	A stacking link is established on that link.
	Green (flashing)	The link is transmitting or receiving data.
PRES	Off	The stacking link is down.
	On	An AT-STACKXG module is correctly installed in the switch.
	Off	There is no AT-STACKXG installed in the switch, or the module is installed incorrectly.

## Show stack detail command

The **show stack detail** command provides a snapshot of the stack status. It includes a full summary of the status of all the stack members and the status of their connections to the other member. This allows you to check that all the stack members have active connections to each other and have recognised correctly which neighbouring switch is connected to each of their stacking ports.

```
x900#show stack detail
Virtual Chassis Stacking detailed information

Stack Status:
-----
Operational Status           Normal operation
Management VLAN ID          4094
Management VLAN subnet address 192.168.255.0
Virtual Chassis ID           1010 (0x3f2)
Virtual MAC address           Disabled
Stack member 1:
-----
ID                             1
Pending ID                       -
MAC address                       0015.77c2.4b7d
Last role change                 Sun May 16 23:39:55 2010
Product type                       x600-24Ts
Role                               Active Master
Status                             Ready
Priority                           128
Host name                          BlueCore
S/W version auto synchronization  On
Resiliency link                   Not configured
Port stk1.0.1 status              Learnt neighbor 4
Port stk1.0.2 status              Learnt neighbor 3
Stack member 2:
-----
ID                             2
Pending ID                       -
MAC address                       0015.77e8.a87d
Last role change                 Sun May 16 23:39:54 2010
Product type                       x600-24Ts-POE
Role                               Backup Member
Status                             Ready
Priority                           128
Host name                          BlueCore-2
S/W version auto synchronization  On
Resiliency link                   Not configured
Port stk2.0.1 status              Learnt neighbor 3
Port stk2.0.2 status              Learnt neighbor 4
Stack member 3:

[continued on next page]
```

-----	
ID	3
Pending ID	-
MAC address	0015.77e8.a892
Last role change	Sun May 16 23:39:55 2010
Product type	x600-24Ts-POE
Role	Backup Member
Status	Ready
Priority	128
Host name	BlueCore-3
S/W version auto synchronization	On
Resiliency link	Not configured
Port stk3.0.1 status	Learnt neighbor 1
Port stk3.0.2 status	Learnt neighbor 2
Stack member 4:	
-----	
ID	4
Pending ID	-
MAC address	0015.77c9.73cb
Last role change	Sun May 16 23:39:56 2010
Product type	x600-48Ts
Role	Backup Member
Status	Ready
Priority	128
Host name	BlueCore-4
[continued on next page]	
-----	
S/W version auto synchronization	On
Resiliency link	Not configured
Port stk4.0.1 status	Learnt neighbor 2
Port stk4.0.2 status	Learnt neighbor 1

## Stack debug output

The clearest way to receive stack debug output is to enable console logging of VCStack messages. Using this method, you receive just the stack debug messages without the other messages that are seen when you enable terminal monitor.

To enable console logging of VCStack messages, use the command:

```
log console program VCS
```

You must enter the parameter **VCS** in uppercase letters when you enter this command.

This command enables you to receive a record of stack link and communication events. For example if a stack port goes down, and then comes up again, the series of messages output is:

```

BlueCore(config)#log console program VCS
BlueCore(config)#
BlueCore#
BlueCore#
BlueCore#
BlueCore#
BlueCore#01:02:45 BlueCore VCS[994]: STK TRACE: < Rxd Link-down msg on L2: 4 (00
15.77c9.
73cb) <--> 2 (0015.77e8.a87d)
01:02:45 BlueCore VCS[994]: Link between members 4 and 2 is down
01:02:45 BlueCore VCS[994]: STK TRACE: Link 1.0.1: --> 4 (0015.77c9.73cb)
01:02:45 BlueCore VCS[994]: STK TRACE: Link 1.0.2: --> 3 (0015.77e8.a892) --> 2
(0015.77e8.a87d)
01:02:45 BlueCore VCS[994]: STK TRACE: Stack topology has changed - updating st
ack H/W routes for L2 connectivity
01:02:45 BlueCore-2 VCS[1011]: Link down event on stack link 2.0.2
01:02:45 BlueCore-4 VCS[995]: Link down event on stack link 4.0.1
01:02:45 BlueCore VCS[994]: STK TRACE: stack H/W route update complete
01:02:45 BlueCore VCS[994]: STK TRACE: Stack topology has changed - updating st
ack H/W routes for L2 connectivity
01:02:45 BlueCore VCS[994]: STK TRACE: stack H/W route update complete
01:02:45 BlueCore VCS[994]: STK TRACE: < Rxd Link-down msg on L2: 2 (0015.77e8.
a87d) <--> 4 (0015.77c9.73cb)
01:02:46 BlueCore-4 VCS[995]: Link up event on stack link 4.0.1
01:02:46 BlueCore-2 VCS[1011]: Link up event on stack link 2.0.2
01:02:48 BlueCore VCS[994]: STK TRACE: < Rxd Ring complete msg on L2: 2 (0015.7
7e8.a87d) --> 3 (0015.77e8.a892) -->
01:02:48 BlueCore VCS[994]: STK TRACE: 1 (0015.77c2.4b7d) --> 4 (0015.77c9.7
3cb) -->
01:02:48 BlueCore VCS[994]: Link between members 4 and 2 is up
01:02:48 BlueCore VCS[994]: STK TRACE: Link 1.0.1: --> 4 (0015.77c9.73cb) --> 2
(0015.77e8.a87d) --> 3 (0015.77e8.a892)
01:02:48 BlueCore VCS[994]: STK TRACE: Link 1.0.2: --> 3 (0015.77e8.a892) --> 2
(0015.77e8.a87d) --> 4 (0015.77c9.73cb)
01:02:48 BlueCore VCS[994]: STK TRACE: Ring topology is now complete
01:02:48 BlueCore VCS[994]: STK TRACE: > Txd Ring complete msg on L2: 1 (0015.7
7c2.4b7d) --> 4 (0015.77c9.73cb) -->
01:02:48 BlueCore VCS[994]: STK TRACE: 2 (0015.77e8.a87d) --> 3 (0015.77e8.a
892) -->
01:02:48 BlueCore VCS[994]: STK TRACE: Stack topology has changed - updating st
ack H/W routes for L2 connectivity
01:02:49 BlueCore VCS[994]: STK TRACE: stack H/W route update complete
01:02:49 BlueCore VCS[994]: STK TRACE: < Rxd Ring complete msg on L2: 4 (0015.7
7c9.73cb) --> 2 (0015.77e8.a87d) -->
01:02:49 BlueCore VCS[994]: STK TRACE: 3 (0015.77e8.a892) --> 1 (0015.77c2.4
b7d) -->
01:02:49 BlueCore VCS[994]: STK TRACE: < Rxd Ring complete msg on L2: 3 (0015.7
7e8.a892) --> 1 (0015.77c2.4b7d) -->
01:02:49 BlueCore VCS[994]: STK TRACE: 4 (0015.77c9.73cb) --> 2 (0015.77e8.a
87d) -->

```

If stack-link communication to a stack member is completely lost, the series of messages output is:

```
BlueCore#
BlueCore#01:04:55 BlueCore VCS[994]: STK TRACE: < Rxd Link-down msg on L2: 4 (00
15.77c9.
73cb) <--> 2 (0015.77e8.a87d)
01:04:55 BlueCore VCS[994]: Link between members 4 and 2 is down
01:04:55 BlueCore VCS[994]: STK TRACE: Link 1.0.1: --> 4 (0015.77c9.73cb)
01:04:55 BlueCore VCS[994]: STK TRACE: Link 1.0.2: --> 3 (0015.77e8.a892) --> 2
(0015.77e8.a87d)
01:04:55 BlueCore VCS[994]: STK TRACE: Stack topology has changed - updating st
ack H/W routes for L2 connectivity
01:04:55 BlueCore-3 VCS[996]: Link down event on stack link 3.0.2
01:04:56 BlueCore VCS[994]: STK TRACE: stack H/W route update complete
01:04:55 BlueCore-4 VCS[995]: Link down event on stack link 4.0.1
01:04:56 BlueCore VCS[994]: STK TRACE: < Rxd Link-down msg on L2: 3 (0015.77e8.
a892) <--> 2 (0015.77e8.a87d)
01:04:56 BlueCore VCS[994]: Link between members 3 and 2 is down
01:04:56 BlueCore VCS[994]: STK TRACE: Link 1.0.1: --> 4 (0015.77c9.73cb)
01:04:56 BlueCore VCS[994]: STK TRACE: Link 1.0.2: --> 3 (0015.77e8.a892)
01:04:56 BlueCore VCS[994]: Member 2 (0015.77e8.a87d) is leaving the stack (unr
eachable via stack links)
01:04:56 BlueCore VCS[994]: STK TRACE: Shutting down access to member 2's file
system
01:04:56 BlueCore VCS[994]: STK TRACE: Member 2 (0015.77e8.a87d) state Backup M
ember --> Leaving
01:04:56 BlueCore VCS[994]: Member 2 (0015.77e8.a87d) has left stack
01:04:56 BlueCore VCS[994]: STK TRACE: Stack topology has changed - updating st
ack H/W routes for L2 connectivity
01:04:56 BlueCore VCS[994]: STK TRACE: stack H/W route update complete
01:04:56 BlueCore VCS: Updating stack file system access...
01:04:57 BlueCore VCS: Shutting down access to member-2's file system
01:04:59 BlueCore VCS[994]: HA monitoring detected member-1 no change
01:04:59 BlueCore VCS[994]: HA monitoring detected member-3 no change
01:04:59 BlueCore VCS[994]: HA monitoring detected member-4 no change
01:04:59 BlueCore VCS[994]: HA monitoring detected member-2 left stack
01:04:59 BlueCore NSM[997]: Removal event on unit 2.0 has been completed
```

To see a very detailed log of stacking related events after they have occurred, and other VCStack debug information, use the command:

```
show stack full-debug [<1-8>]
```

Even if you had not been capturing stack log output at the moment an event occurred, you can still retrospectively obtain the logging information by using this command. If you do not specify a stack ID, then each stack member's output is displayed, one after the other.

This command produces a large amount of output, as shown in the following figure. The VCS debug section contains the detailed log information.

```

BlueCore#show stack full-debug
Detailed debugging information for stack member 1

VCS host configuration
-----
Unit stackable, stack H/W present
VCS mgmt VLAN 4094, subnet 192.168.255.0
Topology: Ring
Neighbor port-1: 4
Neighbor port-2: 3
Stack Virtual MAC: feature disabled
  ID Mac Address      PP  LJ  Cfg  IP              Status  Role
> 1 0015.77c2.4b7d    00  N   Y   192.168.255.1  Ready  Active Master
  2 0015.77e8.a87d    00  Y   Y   192.168.255.2  Syncing Backup Member
  3 0015.77e8.a892    00  N   Y   192.168.255.3  Ready  Backup Member
  4 0015.77c9.73cb    00  N   Y   192.168.255.4  Ready  Backup Member
VCS Inter-process connectivity configuration
-----
Type      Lower      Upper      Port Identity      Publication
-----
0         16781313  16781313  <1.1.1:14753793>    14753794  zone
         16781314  16781314  <1.1.2:3494871041>  3494871042
         16781315  16781315  <1.1.3:3173122049>  3173122050
         16781316  16781316  <1.1.4:1199587329>  1199587330
1         1         1         <1.1.1:14761987>    14761988  node
9500     1         1         <1.1.1:14745638>    14745639  cluster
         2         2         <1.1.2:3494862887>  3494862888
         3         3         <1.1.3:3173113895>  3173113896
         4         4         <1.1.4:1199579174>  1199579175
         100      100      <1.1.1:14753794>    14753795  node
         200      200      <1.1.1:14786564>    14786565  node
         <1.1.1:14745679>    14745680  node
         <1.1.1:14753852>    14753853  node
         <1.1.1:14745674>    14745675  node
         <1.1.1:14745672>    14745673  node
         <1.1.1:14745670>    14745671  node
         <1.1.1:14745663>    14745664  node
         <1.1.1:14876727>    14876728  node
         <1.1.1:14819381>    14819382  node
         <1.1.1:14745651>    14745652  node
         <1.1.1:15073328>    15073329  node
         <1.1.1:14991406>    14991407  node
         <1.1.1:14745630>    14745631  node
         <1.1.1:14745629>    14745630  node
         <1.1.1:14745625>    14745626  node
         <1.1.1:14753816>    14753817  node
         <1.1.1:14745623>    14745624  node
         <1.1.1:14770198>    14770199  node
         <1.1.1:14753813>    14753814  node
         <1.1.1:14745620>    14745621  node
         <1.1.1:14762003>    14762004  node
         <1.1.1:14762002>    14762003  node
         <1.1.1:14745617>    14745618  node
         <1.1.1:14762000>    14762001  node
         <1.1.1:14753807>    14753808  node
         <1.1.1:14753805>    14753806  node
         <1.1.1:14761996>    14761997  node
         <1.1.1:14761995>    14761996  node
         <1.1.1:14745609>    14745610  node
[Continued on next page]

```

			<1.1.1:14753802>	14753803	node
			<1.1.1:14770184>	14770185	node
			<1.1.1:14761991>	14761992	node
			<1.1.1:14761990>	14761991	node
			<1.1.1:14958597>	14958598	node
9501	1	1	<1.1.1:14770221>	14770222	cluster
	2	2	<1.1.2:3494862892>	3494862893	
	3	3	<1.1.3:3173163049>	3173163050	
	4	4	<1.1.4:1199579177>	1199579178	
9506	1	1	<1.1.1:14852155>	14852156	cluster
	2	2	<1.1.2:3494862922>	3494862923	
	3	3	<1.1.3:3173220410>	3173220411	
	4	4	<1.1.4:1199579196>	1199579197	
9513	1	1	<1.1.1:14753806>	14753807	cluster
	2	2	<1.1.2:3494879245>	3494879246	
	3	3	<1.1.3:3173130253>	3173130254	
	4	4	<1.1.4:1199587342>	1199587343	
9550	9550	9550	<1.1.4:1199710258>	1199710259	
			<1.1.2:3494862906>	3494862907	
			<1.1.1:14934066>	14934067	cluster
			<1.1.3:3173187633>	3173187634	
9551	9551	9551	<1.1.1:14745676>	14745677	cluster
			<1.1.2:3494862919>	3494862920	
			<1.1.4:1199579208>	1199579209	
			<1.1.3:3173113917>	3173113918	
9600	1	1	<1.1.1:14794792>	14794793	cluster
	2	2	<1.1.2:3494862893>	3494862894	
	3	3	<1.1.3:3173113898>	3173113899	
	4	4	<1.1.4:1199628328>	1199628329	
9601	1	1	<1.1.1:14745641>	14745642	cluster
	2	2	<1.1.2:3494862894>	3494862895	
	3	3	<1.1.3:3173113899>	3173113900	
	4	4	<1.1.4:1199595562>	1199595563	
9602	1	1	<1.1.1:14745643>	14745644	cluster
	2	2	<1.1.2:3494862896>	3494862897	
	3	3	<1.1.3:3173113901>	3173113902	
	4	4	<1.1.4:1199579180>	1199579181	
9603	1	1	<1.1.1:14745642>	14745643	cluster
	2	2	<1.1.2:3494862895>	3494862896	
	3	3	<1.1.3:3173122092>	3173122093	
	4	4	<1.1.4:1199579179>	1199579180	
9604	1	1	<1.1.1:14745644>	14745645	cluster
	2	2	<1.1.2:3494862897>	3494862898	
	3	3	<1.1.3:3173138478>	3173138479	
	4	4	<1.1.4:1199579181>	1199579182	
9034	1	1	<1.1.1:15351834>	15351835	cluster
	2	2	<1.1.2:3495411739>	3495411740	
	3	3	<1.1.3:3173662747>	3173662748	
	4	4	<1.1.4:1200439322>	1200439323	
9040	2	2	<1.1.2:3494871042>	3494871043	
	3	3	<1.1.3:3173122050>	3173122051	
9041	1	1	<1.1.1:14745657>	14745658	cluster
	2	2	<1.1.2:3494862915>	3494862916	
	3	3	<1.1.3:3173113935>	3173113936	
	4	4	<1.1.4:1199587385>	1199587386	

[Continued on next page]

```

Link <multicast-link>
  Window:20 packets
  RX packets:102 fragments:0/0 bundles:0/0
  TX packets:42244 fragments:0/0 bundles:0/0
  RX naks:118 defs:2 dups:115
  TX naks:4 acks:5 dups:231
  Congestion bearer:0 link:0  Send queue max:18 avg:5

Link <1.1.1:vlan4094-1.1.2:vlan4094>
  ACTIVE MTU:1500 Priority:10 Tolerance:3000 ms Window:50 packets
  RX packets:261 fragments:0/0 bundles:0/0
  TX packets:236 fragments:0/0 bundles:0/0
  TX profile sample:28 packets average:42 octets
  0-64:93% -256:7% -1024:0% -4096:0% -16354:0% -32768:0% -66000:0%
  RX states:115 probes:29 naks:0 defs:0 dups:0 tos:0
  TX states:130 probes:41 naks:0 acks:1 dups:0
  Congestion bearer:0 link:0  Send queue max:1 avg:0
Link <1.1.1:vlan4094-1.1.3:vlan4094>
  ACTIVE MTU:1500 Priority:10 Tolerance:3000 ms Window:50 packets
  RX packets:1401 fragments:0/0 bundles:0/0
  TX packets:1368 fragments:0/0 bundles:0/0
  TX profile sample:86 packets average:43 octets
  0-64:98% -256:2% -1024:0% -4096:0% -16354:0% -32768:0% -66000:0%
  RX states:7745 probes:1234 naks:0 defs:0 dups:0 tos:0
  TX states:10440 probes:3858 naks:0 acks:1 dups:0
  Congestion bearer:0 link:0  Send queue max:5 avg:0

Link <1.1.1:vlan4094-1.1.4:vlan4094>
  ACTIVE MTU:1500 Priority:10 Tolerance:3000 ms Window:50 packets
  RX packets:187 fragments:0/0 bundles:0/0
  TX packets:179 fragments:0/0 bundles:0/0
  TX profile sample:48 packets average:42 octets
  0-64:98% -256:2% -1024:0% -4096:0% -16354:0% -32768:0% -66000:0%
  RX states:7994 probes:1333 naks:0 defs:0 dups:0 tos:0
  TX states:10532 probes:4009 naks:0 acks:0 dups:0
  Congestion bearer:0 link:0  Send queue max:5 avg:0
VCS management traffic
-----
Mon May 17 01:09:05 UTC 2010

vlan4094 Link encap:Ethernet HWaddr 00:15:77:C2:4B:7D
  inet addr:192.168.255.1 Bcast:192.168.255.15 Mask:255.255.255.240
  inet6 addr: fe80::215:77ff:fec2:4b7d/64 Scope:Link
  UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
  RX packets:138575 errors:0 dropped:0 overruns:0 frame:0
  TX packets:205773 errors:0 dropped:0 overruns:0 carrier:0
  collisions:0 txqueuelen:0
  RX bytes:13436002 (12.8 MiB) TX bytes:23187142 (22.1 MiB)

          Pkts          Bytes
Non-VCS Q7: 0              0
Rx AIS:    21173         2559494
Rx mcast:  1912

VCS packet Replication
Type      Tx      Rx      Drops
Total     0       0       0
Bytes     0       0
STP       0       0       0

```

[Continued on next page]

```

EPSR          0          0          0
LACP          0          0          0
Mcast        0          0          0
sflow        0          0          0
BcExc        0          0          0
Other        0          0          0
PortAuth     0          0          0
TxBuff min: 952 86268ms ago
TxBuff cur: 1008
Last Rx:     11ms ago
Last Tx:     14ms ago

```

```

          CPU0
16:      4135  IPIC  Level Enabled  0    serial
17:     186739 IPIC  Level Enabled  0    linux-kernel-bde
22:         0  IPIC  Level Enabled  0    LM81 1
23:         0  IPIC  Level Enabled  0    LM81 2
24:        287  IPIC  Level Enabled  0    mpc83xx_spi
25:        749  IPIC  Level Enabled  0    i2c-mpc
26:         0  IPIC  Level Enabled  0    i2c-mpc
74:         0  IPIC  Edge  Enabled  0    GPIO
75:         0  IPIC  Edge  Enabled  0    GPIO

```

```
BAD:        0
```

```
VCS debug
```

```

-----
2010 May 16 23:39:23 kern.info awplus kernel: TIPC: Activated (version 1.6.4 com
piled May  4 2010 11:45:08)
2010 May 16 23:39:23 kern.info awplus kernel: TIPC: Started in single node mode
2010 May 16 23:39:25 user.info awplus VCS[960]: Parsing 'stack' commands from co
nfig file /flash/default.cfg
2010 May 16 23:39:25 kern.info awplus kernel: TIPC: Started in network mode
2010 May 16 23:39:25 kern.info awplus kernel: TIPC: Own node address <1.1.1>, ne
twork identity 4711
2010 May 16 23:39:31 user.debug awplus VCS[994]: SFL: Base feature license alloc
ated
2010 May 16 23:39:31 user.info awplus VCS[994]: SFL: [stackd] LicenceCheck: Virt
ual-MAC is active
2010 May 16 23:39:31 user.info awplus VCS[994]: SFL: [stackd] LicenceCheck: retu
rns Success.
2010 May 16 23:39:31 user.debug awplus VCS[994]: 2. init app stk-topo-event sess
Pt=0x100696e8, addr=0x4800e000
2010 May 16 23:39:31 user.debug awplus VCS[994]: 2. init app stk-port-1 sessPt=0
x10067480, addr=0x4800f000
2010 May 16 23:39:31 user.debug awplus VCS[994]: 2. init app stk-port-2 sessPt=0
x10069c90, addr=0x48010000
2010 May 16 23:39:31 user.debug awplus VCS[994]: 2. init app stk-topo-msg sessPt
=0x10069b40, addr=0x48011000
2010 May 16 23:39:31 user.debug awplus VCS[994]: 2. init app stk-topo-error sess
Pt=0x10069b58, addr=0x48012000
2010 May 16 23:39:31 user.debug awplus VCS[994]: 2. init app stk-resiliency-link
sessPt=0x10066418, addr=0x48013000
2010 May 16 23:39:31 user.debug awplus VCS[994]: STK DEV  : Stacking Resiliency
Link counters register is successful.
2010 May 16 23:39:45 user.debug awplus VCS[994]: STK DISC : Member-1 XEMs presen
t: Bay 0: XEM-STK,
2010 May 16 23:39:45 user.info awplus VCS[994]: Parsing 'stack resiliencylink vl
an' commands from config file /flash/default.cfg
2010 May 16 23:39:45 user.info awplus VCS[994]: Stacking Ports were discovered o
n the mainboard of member 1

```

## Counters

You can obtain detailed counters relating to stack events and signalling packets with the **show counter stack** command. You can use these for tracking down whether signalling packets are being lost, by checking if there are discrepancies between the number sent from one switch and the number received by its neighbour. The event counters make it possible to see if unexpected events have been occurring on the stack. The following figure is an output example.

```
BlueCore#sh counter stack
Virtual Chassis Stacking counters

Stack member 1:

Topology Event counters
Units joined          ..... 4
Units left            ..... 1
Links up              ..... 8
Links down            ..... 4
ID conflict           ..... 0
Master conflict       ..... 0
Master failover       ..... 0
Master elected         ..... 1
Master discovered     ..... 0
SW autoupgrades       ..... 0

Stack Port 1 Topology Event counters
Link up               ..... 2
Link down             ..... 1
Nbr re-init           ..... 0
Nbr incompatible     ..... 0
Nbr 2way comms       ..... 1
Nbr full comms       ..... 2

Stack Port 2 Topology Event counters
Link up               ..... 1
Link down             ..... 0
Nbr re-init           ..... 0
Nbr incompatible     ..... 0
Nbr 2way comms       ..... 1
Nbr full comms       ..... 0

Topology Message counters
Tx Total              ..... 37
Tx Hellos              ..... 3
Tx Topo DB            ..... 2
Tx Topo update        ..... 2
Tx Link event         ..... 0
Tx Reinitialise       ..... 0
Tx Port 1              ..... 3
Tx Port 2              ..... 2
Tx 1-hop transport    ..... 5
Tx Layer-2 transport  ..... 32
Rx Total              ..... 87
Rx Hellos              ..... 4
Rx Topo DB            ..... 2
Rx Topo update        ..... 10
Rx Link event         ..... 6
```

[Continued on next page]

```

Rx Reinitialise          ..... 0
Rx Port 1                ..... 3
Rx Port 2                ..... 2
Rx 1-hop transport      ..... 5
Rx Layer-2 transport    ..... 82

Topology Error counters
Version unsupported     ..... 0
Product unsupported     ..... 0
XEM unsupported         ..... 0
Too many units         ..... 0
Invalid messages       ..... 0

Resiliency Link counters
Health status good     ..... 0
Health status bad      ..... 0
Tx                     ..... 0
Tx Error               ..... 0
Rx                     ..... 0
Rx Error               ..... 0
Stack member 2:

Topology Event counters
Units joined           ..... 3
Units left             ..... 0
Links up               ..... 4
Links down             ..... 0
ID conflict            ..... 0
Master conflict        ..... 0
Master failover        ..... 0
Master elected          ..... 0
Master discovered      ..... 1
SW autoupgrades        ..... 0

Stack Port 1 Topology Event counters
Link up                ..... 1
Link down              ..... 0
Nbr re-init            ..... 0
Nbr incompatible      ..... 0
Nbr 2way comms         ..... 1
Nbr full comms         ..... 2

Stack Port 2 Topology Event counters
Link up                ..... 1
Link down              ..... 0
Nbr re-init            ..... 0
Nbr incompatible      ..... 0
Nbr 2way comms         ..... 1
Nbr full comms         ..... 0

Topology Message counters
Tx Total                ..... 24
Tx Hellos               ..... 4
Tx Topo DB              ..... 2
Tx Topo update          ..... 1
Tx Link event           ..... 0
Tx Reinitialise        ..... 0
Tx Port 1               ..... 3
Tx Port 2               ..... 3
Tx 1-hop transport     ..... 6

```

[Continued on next page]

```

Tx Layer-2 transport ..... 18
Rx Total ..... 18
Rx Hellos ..... 2
Rx Topo DB ..... 2
Rx Topo update ..... 2
Rx Link event ..... 0
Rx Reinitialise ..... 0
Rx Port 1 ..... 2
Rx Port 2 ..... 2
Rx 1-hop transport ..... 4
Rx Layer-2 transport ..... 14

Topology Error counters
Version unsupported ..... 0
Product unsupported ..... 0
XEM unsupported ..... 0
Too many units ..... 0
Invalid messages ..... 0

Resiliency Link counters
Health status good ..... 0
Health status bad ..... 0
Tx ..... 0
Tx Error ..... 0
Rx ..... 0
Rx Error ..... 0
Stack member 3:

Topology Event counters
Units joined ..... 4
Units left ..... 1
Links up ..... 8
Links down ..... 4
ID conflict ..... 0
Master conflict ..... 0
Master failover ..... 0
Master elected ..... 0
Master discovered ..... 1
SW autoupgrades ..... 0

Stack Port 1 Topology Event counters
Link up ..... 1
Link down ..... 0
Nbr re-init ..... 0
Nbr incompatible ..... 0
Nbr 2way comms ..... 1
Nbr full comms ..... 3

Stack Port 2 Topology Event counters
Link up ..... 4
Link down ..... 3
Nbr re-init ..... 0
Nbr incompatible ..... 0
Nbr 2way comms ..... 2
Nbr full comms ..... 0

```

[Continued on next page]

```

Topology Message counters
Tx Total ..... 37
Tx Hellos ..... 4
Tx Topo DB ..... 3
Tx Topo update ..... 3
Tx Link event ..... 1
Tx Reinitialise ..... 0
Tx Port 1 ..... 2
Tx Port 2 ..... 5
Tx 1-hop transport ..... 7
Tx Layer-2 transport ..... 30
Rx Total ..... 87
Rx Hellos ..... 6
Rx Topo DB ..... 3
Rx Topo update ..... 9
Rx Link event ..... 5
Rx Reinitialise ..... 0
Rx Port 1 ..... 2
Rx Port 2 ..... 5
Rx 1-hop transport ..... 7
Rx Layer-2 transport ..... 80

```

```

Topology Error counters
Version unsupported ..... 0
Product unsupported ..... 0
XEM unsupported ..... 0
Too many units ..... 0
Invalid messages ..... 0

```

```

Resiliency Link counters
Health status good ..... 0
Health status bad ..... 0
Tx ..... 0
Tx Error ..... 0
Rx ..... 0
Rx Error ..... 0

```

Stack member 4:

```

Topology Event counters
Units joined ..... 4
Units left ..... 1
Links up ..... 8
Links down ..... 4
ID conflict ..... 0
Master conflict ..... 0
Master failover ..... 0
Master elected ..... 0
Master discovered ..... 1
SW autoupgrades ..... 0

```

```

Stack Port 1 Topology Event counters
Link up ..... 5
Link down ..... 4
Nbr re-init ..... 0
Nbr incompatible ..... 0
Nbr 2way comms ..... 4
Nbr full comms ..... 5

```

[Continued on next page]

Stack Port 2 Topology Event counters

Link up	.....	1
Link down	.....	0
Nbr re-init	.....	0
Nbr incompatible	.....	0
Nbr 2way comms	.....	1
Nbr full comms	.....	0

Topology Message counters

Tx Total	.....	52
Tx Hellos	.....	9
Tx Topo DB	.....	5
Tx Topo update	.....	4
Tx Link event	.....	3
Tx Reinitialise	.....	0
Tx Port 1	.....	9
Tx Port 2	.....	3
Tx 1-hop transport	.....	12
Tx Layer-2 transport	.....	40
Rx Total	.....	76
Rx Hellos	.....	9
Rx Topo DB	.....	5
Rx Topo update	.....	7
Rx Link event	.....	3
Rx Reinitialise	.....	0
Rx Port 1	.....	10
Rx Port 2	.....	3
Rx 1-hop transport	.....	13
Rx Layer-2 transport	.....	63

Topology Error counters

Version unsupported	.....	0
Product unsupported	.....	0
XEM unsupported	.....	0
Too many units	.....	0
Invalid messages	.....	0

Resiliency Link counters

Health status good	.....	0
Health status bad	.....	0
Tx	.....	0
Tx Error	.....	0
Rx	.....	0
Rx Error	.....	0

BlueCore#