GRAPHCORE

# Graphcore IPU Info Library (gcipuinfo)

*Version latest*

**Graphcore Ltd**

**Oct 10, 2023**

# CONTENTS

CHAPTER

ONE

# INTRODUCTION

The Graphcore IPU Info library (gcipuinfo) provides an API for monitoring and gathering information about the IPUs available in a system, and the applications using them. A typical user of this library would be something like a datacentre monitoring system.

The library has three areas of functionality:

- Looking up information about the devices installed in a system, and statistics about the applications currently using the devices. See Section 3, Device attributes.
- Retrieving application "event records" (warnings or errors) which describe any problems encountered by the last executed Poplar application. See Section 4, Application event record.
- Carrying out basic health checks on the available devices, to rapidly discover hardware or configuration problems. See Section 5, Device health check.

The gcipuinfo API is implemented as part of the standard IPU support libraries already used by Poplar applications and the programs in the Graphcore command line tools package.

There are C++, Python and Go interfaces. The API section of this document describes the C++ API. The Python and Go APIs are very similar, the main difference being that the Go function names all start with an uppercase letter. For example, `getNamedAttributeForAll()` in C++ and Python is `GetNamedAttributeForAll()` in Go.

## 1.1 C++

There are three mechanisms for using gcipuinfo through C++. The most suitable choice depends on the application.

- For programs that are also using the Poplar API, the gcipuinfo interface is directly available from `libpoplar.so`.
- For programs that do not need Poplar, for example system monitoring or configuration tools, you can link against the `libgraphcore_target_access.a` static library from the command line tools package.
- The "gcipuinfo wrapper library". This is a variation of the last method, but intended to work around binary incompatibility issues between C++ code compiled with incompatible compiler versions. See Section 1.1.1, gcipuinfo wrapper.

Note that in all three of these cases the API is the same, and the same header files - `gcipuinfo.h` and `IPUAttributeLabels.h` - are used.

The lookup functions are defined in the `gcipuinfo.h` header, and the attributes are defined in `IPUAttributeLabels.h`

### 1.1.1 gcipuinfo wrapper

libpoplar.so and libgraphcore_target_access.a use a C++ interface. The C++ application binary interface used by these libraries is defined by the standard toolchain in the platforms supported by Graphcore (for example, CentOS 7.6 or Ubuntu 18.04). For programs that must be built with unsupported compilers, a C++ ABI can be problematic due to binary incompatibility issues. For this reason, we also supply a library that exports a plain C interface to the device information database: gcipuinfo_ext.so. The share/gcipuinfo/gcipuinfo_wrapper directory in the Poplar SDK contains two source files that can be compiled along with user code to re-implement the gcipuinfo API using the information from gcipuinfo_ext.so.

## 1.2 Python

Both lookup functions and attributes live within the gcipuinfo module.

## 1.3 Go

Both lookup functions and attributes live within the gcipuinfo package.

CHAPTER

# TWO

# QUICK START

To set up your environment to use gcipuinfo, you will need to source the enable.sh script provided in the Poplar SDK or the Graphcore command line tools package:

```
$ source enable.sh
```

The enable.sh script in the Graphcore command line tools package sets an environment variable HOST_RUNTIME_ROOT to the base path of the package. The equivalent variable for the Poplar SDK is POPLAR_ROOT.

All the remaining examples will assume the command line tools package is being used.

In all the examples below, the output indicates that gc-powertest is currently using an IPU.

## 2.1 Running a Python example

To run one of the Python examples:

```
$ cd $HOST_RUNTIME_ROOT/share/gcipuinfo/examples/python
$ python3 ./gc_active_apps.py

Graphcore active apps:  1
 o application #0: [gc-powertest]
```

## 2.2 Running a C++ example

To run one of the C++ examples using the direct C++ interface:

```
$ cd $HOST_RUNTIME_ROOT/share/gcipuinfo/examples/cpp
$ make -f Makefile.gcda
$ ./gc-active-apps

Graphcore active apps:  1
 o application #0: [gc-powertest]
```

To run one of the C++ examples using the gcipuinfo_wrapper interface:

```
$ cd $HOST_RUNTIME_ROOT/share/gcipuinfo/examples/cpp
$ make -f Makefile.gcipuinfo_wrapper
$ ./gc-active-apps

Graphcore active apps:  1
 o application #0: [gc-powertest]
```

## 2.3 Running a Go example

To run one of the Go examples:

```
$ cd $HOST_RUNTIME_ROOT/share/gcipuinfo/examples/go
$ export GOPATH=$HOST_RUNTIME_ROOT/lib/go/
$ go run gc_active_apps.go

Graphcore active apps:  1
 o application #0: [gc-powertest]
```

CHAPTER

# THREE

# DEVICE ATTRIBUTES

Each IPU device within a system has a set of attributes associated with it. The attributes contain information relating to the IPU such as the current power usage, current process, IPU activity level, and so on.

The gcipuinfo library provides a public API to retrieve this information. The library defines both the lookup functions and the available attributes.

You can query device attributes at the command line with the gc-info and gc-inventory tools. These tools are part of the Graphcore command line tools.

Note that there are some additional attributes visible in the gc-info and gc-inventory tools which are only used internally. These are not exported or documented through the gcipuinfo library.

The lookup functions are defined in gcipuinfo. The available attributes and the *labels* used to access them are defined in Section 6.1, Attribute labels.

## 3.1 Querying attributes

gcipuinfo provides a collection of methods for querying attributes.

gcipuinfo will query all of the available attributes for all devices when the class is created.

By default, subsequent calls to query any of the data will return the original data. Some attributes can dynamically change (such as power usage or IPU utilization). To force gcipuinfo to update the attributes you can call the gcipuinfo::updateData() method. Alternatively, you can call the gcipuinfo::setUpdateMode() method to force gcipuinfo to update the IPU attributes on each and every call to query the data.

By default only attributes of devices in the current partition will be retrieved. You can request that gcipuinfo return device attributes of devices in other partitions, by supplying DiscoverAllPartitionIPUs as the discoveryMode argument of the gcipuinfo::gcipuinfo() object constructor. This is demonstrated in the gc-get-attributes-for-all-devices example program.

**Note:** It is not possible to use both DiscoverAllPartitionIPUs and DiscoverActivePartitionIPUs modes in the same process.

**Note:** Information about available partitions is only retrieved once, at program start up. If partitions are added or removed any monitoring programs will need to be restarted to obtain updated partition information.

CHAPTER

# FOUR

# APPLICATION EVENT RECORD

When the application event recording system is activated, and a Poplar application encounters certain types of events that affect or prevent correct operation, it will store an entry into the "application event record". These events can be retrieved through the gcipuinfo API.

By polling Poplar servers for any event record entries, a monitoring system could detect when applications have stopped running correctly — either due to a hardware/configuration problem or an issue in the application itself.

## 4.1 Event record storage

Event records are written to the Poplar server filesystem as a JSON file. The storage location of that file (a directory) is specified by the value of the IPU_APP_EVENT_RECORD_PATH environment variable. To enable the event recording system, you must set this variable to a path that is accessible to the Poplar application. Event recording is disabled if IPU_APP_EVENT_RECORD_PATH is not set.

You can set this on a per-application, per-user or per-container basis. Note that if multiple applications are running on the same Poplar server with the environment variable set to same value, they can potentially overwrite each other's event records.

Events have a *severity* level. If there is already an entry in the event record an application will only be allowed to write a new entry if it is of an equal or higher severity to the existing one.

## 4.2 Accessing event records

gcipuinfo provides the gcipuinfo::getLastAppEventRecord() and gcipuinfo::getLastAppEventRecordAsJSON() functions to retrieve the last recorded event record entry. These functions take an eventRecordPath parameter to specify the location of the event record.

You must provide the program that uses gcipuinfo with the locations of the event records that it will monitor. If the program is monitoring multiple event records then it will need to check each one individually.

**Note:** For backwards compatibility with previous versions of the library, eventRecordPath is actually an optional parameter in C++ and Python. If it is not specified, the library will attempt to use the value in IPU_APP_EVENT_RECORD_PATH. This usage is deprecated - eventRecordPath should always be specified.

gcipuinfo can return an event record either as a JSON-formatted string or as a string-keyed dictionary/map. If there is no event record set, gcipuinfo::getLastAppEventRecordAsJSON() will return "{}" and gcipuinfo::getLastAppEventRecord() will return an empty dictionary.

Note that the event record paths must be accessible to both the Poplar application and the program using gcipuinfo to read the event record. This may require some special system configuration if, for example, they are running from different containers.

## 4.3 Error record contents

Example JSON error record:


```
{
  "event record path": "/tmp/ipu/ipu_app_event_record",
  "pid": "74647",
  "command line": "/home/justina/gbnwp-pipudc019/poplar/Build/build/poplar/tests/ExecutableTest -- --device-type Hw",
  "timestamp": "2021-09-29T12:14:34.901051Z",
  "severity": "nonrecoverable",
  "partition": "",
  "attached ipus": [
    3,
    2,
    1,
    0
  ],
  "specific ipus": [],
  "attached ipu hosts": [
    "10.1.5.10"
  ],
  "specific ipu hosts": [],
  "description": "Link training failed - At least one link failed to train at 2021-09-29T12:14:34.900795Z"
}
```


The contents of an event record entry are described in the table below.

**Table 4.1: Event record attributes**

| C/Python key symbol | Key string | Description |
|---|---|---|
| keyRecordPath | "event record path" | The path the record was stored at |
| keyTimestamp | "timestamp" | When the event was recorded. Uses extended ISO8601 format. |
| keySeverity | "severity" | *Severity* of the event. |
| keyCommandLine | "command line" | Command line of the process that recorded the event. |
| keyPid | "pid" | Process ID of the process that recorded the event |
| keyAttachedIPUs | "attached ipus" | List of all attached IPUs (device IDs). |
| keyAttachedIPUHosts | "attached ipu hosts" | List of the IPU-Machine hosts that contain the devices listed in "attached ipus". Not used on PCIe-based systems. |
| keySpecificIPUs | "specific ipus" | List of specific IPUs named in this event (device IDs). Only used by some events. |
| keySpecificIPUHosts | "specific ipu hosts" | List of the IPU-Machine hosts that contain the devices in "specific ipus". Not used on PCIe-based systems. |
| keyPartition | "partition" | The partition in use by the application, if applicable. |
| keyDescription | "description" | A description of the event. |

## 4.4 Event severity

Event severity levels are described in gcipuinfo::EventSeverity.

An event with a higher severity than EventSevWarning means that the application encountered a fatal error.

CHAPTER

# FIVE

# DEVICE HEALTH CHECK

gcipuinfo contains a "device health check" function, which allows a monitoring system to rapidly detect when any of the configured IPUs or IPU-Machines are non-functional and should be taken out of use. This mechanism is not intended to provide detailed diagnostic information, it is concerned only with identifying which devices have failed.

The types of failure detected include:

- Network misconfiguration, preventing a Poplar server from communicating with an IPU-Machine.
- Hardware failure of an IPU, an RNIC, or another component of an IPU-Machine.

The API for the device health check is a single function gcipuinfo::checkHealthOfDevices() that returns a JSON string. If no problems were detected, this will be an empty object "{}", otherwise it will list the failing IPUs. If the health check failed to run, a description of the error will be returned. A monitoring system would continually call this function to poll for failures.

By default, gcipuinfo will only check devices in the currently active partition. You can configure gcipuinfo to run health checks on devices in other partitions, by supplying DiscoverAllPartitionIPUs as the discoveryMode argument of the gcipuinfo::gcipuinfo() object constructor. This is demonstrated in the gc_health_check example program.

**Note:** It is not possible to use both DiscoverAllPartitionIPUs and DiscoverActivePartitionIPUs modes in the same process.

**Note:** Information about available partitions is only retrieved once, at program start up. If partitions are added or removed any monitoring programs will need to be restarted to obtain updated partition information.

## 5.1 Health check results

Example JSON health check report with two failing IPUs in partition p1 on IPU-Machine 10.1.5.10:


```
{
  "hosts": {
    "10.1.5.10": [
      {
        "error": "device",
        "id": "2",
        "partition": "p1",
        "board ipu index": "1"
      },
      {
        "error": "connection",
        "id": "3",
        "partition": "p1",
        "board ipu index": "0"
```


(continues on next page)

(continued from previous page)

```
    }
   ]
  }
}
```

Device ID 2 has failed due to a “device” error, which may suggest a hardware or device driver problem. Device ID 3 has failed due to a “connection” error, which may indicate a network problem or a disabled IPUoF server.

Example JSON report where the health check failed to run:

```
{
  ”description”: ”configuration error\n”,
  ”error”: ”no devices found”
}
```

In this case the health check could not be run because no IPU devices were found. The description shows that this is due to a problem with the configuration, for example a partition may not be set up.

CHAPTER

# SIX

# API REFERENCE

enum DeviceDiscoveryMode

*Values:*

enumerator DiscoverActivePartitionIPUs

Discover all devices in current partition.

This is the default mode

enumerator DiscoverAllPartitionIPUs

Discover all devices across all partitions.

class gcipuinfo

**Application Event Record key names**

static constexpr const char *keyRecordPath = "event record path"

The path the record was stored at.

static constexpr const char *keyTimestamp = "timestamp"

When the event was recorded.

static constexpr const char *keySeverity = "severity"

Severity level - EventSeverity.

static constexpr const char *keyCommandLine = "command line"

Command line of the process that recorded the event.

static constexpr const char *keyPid = "pid"

Process id of the process that recorded the event.

static constexpr const char *keyAttachedIPUs = "attached ipus"

List of any attached IPUs (device ids)

static constexpr const char *keySpecificIPUs = "specific ipus"

List of any specific IPUs named in this event (device ids)

static constexpr const char *keyAttachedIPUHosts = "attached ipu hosts"

List of all currently attached IPU-Machine hostnames.

static constexpr const char *keySpecificIPUHosts = "specific ipu hosts"

List of any IPU-Machine hostnames associated with the devices named in "specific ipus".

static constexpr const char *keyPartition = "partition"

The partition in use by the application, if applicable.

static constexpr const char *keyDescription = "description"

A description of the event.

**Device attribute query methods.**

bool updateData()

Updates the attribute info.

Calling this function updates the data. Alternatively, you can call setUpdateMode() to ensure the data is always updated.

**Returns** true if the device attributes have been successfully updated, false if an error occurred.

void setUpdateMode(bool autoUpdate)

Sets the attribute update mode.

By default, the attribute data is queried on construction of this class. This function selects between the default behaviour, and always updating on API call.

**Parameters** autoUpdate – true to enable auto-update mode, false to use the original data.

std::vector<std::map<std::string, std::string» getDevices()

Get all attributes, for all devices.

**Returns** A std::vector containing a std::map for each device.

std::string getDevicesAsJSON()

Return a JSON representation of all device attributes.

**Returns** A JSON-formatted tree of devices and device attributes as a std::string

std::map<std::string, std::string> getAttributesForDevice(unsigned deviceId)

Get all attributes, for a specific devices.

**Parameters** deviceId – Device ID to query.

**Returns** A std::map for each device.

std::string getNamedAttributeForDevice(unsigned deviceId, const std::string key)

Get a specific attribute for a specific device.

**Parameters**

- deviceId – Device ID to query.
- key – Device attribute name.

**Returns** The attribute.

std::vector<std::string> getNamedAttributeForAll(const std::string key)

Get a specific attribute for all devices.

**Parameters** key – Device attribute name.

**Returns** A std::vector containing the attribute for all devices.

**Application Event Record query methods.**

std::string getLastAppEventRecordAsJSON(*EventSeverity* minimumSeverity = *EventSevNone*, const std::string &eventRecordPath = "")

Return a JSON-formatted string describing the last recorded application event.

If there is no recorded event, or the event has a EventSeverity below minimumSeverity, an empty JSON dictionary "{}" is returned.

Looks in the path specified by

**Parameters** eventRecordPath – for an event record. If this is empty, falls back to the path in IPU_APP_EVENT_RECORD_PATH. If neither are set, throws an exception.

std::map<std::string, std::string> getLastAppEventRecord(*EventSeverity* minimumSeverity = *EventSevNone*, const std::string &eventRecordPath = "")

Return a map describing the last recorded application event.

If there is no recorded event, or the event has a EventSeverity below minimumSeverity, an empty map is returned.

Looks in the path specified by

**Parameters** eventRecordPath – for an event record. If this is empty, falls back to the path in IPU_APP_EVENT_RECORD_PATH. If neither are set, throws an exception.

*EventSeverity* getLastAppEventRecordSeverity(const std::string &eventRecordPath = "")

Return the EventSeverity of the last event in the event record.

If there is no recorded event, EventSevNone is returned.

Looks in the path specified by

**Parameters** eventRecordPath – for an event record. If this is empty, falls back to the path in IPU_APP_EVENT_RECORD_PATH. If neither are set, throws an exception.

**Device health check methods.**

std::string checkHealthOfDevices(unsigned timeoutMS, bool checkActiveIPUs = false)

Run basic 'health checks' on all currently configured IPUs.

If all devices appear to be operating normally, returns an empty JSON object:

```
{}
```

If any malfunctioning devices were discovered, returns a JSON tree idenfifying the affected IPUs and the IPU-Machine host and partition they belong to. e.g.

```
{
 "hosts": {
  "10.1.5.10": [
    {
      "error": "device",
      "id": "2",
      "partition": "p1"
    },
    {
      "error": "connection",
      "id": "3",
      "partition": "p1"
    }
  ]
```

(continues on next page)

(continued from previous page)

```
 }
}
```

There are two error types defined:

- ”connection” - the client was unable to communicate with the IPUoF server (either because of network issues or server error) within the specified timeout.
- ”device” - the IPUoF server discovered a problem with the IPU or RNIC device.

Each IPU health check must complete within timeoutMS, or else a “connection” error will be recorded.

By default, devices which are currently in use by applications are not checked, unless checkActiveIPUs is true.

**Public Types**

enum EventSeverity

Application Event Record severity level.

The severity level of an application event indicates how serious it is and potential ways of resolving the issue.

*Values:*

enumerator EventSevNone = 0

enumerator EventSevWarning = 1

An event which may indicate a system problem.

text: “warning”

enumerator EventSevApplicationError = 2

An error likely in the application code or configuration.

poplar::poplar_error, poplar::application_runtime_error

text: “application”

enumerator EventSevUndeterminedError = 3

It is not known if this is a system error, or an error in the application code or configuration.

poplar::unknown_runtime_error

text: “undetermined”

enumerator EventSevRequiresUserReset = 4

The error may be resolvable by an IPU reset or a partition reset (for POD systems) or a link reset (for non-Pod systems).

poplar::recoverable_runtime_error + IPU_RESET or PARTITION_RESET or LINK_RESET

text: “requires_user_reset”

enumerator EventSevRequiresSystemReset = 5

The error may be resolvable by a full reboot of the IPU-M system or Poplar server.

poplar::recoverable_runtime_error + FULL_RESET

text: “requires_system_reset”

enumerator EventSevNonRecoverable = 6

The error may require admin-level system reconfiguration or hardware replacement.

poplar::unrecoverable_runtime_error

text: “nonrecoverable”

**Public Functions**

gcipuinfo(*DeviceDiscoveryMode* = *DiscoverActivePartitionIPUs*)

## 6.1 Attribute labels

const std::string IPUAttributeLabels::DeviceId

Unique identifier of a single-IPU or multi-IPU device.

text: “id”

const std::string IPUAttributeLabels::AverageBoardTemp

Average temperature in degrees Celsius as read by the sensors on the board.

text: “average board temp”

const std::string IPUAttributeLabels::AverageDieTemp

Average temperature in degrees Celsius as read by IPU sensors.

text: “average die temp”

const std::string IPUAttributeLabels::BoardIpuIndex

IPU number on board (0-1 for PCIe cards, 0-3 for IPU-Machines).

text: “board ipu index”

const std::string IPUAttributeLabels::BoardType

The IPU board type ‘family’, for example C600 or M2000.

Note: M2000 includes IPU-M2000 and Bow-2000. text: “board type”

const std::string IPUAttributeLabels::ClockFrequency

Current clock frequency.

text: “clock”

const std::string IPUAttributeLabels::DriverVersion

PCIe driver version, specified as a <major.minor.patch> triple.

text: “driver version”

const std::string IPUAttributeLabels::GatewaySoftwareVersion

(IPUoF) IPU-Gateway software version, specified as a <major.minor.patch> triple.

text: “gateway software version”

const std::string IPUAttributeLabels::GcdId
: (IPUoF) Graphcore Compile Domain ID.
: text: “gcd id”

const std::string IPUAttributeLabels::HexoattTotalSize
: Total remote-buffer memory available.
: text: “hexoatt total size (bytes)”

const std::string IPUAttributeLabels::HexoattActiveSize
: Total remote buffer-memory in use by the IPU.
: text: “hexoatt active size (bytes)”

const std::string IPUAttributeLabels::HexoptTotalSize
: Total host exchange memory available.
: text: “hexopt total size (bytes)”

const std::string IPUAttributeLabels::HexoptActiveSize
: Total host exchange memory in use by the IPU.
: text: “hexopt active size (bytes)”

const std::string IPUAttributeLabels::IpuArchitecture
: IPU hardware architecture version.
: text: “ipu architecture”

const std::string IPUAttributeLabels::IpuofHost
: (IPUoF) IP address of IPU-Gateway.
: text: “ipuof host”

const std::string IPUAttributeLabels::IpuofServerVersion
: (IPUoF) Fabric server version.
: text: “ipuof server version”

const std::string IPUAttributeLabels::IpuUtilisation
: Percentage of time spent waiting for one or more IPU syncs, measured in the last second.
: text: “ipu utilisation”

const std::string IPUAttributeLabels::IpuUtilisationSession
: Percentage of time spent waiting for one or more IPU syncs since the HSPs were set up.
: text: “ipu utilisation (session)”

const std::string IPUAttributeLabels::LinkCorrectableErrorCount
: IPU Link correctable error count.
: text: “link correctable error count”

const std::string IPUAttributeLabels::LinkSpeed

(PCIe) PCIe link speed available.

text: “link speed”

const std::string IPUAttributeLabels::LinkWidth

(PCIe) Number of PCIe lanes available.

text: “link width”

const std::string IPUAttributeLabels::MaxActiveCodeSize

Maximum active code size (bytes).

text: “max active code size (bytes)”

const std::string IPUAttributeLabels::MaxActiveDataSize

Maximum active data size (bytes).

text: “max active data size (bytes)”

const std::string IPUAttributeLabels::MaxActiveStackSize

Maximum active stack size (bytes).

text: “max active stack size (bytes)”

const std::string IPUAttributeLabels::MultiIpuDeviceId

Multi-IPU device the IPU belongs to.

text: “multi-ipu device id”

const std::string IPUAttributeLabels::MultiIpuDiscoveryMethod

Method used to discover multi-IPU groups.

text: “multi-ipu discovery method”

const std::string IPUAttributeLabels::NumaNode

NUMA node the IPU is on.

text: “numa node”

const std::string IPUAttributeLabels::NumIpuLinkSegments

(IPUoF) Number of IPU-Link segments.

text: “number of ipu-link segments”

const std::string IPUAttributeLabels::NumReplicas

(IPUoF) Number of replicas in the partition.

text: “number of replicas”

const std::string IPUAttributeLabels::PartitionId

(IPUoF) partition ID.

text: “ipuof partition id”

const std::string IPUAttributeLabels::PartitionSyncType

(IPUoF) sync configuration type, for example 'c2-compatible'.

text: "partition sync type"

const std::string IPUAttributeLabels::PciId

PCIe device identifier.

text: "pci id"

const std::string IPUAttributeLabels::PhysicalSlot

PCIe physical slot.

text: "pcie physical slot"

const std::string IPUAttributeLabels::ProcessStartTime

The start time of the process currently using the IPU.

text: "process start time"

const std::string IPUAttributeLabels::ReconfigurablePartition

(IPUoF) Set to 1 if the IPU is part of a reconfigurable partition.

text: "reconfigurable partition"

const std::string IPUAttributeLabels::RemoteBuffersSupported

Set to 1 if remote buffers are supported.

text: "remote buffers supported"

const std::string IPUAttributeLabels::SerialNumber

Serial number of the board.

text: "board serial number"

const std::string IPUAttributeLabels::TotalBoardPower

Total current power consumption as read by board level sensors.

Not used on IPU-Machines text: "total board power"

const std::string IPUAttributeLabels::UserExecutable

The name of the process using the device.

text: "user executable"

const std::string IPUAttributeLabels::UserName

The username of the user using the device.

text: "user name"

const std::string IPUAttributeLabels::UserProcessId

The process IDs of the process using the device.

text: "user process id"

const std::string IPUAttributeLabels::GatewayRoutingType

(IPUoF) GW-Link routing type.

text: “gateway routing type”

const std::string IPUAttributeLabels::IpuLinkSegmentId

(IPUoF) Identifier of IPU-Link segment.

text: “ipu link segment id”

const std::string IPUAttributeLabels::NumGcds

Number of Graph Compile Domains.

text: “number of gcds”

const std::string IPUAttributeLabels::FirmwareVersion

ICU Firmware version, specified as a <major.minor.patch> triple.

In development builds, this will be suffixed with branch and build information. text: “firmware version”

const std::string IPUAttributeLabels::IpuofServerError

(IPUoF) Set if error occurred while attempting to communicate with the IPUoF server (a ‘connection’ error), or if the IPUoF server was unable to use the device (a ‘device’ error) text: “ipuof server error”

const std::string IPUAttributeLabels::HostLinkCorrectableErrorCount

(PCIe) Host Link correctable error count.

text: “host link correctable error count”

const std::string IPUAttributeLabels::ApplicationHost

(IPUoF) IP address of the headnode where the application using this IPU is running.

text: “application host”

const std::string IPUAttributeLabels::IpuErrorState

Error state of the IPU.

Set to ‘ipu memory failure’ if the tile parity error thresholds have been exceeded. text: “ipu error state”

const std::string IPUAttributeLabels::ParityErrorCountThreshold

Threshold for number of parity errors to promote to a unrecoverable error.

text: “parity error count threshold”

const std::string IPUAttributeLabels::ParityErrorThresholdInterval

Threshold in seconds at which ‘num parity errors’ are promoted to an uncorrectable error.

text: “parity error threshold interval”

const std::string IPUAttributeLabels::IpumSoftwareVersion

(IPUoF) IPU-M software version.

text: “ipum software version”

const std::string IPUAttributeLabels::IpuPower

Power consumption of a single IPU.

Only available on IPU-Machines text: “ipu power”

const std::string IPUAttributeLabels::LinkCorrectableErrorCountSession

IPU Link correctable error count since device was last reset.

text: “link correctable error count (session)”

const std::string IPUAttributeLabels::HostLinkCorrectableErrorCountSession

(PCIe) Host-Link correctable error count since device was last reset.

text: “host link correctable error count (session)”

const std::string IPUAttributeLabels::BoardVariant

IPU board model name.

This will be identical to BoardType if this product only has a single variant. text: “board variant”

const std::string IPUAttributeLabels::GatewayWriteCombining

Gateway write combining status.

text: “gateway write combining”

const std::string IPUAttributeLabels::SecondaryPcieInterfaceSupported

Set to 1 if the secondary interface is supported.

text: “secondary pcie interface supported”

const std::string IPUAttributeLabels::ICUBootloaderVersion

ICU bootloader version, specified as a <major.minor.patch> triple.

In development builds, this will be suffixed with branch and build information. text: “icu bootloader version”

CHAPTER

# SEVEN

# EXAMPLES

## 7.1 Listing running applications

This example provides a list of the user processes currently running on IPUs within the system.

Example output:

```
$ gc-active-apps
Graphcore active apps:  1
 o application #0: [gc-powertest]
```

### 7.1.1 Python

```
import sys
import gcipuinfo

discovery_mode = gcipuinfo.DiscoverActivePartitionIPUs
if len(sys.argv) > 1:
  if sys.argv[1] == "--all-partitions":
    discovery_mode = gcipuinfo.DiscoverAllPartitionIPUs
  else:
    print("Error, unrecognised option.")
    print("Specify --all-partitions to show IPUs in all partitions")
    sys.exit(1)

inventory = gcipuinfo.gcipuinfo(discovery_mode)
apps = inventory.getNamedAttributeForAll(gcipuinfo.UserExecutable)

print("Graphcore active apps: " + str(len(apps)))
for index, exe_name in enumerate(apps):
  if exe_name:
    print(" o application #{}: [{}]".format(index, exe_name))
```

### 7.1.2 C++

```
#include <iostream>
#include <string.h>

#include "graphcore_target_access/gcipuinfo/IPUAttributeLabels.h"
#include "graphcore_target_access/gcipuinfo/gcipuinfo.h"

int main(int argc, char *argv[]) {

  DeviceDiscoveryMode discoveryMode = DiscoverActivePartitionIPUs;
  if (argc > 1) {
    if (strcmp(argv[1], "--all-partitions") == 0) {
      discoveryMode = DiscoverAllPartitionIPUs;
    } else {
      std::cerr << "Error, unrecognised option.\n";
```

(continues on next page)

(continued from previous page)

```
      std::cerr << "Specify --all-partitions to show IPUs in all partitions\n";
      std::exit(1);
    }
  }
  gcipuinfo inventory(discoveryMode);

  auto apps =
      inventory.getNamedAttributeForAll(IPUAttributeLabels::UserExecutable);

  std::cout << "Graphcore active apps: " << apps.size() << "\n";
  unsigned index = 0;
  for (auto &exeName : apps) {

    if (exeName.size()) {
      std::cout << " o application #" << index++ << ": [" << exeName << "]\n";
    }
    index++;
  }

  return 0;
}
```

### 7.1.3 Go

```
package main

import (
        "os"
        "fmt"
        "gcipuinfo"
)

func main() {

        discoveryMode := gcipuinfo.DiscoverActivePartitionIPUs
        if len(os.Args) > 1 {
                if os.Args[1] == "--all-partitions" {
                        discoveryMode = gcipuinfo.DiscoverAllPartitionIPUs
                } else {
                        fmt.Println("Error, unrecognised option.")
                        fmt.Println("Specify --all-partitions to show IPUs in all partitions")
                        os.Exit(1)
                }
        }

        inventory := gcipuinfo.NewGcipuinfo(discoveryMode)
        var exeNames []string = inventory.GetNamedAttributeForAll(gcipuinfo.UserExecutable)
        fmt.Println("Graphcore active apps: ", len(exeNames))
        for index, exeName := range exeNames {
                        if len(exeName) > 0 {
                                fmt.Printf(" o application #%d: [%s]\n", index, exeName)
                        }
        }
}
```

## 7.2 Listing a collection of attributes across all IPUs in the system

This example loops over the IPUs in the system and displays a subset of the attributes for each device. It runs forever, polling for updated attribute values. By default, only IPUs in the currently active partition or GCD are displayed. If you specify the --all-partitions flag, the gcipuinfo object is configured with the DiscoverAllPartitionIPUs option, which will retrieve information for IPUs in all known partitions.

Example output:

```
$ gc-get-attributes-for-all-devices
*** Iteration 0 ***

Device 0
  user process id : 62931
  user executable : ./example
  user name : ipuuser
  board ipu index : 3
  board serial number : 0026.0002.8203321
  clock : 1300MHz
  total board power : 46.0 C
  average board temp : 41.3 C
  ipu utilisation : 100.00%
  max active code size (bytes) : 36420
  max active data size (bytes) : 318
  max active stack size (bytes) : 1344

Device 1
  (device not in use by any known process)
  board ipu index : 2
  board serial number : 0026.0002.8203321
  clock : 1300MHz
  total board power : N/A
  average board temp : N/A
  ipu utilisation : 0.00%

Device 2
  (device not in use by any known process)
  board ipu index : 1
  board serial number : 0026.0001.8203321
  clock : 1300MHz
  total board power : N/A
  average board temp : N/A
  ipu utilisation : 0.00%
```

### 7.2.1 C++

```
#include <iostream>
#include <string.h>
#include <unistd.h>

#include "graphcore_target_access/gcipuinfo/IPUAttributeLabels.h"
#include "graphcore_target_access/gcipuinfo/gcipuinfo.h"

void printAttribute(const std::map<std::string, std::string> &map,
                    const std::string &key) {
  if (map.count(key) == 0)
    return;
  std::cout << "  " << key << " : " << map.at(key) << "\n";
}

int main(int argc, char *argv[]) {

  DeviceDiscoveryMode discoveryMode = DiscoverActivePartitionIPUs;
  if (argc > 1) {
    if (strcmp(argv[1], "--all-partitions") == 0) {
      discoveryMode = DiscoverAllPartitionIPUs;
    } else {
```

(continues on next page)

(continued from previous page)

```
      std::cerr << "Error, unrecognised option.\n";
      std::cerr << "Specify --all-partitions to show IPUs in all partitions\n";
      std::exit(1);
    }
  }
  gcipuinfo inventory(discoveryMode);

  std::cout << "Devices:\n";

  unsigned count = 0;
  while (1) {
    std::cout << "*** Iteration " << count++ << " ***\n\n";

    inventory.updateData(); // Refresh the device attribute values

    auto deviceMaps = inventory.getDevices();
    for (auto deviceMap : deviceMaps) {
      std::cout << "Device " << deviceMap.at(IPUAttributeLabels::DeviceId)
                << "\n";

      // Only devices that are in use will have a process id associated with
      // them
      if (deviceMap.count(IPUAttributeLabels::UserProcessId)) {
        printAttribute(deviceMap, IPUAttributeLabels::UserProcessId);
        printAttribute(deviceMap, IPUAttributeLabels::UserExecutable);
        printAttribute(deviceMap, IPUAttributeLabels::UserName);
      } else {
        std::cout << "  (device not in use by any known process)\n";
      }
      printAttribute(deviceMap, IPUAttributeLabels::BoardIpuIndex);
      printAttribute(deviceMap, IPUAttributeLabels::SerialNumber);
      printAttribute(deviceMap, IPUAttributeLabels::ClockFrequency);
      printAttribute(deviceMap, IPUAttributeLabels::TotalBoardPower);
      printAttribute(deviceMap, IPUAttributeLabels::AverageBoardTemp);
      printAttribute(deviceMap, IPUAttributeLabels::IpuUtilisation);
      printAttribute(deviceMap, IPUAttributeLabels::MaxActiveCodeSize);
      printAttribute(deviceMap, IPUAttributeLabels::MaxActiveDataSize);
      printAttribute(deviceMap, IPUAttributeLabels::MaxActiveStackSize);
      std::cout << "\n";
    }
    sleep(1);
  }

  return 0;
}
```

### 7.2.2 Go

```
package main

import (
        "os"
        "fmt"
        "time"
        "gcipuinfo"
)

func printAttribute(deviceMap map[string]interface{}, key string) {
        if deviceMap[key] != nil {
                fmt.Println("  ", key, " : ", deviceMap[key])
        }
}

func main() {
        discoveryMode := gcipuinfo.DiscoverActivePartitionIPUs
        if len(os.Args) > 1 {
                if os.Args[1] == "--all-partitions" {
```

(continues on next page)

(continued from previous page)

```
                discoveryMode = gcipuinfo.DiscoverAllPartitionIPUs
        } else {
                fmt.Println("Error, unrecognised option.")
                fmt.Println("Specify --all-partitions to show IPUs in all partitions")
                os.Exit(1)
        }
    }

    inventory := gcipuinfo.NewGcipuinfo(discoveryMode)

    count := 0
    for {
        fmt.Println("*** Iteration ", count, " ***\n")
        count++

        inventory.UpdateData()
        deviceMaps := inventory.GetDevices()
        for _, attribs := range deviceMaps {
            deviceMap := attribs.(map[string]interface{})

            fmt.Println("Device ", deviceMap[gcipuinfo.DeviceId])

            if deviceMap[gcipuinfo.UserProcessId] != nil {
                printAttribute(deviceMap, gcipuinfo.UserProcessId)
                printAttribute(deviceMap, gcipuinfo.UserExecutable)
                printAttribute(deviceMap, gcipuinfo.UserName)
            } else {
                fmt.Println("  (device not in use by any known process)")
            }
            printAttribute(deviceMap, gcipuinfo.BoardIpuIndex)
            printAttribute(deviceMap, gcipuinfo.SerialNumber)
            printAttribute(deviceMap, gcipuinfo.ClockFrequency)
            printAttribute(deviceMap, gcipuinfo.TotalBoardPower)
            printAttribute(deviceMap, gcipuinfo.AverageBoardTemp)
            printAttribute(deviceMap, gcipuinfo.IpuUtilisation)
            printAttribute(deviceMap, gcipuinfo.MaxActiveCodeSize)
            printAttribute(deviceMap, gcipuinfo.MaxActiveDataSize)
            printAttribute(deviceMap, gcipuinfo.MaxActiveStackSize)
        }
        fmt.Println("")
        time.Sleep(1 * time.Second)
    }
}
```

## 7.3 Graph power consumption of IPUs

### 7.3.1 Python

```
import argparse
import time
import sys
import math
import os

# Requires asciichartpy: pip3 install --user asciichartpy
import asciichartpy
import gcipuinfo

ipu_info = gcipuinfo.gcipuinfo()
num_devices = len(ipu_info.getDevices())
if num_devices == 0:
    print("gc_power_consumption.py: error: no IPUs detected", file=sys.stderr)
    exit(-1)


```

(continues on next page)

(continued from previous page)

```
def get_ipu_power_single(device_id):
    if 0 <= device_id and device_id < num_devices:
        return pow_to_float(
            ipu_info.getNamedAttributeForDevice(device_id, gcipuinfo.IpuPower)
        )
    else:
        print(
            f"gc_power_consumption.py: error: device id {device_id} does not exist (valid range is 0-{num_devices-1})",
            file=sys.stderr,
        )
        exit(-1)


def get_ipu_power_from_device_list(devices):
    powers = []
    for device_id in devices:
        pow = ipu_info.getNamedAttributeForDevice(device_id, gcipuinfo.IpuPower)
        if pow != "N/A":
            powers.append(pow_to_float(pow))
    return powers

def get_ipu_power_all():
    device_powers = ipu_info.getNamedAttributeForAll(gcipuinfo.IpuPower)
    return [pow_to_float(pow) for pow in device_powers if pow != "N/A"]


def pow_to_float(pow):
    # Power is reported in the format xxx.xxW, so remove the last character.
    # We also handle the case when the power reports as N/A.
    try:
        return float(pow[:-1])
    except ValueError:
        return 0


def draw_graph(power_history, mode, num_devices, device_ids, min, max, width, height):
    graph_cfg = {
        "height": height - 3,  # Leave room for the title at the top
        "format": "{:8.2f}W ",
        "min": min if min else 0,
    }
    if max and max > graph_cfg["min"]:
        graph_cfg["max"] = max

    if device_ids:
        title_str = mode.capitalize() + " power consumption for IPUs: " + ", ".join(map(str, device_ids))
    else:
        title_str = mode.capitalize() + " power consumption for " + str(num_devices) + " IPUs"

    print(title_str.center(width))
    graph = asciichartpy.plot(power_history, graph_cfg) + "\n"
    sys.stdout.buffer.write(graph.encode("utf-8"))
    sys.stdout.flush()


def main():
    parser = argparse.ArgumentParser(description="Display a console graph of IPU power consumption over time")
    parser.add_argument("--min", type=float, help="Minimum y-axis value, in watts")
    parser.add_argument("--max", type=float, help="Maximum y-axis value, in watts")
    parser.add_argument(
        "--interval",
        type=float,
        default=1,
        help="Interval between power queries, in seconds",
    )
    parser.add_argument(
        "--devices", type=int, nargs="+", help="only query specific devices"
    )
    parser.add_argument('--mode',  help='Simulator IPU architecture',
                        choices=["mean", "total"],
```

(continues on next page)

(continued from previous page)

```
                    required=False, default="mean")

    # This example assumes per-IPU power sensors, which are not available on
    # C2 devices
    if ipu_info.getNamedAttributeForDevice(0, gcipuinfo.BoardType) != "M2000":
        print("This program is only supported on IPU-Machine devices")
        sys.exit(1)

    args = parser.parse_args()

    try:
        term_width, term_height = os.get_terminal_size()
    except OSError:
        print(
            "gc_power_consumption.py: warning: stdout is not attached to a tty, using 50x50 graph",
            file=sys.stderr,
        )
        term_width, term_height = 50

    power_history = []
    max_entries = term_width - 15  # Leave enough room for the y-axis labels

    while True:
        if not args.devices:
            powers = get_ipu_power_all()
        else:
            powers = get_ipu_power_from_device_list(args.devices)
        if len(powers) > 0:
            if args.mode == "mean":
                val = sum(powers) / len(powers)
            else:
                val = sum(powers)
            power_history.append(val)
        if len(power_history) > max_entries:
            power_history = power_history[1:]

        if any([power != 0 for power in power_history]):
            draw_graph(
                power_history,
                args.mode,
                len(powers),
                args.devices,
                min=args.min,
                max=args.max,
                width=term_width,
                height=term_height,
            )
        else:
            print("  -- Waiting for devices to power on...", end="\r")

        time.sleep(args.interval)


if __name__ == "__main__":
    main()
```

## 7.4 Display a message when device 0 is in use

Example output when device 0 is idle:

```
$ go run gc_track_device_0.go
Device 0 is idle
```

Example output when device 0 is in use:

```
$ go run gc_track_device_0.go
User exampleuser is running application gc-powertest on device 0
```

### 7.4.1 C++

```cpp
#include <iostream>
#include <string>

#include "graphcore_target_access/gcipuinfo/IPUAttributeLabels.h"
#include "graphcore_target_access/gcipuinfo/gcipuinfo.h"

int main() {
  gcipuinfo inventory;
  int deviceOfInterest = 0; // device id

  std::string exeName = inventory.getNamedAttributeForDevice(
      deviceOfInterest, IPUAttributeLabels::UserExecutable);
  std::string userName = inventory.getNamedAttributeForDevice(
      deviceOfInterest, IPUAttributeLabels::UserName);

  if (!exeName.empty()) {
    std::cout << "User " << userName << " is running application " << exeName
              << " on device " << deviceOfInterest << "\n";
  } else {
    std::cout << "Device " << deviceOfInterest << " is idle\n";
  }

  return 0;
}
```

### 7.4.2 Python

```python
import gcipuinfo

inventory = gcipuinfo.gcipuinfo()
device_of_interest = 0
exe_name = inventory.getNamedAttributeForDevice(device_of_interest, gcipuinfo.UserExecutable)
user_name = inventory.getNamedAttributeForDevice(device_of_interest, gcipuinfo.UserName)

if exe_name:
  print("User " + user_name + " is running " + exe_name + " on device " + str(device_of_interest))
else:
  print("Device " + str(device_of_interest) + " is idle")
```

### 7.4.3 Go

```go
package main

import (
        "fmt"
        "gcipuinfo"
)

func main() {

        inventory := gcipuinfo.NewGcipuinfo()
        deviceOfInterest := 0 // device id
        exeName := inventory.GetNamedAttributeForDevice(deviceOfInterest, gcipuinfo.UserExecutable)
        userName := inventory.GetNamedAttributeForDevice(deviceOfInterest, gcipuinfo.UserName)
        if len(exeName) > 0 {
                fmt.Printf("User %s is running application %s on device %d\n", userName, exeName, deviceOfInterest)
        } else {
```

(continues on next page)

(continued from previous page)

```
17              fmt.Printf("Device %d is idle\n", deviceOfInterest)
18      }
19  }
```

## 7.5 Display application event record entries

This example program demonstrates how to retrieve and interpret application event records. The program takes one or more application event record paths as command line parameters and displays the contents of any events it finds in the specified paths, along with a list of all IPU-Machine hosts named in at least one event.

**Note:** For demonstration purposes there is a write_example_event_record_entry.py script included in the examples/python directory which can write example application event record entries.

Example output when run with two application event records that are both empty:

```
$ python gc_event_record.py /tmp/ipu_app_event_record_1 /tmp/ipu_app_event_record_2
== Checking application event record:  /tmp/ipu_app_event_record_1
No event record entry found.
== Checking application event record:  /tmp/ipu_app_event_record_2
No event record entry found.
```

Example output when there is an entry in /tmp/ipu_app_event_record_1 but not in /tmp/ipu_app_event_record_2:

```
$ python gc_event_record.py /tmp/ipu_app_event_record_1 /tmp/ipu_app_event_record_2
== Checking application event record: /tmp/ipu_app_event_record_1
{
  "attached ipu hosts": [
    "10.1.5.10"
  ],
  "attached ipus": [
    0,
    1
  ],
  "command line": "./example program with args",
  "description": "A nonrecoverable error has occurred.",
  "event record path": "/tmp/ipu_app_event_record_1/last_event.json",
  "partition": "p1",
  "pid": "35024",
  "severity": "nonrecoverable",
  "specific ipu hosts": [
    "10.1.5.10"
  ],
  "specific ipus": [
    0
  ],
  "timestamp": "2021-11-29T17:23:54.989896Z"
}
Last event was an error.

== Checking application event record: /tmp/ipu_app_event_record_2
No event record found.

== IPU-Machine hosts involved in errors, across all application event records:
  10.1.5.10
```

### 7.5.1 C++

```cpp
#include <iostream>
#include <set>

#include "graphcore_target_access/gcipuinfo/gcipuinfo.h"
#include <nlohmann/json.hpp>
using json = nlohmann::json;

int main(int argc, char *argv[]) {

  if (argc < 2) {
    std::cerr << __FILE__ << ": error requires at least one argument\n";
    return 1;
  }

  gcipuinfo inventory;

  std::set<std::string> hostsNamedInErrors;

  for (int i = 1; i < argc; i++) {
    std::cout << "== Checking application event record: " << argv[i] << "\n";

    std::string eventRecordJSON =
        inventory.getLastAppEventRecordAsJSON(gcipuinfo::EventSevNone, argv[i]);

    json j = json::parse(eventRecordJSON);
    if (j.empty()) {
      std::cout << "No event record entry found.\n";
    } else {
      std::cout << j.dump(2) << "\n";

      // If the event has named specific IPUs
      // record their associated machine names
      bool foundSpecificHosts = false;
      if (j.contains(gcipuinfo::keySpecificIPUHosts)) {
        auto hosts = j[gcipuinfo::keySpecificIPUHosts];
        foundSpecificHosts = hosts.size() > 0;
        for (const auto &element : hosts.items()) {
          hostsNamedInErrors.insert(element.value().dump());
        }
      }
      if (!foundSpecificHosts && j.contains(gcipuinfo::keyAttachedIPUHosts)) {
        // if no specific IPUs have been mentioned in this error,
        // fall back to recording hosts of all attached IPUs
        auto hosts = j[gcipuinfo::keyAttachedIPUHosts];
        for (const auto &element : hosts.items()) {
          hostsNamedInErrors.insert(element.value().dump());
        }
      }
      auto severity = inventory.getLastAppEventRecordSeverity(argv[i]);
      if (severity > gcipuinfo::EventSevWarning) {
        std::cout << "Last event was an error.\n";
      }
    }
    std::cout << "\n";
  }

  if (hostsNamedInErrors.size() > 0) {
    std::cout << "== IPU-Machine hosts involved in errors, across all "
                 "application event records:\n";
    for (auto host : hostsNamedInErrors) {
      std::cout << "  " << host << "\n";
    }
  }

  return 0;
}
```

### 7.5.2 Python

```python
import sys
import json
import gcipuinfo

if len(sys.argv) < 2:
  print("Error, requires at least one argument")
  sys.exit(1)

inventory = gcipuinfo.gcipuinfo()

hosts_named_in_errors = set()

for i in range(1,len(sys.argv)):
  path = sys.argv[i]
  print("== Checking application event record: " + path)
  event_record_json = inventory.getLastAppEventRecordAsJSON(inventory.EventSevNone, path)
  j = json.loads(event_record_json)
  if not j:
    print("No event record entry found.")
  else:
    print(event_record_json)

    # If the event has named specific IPUs
    # record their associated machine names
    found_specific_hosts = False
    if inventory.keySpecificIPUHosts in j:
      hosts = j[inventory.keySpecificIPUHosts]
      found_specific_hosts = len(hosts) > 0
      for host in hosts:
        hosts_named_in_errors.add(host)

    # if no specific IPUs have been mentioned in this error,
    # fall back to recording hosts of all attached IPUs
    if not found_specific_hosts and inventory.keyAttachedIPUHosts in j:
      hosts = j[inventory.keyAttachedIPUHosts]
      for host in hosts:
        hosts_named_in_errors.add(host)
    if (inventory.getLastAppEventRecordSeverity(path) > inventory.EventSevWarning):
      print("Last event was an error.")
  print("")

if len(hosts_named_in_errors) > 0:
  print("== IPU-Machine hosts involved in errors, across all application event records:")
  for host in hosts_named_in_errors:
    print("  " + host)
```

### 7.5.3 Go

```go
package main

import (
        "fmt"
        "os"
        "gcipuinfo"
        "encoding/json"
)

func main() {
        inventory := gcipuinfo.NewGcipuinfo()

        if len(os.Args) < 2 {
                panic("Error, requires at least one argument")
        }

        hostsNamedInErrors := map[string]bool{}

```

(continues on next page)

(continued from previous page)

```
        for i := 1; i < len(os.Args); i++ {
                path := os.Args[i]
                fmt.Println("== Checking application event record: ", path)
                eventRecordJSON := inventory.GetLastAppEventRecordAsJSON(gcipuinfo.EventSevNone, path)
                var eventRecord map[string]interface{}
                json.Unmarshal([]byte(eventRecordJSON), &eventRecord)
                if len(eventRecord) == 0 {
                        fmt.Printf("No event record entry found.\n")
                } else {
                        fmt.Println(eventRecordJSON)

                        // If the event has named specific IPUs
                        // record their associated machine names
                        foundSpecificHosts := false
                        if _, ok := eventRecord[gcipuinfo.KeySpecificIPUHosts]; ok {
                                hosts := eventRecord[gcipuinfo.KeySpecificIPUHosts].([]interface{})
                                foundSpecificHosts = len(hosts) > 0
                                for _, host := range hosts {
                                        hostsNamedInErrors[fmt.Sprint(host)] = true
                                }
                        }
                        if !foundSpecificHosts {
                                // if no specific IPUs have been mentioned in this error,
                                // fall back to recording hosts of all attached IPUs
                                if _, ok := eventRecord[gcipuinfo.KeyAttachedIPUHosts]; ok {
                                        hosts := eventRecord[gcipuinfo.KeyAttachedIPUHosts].([]interface{})
                                        for _, host := range hosts {
                                                hostsNamedInErrors[fmt.Sprint(host)] = true
                                        }
                                }
                        }
                        severity := inventory.GetLastAppEventRecordSeverity(path);
                        if (severity > gcipuinfo.EventSevWarning) {
                                fmt.Println("Last event was an error.\n");
                        }
                }
        }
        if len(hostsNamedInErrors) > 0 {
                fmt.Printf("== IPU-Machine hosts involved in errors, across all application event records:\n");
                for host, _ := range hostsNamedInErrors {
                        fmt.Printf("  %s\n", host)
                }
        }
}
```

## 7.6 Display device health-check result

This example runs forever displaying device health-check results, both as a raw JSON string and then, to demonstrate how to parse the output, in a summarised message. By default, the health checks will only be run on IPUs in the active partition. If you specify the --all-partitions flag, the gcipuinfo object is configured with the DiscoverAllPartitionIPUs option, and health checks will be run on IPUs in all known partitions.

Example output when there are no failing devices:

```
$ python gc_health_check.py
** Iteration 0 ***
Raw JSON:
{}
No errors

** Iteration 1 ***
Raw JSON:
{}
No errors

** Iteration 2 ***
```

(continues on next page)

(continued from previous page)

```
Raw JSON:
{}
No errors
...
```

Example output when device 1 on 10.1.5.10 has failed:

```
$ python gc_health_check.py
** Iteration 0 ***
Raw JSON:
{
  "hosts": {
    "10.1.5.10": [
      {
        "error": "connection",
        "id": "1",
        "partition": "p1",
        "board ipu index": "2"
      }
    ]
  }
}

Parsed:
  host: 10.1.5.10
    device id: 1, partition: p1, error: connection


...
```

### 7.6.1 C++

```
#include <chrono>
#include <iostream>
#include <string.h>
#include <thread>

#include "graphcore_target_access/gcipuinfo/gcipuinfo.h"
#include <nlohmann/json.hpp>

using json = nlohmann::json;

int main(int argc, char *argv[]) {

  DeviceDiscoveryMode discoveryMode = DiscoverActivePartitionIPUs;
  if (argc > 1) {
    if (strcmp(argv[1], "--all-partitions") == 0) {
      discoveryMode = DiscoverAllPartitionIPUs;
    } else {
      std::cerr << "Error, unrecognised option.\n";
      std::cerr << "Specify --all-partitions to show IPUs in all partitions\n";
      std::exit(1);
    }
  }
  gcipuinfo inventory(discoveryMode);

  unsigned count = 0;
  unsigned numIPUs = inventory.getDevices().size();
  while (true) {
    std::cout << "*** Iteration " << count++ << " ***\n\n";
    std::cout << "Checking " << numIPUs << " devices:\n";
    // Set 100ms timeout on response from each IPU.  Don't check active IPUs.
    std::string devicesHealth = inventory.checkHealthOfDevices(100, false);
    std::cout << "Raw JSON: " << devicesHealth << "\n";
    json j = json::parse(devicesHealth);
    if (j.contains("hosts")) {
      std::cout << "Parsed:\n";
```

(continues on next page)

(continued from previous page)

```
    auto hosts = j["hosts"];
    for (auto host : hosts.items()) {
      std::cout << "  host: " << host.key() << "\n";
      for (auto device : host.value().items()) {
        std::cout << "    device id: " << device.value()["id"]
                  << ", partition: " << device.value()["partition"]
                  << ", error: " << device.value()["error"] << "\n";
      }
    }
  } else if (j.contains("error")) {
    std::cout << "Parsed:\n";
    std::cout << "    error: " << j["error"]
              << ", description: " << j["description"] << "\n";
  } else {
    std::cout << "No errors\n";
  }
  std::cout << "\n";
  std::this_thread::sleep_for(std::chrono::milliseconds(10));
 }

 return 0;
}
```

### 7.6.2 Python

```
import json
import time
import sys
import gcipuinfo

discovery_mode = gcipuinfo.DiscoverActivePartitionIPUs
if len(sys.argv) > 1:
  if sys.argv[1] == "--all-partitions":
    discovery_mode = gcipuinfo.DiscoverAllPartitionIPUs
  else:
    print("Error, unrecognised option.")
    print("Specify --all-partitions to show IPUs in all partitions")
    sys.exit(1)

inventory = gcipuinfo.gcipuinfo(discovery_mode)
num_ipus = len(inventory.getDevices())

count = 0
while True:
  print("** Iteration " + str(count) + " ***")
  print("Checking " + str(num_ipus) + " devices")
  devicesHealth = inventory.checkHealthOfDevices(100, False)
  print("Raw JSON: \n" + devicesHealth)
  j = json.loads(devicesHealth)
  if "hosts" in j:
    print("Parsed:")
    hosts = j["hosts"]
    for host_name in hosts:
      print("  host: " + host_name)
      for device in hosts[host_name]:
        print("    device id: " + device["id"] + ", partition: " + device["partition"] + ", error: " + device["error"])
  elif "error" in j:
    print("Parsed:")
    print("  error: " + j["error"] + ", description: " + j["description"])
  else:
    print("No errors")
  print("")
  count = count + 1
  time.sleep(0.01)
```

### 7.6.3 Go

```go
package main

import (
	"os"
	"fmt"
	"gcipuinfo"
	"encoding/json"
	"time"
)

func main() {
	discoveryMode := gcipuinfo.DiscoverActivePartitionIPUs
	if len(os.Args) > 1 {
		if os.Args[1] == "--all-partitions" {
			discoveryMode = gcipuinfo.DiscoverAllPartitionIPUs
		} else {
			fmt.Println("Error, unrecognised option.")
			fmt.Println("Specify --all-partitions to show IPUs in all partitions")
			os.Exit(1)
		}
	}

	inventory := gcipuinfo.NewGcipuinfo(discoveryMode)
	numIPUs := len(inventory.GetDevices())

	count := 0
	for {
		fmt.Println("*** Iteration ", count, " ***\n")
		fmt.Println("Checking ", numIPUs, " devices\n")
		count++

		var devicesHealth string
		devicesHealth = inventory.CheckHealthOfDevices(100, false)
		fmt.Println("Raw JSON:", devicesHealth)

		var result map[string]interface{}
		json.Unmarshal([]byte(devicesHealth), &result)
		if _, ok := result["hosts"]; ok {
			fmt.Println("Parsed:")
			hosts := result["hosts"].(map[string]interface{})
			for hostName, hostVal := range hosts {
				fmt.Println("  host: ", hostName)
				devices := hostVal.([]interface{})
				for _, deviceVal := range devices {
					device := deviceVal.(map[string]interface{})
					fmt.Println("    device id: ", device["id"], "partition: ", device["partition"], ", error: ", device["error
↪"])
				}
			}
		} else if _, ok := result["error"]; ok {
			fmt.Println("Parsed:")
			fmt.Println("  error: ", result["error"], ", description: ", result["description"])
		} else {
			fmt.Println("No errors")
		}
		time.Sleep(time.Duration(10) * time.Millisecond)
	}
}
```

CHAPTER

# EIGHT

# INDEX

CHAPTER

NINE

# TRADEMARKS & COPYRIGHT

# INDEX

## H



## I



## K



## L



## M



## N



## P

## R



## S



## T



## U