



# DATACENTER GPU MANAGER API MANUAL

v2.0 | July 2020

**Reference Manual**



# TABLE OF CONTENTS

<b>Chapter 1. Modules</b> .....	<b>1</b>
1.1. Administrative.....	<b>2</b>
Init and Shutdown.....	2
Auxiliary information about DCGM engine.....	2
1.1.1. Init and Shutdown.....	2
dcmgInit.....	2
dcmgShutdown.....	2
dcmgStartEmbedded.....	3
dcmgStartEmbedded_v2.....	3
dcmgStopEmbedded.....	4
dcmgConnect.....	4
dcmgConnect_v2.....	5
dcmgDisconnect.....	6
1.1.2. Auxiliary information about DCGM engine.....	6
dcmgVersionInfo.....	6
dcmgHostengineVersionInfo.....	7
dcmgHostengineSetLoggingSeverity.....	7
dcmgHostengineIsHealthy.....	8
1.2. System.....	<b>8</b>
Discovery.....	8
Grouping.....	8
Field Grouping.....	8
Status handling.....	8
1.2.1. Discovery.....	9
dcmgGetAllDevices.....	9
dcmgGetAllSupportedDevices.....	9
dcmgGetDeviceAttributes.....	10
dcmgGetEntityGroupEntities.....	11
dcmgGetGpuInstanceHierarchy.....	11
dcmgGetNvLinkLinkStatus.....	12
1.2.2. Grouping.....	12
dcmgGroupCreate.....	13
dcmgGroupDestroy.....	14
dcmgGroupAddDevice.....	14
dcmgGroupAddEntity.....	15
dcmgGroupRemoveDevice.....	15
dcmgGroupRemoveEntity.....	16
dcmgGroupGetInfo.....	17
dcmgGroupGetAllIds.....	17
1.2.3. Field Grouping.....	18

dcgmFieldGroupCreate.....	18
dcgmFieldGroupDestroy.....	19
dcgmFieldGroupGetInfo.....	19
dcgmFieldGroupGetAll.....	20
1.2.4. Status handling.....	20
dcgmStatusCreate.....	20
dcgmStatusDestroy.....	21
dcgmStatusGetCount.....	21
dcgmStatusPopError.....	22
dcgmStatusClear.....	22
1.3. Configuration.....	23
Setup and management.....	23
Manual Invocation.....	23
1.3.1. Setup and management.....	23
dcgmConfigSet.....	23
dcgmConfigGet.....	24
1.3.2. Manual Invocation.....	25
dcgmConfigEnforce.....	26
1.4. Field APIs.....	26
dcgmWatchFields.....	27
dcgmUnwatchFields.....	28
dcgmGetValuesSince.....	28
dcgmGetValuesSince_v2.....	29
dcgmGetLatestValues.....	30
dcgmGetLatestValues_v2.....	31
dcgmGetLatestValuesForFields.....	32
dcgmEntityGetLatestValues.....	32
dcgmEntitiesGetLatestValues.....	33
dcgmGetFieldSummary.....	34
1.5. Process Statistics.....	34
dcgmWatchPidFields.....	34
dcgmGetPidInfo.....	35
1.6. Job Statistics.....	36
dcgmWatchJobFields.....	36
dcgmJobStartStats.....	37
dcgmJobStopStats.....	37
dcgmJobGetStats.....	38
dcgmJobRemove.....	38
dcgmJobRemoveAll.....	39
1.7. Health Monitor.....	39
dcgmHealthSet.....	40
dcgmHealthSet_v2.....	40
dcgmHealthGet.....	41

dcgmHealthCheck.....	41
1.8. Policies.....	42
Setup and Management.....	42
Manual Invocation.....	42
1.8.1. Setup and Management.....	42
dcgmPolicySet.....	43
dcgmPolicyGet.....	43
dcgmPolicyRegister.....	44
dcgmPolicyUnregister.....	45
1.8.2. Manual Invocation.....	46
dcgmActionValidate.....	46
dcgmActionValidate_v2.....	47
dcgmRunDiagnostic.....	47
1.9. Topology.....	48
dcgmGetDeviceTopology.....	48
dcgmGetGroupTopology.....	49
1.10. Metadata.....	49
dcgmIntrospectToggleState.....	50
dcgmIntrospectGetFieldsMemoryUsage.....	50
dcgmIntrospectGetHostengineMemoryUsage.....	51
dcgmIntrospectGetFieldsExecTime.....	52
dcgmIntrospectGetHostengineCpuUtilization.....	52
dcgmIntrospectUpdateAll.....	53
1.11. Topology.....	54
dcgmSelectGpusByTopology.....	54
1.12. Modules.....	55
dcgmModuleBlacklist.....	55
dcgmModuleGetStatuses.....	55
1.13. Profiling.....	56
dcgmProfGetSupportedMetricGroups.....	56
dcgmProfWatchFields.....	57
dcgmProfUnwatchFields.....	58
dcgmProfPause.....	58
dcgmProfResume.....	59
1.14. Enums and Macros.....	59
dcgmOperationMode_t.....	59
dcgmOrder_t.....	60
dcgmReturn_t.....	60
dcgmGroupType_t.....	63
dcgmChipArchitecture_t.....	63
dcgmConfigType_t.....	64
dcgmConfigPowerLimitType_t.....	64
MAKE_DCGM_VERSION.....	64

DCGM_INT32_BLANK.....	64
DCGM_INT64_BLANK.....	64
DCGM_FP64_BLANK.....	64
DCGM_STR_BLANK.....	65
DCGM_INT32_NOT_FOUND.....	65
DCGM_INT64_NOT_FOUND.....	65
DCGM_FP64_NOT_FOUND.....	65
DCGM_STR_NOT_FOUND.....	65
DCGM_INT32_NOT_SUPPORTED.....	65
DCGM_INT64_NOT_SUPPORTED.....	65
DCGM_FP64_NOT_SUPPORTED.....	65
DCGM_STR_NOT_SUPPORTED.....	65
DCGM_INT32_NOT_PERMISSIONED.....	66
DCGM_INT64_NOT_PERMISSIONED.....	66
DCGM_FP64_NOT_PERMISSIONED.....	66
DCGM_STR_NOT_PERMISSIONED.....	66
DCGM_INT32_IS_BLANK.....	66
DCGM_INT64_IS_BLANK.....	66
DCGM_FP64_IS_BLANK.....	66
DCGM_STR_IS_BLANK.....	67
DCGM_MAX_NUM_DEVICES.....	67
DCGM_NVLINK_MAX_LINKS_PER_GPU.....	67
DCGM_NVLINK_MAX_LINKS_PER_GPU_LEGACY1.....	67
DCGM_MAX_NUM_SWITCHES.....	67
DCGM_NVLINK_MAX_LINKS_PER_NVSWITCH.....	67
DCGM_MAX_VGPU_INSTANCES_PER_PGPU.....	67
DCGM_MAX_STR_LENGTH.....	67
DCGM_MAX_CLOCKS.....	67
DCGM_MAX_NUM_GROUPS.....	67
DCGM_MAX_FBC_SESSIONS.....	68
DCGM_VGPU_NAME_BUFFER_SIZE.....	68
DCGM_GRID_LICENSE_BUFFER_SIZE.....	68
DCGM_CONFIG_COMPUTEMODE_DEFAULT.....	68
DCGM_CONFIG_COMPUTEMODE_PROHIBITED.....	68
DCGM_CONFIG_COMPUTEMODE_EXCLUSIVE_PROCESS.....	68
DCGM_HE_PORT_NUMBER.....	68
DCGM_GROUP_ALL_GPUS.....	68
DCGM_GROUP_MAX_ENTITIES.....	68
1.16. Field Types.....	68
DCGM_FT_BINARY.....	69
DCGM_FT_DOUBLE.....	69
DCGM_FT_INT64.....	69
DCGM_FT_STRING.....	69

DCGM_FT_TIMESTAMP.....	69
1.17. Field Scope.....	69
DCGM_FS_GLOBAL.....	69
DCGM_FS_ENTITY.....	69
DCGM_FS_DEVICE.....	69
1.18. Field Constants.....	69
dcmGpuVirtualizationMode_t.....	70
DCGM_CUDA_COMPUTE_CAPABILITY_MAJOR.....	70
DCGM_CLOCKS_THROTTLE_REASON_GPU_IDLE.....	70
DCGM_CLOCKS_THROTTLE_REASON_CLOCKS_SETTING.....	70
DCGM_CLOCKS_THROTTLE_REASON_SW_POWER_CAP.....	71
DCGM_CLOCKS_THROTTLE_REASON_HW_SLOWDOWN.....	71
DCGM_CLOCKS_THROTTLE_REASON_SYNC_BOOST.....	71
DCGM_CLOCKS_THROTTLE_REASON_SW_THERMAL.....	71
DCGM_CLOCKS_THROTTLE_REASON_HW_THERMAL.....	72
DCGM_CLOCKS_THROTTLE_REASON_HW_POWER_BRAKE.....	72
DCGM_CLOCKS_THROTTLE_REASON_DISPLAY_CLOCKS.....	72
1.19. Field Entity.....	72
dcmFieldEntityGroup_t.....	72
dcmFieldEid_t.....	73
1.20. Field Identifiers.....	73
DcmFieldGetById.....	73
DcmFieldGetByTag.....	73
DcmFieldsInit.....	74
DcmFieldsTerm.....	74
DcmFieldsGetEntityGroupString.....	74
DCGM_FI_UNKNOWN.....	74
DCGM_FI_DRIVER_VERSION.....	74
DCGM_FI_DEV_COUNT.....	75
DCGM_FI_CUDA_DRIVER_VERSION.....	75
DCGM_FI_DEV_NAME.....	75
DCGM_FI_DEV_BRAND.....	75
DCGM_FI_DEV_NVML_INDEX.....	75
DCGM_FI_DEV_SERIAL.....	75
DCGM_FI_DEV_UUID.....	75
DCGM_FI_DEV_MINOR_NUMBER.....	75
DCGM_FI_DEV_OEM_INFOROM_VER.....	75
DCGM_FI_DEV_PCI_BUSID.....	75
DCGM_FI_DEV_PCI_COMBINED_ID.....	75
DCGM_FI_DEV_PCI_SUBSYS_ID.....	76
DCGM_FI_GPU_TOPOLOGY_PCI.....	76
DCGM_FI_GPU_TOPOLOGY_NVLINK.....	76
DCGM_FI_GPU_TOPOLOGY_AFFINITY.....	76

DCGM_FI_DEV_CUDA_COMPUTE_CAPABILITY.....	76
DCGM_FI_DEV_COMPUTE_MODE.....	76
DCGM_FI_DEV_PERSISTENCE_MODE.....	76
DCGM_FI_DEV_MIG_MODE.....	76
DCGM_FI_DEV_CUDA_VISIBLE_DEVICES_STR.....	76
DCGM_FI_DEV_MIG_MAX_SLICES.....	76
DCGM_FI_DEV_CPU_AFFINITY_0.....	76
DCGM_FI_DEV_CPU_AFFINITY_1.....	77
DCGM_FI_DEV_CPU_AFFINITY_2.....	77
DCGM_FI_DEV_CPU_AFFINITY_3.....	77
DCGM_FI_DEV_ECC_INFOROM_VER.....	77
DCGM_FI_DEV_POWER_INFOROM_VER.....	77
DCGM_FI_DEV_INFOROM_IMAGE_VER.....	77
DCGM_FI_DEV_INFOROM_CONFIG_CHECK.....	77
DCGM_FI_DEV_INFOROM_CONFIG_VALID.....	77
DCGM_FI_DEV_VBIOS_VERSION.....	77
DCGM_FI_DEV_BAR1_TOTAL.....	77
DCGM_FI_SYNC_BOOST.....	77
DCGM_FI_DEV_BAR1_USED.....	78
DCGM_FI_DEV_BAR1_FREE.....	78
DCGM_FI_DEV_SM_CLOCK.....	78
DCGM_FI_DEV_MEM_CLOCK.....	78
DCGM_FI_DEV_VIDEO_CLOCK.....	78
DCGM_FI_DEV_APP_SM_CLOCK.....	78
DCGM_FI_DEV_APP_MEM_CLOCK.....	78
DCGM_FI_DEV_CLOCK_THROTTLE_REASONS.....	78
DCGM_FI_DEV_MAX_SM_CLOCK.....	78
DCGM_FI_DEV_MAX_MEM_CLOCK.....	78
DCGM_FI_DEV_MAX_VIDEO_CLOCK.....	78
DCGM_FI_DEV_AUTOBOOST.....	79
DCGM_FI_DEV_SUPPORTED_CLOCKS.....	79
DCGM_FI_DEV_MEMORY_TEMP.....	79
DCGM_FI_DEV_GPU_TEMP.....	79
DCGM_FI_DEV_MEM_MAX_OP_TEMP.....	79
DCGM_FI_DEV_GPU_MAX_OP_TEMP.....	79
DCGM_FI_DEV_POWER_USAGE.....	79
DCGM_FI_DEV_TOTAL_ENERGY_CONSUMPTION.....	79
DCGM_FI_DEV_SLOWDOWN_TEMP.....	79
DCGM_FI_DEV_SHUTDOWN_TEMP.....	79
DCGM_FI_DEV_POWER_MGMT_LIMIT.....	79
DCGM_FI_DEV_POWER_MGMT_LIMIT_MIN.....	80
DCGM_FI_DEV_POWER_MGMT_LIMIT_MAX.....	80
DCGM_FI_DEV_POWER_MGMT_LIMIT_DEF.....	80

DCGM_FI_DEV_ENFORCED_POWER_LIMIT.....	80
DCGM_FI_DEV_PSTATE.....	80
DCGM_FI_DEV_FAN_SPEED.....	80
DCGM_FI_DEV_PCIE_TX_THROUGHPUT.....	80
DCGM_FI_DEV_PCIE_RX_THROUGHPUT.....	80
DCGM_FI_DEV_PCIE_REPLAY_COUNTER.....	80
DCGM_FI_DEV_GPU_UTIL.....	80
DCGM_FI_DEV_MEM_COPY_UTIL.....	81
DCGM_FI_DEV_ACCOUNTING_DATA.....	81
DCGM_FI_DEV_ENC_UTIL.....	81
DCGM_FI_DEV_DEC_UTIL.....	81
DCGM_FI_DEV_MEM_COPY_UTIL_SAMPLES.....	81
DCGM_FI_DEV_GRAPHICS_PIDS.....	81
DCGM_FI_DEV_COMPUTE_PIDS.....	81
DCGM_FI_DEV_XID_ERRORS.....	81
DCGM_FI_DEV_PCIE_MAX_LINK_GEN.....	81
DCGM_FI_DEV_PCIE_MAX_LINK_WIDTH.....	81
DCGM_FI_DEV_PCIE_LINK_GEN.....	82
DCGM_FI_DEV_PCIE_LINK_WIDTH.....	82
DCGM_FI_DEV_POWER_VIOLATION.....	82
DCGM_FI_DEV_THERMAL_VIOLATION.....	82
DCGM_FI_DEV_SYNC_BOOST_VIOLATION.....	82
DCGM_FI_DEV_BOARD_LIMIT_VIOLATION.....	82
DCGM_FI_DEV_LOW_UTIL_VIOLATION.....	82
DCGM_FI_DEV_RELIABILITY_VIOLATION.....	82
DCGM_FI_DEV_TOTAL_APP_CLOCKS_VIOLATION.....	82
DCGM_FI_DEV_TOTAL_BASE_CLOCKS_VIOLATION.....	82
DCGM_FI_DEV_FB_TOTAL.....	83
DCGM_FI_DEV_FB_FREE.....	83
DCGM_FI_DEV_FB_USED.....	83
DCGM_FI_DEV_ECC_CURRENT.....	83
DCGM_FI_DEV_ECC_PENDING.....	83
DCGM_FI_DEV_ECC_SBE_VOL_TOTAL.....	83
DCGM_FI_DEV_ECC_DBE_VOL_TOTAL.....	83
DCGM_FI_DEV_ECC_SBE_AGG_TOTAL.....	83
DCGM_FI_DEV_ECC_DBE_AGG_TOTAL.....	83
DCGM_FI_DEV_ECC_SBE_VOL_L1.....	83
DCGM_FI_DEV_ECC_DBE_VOL_L1.....	83
DCGM_FI_DEV_ECC_SBE_VOL_L2.....	84
DCGM_FI_DEV_ECC_DBE_VOL_L2.....	84
DCGM_FI_DEV_ECC_SBE_VOL_DEV.....	84
DCGM_FI_DEV_ECC_DBE_VOL_DEV.....	84
DCGM_FI_DEV_ECC_SBE_VOL_REG.....	84



DCGM_FI_DEV_ECC_DBE_VOL_REG.....	84
DCGM_FI_DEV_ECC_SBE_VOL_TEX.....	84
DCGM_FI_DEV_ECC_DBE_VOL_TEX.....	84
DCGM_FI_DEV_ECC_SBE_AGG_L1.....	84
DCGM_FI_DEV_ECC_DBE_AGG_L1.....	84
DCGM_FI_DEV_ECC_SBE_AGG_L2.....	84
DCGM_FI_DEV_ECC_DBE_AGG_L2.....	85
DCGM_FI_DEV_ECC_SBE_AGG_DEV.....	85
DCGM_FI_DEV_ECC_DBE_AGG_DEV.....	85
DCGM_FI_DEV_ECC_SBE_AGG_REG.....	85
DCGM_FI_DEV_ECC_DBE_AGG_REG.....	85
DCGM_FI_DEV_ECC_SBE_AGG_TEX.....	85
DCGM_FI_DEV_ECC_DBE_AGG_TEX.....	85
DCGM_FI_DEV_RETIRED_SBE.....	85
DCGM_FI_DEV_RETIRED_DBE.....	85
DCGM_FI_DEV_RETIRED_PENDING.....	85
DCGM_FI_DEV_UNCORRECTABLE_REMAPPED_ROWS.....	86
DCGM_FI_DEV_CORRECTABLE_REMAPPED_ROWS.....	86
DCGM_FI_DEV_ROW_REMAP_FAILURE.....	86
DCGM_FI_DEV_VIRTUAL_MODE.....	86
DCGM_FI_DEV_SUPPORTED_TYPE_INFO.....	86
DCGM_FI_DEV_CREATABLE_VGPU_TYPE_IDS.....	86
DCGM_FI_DEV_VGPU_INSTANCE_IDS.....	86
DCGM_FI_DEV_VGPU_UTILIZATIONS.....	86
DCGM_FI_DEV_VGPU_PER_PROCESS_UTILIZATION.....	86
DCGM_FI_DEV_ENC_STATS.....	86
DCGM_FI_DEV_FBC_STATS.....	87
DCGM_FI_DEV_FBC_SESSIONS_INFO.....	87
DCGM_FI_DEV_VGPU_VM_ID.....	87
DCGM_FI_DEV_VGPU_VM_NAME.....	87
DCGM_FI_DEV_VGPU_TYPE.....	87
DCGM_FI_DEV_VGPU_UUID.....	87
DCGM_FI_DEV_VGPU_DRIVER_VERSION.....	87
DCGM_FI_DEV_VGPU_MEMORY_USAGE.....	87
DCGM_FI_DEV_VGPU_LICENSE_STATUS.....	87
DCGM_FI_DEV_VGPU_FRAME_RATE_LIMIT.....	87
DCGM_FI_DEV_VGPU_ENC_STATS.....	87
DCGM_FI_DEV_VGPU_ENC_SESSIONS_INFO.....	88
DCGM_FI_DEV_VGPU_FBC_STATS.....	88
DCGM_FI_DEV_VGPU_FBC_SESSIONS_INFO.....	88
DCGM_FI_DEV_VGPU_LICENSE_INSTANCE_STATUS.....	88
DCGM_FI_FIRST_VGPU_FIELD_ID.....	88
DCGM_FI_LAST_VGPU_FIELD_ID.....	88

DCGM_FI_MAX_VGPU_FIELDS.....	88
DCGM_FI_INTERNAL_FIELDS_0_START.....	88
DCGM_FI_INTERNAL_FIELDS_0_END.....	88
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P00.....	89
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P00.....	89
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P00.....	89
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P00.....	89
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P01.....	89
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P01.....	89
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P01.....	89
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P01.....	90
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P02.....	90
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P02.....	90
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P02.....	90
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P02.....	90
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P03.....	90
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P03.....	90
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P03.....	91
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P03.....	91
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P04.....	91
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P04.....	91
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P04.....	91
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P04.....	91
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P05.....	91
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P05.....	92
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P05.....	92
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P05.....	92
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P06.....	92
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P06.....	92
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P06.....	92
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P06.....	92
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P07.....	93
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P07.....	93
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P07.....	93
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P07.....	93
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P08.....	93
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P08.....	93
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P08.....	93
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P08.....	93
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P09.....	94
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P09.....	94
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P09.....	94
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P09.....	94

DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P10.....	94
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P10.....	94
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P10.....	94
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P10.....	95
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P11.....	95
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P11.....	95
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P11.....	95
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P11.....	95
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P12.....	95
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P12.....	95
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P12.....	96
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P12.....	96
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P13.....	96
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P13.....	96
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P13.....	96
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P13.....	96
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P14.....	96
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P14.....	97
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P14.....	97
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P14.....	97
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P15.....	97
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P15.....	97
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P15.....	97
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P15.....	97
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P16.....	98
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P16.....	98
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P16.....	98
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P16.....	98
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P17.....	98
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P17.....	98
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P17.....	98
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P17.....	98
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P00.....	99
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P00.....	99
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P01.....	99
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P01.....	99
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P02.....	99
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P02.....	99
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P03.....	99
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P03.....	100
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P04.....	100
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P04.....	100
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P05.....	100

DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P05.....	100
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P06.....	100
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P06.....	100
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P07.....	100
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P07.....	101
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P08.....	101
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P08.....	101
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P09.....	101
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P09.....	101
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P10.....	101
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P10.....	101
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P11.....	101
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P11.....	102
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P12.....	102
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P12.....	102
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P13.....	102
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P13.....	102
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P14.....	102
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P14.....	102
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P15.....	102
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P15.....	103
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P16.....	103
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P16.....	103
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P17.....	103
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P17.....	103
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P00.....	103
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P00.....	103
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P01.....	104
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P01.....	104
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P02.....	104
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P02.....	104
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P03.....	104
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P03.....	104
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P04.....	104
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P04.....	104
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P05.....	105
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P05.....	105
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P06.....	105
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P06.....	105
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P07.....	105
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P07.....	105
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P08.....	105
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P08.....	105

DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P09.....	106
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P09.....	106
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P10.....	106
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P10.....	106
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P11.....	106
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P11.....	106
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P12.....	106
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P12.....	106
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P13.....	107
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P13.....	107
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P14.....	107
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P14.....	107
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P15.....	107
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P15.....	107
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P16.....	107
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P16.....	107
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P17.....	108
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P17.....	108
DCGM_FI_DEV_NVSWITCH_FATAL_ERRORS.....	108
DCGM_FI_DEV_NVSWITCH_NON_FATAL_ERRORS.....	108
DCGM_FI_FIRST_NVSWITCH_FIELD_ID.....	108
DCGM_FI_LAST_NVSWITCH_FIELD_ID.....	108
DCGM_FI_MAX_NVSWITCH_FIELDS.....	108
DCGM_FI_PROF_GR_ENGINE_ACTIVE.....	109
DCGM_FI_PROF_SM_ACTIVE.....	109
DCGM_FI_PROF_SM_OCCUPANCY.....	109
DCGM_FI_PROF_PIPE_TENSOR_ACTIVE.....	109
DCGM_FI_PROF_DRAM_ACTIVE.....	109
DCGM_FI_PROF_PIPE_FP64_ACTIVE.....	109
DCGM_FI_PROF_PIPE_FP32_ACTIVE.....	109
DCGM_FI_PROF_PIPE_FP16_ACTIVE.....	109
DCGM_FI_PROF_PCIE_TX_BYTES.....	109
DCGM_FI_PROF_PCIE_RX_BYTES.....	110
DCGM_FI_PROF_NVLINK_TX_BYTES.....	110
DCGM_FI_PROF_NVLINK_RX_BYTES.....	110
DCGM_FI_MAX_FIELDS.....	110
1.21. DCGMAPI_Admin_ExecCtrl.....	110
dcmUpdateAllFields.....	110
dcmPolicyTrigger.....	111
1.15. Modules.....	111
Init and Shutdown.....	112
Auxiliary information about DCGM engine.....	112
Discovery.....	119

Grouping.....	119
Field Grouping.....	119
Status handling.....	119
Setup and management.....	132
Manual Invocation.....	132
dcgmWatchFields.....	136
dcgmUnwatchFields.....	137
dcgmGetValuesSince.....	137
dcgmGetValuesSince_v2.....	138
dcgmGetLatestValues.....	139
dcgmGetLatestValues_v2.....	140
dcgmGetLatestValuesForFields.....	141
dcgmEntityGetLatestValues.....	141
dcgmEntitiesGetLatestValues.....	142
dcgmGetFieldSummary.....	143
dcgmWatchPidFields.....	144
dcgmGetPidInfo.....	144
dcgmWatchJobFields.....	145
dcgmJobStartStats.....	146
dcgmJobStopStats.....	147
dcgmJobGetStats.....	147
dcgmJobRemove.....	148
dcgmJobRemoveAll.....	148
dcgmHealthSet.....	149
dcgmHealthSet_v2.....	149
dcgmHealthGet.....	150
dcgmHealthCheck.....	150
Setup and Management.....	151
Manual Invocation.....	151
dcgmGetDeviceTopology.....	157
dcgmGetGroupTopology.....	158
dcgmIntrospectToggleState.....	158
dcgmIntrospectGetFieldsMemoryUsage.....	159
dcgmIntrospectGetHostengineMemoryUsage.....	160
dcgmIntrospectGetFieldsExecTime.....	160
dcgmIntrospectGetHostengineCpuUtilization.....	161
dcgmIntrospectUpdateAll.....	162
dcgmSelectGpusByTopology.....	162
dcgmModuleBlacklist.....	163
dcgmModuleGetStatuses.....	164
dcgmProfGetSupportedMetricGroups.....	164
dcgmProfWatchFields.....	165
dcgmProfUnwatchFields.....	166

dcgmProfPause.....	166
dcgmProfResume.....	167
dcgmOperationMode_t.....	168
dcgmOrder_t.....	168
dcgmReturn_t.....	168
dcgmGroupType_t.....	171
dcgmChipArchitecture_t.....	171
dcgmConfigType_t.....	172
dcgmConfigPowerLimitType_t.....	172
MAKE_DCGM_VERSION.....	172
DCGM_INT32_BLANK.....	172
DCGM_INT64_BLANK.....	172
DCGM_FP64_BLANK.....	173
DCGM_STR_BLANK.....	173
DCGM_INT32_NOT_FOUND.....	173
DCGM_INT64_NOT_FOUND.....	173
DCGM_FP64_NOT_FOUND.....	173
DCGM_STR_NOT_FOUND.....	173
DCGM_INT32_NOT_SUPPORTED.....	173
DCGM_INT64_NOT_SUPPORTED.....	173
DCGM_FP64_NOT_SUPPORTED.....	173
DCGM_STR_NOT_SUPPORTED.....	173
DCGM_INT32_NOT_PERMISSIONED.....	173
DCGM_INT64_NOT_PERMISSIONED.....	174
DCGM_FP64_NOT_PERMISSIONED.....	174
DCGM_STR_NOT_PERMISSIONED.....	174
DCGM_INT32_IS_BLANK.....	174
DCGM_INT64_IS_BLANK.....	174
DCGM_FP64_IS_BLANK.....	174
DCGM_STR_IS_BLANK.....	174
DCGM_MAX_NUM_DEVICES.....	174
DCGM_NVLINK_MAX_LINKS_PER_GPU.....	174
DCGM_NVLINK_MAX_LINKS_PER_GPU_LEGACY1.....	174
DCGM_MAX_NUM_SWITCHES.....	175
DCGM_NVLINK_MAX_LINKS_PER_NVSWITCH.....	175
DCGM_MAX_VGPU_INSTANCES_PER_PGPU.....	175
DCGM_MAX_STR_LENGTH.....	175
DCGM_MAX_CLOCKS.....	175
DCGM_MAX_NUM_GROUPS.....	175
DCGM_MAX_FBC_SESSIONS.....	175
DCGM_VGPU_NAME_BUFFER_SIZE.....	175
DCGM_GRID_LICENSE_BUFFER_SIZE.....	175
DCGM_CONFIG_COMPUTEMODE_DEFAULT.....	175

DCGM_CONFIG_COMPUTEMODE_PROHIBITED.....	175
DCGM_CONFIG_COMPUTEMODE_EXCLUSIVE_PROCESS.....	176
DCGM_HE_PORT_NUMBER.....	176
DCGM_GROUP_ALL_GPUS.....	176
DCGM_GROUP_MAX_ENTITIES.....	176
DCGM_FT_BINARY.....	176
DCGM_FT_DOUBLE.....	176
DCGM_FT_INT64.....	176
DCGM_FT_STRING.....	176
DCGM_FT_TIMESTAMP.....	176
DCGM_FS_GLOBAL.....	176
DCGM_FS_ENTITY.....	177
DCGM_FS_DEVICE.....	177
dcmGpuVirtualizationMode_t.....	177
DCGM_CUDA_COMPUTE_CAPABILITY_MAJOR.....	177
DCGM_CLOCKS_THROTTLE_REASON_GPU_IDLE.....	177
DCGM_CLOCKS_THROTTLE_REASON_CLOCKS_SETTING.....	178
DCGM_CLOCKS_THROTTLE_REASON_SW_POWER_CAP.....	178
DCGM_CLOCKS_THROTTLE_REASON_HW_SLOWDOWN.....	178
DCGM_CLOCKS_THROTTLE_REASON_SYNC_BOOST.....	178
DCGM_CLOCKS_THROTTLE_REASON_SW_THERMAL.....	178
DCGM_CLOCKS_THROTTLE_REASON_HW_THERMAL.....	178
DCGM_CLOCKS_THROTTLE_REASON_HW_POWER_BRAKE.....	179
DCGM_CLOCKS_THROTTLE_REASON_DISPLAY_CLOCKS.....	179
dcm_field_entity_group_t.....	179
dcm_field_eid_t.....	180
DcmFieldGetById.....	180
DcmFieldGetByTag.....	180
DcmFieldsInit.....	180
DcmFieldsTerm.....	181
DcmFieldsGetEntityGroupString.....	181
DCGM_FI_UNKNOWN.....	181
DCGM_FI_DRIVER_VERSION.....	181
DCGM_FI_DEV_COUNT.....	181
DCGM_FI_CUDA_DRIVER_VERSION.....	181
DCGM_FI_DEV_NAME.....	181
DCGM_FI_DEV_BRAND.....	182
DCGM_FI_DEV_NVML_INDEX.....	182
DCGM_FI_DEV_SERIAL.....	182
DCGM_FI_DEV_UUID.....	182
DCGM_FI_DEV_MINOR_NUMBER.....	182
DCGM_FI_DEV_OEM_INFOROM_VER.....	182
DCGM_FI_DEV_PCI_BUSID.....	182



DCGM_FI_DEV_PCI_COMBINED_ID.....	182
DCGM_FI_DEV_PCI_SUBSYS_ID.....	182
DCGM_FI_GPU_TOPOLOGY_PCI.....	182
DCGM_FI_GPU_TOPOLOGY_NVLINK.....	182
DCGM_FI_GPU_TOPOLOGY_AFFINITY.....	182
DCGM_FI_DEV_CUDA_COMPUTE_CAPABILITY.....	183
DCGM_FI_DEV_COMPUTE_MODE.....	183
DCGM_FI_DEV_PERSISTENCE_MODE.....	183
DCGM_FI_DEV_MIG_MODE.....	183
DCGM_FI_DEV_CUDA_VISIBLE_DEVICES_STR.....	183
DCGM_FI_DEV_MIG_MAX_SLICES.....	183
DCGM_FI_DEV_CPU_AFFINITY_0.....	183
DCGM_FI_DEV_CPU_AFFINITY_1.....	183
DCGM_FI_DEV_CPU_AFFINITY_2.....	183
DCGM_FI_DEV_CPU_AFFINITY_3.....	183
DCGM_FI_DEV_ECC_INFOROM_VER.....	183
DCGM_FI_DEV_POWER_INFOROM_VER.....	184
DCGM_FI_DEV_INFOROM_IMAGE_VER.....	184
DCGM_FI_DEV_INFOROM_CONFIG_CHECK.....	184
DCGM_FI_DEV_INFOROM_CONFIG_VALID.....	184
DCGM_FI_DEV_VBIOS_VERSION.....	184
DCGM_FI_DEV_BAR1_TOTAL.....	184
DCGM_FI_SYNC_BOOST.....	184
DCGM_FI_DEV_BAR1_USED.....	184
DCGM_FI_DEV_BAR1_FREE.....	184
DCGM_FI_DEV_SM_CLOCK.....	184
DCGM_FI_DEV_MEM_CLOCK.....	184
DCGM_FI_DEV_VIDEO_CLOCK.....	184
DCGM_FI_DEV_APP_SM_CLOCK.....	185
DCGM_FI_DEV_APP_MEM_CLOCK.....	185
DCGM_FI_DEV_CLOCK_THROTTLE_REASONS.....	185
DCGM_FI_DEV_MAX_SM_CLOCK.....	185
DCGM_FI_DEV_MAX_MEM_CLOCK.....	185
DCGM_FI_DEV_MAX_VIDEO_CLOCK.....	185
DCGM_FI_DEV_AUTOBOOST.....	185
DCGM_FI_DEV_SUPPORTED_CLOCKS.....	185
DCGM_FI_DEV_MEMORY_TEMP.....	185
DCGM_FI_DEV_GPU_TEMP.....	185
DCGM_FI_DEV_MEM_MAX_OP_TEMP.....	185
DCGM_FI_DEV_GPU_MAX_OP_TEMP.....	185
DCGM_FI_DEV_POWER_USAGE.....	186
DCGM_FI_DEV_TOTAL_ENERGY_CONSUMPTION.....	186
DCGM_FI_DEV_SLOWDOWN_TEMP.....	186

DCGM_FI_DEV_SHUTDOWN_TEMP.....	186
DCGM_FI_DEV_POWER_MGMT_LIMIT.....	186
DCGM_FI_DEV_POWER_MGMT_LIMIT_MIN.....	186
DCGM_FI_DEV_POWER_MGMT_LIMIT_MAX.....	186
DCGM_FI_DEV_POWER_MGMT_LIMIT_DEF.....	186
DCGM_FI_DEV_ENFORCED_POWER_LIMIT.....	186
DCGM_FI_DEV_PSTATE.....	186
DCGM_FI_DEV_FAN_SPEED.....	186
DCGM_FI_DEV_PCIE_TX_THROUGHPUT.....	186
DCGM_FI_DEV_PCIE_RX_THROUGHPUT.....	187
DCGM_FI_DEV_PCIE_REPLAY_COUNTER.....	187
DCGM_FI_DEV_GPU_UTIL.....	187
DCGM_FI_DEV_MEM_COPY_UTIL.....	187
DCGM_FI_DEV_ACCOUNTING_DATA.....	187
DCGM_FI_DEV_ENC_UTIL.....	187
DCGM_FI_DEV_DEC_UTIL.....	187
DCGM_FI_DEV_MEM_COPY_UTIL_SAMPLES.....	187
DCGM_FI_DEV_GRAPHICS_PIDS.....	187
DCGM_FI_DEV_COMPUTE_PIDS.....	187
DCGM_FI_DEV_XID_ERRORS.....	187
DCGM_FI_DEV_PCIE_MAX_LINK_GEN.....	188
DCGM_FI_DEV_PCIE_MAX_LINK_WIDTH.....	188
DCGM_FI_DEV_PCIE_LINK_GEN.....	188
DCGM_FI_DEV_PCIE_LINK_WIDTH.....	188
DCGM_FI_DEV_POWER_VIOLATION.....	188
DCGM_FI_DEV_THERMAL_VIOLATION.....	188
DCGM_FI_DEV_SYNC_BOOST_VIOLATION.....	188
DCGM_FI_DEV_BOARD_LIMIT_VIOLATION.....	188
DCGM_FI_DEV_LOW_UTIL_VIOLATION.....	188
DCGM_FI_DEV_RELIABILITY_VIOLATION.....	188
DCGM_FI_DEV_TOTAL_APP_CLOCKS_VIOLATION.....	188
DCGM_FI_DEV_TOTAL_BASE_CLOCKS_VIOLATION.....	188
DCGM_FI_DEV_FB_TOTAL.....	189
DCGM_FI_DEV_FB_FREE.....	189
DCGM_FI_DEV_FB_USED.....	189
DCGM_FI_DEV_ECC_CURRENT.....	189
DCGM_FI_DEV_ECC_PENDING.....	189
DCGM_FI_DEV_ECC_SBE_VOL_TOTAL.....	189
DCGM_FI_DEV_ECC_DBE_VOL_TOTAL.....	189
DCGM_FI_DEV_ECC_SBE_AGG_TOTAL.....	189
DCGM_FI_DEV_ECC_DBE_AGG_TOTAL.....	189
DCGM_FI_DEV_ECC_SBE_VOL_L1.....	189
DCGM_FI_DEV_ECC_DBE_VOL_L1.....	189

DCGM_FI_DEV_ECC_SBE_VOL_L2.....	189
DCGM_FI_DEV_ECC_DBE_VOL_L2.....	190
DCGM_FI_DEV_ECC_SBE_VOL_DEV.....	190
DCGM_FI_DEV_ECC_DBE_VOL_DEV.....	190
DCGM_FI_DEV_ECC_SBE_VOL_REG.....	190
DCGM_FI_DEV_ECC_DBE_VOL_REG.....	190
DCGM_FI_DEV_ECC_SBE_VOL_TEX.....	190
DCGM_FI_DEV_ECC_DBE_VOL_TEX.....	190
DCGM_FI_DEV_ECC_SBE_AGG_L1.....	190
DCGM_FI_DEV_ECC_DBE_AGG_L1.....	190
DCGM_FI_DEV_ECC_SBE_AGG_L2.....	190
DCGM_FI_DEV_ECC_DBE_AGG_L2.....	190
DCGM_FI_DEV_ECC_SBE_AGG_DEV.....	190
DCGM_FI_DEV_ECC_DBE_AGG_DEV.....	191
DCGM_FI_DEV_ECC_SBE_AGG_REG.....	191
DCGM_FI_DEV_ECC_DBE_AGG_REG.....	191
DCGM_FI_DEV_ECC_SBE_AGG_TEX.....	191
DCGM_FI_DEV_ECC_DBE_AGG_TEX.....	191
DCGM_FI_DEV_RETIRED_SBE.....	191
DCGM_FI_DEV_RETIRED_DBE.....	191
DCGM_FI_DEV_RETIRED_PENDING.....	191
DCGM_FI_DEV_UNCORRECTABLE_REMAPPED_ROWS.....	191
DCGM_FI_DEV_CORRECTABLE_REMAPPED_ROWS.....	191
DCGM_FI_DEV_ROW_REMAP_FAILURE.....	191
DCGM_FI_DEV_VIRTUAL_MODE.....	192
DCGM_FI_DEV_SUPPORTED_TYPE_INFO.....	192
DCGM_FI_DEV_CREATABLE_VGPU_TYPE_IDS.....	192
DCGM_FI_DEV_VGPU_INSTANCE_IDS.....	192
DCGM_FI_DEV_VGPU_UTILIZATIONS.....	192
DCGM_FI_DEV_VGPU_PER_PROCESS_UTILIZATION.....	192
DCGM_FI_DEV_ENC_STATS.....	192
DCGM_FI_DEV_FBC_STATS.....	192
DCGM_FI_DEV_FBC_SESSIONS_INFO.....	192
DCGM_FI_DEV_VGPU_VM_ID.....	192
DCGM_FI_DEV_VGPU_VM_NAME.....	192
DCGM_FI_DEV_VGPU_TYPE.....	192
DCGM_FI_DEV_VGPU_UUID.....	193
DCGM_FI_DEV_VGPU_DRIVER_VERSION.....	193
DCGM_FI_DEV_VGPU_MEMORY_USAGE.....	193
DCGM_FI_DEV_VGPU_LICENSE_STATUS.....	193
DCGM_FI_DEV_VGPU_FRAME_RATE_LIMIT.....	193
DCGM_FI_DEV_VGPU_ENC_STATS.....	193
DCGM_FI_DEV_VGPU_ENC_SESSIONS_INFO.....	193

DCGM_FI_DEV_VGPU_FBC_STATS.....	193
DCGM_FI_DEV_VGPU_FBC_SESSIONS_INFO.....	193
DCGM_FI_DEV_VGPU_LICENSE_INSTANCE_STATUS.....	193
DCGM_FI_FIRST_VGPU_FIELD_ID.....	193
DCGM_FI_LAST_VGPU_FIELD_ID.....	193
DCGM_FI_MAX_VGPU_FIELDS.....	194
DCGM_FI_INTERNAL_FIELDS_0_START.....	194
DCGM_FI_INTERNAL_FIELDS_0_END.....	194
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P00.....	194
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P00.....	194
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P00.....	194
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P00.....	194
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P01.....	194
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P01.....	194
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P01.....	195
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P01.....	195
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P02.....	195
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P02.....	195
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P02.....	195
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P02.....	195
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P03.....	195
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P03.....	195
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P03.....	195
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P03.....	195
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P04.....	196
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P04.....	196
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P04.....	196
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P04.....	196
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P05.....	196
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P05.....	196
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P05.....	196
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P05.....	196
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P06.....	196
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P06.....	196
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P06.....	196
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P06.....	197
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P07.....	197
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P07.....	197
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P07.....	197
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P07.....	197
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P08.....	197
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P08.....	197
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P08.....	197

DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P08.....	197
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P09.....	197
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P09.....	198
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P09.....	198
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P09.....	198
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P10.....	198
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P10.....	198
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P10.....	198
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P10.....	198
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P11.....	198
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P11.....	198
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P11.....	198
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P11.....	198
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P12.....	199
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P12.....	199
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P12.....	199
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P12.....	199
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P13.....	199
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P13.....	199
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P13.....	199
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P13.....	199
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P14.....	199
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P14.....	199
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P14.....	200
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P14.....	200
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P15.....	200
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P15.....	200
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P15.....	200
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P15.....	200
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P16.....	200
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P16.....	200
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P16.....	200
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P16.....	200
DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P17.....	201
DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P17.....	201
DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P17.....	201
DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P17.....	201
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P00.....	201
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P00.....	201
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P01.....	201
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P01.....	201
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P02.....	201
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P02.....	201

DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P03.....	201
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P03.....	202
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P04.....	202
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P04.....	202
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P05.....	202
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P05.....	202
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P06.....	202
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P06.....	202
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P07.....	202
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P07.....	202
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P08.....	202
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P08.....	202
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P09.....	202
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P09.....	203
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P10.....	203
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P10.....	203
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P11.....	203
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P11.....	203
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P12.....	203
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P12.....	203
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P13.....	203
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P13.....	203
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P14.....	203
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P14.....	203
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P15.....	203
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P15.....	204
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P16.....	204
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P16.....	204
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P17.....	204
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P17.....	204
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P00.....	204
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P00.....	204
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P01.....	204
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P01.....	204
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P02.....	204
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P02.....	204
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P03.....	205
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P03.....	205
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P04.....	205
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P04.....	205
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P05.....	205
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P05.....	205
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P06.....	205

DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P06.....	205
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P07.....	205
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P07.....	205
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P08.....	205
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P08.....	205
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P09.....	206
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P09.....	206
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P10.....	206
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P10.....	206
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P11.....	206
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P11.....	206
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P12.....	206
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P12.....	206
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P13.....	206
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P13.....	206
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P14.....	206
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P14.....	206
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P15.....	207
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P15.....	207
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P16.....	207
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P16.....	207
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P17.....	207
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P17.....	207
DCGM_FI_DEV_NVSWITCH_FATAL_ERRORS.....	207
DCGM_FI_DEV_NVSWITCH_NON_FATAL_ERRORS.....	207
DCGM_FI_FIRST_NVSWITCH_FIELD_ID.....	207
DCGM_FI_LAST_NVSWITCH_FIELD_ID.....	207
DCGM_FI_MAX_NVSWITCH_FIELDS.....	208
DCGM_FI_PROF_GR_ENGINE_ACTIVE.....	208
DCGM_FI_PROF_SM_ACTIVE.....	208
DCGM_FI_PROF_SM_OCCUPANCY.....	208
DCGM_FI_PROF_PIPE_TENSOR_ACTIVE.....	208
DCGM_FI_PROF_DRAM_ACTIVE.....	208
DCGM_FI_PROF_PIPE_FP64_ACTIVE.....	208
DCGM_FI_PROF_PIPE_FP32_ACTIVE.....	208
DCGM_FI_PROF_PIPE_FP16_ACTIVE.....	208
DCGM_FI_PROF_PCIE_TX_BYTES.....	209
DCGM_FI_PROF_PCIE_RX_BYTES.....	209
DCGM_FI_PROF_NVLINK_TX_BYTES.....	209
DCGM_FI_PROF_NVLINK_RX_BYTES.....	209
DCGM_FI_MAX_FIELDS.....	209
dcgmUpdateAllFields.....	209
dcgmPolicyTrigger.....	210

<b>Chapter 2. Data Structures.....</b>	<b>211</b>
dcgM_field_meta_t.....	213
fieldId.....	213
fieldType.....	213
size.....	213
tag.....	213
scope.....	213
nvmlFieldId.....	213
entityLevel.....	213
valueFormat.....	213
dcgm_field_output_format_t.....	214
shortName.....	214
unit.....	214
width.....	214
dcgmClockSet_v1.....	214
version.....	214
memClock.....	214
smClock.....	214
dcgmConfig_v1.....	214
version.....	215
gpuld.....	215
eccMode.....	215
computeMode.....	215
perfState.....	215
powerLimit.....	215
dcgmConfigPerfStateSettings_t.....	215
syncBoost.....	215
targetClocks.....	215
dcgmConfigPowerLimit_t.....	216
type.....	216
val.....	216
dcgmConnectV2Params_v1.....	216
version.....	216
persistAfterDisconnect.....	216
dcgmConnectV2Params_v2.....	216
version.....	216
persistAfterDisconnect.....	217
timeoutMs.....	217
addressIsUnixSocket.....	217
dcgmDeviceAttributes_v1.....	217
version.....	218
clockSets.....	218
thermalSettings.....	218



powerLimits.....	218
identifiers.....	218
memoryUsage.....	218
unused.....	218
dcgmDeviceEncStats_v1.....	218
version.....	219
sessionCount.....	219
averageFps.....	219
averageLatency.....	219
dcgmDeviceFbcSessionInfo_v1.....	219
version.....	220
sessionId.....	220
pid.....	220
vgpuld.....	220
displayOrdinal.....	220
sessionType.....	220
sessionFlags.....	220
hMaxResolution.....	220
vMaxResolution.....	220
hResolution.....	220
vResolution.....	220
averageFps.....	221
averageLatency.....	221
dcgmDeviceFbcSessions_v1.....	221
version.....	221
sessionCount.....	221
sessionInfo.....	221
dcgmDeviceFbcStats_v1.....	221
version.....	222
sessionCount.....	222
averageFps.....	222
averageLatency.....	222
dcgmDeviceIdentifiers_v1.....	222
version.....	223
brandName.....	223
deviceName.....	223
pciBusId.....	223
serial.....	223
uuid.....	223
vbios.....	223
inforomImageVersion.....	223
pciDeviceId.....	223
pciSubSystemId.....	223

driverVersion.....	223
virtualizationMode.....	223
dcgmDeviceMemoryUsage_v1.....	224
version.....	224
bar1Total.....	224
fbTotal.....	224
fbUsed.....	224
fbFree.....	224
dcgmDevicePidAccountingStats_v1.....	224
version.....	224
pid.....	224
gpuUtilization.....	224
memoryUtilization.....	225
maxMemoryUsage.....	225
startTimestamp.....	225
activeTimeUsec.....	225
dcgmDevicePowerLimits_v1.....	225
version.....	226
curPowerLimit.....	226
defaultPowerLimit.....	226
enforcedPowerLimit.....	226
minPowerLimit.....	226
maxPowerLimit.....	226
dcgmDeviceSupportedClockSets_v1.....	226
version.....	227
count.....	227
clockSet.....	227
dcgmDeviceThermals_v1.....	227
version.....	227
slowdownTemp.....	227
shutdownTemp.....	227
dcgmDeviceTopology_v1.....	227
version.....	227
cpuAffinityMask.....	227
numGpus.....	228
gpuld.....	228
path.....	228
localNvLinkIds.....	228
dcgmDeviceVgpuEncSessions_v1.....	228
version.....	229
vgpuld.....	229
sessionId.....	229
pid.....	229

codecType.....	229
hResolution.....	229
vResolution.....	229
averageFps.....	229
averageLatency.....	229
dcgmDeviceVgpuProcessUtilInfo_v1.....	229
version.....	230
vgpuld.....	230
vgpuProcessSamplesCount.....	230
pid.....	230
processName.....	230
smUtil.....	230
memUtil.....	230
encUtil.....	230
decUtil.....	230
dcgmDeviceVgpuTypeInfo_v1.....	230
version.....	231
vgpuTypeInfo.....	231
vgpuTypeName.....	231
vgpuTypeClass.....	231
vgpuTypeLicense.....	231
deviceId.....	231
subsystemId.....	231
numDisplayHeads.....	231
maxInstances.....	231
frameRateLimit.....	231
maxResolutionX.....	231
maxResolutionY.....	231
fbTotal.....	231
dcgmDeviceVgpuUtilInfo_v1.....	232
version.....	232
vgpuld.....	232
smUtil.....	232
memUtil.....	232
encUtil.....	232
decUtil.....	232
dcgmDiagResponse_v6.....	232
version.....	233
gpuCount.....	233
levelOneTestCount.....	233
levelOneResults.....	233
perGpuResponses.....	233
systemError.....	233

trainingMsg.....	233
dcgmDiagResponsePerGpu_v2.....	233
gpuld.....	234
hwDiagnosticReturn.....	234
results.....	234
dcgmErrorInfo_t.....	234
gpuld.....	234
fieldId.....	234
status.....	234
dcgmFieldGroupInfo_v1.....	234
version.....	235
numFieldIds.....	235
fieldGroupId.....	235
fieldName.....	235
fieldIds.....	235
dcgmFieldValue_v1.....	235
version.....	236
fieldId.....	236
fieldType.....	236
status.....	236
ts.....	236
i64.....	236
dbl.....	236
str.....	236
blob.....	236
value.....	236
dcgmFieldValue_v2.....	236
version.....	237
entityGroupId.....	237
entityId.....	237
fieldId.....	237
fieldType.....	237
status.....	237
unused.....	237
ts.....	237
i64.....	237
dbl.....	237
str.....	237
blob.....	237
value.....	237
dcgmGpuUsageInfo_t.....	238
gpuld.....	239
energyConsumed.....	239

powerUsage.....	239
pcieRxBandwidth.....	239
pcieTxBandwidth.....	239
pcieReplays.....	239
startTime.....	239
endTime.....	239
smUtilization.....	239
memoryUtilization.....	239
eccSingleBit.....	239
eccDoubleBit.....	240
memoryClock.....	240
smClock.....	240
numXidCriticalErrors.....	240
xidCriticalErrorsTs.....	240
numComputePids.....	240
computePidInfo.....	240
numGraphicsPids.....	240
graphicsPidInfo.....	240
maxGpuMemoryUsed.....	241
powerViolationTime.....	241
thermalViolationTime.....	241
reliabilityViolationTime.....	241
boardLimitViolationTime.....	241
lowUtilizationTime.....	242
syncBoostTime.....	242
overallHealth.....	242
system.....	242
health.....	242
dcmGroupEntityPair_t.....	242
entityGroupld.....	242
entityld.....	242
dcmGroupInfo_v2.....	242
version.....	243
count.....	243
groupName.....	243
entityList.....	243
dcmGroupTopology_v1.....	243
version.....	243
groupCpuAffinityMask.....	243
numaOptimalFlag.....	243
slowestPath.....	244
dcmHealthResponse_v4.....	244
version.....	244

overallHealth.....	244
incidentCount.....	244
incidents.....	244
dcgmHealthSetParams_v2.....	244
version.....	244
groupId.....	244
systems.....	245
updateInterval.....	245
maxKeepAge.....	245
dcgmHostengineHealth_v1.....	245
version.....	245
overallHealth.....	245
dcgmIntrospectContext_v1.....	245
version.....	246
introspectLvl.....	246
fieldGroupId.....	246
fieldId.....	246
contextId.....	246
dcgmIntrospectCpuUtil_v1.....	246
version.....	246
total.....	246
kernel.....	246
user.....	246
dcgmIntrospectFieldsExecTime_v1.....	246
version.....	247
meanUpdateFreqUsec.....	247
recentUpdateUsec.....	247
totalEverUpdateUsec.....	247
dcgmIntrospectFullFieldsExecTime_v2.....	247
version.....	248
aggregateInfo.....	248
hasGlobalInfo.....	248
globalInfo.....	248
gpuInfoCount.....	248
gpusForGpuInfo.....	248
gpuInfo.....	248
dcgmIntrospectFullMemory_v1.....	248
version.....	249
aggregateInfo.....	249
hasGlobalInfo.....	249
globalInfo.....	249
gpuInfoCount.....	249
gpusForGpuInfo.....	249

gpulInfo.....	249
dcgmIntrospectMemory_v1.....	249
version.....	250
bytesUsed.....	250
dcgmJobInfo_v3.....	250
version.....	250
numGpus.....	250
summary.....	250
gpus.....	250
dcgmMigEntityInfo_t.....	250
gpuUuid.....	250
nvmiGpuIndex.....	251
nvmiInstancelId.....	251
nvmiComputeInstancelId.....	251
nvmiMigProfileId.....	251
nvmiProfileSlices.....	251
dcgmMigHierarchy_v1.....	251
dcgmMigHierarchyInfo_t.....	251
entity.....	252
parent.....	252
sliceProfile.....	252
dcgmModuleGetStatusesModule_t.....	252
id.....	252
status.....	252
dcgmNvLinkGpuLinkStatus_v1.....	252
entityId.....	253
linkState.....	253
dcgmNvLinkNvSwitchLinkStatus_t.....	253
entityId.....	253
linkState.....	253
dcgmNvLinkStatus_v1.....	253
version.....	254
numGpus.....	254
gpus.....	254
numNvSwitches.....	254
nvSwitches.....	254
dcgmPidInfo_v2.....	254
version.....	255
pid.....	255
numGpus.....	255
summary.....	255
gpus.....	255
dcgmPidSingleInfo_t.....	255

gpuld.....	256
energyConsumed.....	256
pcieRxBandwidth.....	256
pcieTxBandwidth.....	256
pcieReplays.....	256
startTime.....	256
endTime.....	256
processUtilization.....	256
smUtilization.....	256
memoryUtilization.....	256
eccSingleBit.....	257
eccDoubleBit.....	257
memoryClock.....	257
smClock.....	257
numXidCriticalErrors.....	257
xidCriticalErrorsTs.....	257
numOtherComputePids.....	257
otherComputePids.....	257
numOtherGraphicsPids.....	257
otherGraphicsPids.....	257
maxGpuMemoryUsed.....	257
powerViolationTime.....	257
thermalViolationTime.....	258
reliabilityViolationTime.....	258
boardLimitViolationTime.....	258
lowUtilizationTime.....	258
syncBoostTime.....	258
overallHealth.....	258
system.....	258
health.....	258
dcmPolicy_v1.....	258
version.....	259
condition.....	259
mode.....	259
isolation.....	259
action.....	259
validation.....	259
response.....	259
parms.....	259
dcmPolicyCallbackResponse_v1.....	259
version.....	260
condition.....	260
dbe.....	260



pci.....	260
mpr.....	260
thermal.....	260
power.....	260
nvlink.....	260
xid.....	260
dcgmPolicyConditionDbe_t.....	260
timestamp.....	261
location.....	261
numerrors.....	261
dcgmPolicyConditionMpr_t.....	261
timestamp.....	261
sbepages.....	261
dbepages.....	261
dcgmPolicyConditionNvlink_t.....	261
timestamp.....	262
fieldId.....	262
counter.....	262
dcgmPolicyConditionParams_t.....	262
dcgmPolicyConditionPci_t.....	262
timestamp.....	262
counter.....	262
dcgmPolicyConditionPower_t.....	262
timestamp.....	263
powerViolation.....	263
dcgmPolicyConditionThermal_t.....	263
timestamp.....	263
thermalViolation.....	263
dcgmPolicyConditionXID_t.....	263
timestamp.....	263
errnum.....	263
dcgmPolicyViolationNotify_t.....	263
gpuld.....	264
violationOccurred.....	264
dcgmProcessUtilInfo_t.....	264
dcgmProcessUtilSample_t.....	264
dcgmProfMetricGroupInfo_t.....	264
majorId.....	264
minorId.....	264
numFieldIds.....	264
fieldIds.....	264
dcgmProfUnwatchFields_v1.....	265
version.....	265

groupId.....	265
flags.....	265
dcgmProfWatchFields_v1.....	265
version.....	265
groupId.....	265
numFieldIds.....	266
fieldIds.....	266
updateFreq.....	266
maxKeepAge.....	266
maxKeepSamples.....	266
flags.....	266
dcgmRunningProcess_v1.....	266
version.....	267
pid.....	267
memoryUsed.....	267
dcgmSettingsSetLoggingSeverity_v1.....	267
dcgmStartEmbeddedV2Params_v1.....	267
version.....	267
opMode.....	267
dcgmHandle.....	267
logFile.....	267
severity.....	268
blackListCount.....	268
unused.....	268
dcgmStatSummaryFp64_t.....	268
minValue.....	268
maxValue.....	268
average.....	268
dcgmStatSummaryInt32_t.....	268
minValue.....	269
maxValue.....	269
average.....	269
dcgmStatSummaryInt64_t.....	269
minValue.....	269
maxValue.....	269
average.....	269
dcgmVersionInfo_v2.....	269
rawBuildInfoString.....	269
<b>Chapter 3. Data Fields.....</b>	<b>271</b>

# Chapter 1.

## MODULES

Here is a list of all modules:

- ▶ Administrative
  - ▶ Init and Shutdown
  - ▶ Auxilary information about DCGM engine.
- ▶ System
  - ▶ Discovery
  - ▶ Grouping
  - ▶ Field Grouping
  - ▶ Status handling
- ▶ Configuration
  - ▶ Setup and management
  - ▶ Manual Invocation
- ▶ Field APIs
- ▶ Process Statistics
- ▶ Job Statistics
- ▶ Health Monitor
- ▶ Policies
  - ▶ Setup and Management
  - ▶ Manual Invocation
- ▶ Topology
- ▶ Metadata
- ▶ Topology
- ▶ Modules
- ▶ Profiling
- ▶ Enums and Macros
- ▶ Structure definitions

- ▶ Field Types
- ▶ Field Scope
- ▶ Field Constants
- ▶ Field Entity
- ▶ Field Identifiers
- ▶ DCGMAPI\_Admin\_ExecCtrl

## 1.1. Administrative

This chapter describes the administration interfaces for DCGM. It is the user's responsibility to call `dcgmInit()` before calling any other methods, and `dcgmShutdown()` once DCGM is no longer being used. The APIs in Administrative module can be broken down into following categories:

### Init and Shutdown

### Auxiliary information about DCGM engine.

#### 1.1.1. Init and Shutdown

##### Administrative

Describes APIs to Initialize and Shutdown the DCGM Engine.

##### `dcgmReturn_t dcgmInit (void)`

##### Returns

- ▶ `DCGM_ST_OK` if DCGM has been properly initialized
- ▶ `DCGM_ST_INIT_ERROR` if there was an error initializing the library

##### Description

This method is used to initialize DCGM within this process. This must be called before `dcgmStartEmbedded()` or `dcgmConnect()`

\*

##### `dcgmReturn_t dcgmShutdown (void)`

##### Returns

- ▶ `DCGM_ST_OK` if DCGM has been properly shut down
- ▶ `DCGM_ST_UNINITIALIZED` if the library was not shut down properly

**Description**

This method is used to shut down DCGM. Any embedded host engines or remote connections will automatically be shut down as well.

`dcgmReturn_t dcgmStartEmbedded (dcgmOperationMode_t opMode, dcgmHandle_t *pDcgmHandle)`

**Parameters****opMode**

IN: Collect data automatically or manually when asked by the user.

**pDcgmHandle**

OUT: DCGM Handle to use for API calls

**Returns**

- ▶ DCGM\_ST\_OK if DCGM was started successfully within our process
- ▶ DCGM\_ST\_UNINITIALIZED if DCGM has not been initialized with `dcgmInit` yet

**Description**

Start an embedded host engine agent within this process.

The agent is loaded as a shared library. This mode is provided to avoid any extra jitter associated with an additional autonomous agent needs to be managed. In this mode, the user has to periodically call APIs such as `dcgmPolicyTrigger` and `dcgmUpdateAllFields` which tells DCGM to wake up and perform data collection and operations needed for policy management.

`dcgmReturn_t dcgmStartEmbedded_v2 (dcgmStartEmbeddedV2Params_v1 *params[])`

**Parameters****params**

IN/OUT: See `dcgmStartEmbeddedV2Params_v1` for details.

**Returns**

- ▶ DCGM\_ST\_OK if DCGM was started successfully within our process
- ▶ DCGM\_ST\_UNINITIALIZED if DCGM has not been initialized with `dcgmInit` yet

**Description**

Start an embedded host engine agent within this process.

The agent is loaded as a shared library. This mode is provided to avoid any extra jitter associated with an additional autonomous agent needs to be managed. In this mode, the user has to periodically call APIs such as `dcgmPolicyTrigger` and `dcgmUpdateAllFields` which tells DCGM to wake up and perform data collection and operations needed for policy management.

## `dcgmReturn_t dcgmStopEmbedded (dcgmHandle_t pDcgmHandle)`

### Parameters

#### `pDcgmHandle`

IN : DCGM Handle of the embedded host engine that came from `dcgmStartEmbedded`

### Returns

- ▶ `DCGM_ST_OK` if DCGM was stopped successfully within our process
- ▶ `DCGM_ST_UNINITIALIZED` if DCGM has not been initialized with `dcgmInit` or the embedded host engine was not running.
- ▶ `DCGM_ST_BADPARAM` if an invalid parameter was provided
- ▶ `DCGM_ST_INIT_ERROR` if an error occurred while trying to start the host engine.

### Description

Stop the embedded host engine within this process that was started with `dcgmStartEmbedded`

## `dcgmReturn_t dcgmConnect (char *ipAddress, dcgmHandle_t *pDcgmHandle)`

### Parameters

#### `ipAddress`

IN: Valid IP address for the remote host engine to connect to. If `ipAddress` is specified as `x.x.x.x` it will attempt to connect to the default port specified by `DCGM_HE_PORT_NUMBER`. If `ipAddress` is specified as `x.x.x.x:yyyy` it will attempt to connect to the port specified by `yyyy`

#### `pDcgmHandle`

OUT: DCGM Handle of the remote host engine

### Returns

- ▶ `DCGM_ST_OK` if we successfully connected to the remote host engine

- ▶ DCGM\_ST\_CONNECTION\_NOT\_VALID if the remote host engine could not be reached
- ▶ DCGM\_ST\_UNINITIALIZED if DCGM has not been initialized with `dcmgInit`.
- ▶ DCGM\_ST\_BADPARAM if `pDcmgHandle` is NULL or `ipAddress` is invalid
- ▶ DCGM\_ST\_INIT\_ERROR if DCGM encountered an error while initializing the remote client library
- ▶ DCGM\_ST\_UNINITIALIZED if DCGM has not been initialized with `dcmgInit`

### Description

This method is used to connect to a stand-alone host engine process. Remote host engines are started by running the `nv-hostengine` command.

NOTE: `dcmgConnect_v2` provides additional connection options.

`dcmgReturn_t dcmgConnect_v2 (char *ipAddress,  
dcmgConnectV2Params_t *connectParams, dcmgHandle_t  
*pDcmgHandle)`

### Parameters

#### `ipAddress`

IN: Valid IP address for the remote host engine to connect to. If `ipAddress` is specified as `x.x.x.x` it will attempt to connect to the default port specified by `DCGM_HE_PORT_NUMBER`. If `ipAddress` is specified as `x.x.x.x:yyyy` it will attempt to connect to the port specified by `yyyy`

#### `connectParams`

IN: Additional connection parameters. See `dcmgConnectV2Params_t` for details.

#### `pDcmgHandle`

OUT: DCGM Handle of the remote host engine

### Returns

- ▶ DCGM\_ST\_OK if we successfully connected to the remote host engine
- ▶ DCGM\_ST\_CONNECTION\_NOT\_VALID if the remote host engine could not be reached
- ▶ DCGM\_ST\_UNINITIALIZED if DCGM has not been initialized with `dcmgInit`.
- ▶ DCGM\_ST\_BADPARAM if `pDcmgHandle` is NULL or `ipAddress` is invalid
- ▶ DCGM\_ST\_INIT\_ERROR if DCGM encountered an error while initializing the remote client library
- ▶ DCGM\_ST\_UNINITIALIZED if DCGM has not been initialized with `dcmgInit`

**Description**

This method is used to connect to a stand-alone host engine process. Remote host engines are started by running the nv-hostengine command.

**dcgmReturn\_t dcgmDisconnect (dcgmHandle\_t pDcgmHandle)****Parameters****pDcgmHandle**

IN: DCGM Handle that came from dcgmConnect

**Returns**

- ▶ DCGM\_ST\_OK if we successfully disconnected from the host engine
- ▶ DCGM\_ST\_UNINITIALIZED if DCGM has not been initialized with dcgmInit
- ▶ DCGM\_ST\_BADPARAM if pDcgmHandle is not a valid DCGM handle
- ▶ DCGM\_ST\_GENERIC\_ERROR if an unspecified internal error occurred

**Description**

This method is used to disconnect from a stand-alone host engine process.

**1.1.2. Auxiliary information about DCGM engine.**

## Administrative

Describes APIs to get generic information about the DCGM Engine.

**dcgmReturn\_t dcgmVersionInfo (dcgmVersionInfo\_t \*pVersionInfo)****Parameters****pVersionInfo**

OUT: Build environment information

**Returns**

- ▶ DCGM\_ST\_OK if build information is successfully obtained
- ▶ DCGM\_ST\_BADPARAM if pVersionInfo is null
- ▶ DCGM\_ST\_VER\_MISMATCH if the expected and provided versions of dcgmVersionInfo\_t do not match



**Description**

This method is used to return information about the build environment where DCGM was built.

**dcgmReturn\_t dcgmHostengineVersionInfo (dcgmHandle\_t pDcgmHandle, dcgmVersionInfo\_t \*pVersionInfo)**

**Parameters****pDcgmHandle**

IN: DCGM Handle that came from dcgmConnect

**pVersionInfo**

OUT: Build environment information

**Returns**

- ▶ DCGM\_ST\_OK if build information is successfully obtained
- ▶ DCGM\_ST\_BADPARAM if pVersionInfo is null
- ▶ DCGM\_ST\_VER\_MISMATCH if the expected and provided versions of dcgmVersionInfo\_t do not match

**Description**

This method is used to return information about the build environment of the hostengine.

**dcgmReturn\_t dcgmHostengineSetLoggingSeverity (dcgmHandle\_t pDcgmHandle, dcgmSettingsSetLoggingSeverity\_t \*logging)**

**Parameters****pDcgmHandle**

IN: DCGM Handle

**logging**

IN: dcgmSettingsSetLoggingSeverity\_t struct containing the target logger and severity

**Returns**

- ▶ DCGM\_ST\_OK Severity successfully set
- ▶ DCGM\_ST\_BADPARAM Bad logger/severity string
- ▶ DCGM\_ST\_VER\_MISMATCH if the expected and provided versions of dcgmSettingsSetLoggingSeverity\_t do not match

**Description**

This method is used to set the logging severity on HostEngine for the specified logger

```
dcgmReturn_t dcgmHostengineIsHealthy (dcgmHandle_t
pDcgmHandle, dcgmHostengineHealth_t *heHealth)
```

**Parameters****pDcgmHandle**

- the handle to DCGM

**heHealth**

- struct describing the health of the hostengine. if heHealth.hostengineHealth is 0, then the hostengine is healthy. Non-zero indicates not healthy with error codes determining the cause.

**Returns**

- ▶ DCGM\_ST\_OK Able to gauge health
- ▶ DCGM\_ST\_BADPARAM isHealthy is not a valid pointer

**Description**

This function is used to return whether or not the host engine considers itself healthy

## 1.2. System

This chapter describes the APIs used to identify set of GPUs on the node, grouping functions to provide mechanism to operate on a group of GPUs, and status management APIs in order to get individual statuses for each operation. The APIs in System module can be broken down into following categories:

### Discovery

### Grouping

### Field Grouping

### Status handling

## 1.2.1. Discovery

System

The following APIs are used to discover GPUs and their attributes on a Node.

**dcgmReturn\_t dcgmGetAllDevices (dcgmHandle\_t pDcgmHandle, unsigned int gpuIdList, int \*count)**

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **gpuIdList**

OUT: Array reference to fill GPU Ids present on the system.

#### **count**

OUT: Number of GPUs returned in gpuIdList.

### Returns

- ▶ DCGM\_ST\_OK if the call was successful.
- ▶ DCGM\_ST\_BADPARAM if gpuIdList or count were not valid.

### Description

This method is used to get identifiers corresponding to all the devices on the system. The identifier represents DCGM GPU Id corresponding to each GPU on the system and is immutable during the lifespan of the engine. The list should be queried again if the engine is restarted.

The GPUs returned from this function include gpuIds of GPUs that are not supported by DCGM. To only get gpuIds of GPUs that are supported by DCGM, use [dcgmGetAllSupportedDevices\(\)](#).

**dcgmReturn\_t dcgmGetAllSupportedDevices (dcgmHandle\_t pDcgmHandle, unsigned int gpuIdList, int \*count)**

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **gpuIdList**

OUT: Array reference to fill GPU Ids present on the system.

#### **count**

OUT: Number of GPUs returned in gpuIdList.

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful.
- ▶ DCGM\_ST\_BADPARAM if gpuIdList or count were not valid.

**Description**

This method is used to get identifiers corresponding to all the DCGM-supported devices on the system. The identifier represents DCGM GPU Id corresponding to each GPU on the system and is immutable during the lifespan of the engine. The list should be queried again if the engine is restarted.

The GPUs returned from this function ONLY includes gpuIds of GPUs that are supported by DCGM. To get gpuIds of all GPUs in the system, use [dcmGetAllDevices\(\)](#).

```
dcgmReturn_t dcmGetDeviceAttributes (dcgmHandle_t
pDcmHandle, unsigned int gpuId, dcmDeviceAttributes_t
*pDcmAttr)
```

**Parameters****pDcmHandle**

IN: DCGM Handle

**gpuId**

IN: GPU Id corresponding to which the attributes should be fetched

**pDcmAttr**

IN/OUT: Device attributes corresponding to gpuId. pDcmAttr->version should be set to [dcmDeviceAttributes\\_version](#) before this call.

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful.
- ▶ DCGM\_ST\_VER\_MISMATCH if pDcmAttr->version is not set or is invalid.

**Description**

Gets device attributes corresponding to the gpuId. If operation is not successful for any of the requested fields then the field is populated with one of DCGM\_BLANK\_VALUES defined in [dcm\\_structs.h](#).

`dcgmReturn_t dcgmGetEntityGroupEntities (dcgmHandle_t dcgmHandle, dcgm_field_entity_group_t entityGroup, dcgm_field_eid_t *entities, int *numEntities, unsigned int flags)`

### Parameters

#### **dcgmHandle**

IN: DCGM Handle

#### **entityGroup**

IN: Entity group to list entities of

#### **entities**

OUT: Array of entities for entityGroup

#### **numEntities**

IN/OUT: Upon calling, this should be the number of entities that entityList[] can hold. Upon return, this will contain the number of entities actually saved to entityList.

#### **flags**

IN: Flags to modify the behavior of this request. See DCGM\_GEGE\_FLAG\_\* defines in dcgm\_structs.h

### Returns

- ▶ DCGM\_ST\_OK if the call was successful.
- ▶ DCGM\_ST\_INSUFFICIENT\_SIZE if numEntities was not large enough to hold the number of entities in the entityGroup. numEntities will contain the capacity needed to complete this request successfully.
- ▶ DCGM\_ST\_NOT\_SUPPORTED if the given entityGroup does not support enumeration.
- ▶ DCGM\_ST\_BADPARAM if any parameter is invalid

### Description

Gets the list of entities that exist for a given entity group. This API can be used in place of [dcgmGetAllDevices](#).

`dcgmReturn_t dcgmGetGpuInstanceHierarchy (dcgmHandle_t dcgmHandle, dcgmMigHierarchy_v2 *hierarchy)`

### Parameters

#### **dcgmHandle**

IN: DCGM Handle

## hierarchy

### Returns

- ▶ DCGM\_ST\_OK if the call was successful.
- ▶ DCGM\_ST\_VER\_MISMATCH if the struct version is incorrect
- ▶ DCGM\_ST\_BADPARAM if any parameter is invalid

### Description

Gets the hierarchy of GPUs, GPU Instances, and Compute Instances by populating a list of each entity with a reference to their parent

`dcgmReturn_t dcgmGetNvLinkLinkStatus (dcgmHandle_t dcgmHandle, dcgmNvLinkStatus_v2 *linkStatus)`

### Parameters

#### **dcgmHandle**

IN: DCGM Handle

#### **linkStatus**

OUT: Structure in which to store NvLink link statuses. .version should be set to dcgmNvLinkStatus\_version1 before calling this.

### Returns

- ▶ DCGM\_ST\_OK if the call was successful.
- ▶ DCGM\_ST\_NOT\_SUPPORTED if the given entityGroup does not support enumeration.
- ▶ DCGM\_ST\_BADPARAM if any parameter is invalid

### Description

Get the NvLink link status for every NvLink in this system. This includes the NvLinks of both GPUs and NvSwitches. Note that only NvSwitches and GPUs that are visible to the current environment will be returned in this structure.

## 1.2.2. Grouping

### System

The following APIs are used for group management. The user can create a group of entities and perform an operation on a group of entities. If grouping is not needed and the user wishes to run commands on all GPUs seen by DCGM then the user can use

DCGM\_GROUP\_ALL\_GPUS or DCGM\_GROUP\_ALL\_NVSWITCHES in place of group IDs when needed.

```
dcgmReturn_t dcgmGroupCreate (dcgmHandle_t pDcgmHandle,
dcgmGroupType_t type, char *groupName, dcgmGpuGrp_t
*pDcgmGrpId)
```

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **type**

IN: Type of Entity Group to be formed

#### **groupName**

IN: Desired name of the GPU group specified as NULL terminated C string

#### **pDcgmGrpId**

OUT: Reference to group ID

### Returns

- ▶ DCGM\_ST\_OK if the group has been created
- ▶ DCGM\_ST\_BADPARAM if any of type, groupName, length or pDcgmGrpId is invalid
- ▶ DCGM\_ST\_MAX\_LIMIT if number of groups on the system has reached the max limit DCGM\_MAX\_NUM\_GROUPS
- ▶ DCGM\_ST\_INIT\_ERROR if the library has not been successfully initialized

### Description

Used to create a entity group handle which can store one or more entity Ids as an opaque handle returned in pDcgmGrpId. Instead of executing an operation separately for each entity, the DCGM group enables the user to execute same operation on all the entities present in the group as a single API call.

To create the group with all the entities present on the system, the type field should be specified as DCGM\_GROUP\_DEFAULT or DCGM\_GROUP\_ALL\_NVSWITCHES. To create an empty group, the type field should be specified as DCGM\_GROUP\_EMPTY. The empty group can be updated with the desired set of entities using the APIs [dcgmGroupAddDevice](#), [dcgmGroupAddEntity](#), [dcgmGroupRemoveDevice](#), and [dcgmGroupRemoveEntity](#).

**dcgmReturn\_t dcgmGroupDestroy (dcgmHandle\_t pDcgmHandle, dcgmGpuGrp\_t groupId)**

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **groupId**

IN: Group ID

### Returns

- ▶ DCGM\_ST\_OK if the group has been destroyed
- ▶ DCGM\_ST\_BADPARAM if groupId is invalid
- ▶ DCGM\_ST\_INIT\_ERROR if the library has not been successfully initialized
- ▶ DCGM\_ST\_NOT\_CONFIGURED if entry corresponding to the group does not exist

### Description

Used to destroy a group represented by groupId. Since DCGM group is a logical grouping of entities, the properties applied on the group stay intact for the individual entities even after the group is destroyed.

**dcgmReturn\_t dcgmGroupAddDevice (dcgmHandle\_t pDcgmHandle, dcgmGpuGrp\_t groupId, unsigned int gpuId)**

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **groupId**

IN: Group Id to which device should be added

#### **gpuId**

IN: DCGM GPU Id

### Returns

- ▶ DCGM\_ST\_OK if the GPU Id has been successfully added to the group
- ▶ DCGM\_ST\_INIT\_ERROR if the library has not been successfully initialized
- ▶ DCGM\_ST\_NOT\_CONFIGURED if entry corresponding to the group (groupId) does not exist
- ▶ DCGM\_ST\_BADPARAM if gpuId is invalid or already part of the specified group



**Description**

Used to add specified GPU Id to the group represented by groupId.

```
dcgmReturn_t dcgmGroupAddEntity (dcgmHandle_t pDcgmHandle,
dcgmGpuGrp_t groupId, dcgm_field_entity_group_t entityGroupId,
dcgm_field_eid_t entityId)
```

**Parameters****pDcgmHandle**

IN: DCGM Handle

**groupId**

IN: Group Id to which device should be added

**entityGroupId**

IN: Entity group that entityId belongs to

**entityId**

IN: DCGM entityId

**Returns**

- ▶ DCGM\_ST\_OK if the entity has been successfully added to the group
- ▶ DCGM\_ST\_INIT\_ERROR if the library has not been successfully initialized
- ▶ DCGM\_ST\_NOT\_CONFIGURED if entry corresponding to the group (groupId) does not exist
- ▶ DCGM\_ST\_BADPARAM if entityId is invalid or already part of the specified group

**Description**

Used to add specified entity to the group represented by groupId.

```
dcgmReturn_t dcgmGroupRemoveDevice (dcgmHandle_t
pDcgmHandle, dcgmGpuGrp_t groupId, unsigned int gpuld)
```

**Parameters****pDcgmHandle**

IN: DCGM Handle

**groupId**

IN: Group ID from which device should be removed

**gpuld**

IN: DCGM GPU Id

**Returns**

- ▶ DCGM\_ST\_OK if the GPU Id has been successfully removed from the group
- ▶ DCGM\_ST\_INIT\_ERROR if the library has not been successfully initialized
- ▶ DCGM\_ST\_NOT\_CONFIGURED if entry corresponding to the group (groupId) does not exist
- ▶ DCGM\_ST\_BADPARAM if groupId is invalid or not part of the specified group

**Description**

Used to remove specified GPU Id from the group represented by groupId.

```
dcgmReturn_t dcgmGroupRemoveEntity (dcgmHandle_t
pDcgmHandle, dcgmGpuGrp_t groupId, dcgm_field_entity_group_t
entityGroupId, dcgm_field_eid_t entityId)
```

**Parameters****pDcgmHandle**

IN: DCGM Handle

**groupId**

IN: Group ID from which device should be removed

**entityGroupId**

IN: Entity group that entityId belongs to

**entityId**

IN: DCGM entityId

**Returns**

- ▶ DCGM\_ST\_OK if the entity has been successfully removed from the group
- ▶ DCGM\_ST\_INIT\_ERROR if the library has not been successfully initialized
- ▶ DCGM\_ST\_NOT\_CONFIGURED if entry corresponding to the group (groupId) does not exist
- ▶ DCGM\_ST\_BADPARAM if entityId is invalid or not part of the specified group

**Description**

Used to remove specified entity from the group represented by groupId.

**dcgmReturn\_t dcgmGroupGetInfo (dcgmHandle\_t pDcgmHandle, dcgmGpuGrp\_t groupId, dcgmGroupInfo\_t \*pDcgmGroupInfo)**

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **groupId**

IN: Group ID for which information to be fetched

#### **pDcgmGroupInfo**

OUT: Group Information

### Returns

- ▶ DCGM\_ST\_OK if the group info is successfully received.
- ▶ DCGM\_ST\_BADPARAM if any of groupId or pDcgmGroupInfo is invalid.
- ▶ DCGM\_ST\_INIT\_ERROR if the library has not been successfully initialized.
- ▶ DCGM\_ST\_MAX\_LIMIT if the group does not contain the GPU
- ▶ DCGM\_ST\_NOT\_CONFIGURED if entry corresponding to the group (groupId) does not exist

### Description

Used to get information corresponding to the group represented by groupId. The information returned in pDcgmGroupInfo consists of group name, and the list of entities present in the group.

**dcgmReturn\_t dcgmGroupGetAllIds (dcgmHandle\_t pDcgmHandle, dcgmGpuGrp\_t groupIdList, unsigned int \*count)**

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **groupIdList**

OUT: List of Group Ids

#### **count**

OUT: The number of Group ids in the list

### Returns

- ▶ DCGM\_ST\_OK if the ids of the groups were successfully retrieved
- ▶ DCGM\_ST\_BADPARAM if either of the groupIdList or count is null
- ▶ DCGM\_ST\_GENERIC\_ERROR if an unknown error has occurred

## Description

Used to get the Ids of all groups of entities. The information returned is a list of group ids in `groupIdList` as well as a count of how many ids there are in `count`. Please allocate enough memory for `groupIdList`. Memory of size `MAX_NUM_GROUPS` should be allocated for `groupIdList`.

## 1.2.3. Field Grouping

### System

The following APIs are used for field group management. The user can create a group of fields and perform an operation on a group of fields at once.

```
dcgmReturn_t dcgmFieldGroupCreate (dcgmHandle_t dcgmHandle,
int numFieldIds, unsigned short *fieldIds, char *fieldGroupName,
dcgmFieldGrp_t *dcgmFieldGroupId)
```

### Parameters

#### **dcgmHandle**

IN: DCGM handle

#### **numFieldIds**

IN: Number of field IDs that are being provided in `fieldIds[]`. Must be between 1 and `DCGM_MAX_FIELD_IDS_PER_FIELD_GROUP`.

#### **fieldIds**

IN: Field IDs to be added to the newly-created field group

#### **fieldGroupName**

IN: Unique name for this group of fields. This must not be the same as any existing field groups.

#### **dcgmFieldGroupId**

OUT: Handle to the newly-created field group

### Returns

- ▶ `DCGM_ST_OK` if the field group was successfully created.
- ▶ `DCGM_ST_BADPARAM` if any parameters were bad
- ▶ `DCGM_ST_INIT_ERROR` if the library has not been successfully initialized.
- ▶ `DCGM_ST_MAX_LIMIT` if too many field groups already exist

## Description

Used to create a group of fields and return the handle in `dcgmFieldGroupId`

`dcgmReturn_t dcgmFieldGroupDestroy (dcgmHandle_t dcgmHandle, dcgmFieldGrp_t dcgmFieldGroupId)`

### Parameters

#### **dcgmHandle**

IN: DCGM handle

#### **dcgmFieldGroupId**

IN: Field group to remove

### Returns

- ▶ DCGM\_ST\_OK if the field group was successfully removed
- ▶ DCGM\_ST\_BADPARAM if any parameters were bad
- ▶ DCGM\_ST\_INIT\_ERROR if the library has not been successfully initialized.

### Description

Used to remove a field group that was created with `dcgmFieldGroupCreate`

`dcgmReturn_t dcgmFieldGroupGetInfo (dcgmHandle_t dcgmHandle, dcgmFieldGroupInfo_t *fieldGroupInfo)`

### Parameters

#### **dcgmHandle**

IN: DCGM handle

#### **fieldGroupInfo**

IN/OUT: Info about all of the field groups that exist. `.version` should be set to `dcgmFieldGroupInfo_version` before this call `.fieldGroupId` should contain the `fieldGroupId` you are interested in querying information for.

### Returns

- ▶ DCGM\_ST\_OK if the field group info was returned successfully
- ▶ DCGM\_ST\_BADPARAM if any parameters were bad
- ▶ DCGM\_ST\_INIT\_ERROR if the library has not been successfully initialized.
- ▶ DCGM\_ST\_VER\_MISMATCH if `.version` is not set or is invalid.

### Description

Used to get information about a field group that was created with `dcgmFieldGroupCreate`.

**dcgmReturn\_t dcgmFieldGroupGetAll (dcgmHandle\_t dcgmHandle, dcgmAllFieldGroup\_t \*allGroupInfo)**

### Parameters

#### **dcgmHandle**

IN: DCGM handle

#### **allGroupInfo**

IN/OUT: Info about all of the field groups that exist. `.version` should be set to `dcgmAllFieldGroup_version` before this call.

### Returns

- ▶ DCGM\_ST\_OK if the field group info was successfully returned
- ▶ DCGM\_ST\_BADPARAM if any parameters were bad
- ▶ DCGM\_ST\_INIT\_ERROR if the library has not been successfully initialized.
- ▶ DCGM\_ST\_VER\_MISMATCH if `.version` is not set or is invalid.

### Description

Used to get information about all field groups in the system.

## 1.2.4. Status handling

### System

The following APIs are used to manage statuses for multiple operations on one or more GPUs.

**dcgmReturn\_t dcgmStatusCreate (dcgmStatus\_t \*statusHandle)**

### Parameters

#### **statusHandle**

OUT: Reference to handle for list of statuses

### Returns

- ▶ DCGM\_ST\_OK if the status handle is successfully created
- ▶ DCGM\_ST\_BADPARAM if statusHandle is invalid

### Description

Creates reference to DCGM status handler which can be used to get the statuses for multiple operations on one or more devices.

The multiple statuses are useful when the operations are performed at group level. The status handle provides a mechanism to access error attributes for the failed operations.

The number of errors stored behind the opaque handle can be accessed using the the API `dcgmStatusGetCount`. The errors are accessed from the opaque handle `statusHandle` using the API `dcgmStatusPopError`. The user can invoke `dcgmStatusPopError` for the number of errors or until all the errors are fetched.

When the status handle is not required any further then it should be deleted using the API `dcgmStatusDestroy`.

## `dcgmReturn_t dcgmStatusDestroy (dcgmStatus_t statusHandle)`

### Parameters

#### `statusHandle`

IN: Handle to list of statuses

### Returns

- ▶ `DCGM_ST_OK` if the status handle is successfully created
- ▶ `DCGM_ST_BADPARAM` if `statusHandle` is invalid

### Description

Used to destroy status handle created using `dcgmStatusCreate`.

## `dcgmReturn_t dcgmStatusGetCount (dcgmStatus_t statusHandle, unsigned int *count)`

### Parameters

#### `statusHandle`

IN: Handle to list of statuses

#### `count`

OUT: Number of error entries present in the list of statuses

### Returns

- ▶ `DCGM_ST_OK` if the error count is successfully received
- ▶ `DCGM_ST_BADPARAM` if any of `statusHandle` or `count` is invalid

### Description

Used to get count of error entries stored inside the opaque handle `statusHandle`.

`dcgmReturn_t dcgmStatusPopError (dcgmStatus_t statusHandle, dcgmErrorInfo_t *pDcgmErrorInfo)`

### Parameters

#### **statusHandle**

IN: Handle to list of statuses

#### **pDcgmErrorInfo**

OUT: First error from the list of statuses

### Returns

- ▶ DCGM\_ST\_OK if the error entry is successfully fetched
- ▶ DCGM\_ST\_BADPARAM if any of statusHandle or pDcgmErrorInfo is invalid
- ▶ DCGM\_ST\_NO\_DATA if the status handle list is empty

### Description

Used to iterate through the list of errors maintained behind statusHandle. The method pops the first error from the list of DCGM statuses. In order to iterate through all the errors, the user can invoke this API for the number of errors or until all the errors are fetched.

`dcgmReturn_t dcgmStatusClear (dcgmStatus_t statusHandle)`

### Parameters

#### **statusHandle**

IN: Handle to list of statuses

### Returns

- ▶ DCGM\_ST\_OK if the errors are successfully cleared
- ▶ DCGM\_ST\_BADPARAM if statusHandle is invalid

### Description

Used to clear all the errors in the status handle created by the API [dcgmStatusCreate](#). After one set of operation, the statusHandle can be cleared and reused for the next set of operation.



## 1.3. Configuration

This chapter describes the methods that handle device configuration retrieval and default settings. The APIs in Configuration module can be broken down into following categories:

### Setup and management

#### Manual Invocation

##### 1.3.1. Setup and management

Configuration

Describes APIs to Get/Set configuration on the group of GPUs.

```
dcgmReturn_t dcgmConfigSet (dcgmHandle_t pDcgmHandle,
dcgmGpuGrp_t groupId, dcgmConfig_t *pDeviceConfig,
dcgmStatus_t statusHandle)
```

#### Parameters

##### **pDcgmHandle**

IN: DCGM Handle

##### **groupId**

IN: Group ID representing collection of one or more GPUs. Look at [dcgmGroupCreate](#) for details on creating the group.

##### **pDeviceConfig**

IN: Pointer to memory to hold desired configuration to be applied for all the GPU in the group represented by groupId. The caller must populate the version field of pDeviceConfig.

##### **statusHandle**

IN/OUT: Resulting error status for multiple operations. Pass it as NULL if the detailed error information is not needed. Look at [dcgmStatusCreate](#) for details on creating status handle.

#### Returns

- ▶ DCGM\_ST\_OK if the configuration has been successfully set.
- ▶ DCGM\_ST\_BADPARAM if any of groupId or pDeviceConfig is invalid.
- ▶ DCGM\_ST\_VER\_MISMATCH if pDeviceConfig has the incorrect version.
- ▶ DCGM\_ST\_GENERIC\_ERROR if an unknown error has occurred.

## Description

Used to set configuration for the group of one or more GPUs identified by `groupId`.

The configuration settings specified in `pDeviceConfig` are applied to all the GPUs in the group. Since DCGM group is a logical grouping of GPUs, the configuration settings stays intact for the individual GPUs even after the group is destroyed.

If the user wishes to ignore the configuration of one or more properties in the input `pDeviceConfig` then the property should be specified as one of `DCGM_INT32_BLANK`, `DCGM_INT64_BLANK`, `DCGM_FP64_BLANK` or `DCGM_STR_BLANK` based on the data type of the property to be ignored.

If any of the properties fail to be configured for any of the GPUs in the group then the API returns an error. The status handle `statusHandle` should be further evaluated to access error attributes for the failed operations. Please refer to status management APIs at [Status handling](#) to access the error attributes.

To find out valid supported clock values that can be passed to `dcgmConfigSet`, look at the device attributes of a GPU in the group using the API `dcgmGetDeviceAttributes`.

`dcgmReturn_t dcgmConfigGet (dcgmHandle_t pDcgmHandle, dcgmGpuGrp_t groupId, dcgmConfigType_t type, int count, dcgmConfig_t deviceConfigList, dcgmStatus_t statusHandle)`

## Parameters

### **pDcgmHandle**

IN: DCGM Handle

### **groupId**

IN: Group ID representing collection of one or more GPUs. Look at [dcgmGroupCreate](#) for details on creating the group.

### **type**

IN: Type of configuration values to be fetched.

### **count**

IN: The number of entries that `deviceConfigList` array can store.

### **deviceConfigList**

OUT: Pointer to memory to hold requested configuration corresponding to all the GPUs in the group (`groupId`). The size of the memory must be greater than or equal to hold output information for the number of GPUs present in the group (`groupId`).

### **statusHandle**

IN/OUT: Resulting error status for multiple operations. Pass it as NULL if the detailed error information is not needed. Look at [dcgmStatusCreate](#) for details on creating status handle.

## Returns

- ▶ DCGM\_ST\_OK if the configuration has been successfully fetched.
- ▶ DCGM\_ST\_BADPARAM if any of groupId, type, count, or deviceConfigList is invalid.
- ▶ DCGM\_ST\_NOT\_CONFIGURED if the target configuration is not already set.
- ▶ DCGM\_ST\_VER\_MISMATCH if deviceConfigList has the incorrect version.
- ▶ DCGM\_ST\_GENERIC\_ERROR if an unknown error has occurred.

## Description

Used to get configuration for all the GPUs present in the group.

This API can get the most recent target or desired configuration set by [dcgmConfigSet](#). Set type as DCGM\_CONFIG\_TARGET\_STATE to get target configuration. The target configuration properties are maintained by DCGM and are automatically enforced after a GPU reset or reinitialization is completed.

The method can also be used to get the actual configuration state for the GPUs in the group. Set type as DCGM\_CONFIG\_CURRENT\_STATE to get the actual configuration state. Ideally, the actual configuration state will be exact same as the target configuration state.

If any of the property in the target configuration is unknown then the property value in the output is populated as one of DCGM\_INT32\_BLANK, DCGM\_INT64\_BLANK, DCGM\_FP64\_BLANK or DCGM\_STR\_BLANK based on the data type of the property.

If any of the property in the current configuration state is not supported then the property value in the output is populated as one of DCGM\_INT32\_NOT\_SUPPORTED, DCGM\_INT64\_NOT\_SUPPORTED, DCGM\_FP64\_NOT\_SUPPORTED or DCGM\_STR\_NOT\_SUPPORTED based on the data type of the property.

If any of the properties can't be fetched for any of the GPUs in the group then the API returns an error. The status handle statusHandle should be further evaluated to access error attributes for the failed operations. Please refer to status management APIs at [Status handling](#) to access the error attributes.

## 1.3.2. Manual Invocation

### Configuration

Describes APIs used to manually enforce the desired configuration on a group of GPUs.

`dcgmReturn_t dcgmConfigEnforce (dcgmHandle_t pDcgmHandle, dcgmGpuGrp_t groupId, dcgmStatus_t statusHandle)`

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **groupId**

IN: Group ID representing collection of one or more GPUs. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs.

#### **statusHandle**

IN/OUT: Resulting error status for multiple operations. Pass it as NULL if the detailed error information is not needed. Look at [dcgmStatusCreate](#) for details on creating status handle.

### Returns

- ▶ DCGM\_ST\_OK if the configuration has been successfully enforced.
- ▶ DCGM\_ST\_BADPARAM if groupId is invalid.
- ▶ DCGM\_ST\_NOT\_CONFIGURED if the target configuration is not already set.
- ▶ DCGM\_ST\_GENERIC\_ERROR if an unknown error has occurred.

### Description

Used to enforce previously set configuration for all the GPUs present in the group.

This API provides a mechanism to the users to manually enforce the configuration at any point of time. The configuration can only be enforced if it's already configured using the API [dcgmConfigSet](#).

If any of the properties can't be enforced for any of the GPUs in the group then the API returns an error. The status handle statusHandle should be further evaluated to access error attributes for the failed operations. Please refer to status management APIs at [Status handling](#) to access the error attributes.

## 1.4. Field APIs

These APIs are responsible for watching, unwatching, and updating specific fields as defined by DCGM\_FI\_\*

`dcgmReturn_t dcgmWatchFields (dcgmHandle_t pDcgmHandle, dcgmGpuGrp_t groupId, dcgmFieldGrp_t fieldGroupId, long long updateFreq, double maxKeepAge, int maxKeepSamples)`

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **groupId**

IN: Group ID representing collection of one or more entities. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as `DCGM_GROUP_ALL_GPUS` to perform operation on all the GPUs or `DCGM_GROUP_ALL_NVSITCHES` to perform the operation on all NvSwitches.

#### **fieldGroupId**

IN: Fields to watch.

#### **updateFreq**

IN: How often to update this field in usec

#### **maxKeepAge**

IN: How long to keep data for this field in seconds

#### **maxKeepSamples**

IN: Maximum number of samples to keep. 0=no limit

### Returns

- ▶ `DCGM_ST_OK` if the call was successful
- ▶ `DCGM_ST_BADPARAM` if a parameter is invalid

### Description

Request that DCGM start recording updates for a given field collection.

Note that the first update of the field will not occur until the next field update cycle. To force a field update cycle, call `dcgmUpdateAllFields(1)`.

## dcgmReturn\_t dcgmUnwatchFields (dcgmHandle\_t pDcgmHandle, dcgmGpuGrp\_t groupId, dcgmFieldGrp\_t fieldGroupId)

### Parameters

#### pDcgmHandle

IN: DCGM Handle

#### groupId

IN: Group ID representing collection of one or more entities. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs or DCGM\_GROUP\_ALL\_NVSITCHES to to perform the operation on all NvSwitches.

#### fieldGroupId

IN: Fields to unwatch.

### Returns

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid

### Description

Request that DCGM stop recording updates for a given field collection.

## dcgmReturn\_t dcgmGetValuesSince (dcgmHandle\_t pDcgmHandle, dcgmGpuGrp\_t groupId, dcgmFieldGrp\_t fieldGroupId, long long sinceTimestamp, long long \*nextSinceTimestamp, dcgmFieldValueEnumeration\_f enumCB, void \*userData)

### Parameters

#### pDcgmHandle

IN: DCGM Handle

#### groupId

IN: Group ID representing collection of one or more GPUs. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs.

#### fieldGroupId

IN: Fields to return data for

**sinceTimestamp**

IN: Timestamp to request values since in usec since 1970. This will be returned in nextSinceTimestamp for subsequent calls 0 = request all data

**nextSinceTimestamp**

OUT: Timestamp to use for sinceTimestamp on next call to this function

**enumCB**

IN: Callback to invoke for every field value update. Note that multiple updates can be returned in each invocation

**userData**

IN: User data pointer to pass to the userData field of enumCB.

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_NOT\_SUPPORTED if one of the entities was from a non-GPU type
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid

**Description**

Request updates for all field values that have updated since a given timestamp

This version only works with GPU entities. Use [dcmGetValuesSince\\_v2](#) for entity groups containing NvSwitches.

```
dcgmReturn_t dcmGetValuesSince_v2
(dcmHandle_t pDcmHandle, dcmGpuGrp_t
groupId, dcmFieldGrp_t fieldGroupId, long long
sinceTimestamp, long long *nextSinceTimestamp,
dcmFieldValueEnumeration_f enumCB, void
*userData)
```

**Parameters****pDcmHandle**

IN: DCGM Handle

**groupId**

IN: Group ID representing collection of one or more entities. Look at [dcmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs or DCGM\_GROUP\_ALL\_NVSITCHES to perform the operation on all NvSwitches.

**fieldGroupId**

IN: Fields to return data for

**sinceTimestamp**

IN: Timestamp to request values since in usec since 1970. This will be returned in nextSinceTimestamp for subsequent calls 0 = request all data

**nextSinceTimestamp**

OUT: Timestamp to use for sinceTimestamp on next call to this function

**enumCB**

IN: Callback to invoke for every field value update. Note that multiple updates can be returned in each invocation

**userData**

IN: User data pointer to pass to the userData field of enumCB.

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid

**Description**

Request updates for all field values that have updated since a given timestamp

This version works with non-GPU entities like NvSwitches

```
dcgmReturn_t dcgmGetLatestValues (dcgmHandle_t
pDcgmHandle, dcgmGpuGrp_t groupId, dcgmFieldGrp_t
fieldGroupId, dcgmFieldValueEnumeration_f enumCB,
void *userData)
```

**Parameters****pDcgmHandle**

IN: DCGM Handle

**groupId**

IN: Group ID representing collection of one or more GPUs. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs.

**fieldGroupId**

IN: Fields to return data for.

**enumCB**

IN: Callback to invoke for every field value update. Note that multiple updates can be returned in each invocation

**userData**

IN: User data pointer to pass to the userData field of enumCB.



**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_NOT\_SUPPORTED if one of the entities was from a non-GPU type
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid

**Description**

Request latest cached field value for a field value collection

This version only works with GPU entities. Use `dcgmGetLatestValues_v2` for entity groups containing NvSwitches.

`dcgmReturn_t dcgmGetLatestValues_v2 (dcgmHandle_t pDcgmHandle, dcgmGpuGrp_t groupId, dcgmFieldGrp_t fieldGroupId, dcgmFieldValueEnumeration_f enumCB, void *userData)`

**Parameters****pDcgmHandle**

IN: DCGM Handle

**groupId**

IN: Group ID representing collection of one or more entities. Look at `dcgmGroupCreate` for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs or DCGM\_GROUP\_ALL\_NVSITCHES to perform the operation on all NvSwitches.

**fieldGroupId**

IN: Fields to return data for.

**enumCB**

IN: Callback to invoke for every field value update. Note that multiple updates can be returned in each invocation

**userData**

IN: User data pointer to pass to the userData field of enumCB.

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_NOT\_SUPPORTED if one of the entities was from a non-GPU type
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid

**Description**

Request latest cached field value for a field value collection

This version works with non-GPU entities like NvSwitches

**dcgmReturn\_t dcgmGetLatestValuesForFields**  
(**dcgmHandle\_t pDcgmHandle**, **int gpuId**, **unsigned short fields**, **unsigned int count**, **dcgmFieldValue\_v1 values**)

#### Parameters

##### **pDcgmHandle**

IN: DCGM Handle

##### **gpuId**

IN: Gpu ID representing the GPU for which the fields are being requested.

##### **fields**

IN: Field IDs to return data for. See the definitions in `dcgm_fields.h` that start with `DCGM_FI_`.

##### **count**

IN: Number of field IDs in `fields[]` array.

##### **values**

OUT: Latest field values for the fields in `fields[]`.

#### Description

Request latest cached field value for a GPU

**dcgmReturn\_t dcgmEntityGetLatestValues**  
(**dcgmHandle\_t pDcgmHandle**,  
**dcgm\_field\_entity\_group\_t entityGroup**, **int entityId**, **unsigned short fields**, **unsigned int count**,  
**dcgmFieldValue\_v1 values**)

#### Parameters

##### **pDcgmHandle**

IN: DCGM Handle

##### **entityGroup**

IN: `entity_group_t` (e.g. switch)

##### **entityId**

IN: entity ID representing the entity for which the fields are being requested.

##### **fields**

IN: Field IDs to return data for. See the definitions in `dcgm_fields.h` that start with `DCGM_FI_`.

**count**

IN: Number of field IDs in fields[] array.

**values**

OUT: Latest field values for the fields in fields[].

**Description**

Request latest cached field value for a group of fields for a specific entity

**dcgmReturn\_t dcgmEntitiesGetLatestValues**  
(**dcgmHandle\_t pDcgmHandle**, **dcgmGroupEntityPair\_t**  
**entities**, **unsigned int entityCount**, **unsigned short**  
**fields**, **unsigned int fieldCount**, **unsigned int flags**,  
**dcgmFieldValue\_v2 values**)

**Parameters****pDcgmHandle**

IN: DCGM Handle

**entities**

IN: List of entities to get values for

**entityCount**

IN: Number of entries in entities[]

**fields**

IN: Field IDs to return data for. See the definitions in dcgm\_fields.h that start with DCGM\_FI\_.

**fieldCount**

IN: Number of field IDs in fields[] array.

**flags**

IN: Optional flags that affect how this request is processed. Pass **DCGM\_FV\_FLAG\_LIVE\_DATA** here to retrieve a live driver value rather than a cached value. See that flag's documentation for caveats.

**values**

OUT: Latest field values for the fields requested. This must be able to hold entityCount \* fieldCount field value records.

**Description**

Request the latest cached or live field value for a list of fields for a group of entities

Note: The returned entities are not guaranteed to be in any order. Reordering can occur internally in order to optimize calls to the NVIDIA driver.

## dcgmReturn\_t dcgmGetFieldSummary (dcgmHandle\_t pDcgmHandle, dcgmFieldSummaryRequest\_t \*request)

### Parameters

#### pDcgmHandle

IN: DCGM Handle

#### request

IN/OUT: a pointer to the struct detailing the request and containing the response

### Returns

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_FIELD\_UNSUPPORTED\_BY\_API if the field is not int64 or double type

### Description

Get a summary of the values for a field id over a period of time.

## 1.5. Process Statistics

Describes APIs to investigate statistics such as accounting, performance and errors during the lifetime of a GPU process

## dcgmReturn\_t dcgmWatchPidFields (dcgmHandle\_t pDcgmHandle, dcgmGpuGrp\_t groupId, long long updateFreq, double maxKeepAge, int maxKeepSamples)

### Parameters

#### pDcgmHandle

IN: DCGM Handle

#### groupId

IN: Group ID representing collection of one or more GPUs. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs.

#### updateFreq

IN: How often to update this field in usec

#### maxKeepAge

IN: How long to keep data for this field in seconds

**maxKeepSamples**

IN: Maximum number of samples to keep. 0=no limit

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid
- ▶ DCGM\_ST\_REQUIRES\_ROOT if the host engine is being run as non-root, and accounting mode could not be enabled (requires root). Run "nvidia-smi -am 1" as root on the node before starting DCGM to fix this.

**Description**

Request that DCGM start recording stats for fields that can be queried with [dcgmGetPidInfo\(\)](#).

Note that the first update of the field will not occur until the next field update cycle. To force a field update cycle, call [dcgmUpdateAllFields\(1\)](#).

## dcgmReturn\_t dcgmGetPidInfo (dcgmHandle\_t pDcgmHandle, dcgmGpuGrp\_t groupId, dcgmPidInfo\_t \*pidInfo)

**Parameters****pDcgmHandle**

IN: DCGM Handle

**groupId**

IN: Group ID representing collection of one or more GPUs. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs.

**pidInfo**

IN/OUT: Structure to return information about pid in. pidInfo->pid must be set to the pid in question. pidInfo->version should be set to dcgmPidInfo\_version.

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_NO\_DATA if the PID did not run on any GPU

**Description**

Get information about all GPUs while the provided pid was running

In order for this request to work, you must first call `dcgmWatchPidFields()` to make sure that DCGM is watching the appropriate field IDs that will be populated in `pidInfo`

## 1.6. Job Statistics

The client can invoke DCGM APIs to start and stop collecting the stats at the process boundaries (during prologue and epilogue). This will enable DCGM to monitor all the PIDs while the job is in progress, and provide a summary of active processes and resource usage during the window of interest.

`dcgmReturn_t dcgmWatchJobFields (dcgmHandle_t pDcgmHandle, dcgmGpuGrp_t groupId, long long updateFreq, double maxKeepAge, int maxKeepSamples)`

### Parameters

#### `pDcgmHandle`

IN: DCGM Handle

#### `groupId`

IN: Group ID representing collection of one or more GPUs. Look at `dcgmGroupCreate` for details on creating the group. Alternatively, pass in the group id as `DCGM_GROUP_ALL_GPUS` to perform operation on all the GPUs.

#### `updateFreq`

IN: How often to update this field in usec

#### `maxKeepAge`

IN: How long to keep data for this field in seconds

#### `maxKeepSamples`

IN: Maximum number of samples to keep. 0=no limit

### Returns

- ▶ `DCGM_ST_OK` if the call was successful
- ▶ `DCGM_ST_BADPARAM` if a parameter is invalid
- ▶ `DCGM_ST_REQUIRES_ROOT` if the host engine is being run as non-root, and accounting mode could not be enabled (requires root). Run "nvidia-smi -am 1" as root on the node before starting DCGM to fix this.

### Description

Request that DCGM start recording stats for fields that are queried with `dcgmJobGetStats()`

Note that the first update of the field will not occur until the next field update cycle. To force a field update cycle, call `dcgmUpdateAllFields(1)`.

## `dcgmReturn_t dcgmJobStartStats (dcgmHandle_t pDcgmHandle, dcgmGpuGrp_t groupId, char jobId)`

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **groupId**

IN: Group ID representing collection of one or more GPUs. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as `DCGM_GROUP_ALL_GPUS` to perform operation on all the GPUs.

#### **jobId**

IN: User provided string to represent the job

### Returns

- ▶ `DCGM_ST_OK` if the call was successful
- ▶ `DCGM_ST_BADPARAM` if a parameter is invalid
- ▶ `DCGM_ST_DUPLICATE_KEY` if the specified `jobId` is already in use

### Description

This API is used by the client to notify DCGM about the job to be started. Should be invoked as part of job prologue

## `dcgmReturn_t dcgmJobStopStats (dcgmHandle_t pDcgmHandle, char jobId)`

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **jobId**

IN: User provided string to represent the job

### Returns

- ▶ `DCGM_ST_OK` if the call was successful
- ▶ `DCGM_ST_BADPARAM` if a parameter is invalid
- ▶ `DCGM_ST_NO_DATA` if `jobId` is not a valid job identifier.

**Description**

This API is used by the clients to notify DCGM to stop collecting stats for the job represented by job id. Should be invoked as part of job epilogue. The job Id remains available to view the stats at any point but cannot be used to start a new job. You must call [dcgmWatchJobFields\(\)](#) before this call to enable watching of job

## dcgmReturn\_t dcgmJobGetStats (dcgmHandle\_t pDcgmHandle, char jobId, dcgmJobInfo\_t \*pJobInfo)

**Parameters****pDcgmHandle**

IN: DCGM Handle

**jobId**

IN: User provided string to represent the job

**pJobInfo**

IN/OUT: Structure to return information about the job. .version should be set to [dcgmJobInfo\\_version](#) before this call.

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid
- ▶ DCGM\_ST\_NO\_DATA if jobId is not a valid job identifier.
- ▶ DCGM\_ST\_VER\_MISMATCH if .version is not set or is invalid.

**Description**

Get stats for the job identified by DCGM generated job id. The stats can be retrieved at any point when the job is in process. If you want to reuse this jobId, call [dcgmJobRemove](#) after this call.

## dcgmReturn\_t dcgmJobRemove (dcgmHandle\_t pDcgmHandle, char jobId)

**Parameters****pDcgmHandle**

IN: DCGM Handle

**jobId**

IN: User provided string to represent the job



### Returns

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid
- ▶ DCGM\_ST\_NO\_DATA if jobId is not a valid job identifier.

### Description

This API tells DCGM to stop tracking the job given by jobId. After this call, you will no longer be able to call `dcgmJobGetStats()` on this jobId. However, you will be able to reuse jobId after this call.

## dcgmReturn\_t dcgmJobRemoveAll (dcgmHandle\_t pDcgmHandle)

### Parameters

#### pDcgmHandle

IN: DCGM Handle

### Returns

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid

### Description

This API tells DCGM to stop tracking all jobs. After this call, you will no longer be able to call `dcgmJobGetStats()` any jobs until you call `dcgmJobStartStats` again. You will be able to reuse any previously-used jobIds after this call.

## 1.7. Health Monitor

This chapter describes the methods that handle the GPU health monitor.

## dcgmReturn\_t dcgmHealthSet (dcgmHandle\_t pDcgmHandle, dcgmGpuGrp\_t groupId, dcgmHealthSystems\_t systems)

### Parameters

#### pDcgmHandle

IN: DCGM Handle

#### groupId

IN: Group ID representing collection of one or more entities. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs or DCGM\_GROUP\_ALL\_NVSWITCHES to perform operation on all the NvSwitches.

#### systems

IN: An enum representing systems that should be enabled for health checks logically OR'd together. Refer to [dcgmHealthSystems\\_t](#) for details.

### Returns

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid

### Description

Enable the DCGM health check system for the given systems defined in [dcgmHealthSystems\\_t](#)

## dcgmReturn\_t dcgmHealthSet\_v2 (dcgmHandle\_t pDcgmHandle, dcgmHealthSetParams\_v2 \*params[])

### Parameters

#### pDcgmHandle

IN: DCGM Handle

#### params

### Returns

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid

**Description**

Enable the DCGM health check system for the given systems defined in [dcmHealthSystems\\_t](#)

Since DCGM 2.0

**dcmReturn\_t dcmHealthGet (dcmHandle\_t pDcmHandle, dcmGpuGrp\_t groupId, dcmHealthSystems\_t \*systems)**

**Parameters****pDcmHandle**

IN: DCGM Handle

**groupId**

IN: Group ID representing collection of one or more entities. Look at [dcmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs or DCGM\_GROUP\_ALL\_NVSWITCHES to perform operation on all the NvSwitches.

**systems**

OUT: An integer representing the enabled systems for the given group Refer to [dcmHealthSystems\\_t](#) for details.

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid

**Description**

Retrieve the current state of the DCGM health check system

**dcmReturn\_t dcmHealthCheck (dcmHandle\_t pDcmHandle, dcmGpuGrp\_t groupId, dcmHealthResponse\_t \*results)**

**Parameters****pDcmHandle**

IN: DCGM Handle

**groupId**

IN: Group ID representing a collection of one or more entities. Refer to [dcgmGroupCreate](#) for details on creating a group

**results**

OUT: A reference to the `dcgmHealthResponse_t` structure to populate. `results->version` must be set to `dcgmHealthResponse_version`.

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid
- ▶ DCGM\_ST\_VER\_MISMATCH if `results->version` is not `dcgmHealthResponse_version`

**Description**

Check the configured watches for any errors/failures/warnings that have occurred since the last time this check was invoked. On the first call, stateful information about all of the enabled watches within a group is created but no error results are provided. On subsequent calls, any error information will be returned.

## 1.8. Policies

This chapter describes the methods that handle system policy management and violation settings. The APIs in Policies module can be broken down into following categories:

### Setup and Management

#### Manual Invocation

##### 1.8.1. Setup and Management

###### Policies

Describes APIs for setting up policies and registering callbacks to receive notification in case specific policy condition has been violated.

```
dcgmReturn_t dcgmPolicySet (dcgmHandle_t pDcgmHandle,
dcgmGpuGrp_t groupId, dcgmPolicy_t *policy, dcgmStatus_t
statusHandle)
```

### Parameters

#### pDcgmHandle

IN: DCGM Handle

#### groupId

IN: Group ID representing collection of one or more GPUs. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs.

#### policy

IN: A reference to [dcgmPolicy\\_t](#) that will be applied to all GPUs in the group.

#### statusHandle

IN/OUT: Resulting status for the operation. Pass it as NULL if the detailed error information is not needed. Refer to [dcgmStatusCreate](#) for details on creating a status handle.

### Returns

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if groupId or policy is invalid
- ▶ DCGM\_ST\_NOT\_SUPPORTED if any unsupported GPUs are part of the GPU group specified in groupId
- ▶ DCGM\_ST\_\* a different error has occurred and is stored in statusHandle. Refer to [dcgmReturn\\_t](#)

### Description

Set the current violation policy inside the policy manager. Given the conditions within the [dcgmPolicy\\_t](#) structure, if a violation has occurred, subsequent action(s) may be performed to either report or contain the failure.

```
dcgmReturn_t dcgmPolicyGet (dcgmHandle_t pDcgmHandle,
dcgmGpuGrp_t groupId, int count, dcgmPolicy_t *policy,
dcgmStatus_t statusHandle)
```

### Parameters

#### pDcgmHandle

IN: DCGM Handle

**groupId**

IN: Group ID representing collection of one or more GPUs. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs.

**count**

IN: The size of the policy array. This is the maximum number of policies that will be retrieved and ultimately should correspond to the number of GPUs specified in the group.

**policy**

OUT: A reference to [dcgmPolicy\\_t](#) that will be used as storage for the current policies applied to each GPU in the group.

**statusHandle**

IN/OUT: Resulting status for the operation. Pass it as NULL if the detailed error information for the operation is not needed. Refer to [dcgmStatusCreate](#) for details on creating a status handle.

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if groupId or policy is invalid
- ▶ DCGM\_ST\_\* a different error has occurred and is stored in statusHandle. Refer to [dcgmReturn\\_t](#)

**Description**

Get the current violation policy inside the policy manager. Given a groupId, a number of policy structures are retrieved.

[dcgmReturn\\_t](#) [dcgmPolicyRegister](#) ([dcgmHandle\\_t](#) pDcgmHandle, [dcgmGpuGrp\\_t](#) groupId, [dcgmPolicyCondition\\_t](#) condition, [fpRecvUpdates](#) beginCallback, [fpRecvUpdates](#) finishCallback)

**Parameters****pDcgmHandle**

IN: DCGM Handle

**groupId**

IN: Group ID representing collection of one or more GPUs. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs.

**condition**

IN: The set of conditions specified as an OR'd list (see [dcgmPolicyCondition\\_t](#)) for which to register a callback function

**beginCallback**

IN: A reference to a function that should be called should a violation occur. This function will be called prior to any actions specified by the policy are taken.

**finishCallback**

IN: A reference to a function that should be called should a violation occur. This function will be called after any action specified by the policy are completed.

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if groupId, condition, is invalid, beginCallback, or finishCallback is NULL
- ▶ DCGM\_ST\_NOT\_SUPPORTED if any unsupported GPUs are part of the GPU group specified in groupId

**Description**

Register a function to be called when a specific policy condition (see [dcgmPolicyCondition\\_t](#)) has been violated. This callback(s) will be called automatically when in DCGM\_OPERATION\_MODE\_AUTO mode and only after dcgmPolicyTrigger when in DCGM\_OPERATION\_MODE\_MANUAL mode. All callbacks are made within a separate thread.

**dcgmReturn\_t dcgmPolicyUnregister (dcgmHandle\_t pDcgmHandle, dcgmGpuGrp\_t groupId, dcgmPolicyCondition\_t condition)**

**Parameters****pDcgmHandle**

IN: DCGM Handle

**groupId**

IN: Group ID representing collection of one or more GPUs. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs.

**condition**

IN: The set of conditions specified as an OR'd list (see [dcgmPolicyCondition\\_t](#)) for which to unregister a callback function

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if groupId, condition, is invalid or callback is NULL

## Description

Unregister a function to be called for a specific policy condition (see [dcmgPolicyCondition\\_t](#)). This function will unregister all callbacks for a given condition and handle.

## 1.8.2. Manual Invocation

### Policies

Describes APIs which can be used to perform direct actions (e.g. Perform GPU Reset, Run Health Diagnostics) on a group of GPUs.

```
dcmgReturn_t dcmgActionValidate (dcmgHandle_t pDcmgHandle,
dcmgGpuGrp_t groupId, dcmgPolicyValidation_t validate,
dcmgDiagResponse_t *response)
```

### Parameters

#### pDcmgHandle

IN: DCGM Handle

#### groupId

IN: Group ID representing collection of one or more GPUs. Look at [dcmgGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs.

#### validate

IN: The validation to perform after the action.

#### response

OUT: Result of the validation process. Refer to [dcmgDiagResponse\\_t](#) for details.

### Returns

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_NOT\_SUPPORTED if running the specified validate is not supported. This is usually due to the Tesla recommended driver not being installed on the system.
- ▶ DCGM\_ST\_BADPARAM if groupId, validate, or statusHandle is invalid
- ▶ DCGM\_ST\_GENERIC\_ERROR an internal error has occurred
- ▶ DCGM\_ST\_GROUP\_INCOMPATIBLE if groupId refers to a group of non-homogeneous GPUs. This is currently not allowed.

## Description

Inform the action manager to perform a manual validation of a group of GPUs on the system



\*\*\*\*\* DEPRECATED \*\*\*\*\*

**dcgmReturn\_t dcgmActionValidate\_v2 (dcgmHandle\_t pDcgmHandle, dcgmRunDiag\_v7 \*drd, dcgmDiagResponse\_t \*response)**

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **drd**

IN: Contains the group id, test names, test parameters, struct version, and the validation that should be performed. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs.

#### **response**

OUT: Result of the validation process. Refer to [dcgmDiagResponse\\_t](#) for details.

### Returns

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_NOT\_SUPPORTED if running the specified validate is not supported. This is usually due to the Tesla recommended driver not being installed on the system.
- ▶ DCGM\_ST\_BADPARAM if groupId, validate, or statusHandle is invalid
- ▶ DCGM\_ST\_GENERIC\_ERROR an internal error has occurred
- ▶ DCGM\_ST\_GROUP\_INCOMPATIBLE if groupId refers to a group of non-homogeneous GPUs. This is currently not allowed.

### Description

Inform the action manager to perform a manual validation of a group of GPUs on the system

**dcgmReturn\_t dcgmRunDiagnostic (dcgmHandle\_t pDcgmHandle, dcgmGpuGrp\_t groupId, dcgmDiagnosticLevel\_t diagLevel, dcgmDiagResponse\_t \*diagResponse)**

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

**groupId**

IN: Group ID representing collection of one or more GPUs. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs.

**diagLevel**

IN: Diagnostic level to run

**diagResponse**

IN/OUT: Result of running the DCGM diagnostic. `.version` should be set to [dcgmDiagResponse\\_version](#) before this call.

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_NOT\_SUPPORTED if running the diagnostic is not supported. This is usually due to the Tesla recommended driver not being installed on the system.
- ▶ DCGM\_ST\_BADPARAM if a provided parameter is invalid or missing
- ▶ DCGM\_ST\_GENERIC\_ERROR an internal error has occurred
- ▶ DCGM\_ST\_GROUP\_INCOMPATIBLE if `groupId` refers to a group of non-homogeneous GPUs. This is currently not allowed.
- ▶ DCGM\_ST\_VER\_MISMATCH if `.version` is not set or is invalid.

**Description**

Run a diagnostic on a group of GPUs

## 1.9. Topology

`dcgmReturn_t dcgmGetDeviceTopology`  
(`dcgmHandle_t pDcgmHandle`, `unsigned int gpuId`,  
`dcgmDeviceTopology_t *pDcgmDeviceTopology`)

**Parameters****pDcgmHandle**

IN: DCGM Handle

**gpuId**

IN: GPU Id corresponding to which topology information should be fetched

**pDcgmDeviceTopology**

IN/OUT: Topology information corresponding to `gpuId`. `pDcgmDeviceTopology->version` must be set to `dcgmDeviceTopology_version` before this call.

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful.
- ▶ DCGM\_ST\_BADPARAM if gpuId or pDcgmDeviceTopology were not valid.
- ▶ DCGM\_ST\_VER\_MISMATCH if pDcgmDeviceTopology->version was not set to dcgmDeviceTopology\_version.

**Description**

Gets device topology corresponding to the gpuId.

**dcgmReturn\_t dcgmGetGroupTopology (dcgmHandle\_t pDcgmHandle, dcgmGpuGrp\_t groupId, dcgmGroupTopology\_t \*pDcgmGroupTopology)**

**Parameters****pDcgmHandle**

IN: DCGM Handle

**groupId**

IN: groupId corresponding to which topology information should be fetched

**pDcgmGroupTopology**

IN/OUT: Topology information corresponding to groupId. pDcgmgroupTopology->version must be set to dcgmGroupTopology\_version.

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful.
- ▶ DCGM\_ST\_BADPARAM if groupId or pDcgmGroupTopology were not valid.
- ▶ DCGM\_ST\_VER\_MISMATCH if pDcgmgroupTopology->version was not set to dcgmGroupTopology\_version.

**Description**

Gets group topology corresponding to the groupId.

## 1.10. Metadata

This chapter describes the methods that query for DCGM metadata.

## dcgmReturn\_t dcgmIntrospectToggleState (dcgmHandle\_t pDcgmHandle, dcgmIntrospectState\_t enabledState)

### Parameters

#### pDcgmHandle

IN: DCGM Handle

#### enabledState

IN: The state to set gathering of introspection data to

### Returns

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM enabledState is an invalid state for metadata gathering

### Description

Toggle the state of introspection metadata gathering in DCGM. Metadata gathering will increase the memory usage of DCGM so that it can store the metadata it gathers.

## dcgmReturn\_t dcgmIntrospectGetFieldsMemoryUsage (dcgmHandle\_t pDcgmHandle, dcgmIntrospectContext\_t \*context, dcgmIntrospectFullMemory\_t \*memoryInfo, int waitIfNoData)

### Parameters

#### pDcgmHandle

IN: DCGM Handle

#### context

IN: see [dcgmIntrospectContext\\_t](#). This identifies the level of fields to do introspection for (ex: all fields, field groups) context->version must be set to dcgmIntrospectContext\_version prior to this call.

#### memoryInfo

IN/OUT: see [dcgmIntrospectFullMemory\\_t](#). memoryInfo->version must be set to dcgmIntrospectFullMemory\_version prior to this call.

#### waitIfNoData

IN: if no metadata has been gathered, should this call block until data has been gathered (1), or should this call just return DCGM\_ST\_NO\_DATA (0).

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_NOT\_CONFIGURED if metadata gathering state is DCGM\_INTROSPECT\_STATE\_DISABLED
- ▶ DCGM\_ST\_NO\_DATA if waitIfNoData is false and metadata has not been gathered yet
- ▶ DCGM\_ST\_VER\_MISMATCH if context->version or memoryInfo->version is 0 or invalid.

**Description**

Get the current amount of memory used to store the given field collection.

**dcgmReturn\_t**  
**dcgmIntrospectGetHostengineMemoryUsage**  
 (dcgmHandle\_t pDcgmHandle, dcgmIntrospectMemory\_t \*memoryInfo, int waitIfNoData)

**Parameters****pDcgmHandle**

IN: DCGM Handle

**memoryInfo**

IN/OUT: see [dcgmIntrospectMemory\\_t](#). memoryInfo->version must be set to dcgmIntrospectMemory\_version prior to this call.

**waitIfNoData**

IN: if no metadata is gathered wait till this occurs (!0) or return DCGM\_ST\_NO\_DATA (0)

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_NOT\_CONFIGURED if metadata gathering state is DCGM\_INTROSPECT\_STATE\_DISABLED
- ▶ DCGM\_ST\_NO\_DATA if waitIfNoData is false and metadata has not been gathered yet
- ▶ DCGM\_ST\_VER\_MISMATCH if memoryInfo->version is 0 or invalid.

**Description**

Retrieve the total amount of memory that the hostengine process is currently using. This measurement represents both the resident set size (what is currently in RAM) and the swapped memory that belongs to the process.

**dcgmReturn\_t dcgmIntrospectGetFieldsExecTime**  
**(dcgmHandle\_t pDcgmHandle, dcgmIntrospectContext\_t**  
**\*context, dcgmIntrospectFullFieldsExecTime\_t**  
**\*execTime, int waitIfNoData)**

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **context**

IN: see [dcgmIntrospectContext\\_t](#). This identifies the level of fields to do introspection for (ex: all fields, field group ) context->version must be set to dcgmIntrospectContext\_version prior to this call.

#### **execTime**

IN/OUT: see [dcgmIntrospectFullFieldsExecTime\\_t](#). execTime->version must be set to dcgmIntrospectFullFieldsExecTime\_version prior to this call.

#### **waitIfNoData**

IN: if no metadata is gathered, wait until data has been gathered (1) or return DCGM\_ST\_NO\_DATA (0)

### Returns

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_NOT\_CONFIGURED if metadata gathering state is DCGM\_INTROSPECT\_STATE\_DISABLED
- ▶ DCGM\_ST\_NO\_DATA if waitIfNoData is false and metadata has not been gathered yet
- ▶ DCGM\_ST\_VER\_MISMATCH if context->version or execTime->version is 0 or invalid.

### Description

Get introspection info relating to execution time needed to update the fields identified by context.

**dcgmReturn\_t**  
**dcgmIntrospectGetHostengineCpuUtilization**

## `(dcgmHandle_t pDcgmHandle, dcgmIntrospectCpuUtil_t *cpuUtil, int waitIfNoData)`

### Parameters

#### `pDcgmHandle`

IN: DCGM Handle

#### `cpuUtil`

IN/OUT: see `dcgmIntrospectCpuUtil_t`. `cpuUtil->version` must be set to `dcgmIntrospectCpuUtil_version` prior to this call.

#### `waitIfNoData`

IN: if no metadata is gathered wait till this occurs (!0) or return `DCGM_ST_NO_DATA` (0)

### Returns

- ▶ `DCGM_ST_OK` if the call was successful
- ▶ `DCGM_ST_NOT_CONFIGURED` if metadata gathering state is `DCGM_INTROSPECT_STATE_DISABLED`
- ▶ `DCGM_ST_NO_DATA` if `waitIfNoData` is false and metadata has not been gathered yet
- ▶ `DCGM_ST_VER_MISMATCH` if `cpuUtil->version` or `execTime->version` is 0 or invalid.

### Description

Retrieve the CPU utilization of the DCGM hostengine process.

## `dcgmReturn_t dcgmIntrospectUpdateAll (dcgmHandle_t pDcgmHandle, int waitForUpdate)`

### Parameters

#### `pDcgmHandle`

IN: DCGM Handle

#### `waitForUpdate`

IN: Whether or not to wait for the update loop to complete before returning to the caller

### Returns

- ▶ `DCGM_ST_OK` if the call was successful
- ▶ `DCGM_ST_BADPARAM` if `waitForUpdate` is invalid

**Description**

This method is used to manually tell the the introspection module to update all DCGM introspection data. This is normally performed automatically on an interval of 1 second.

## 1.11. Topology

This chapter describes the methods that query for DCGM topology information.

**dcgmReturn\_t dcgmSelectGpusByTopology**  
(**dcgmHandle\_t pDcgmHandle**, **uint64\_t inputGpuIds**,  
**uint32\_t numGpus**, **uint64\_t \*outputGpuIds**, **uint64\_t**  
**hintFlags**)

**Parameters****pDcgmHandle**

IN: DCGM Handle

**inputGpuIds**

IN: a bitmask of which GPUs DCGM should consider. If some of the GPUs on the system are already in use, they shouldn't be included in the bitmask. 0 means that all of the GPUs in the system should be considered.

**numGpus**

IN: the number of GPUs that are desired from inputGpuIds. If this number is greater than the number of healthy GPUs in inputGpuIds, then less than numGpus gpu will be specified in outputGpuIds.

**outputGpuIds**

OUT: a bitmask of numGpus or fewer GPUs from inputGpuIds that represent the best placement available from inputGpuIds.

**hintFlags**

IN: a bitmask of DCGM\_TOPO\_HINT\_F\_ defines of hints that should be taken into account when assigning outputGpuIds.

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful

**Description**

Get the best group of gpus from the specified bitmask according to topological proximity: cpuAffinity, NUMA node, and NVLink.



## 1.12. Modules

This chapter describes the methods that query and configure DCGM modules.

### `dcgmReturn_t dcgmModuleBlacklist (dcgmHandle_t pDcgmHandle, dcgmModuleId_t moduleId)`

#### Parameters

##### **pDcgmHandle**

IN: DCGM Handle

##### **moduleId**

IN: ID of the module to blacklist. Use `dcgmModuleGetStatuses` to get a list of valid module IDs.

#### Returns

- ▶ `DCGM_ST_OK` if the module has been blacklisted.
- ▶ `DCGM_ST_IN_USE` if the module has already been loaded and cannot be blacklisted.
- ▶ `DCGM_ST_BADPARAM` if a parameter is missing or bad.

#### Description

Set a module to be blacklisted. This module will be prevented from being loaded if it hasn't been loaded already. Modules are lazy-loaded as they are used by DCGM APIs, so it's important to call this API soon after the host engine has been started. You can also pass `--blacklist-modules` to the `nv-hostengine` binary to make sure modules get blacklisted immediately after the host engine starts up.

### `dcgmReturn_t dcgmModuleGetStatuses (dcgmHandle_t pDcgmHandle, dcgmModuleGetStatuses_t *moduleStatuses)`

#### Parameters

##### **pDcgmHandle**

IN: DCGM Handle

##### **moduleStatuses**

OUT: Module statuses. `.version` should be set to `dcgmModuleStatuses_version` upon calling.

**Returns**

- ▶ DCGM\_ST\_OK if the request succeeds.
- ▶ DCGM\_ST\_BADPARAM if a parameter is missing or bad.

**Description**

Get the status of all of the DCGM modules.

## 1.13. Profiling

This chapter describes the methods that watch profiling fields from within DCGM.

**dcgmReturn\_t dcgmProfGetSupportedMetricGroups  
(dcgmHandle\_t pDcgmHandle,  
dcgmProfGetMetricGroups\_t \*metricGroups)**

**Parameters****pDcgmHandle**

IN: DCGM Handle

**metricGroups**

IN/OUT: Metric groups supported for metricGroups->groupId. metricGroups->version should be set to dcgmProfGetMetricGroups\_version upon calling.

**Returns**

- ▶ DCGM\_ST\_OK if the request succeeds.
- ▶ DCGM\_ST\_BADPARAM if a parameter is missing or bad.
- ▶ DCGM\_ST\_GROUP\_INCOMPATIBLE if metricGroups->groupId's GPUs are not identical GPUs.
- ▶ DCGM\_ST\_NOT\_SUPPORTED if profiling metrics are not supported for the given GPU group.

**Description**

Get all of the profiling metric groups for a given GPU group.

Profiling metrics are watched in groups of fields that are all watched together. For instance, if you want to watch DCGM\_FI\_PROF\_GR\_ENGINE\_ACTIVITY, this might also be in the same group as DCGM\_FI\_PROF\_SM\_EFFICIENCY. Watching this group would result in DCGM storing values for both of these metrics.

Some groups cannot be watched concurrently as others as they utilize the same hardware resource. For instance, you may not be able to watch `DCGM_FI_PROF_TENSOR_OP_UTIL` at the same time as `DCGM_FI_PROF_GR_ENGINE_ACTIVITY` on your hardware. At the same time, you may be able to watch `DCGM_FI_PROF_TENSOR_OP_UTIL` at the same time as `DCGM_FI_PROF_NVLINK_TX_DATA`.

Metrics that can be watched concurrently will have different `.majorId` fields in their `dcmProfMetricGroupInfo_t`

See [dcmGroupCreate](#) for details on creating a GPU group See [dcmProfWatchFields](#) to actually watch a metric group

## `dcmReturn_t dcmProfWatchFields (dcmHandle_t pDcmHandle, dcmProfWatchFields_t *watchFields)`

### Parameters

#### `pDcmHandle`

IN: DCGM Handle

#### `watchFields`

IN: Details of which metric groups to watch for which GPUs. See [dcmProfWatchFields\\_v1](#) for details of what should be put in each struct member. `watchFields->version` should be set to `dcmProfWatchFields_version` upon calling.

### Returns

- ▶ `DCGM_ST_OK` if the call was successful
- ▶ `DCGM_ST_BADPARAM` if a parameter is invalid
- ▶ `DCGM_ST_NOT_SUPPORTED` if profiling metric group `metricGroupTag` is not supported for the given GPU group.
- ▶ `DCGM_ST_GROUP_INCOMPATIBLE` if `groupId`'s GPUs are not identical GPUs. Profiling metrics are only support for homogenous groups of GPUs.
- ▶ `DCGM_ST_PROFILING_MULTI_PASS` if any of the metric groups could not be watched concurrently due to requiring the hardware to gather them with multiple passes

### Description

Request that DCGM start recording updates for a given list of profiling field IDs.

Once metrics have been watched by this API, any of the normal DCGM field-value retrieval APIs can be used on the underlying fieldIds of this metric group. See [dcmGetLatestValues\\_v2](#), [dcmGetLatestValuesForFields](#), [dcmEntityGetLatestValues](#), and [dcmEntitiesGetLatestValues](#).

## dcgmReturn\_t dcgmProfUnwatchFields (dcgmHandle\_t pDcgmHandle, dcgmProfUnwatchFields\_t \*unwatchFields)

### Parameters

#### pDcgmHandle

IN: DCGM Handle

#### unwatchFields

IN: Details of which metric groups to unwatch for which GPUs. See [dcgmProfUnwatchFields\\_v1](#) for details of what should be put in each struct member. unwatchFields->version should be set to dcgmProfUnwatchFields\_version upon calling.

### Returns

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid

### Description

Request that DCGM stop recording updates for all profiling field IDs for all GPUs

## dcgmReturn\_t dcgmProfPause (dcgmHandle\_t pDcgmHandle)

### Parameters

#### pDcgmHandle

IN: DCGM Handle

### Returns

- ▶ DCGM\_ST\_OK If the call was successful.
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid.

### Description

Pause profiling activities in DCGM. This should be used when you are monitoring profiling fields from DCGM but want to be able to still run developer tools like nvprof, nsight systems, and nsight compute. Profiling fields start with DCGM\_PROF\_ and are in the field ID range 1001-1012.

Call this API before you launch one of those tools and `dcgmProfResume()` after the tool has completed.

DCGM will save BLANK values while profiling is paused.

Calling this while profiling activities are already paused is fine and will be treated as a no-op.

## dcgmReturn\_t dcgmProfResume (dcgmHandle\_t pDcgmHandle)

### Parameters

#### pDcgmHandle

IN: DCGM Handle

### Returns

- ▶ DCGM\_ST\_OK If the call was successful.
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid.

### Description

Resume profiling activities in DCGM that were previously paused with `dcgmProfPause()`.

Call this API after you have completed running other NVIDIA developer tools to reenable DCGM profiling metrics.

DCGM will save BLANK values while profiling is paused.

Calling this while profiling activities have already been resumed is fine and will be treated as a no-op.

## 1.14. Enums and Macros

### enum dcgmOperationMode\_t

Operation mode for DCGM

DCGM can run in auto-mode where it runs additional threads in the background to collect any metrics of interest and auto manages any operations needed for policy management.

DCGM can also operate in manual-mode where it's execution is controlled by the user. In this mode, the user has to periodically call APIs such as [dcgmPolicyTrigger](#) and [dcgmUpdateAllFields](#) which tells DCGM to wake up and perform data collection and operations needed for policy management.

### Values

**DCGM\_OPERATION\_MODE\_AUTO = 1**  
**DCGM\_OPERATION\_MODE\_MANUAL = 2**

## enum dcgmOrder\_t

When more than one value is returned from a query, which order should it be returned in?

### Values

**DCGM\_ORDER\_ASCENDING = 1**  
 Data with earliest (lowest) timestamps returned first.  
**DCGM\_ORDER\_DESCENDING = 2**  
 Data with latest (highest) timestamps returned first.

## enum dcgmReturn\_t

Return values for DCGM API calls.

### Values

**DCGM\_ST\_OK = 0**  
 Success.  
**DCGM\_ST\_BADPARAM = -1**  
 A bad parameter was passed to a function.  
**DCGM\_ST\_GENERIC\_ERROR = -3**  
 A generic, unspecified error.  
**DCGM\_ST\_MEMORY = -4**  
 An out of memory error occurred.  
**DCGM\_ST\_NOT\_CONFIGURED = -5**  
 Setting not configured.  
**DCGM\_ST\_NOT\_SUPPORTED = -6**  
 Feature not supported.  
**DCGM\_ST\_INIT\_ERROR = -7**  
 DCGM Init error.  
**DCGM\_ST\_NVML\_ERROR = -8**  
 When NVML returns error.  
**DCGM\_ST\_PENDING = -9**  
 Object is in pending state of something else.

**DCGM\_ST\_UNINITIALIZED = -10**

Object is in undefined state.

**DCGM\_ST\_TIMEOUT = -11**

Requested operation timed out.

**DCGM\_ST\_VER\_MISMATCH = -12**

Version mismatch between received and understood API.

**DCGM\_ST\_UNKNOWN\_FIELD = -13**

Unknown field id.

**DCGM\_ST\_NO\_DATA = -14**

No data is available.

**DCGM\_ST\_STALE\_DATA = -15**

Data is considered stale.

**DCGM\_ST\_NOT\_WATCHED = -16**

The given field id is not being updated by the cache manager.

**DCGM\_ST\_NO\_PERMISSION = -17**

Do not have permission to perform the desired action.

**DCGM\_ST\_GPU\_IS\_LOST = -18**

GPU is no longer reachable.

**DCGM\_ST\_RESET\_REQUIRED = -19**

GPU requires a reset.

**DCGM\_ST\_FUNCTION\_NOT\_FOUND = -20**

The function that was requested was not found (bindings only error).

**DCGM\_ST\_CONNECTION\_NOT\_VALID = -21**

The connection to the host engine is not valid any longer.

**DCGM\_ST\_GPU\_NOT\_SUPPORTED = -22**

This GPU is not supported by DCGM.

**DCGM\_ST\_GROUP\_INCOMPATIBLE = -23**

The GPUs of the provided group are not compatible with each other for the requested operation

**DCGM\_ST\_MAX\_LIMIT = -24**

Max limit reached for the object.

**DCGM\_ST\_LIBRARY\_NOT\_FOUND = -25**

DCGM library could not be found.

**DCGM\_ST\_DUPLICATE\_KEY = -26**

Duplicate key passed to a function.

**DCGM\_ST\_GPU\_IN\_SYNC\_BOOST\_GROUP = -27**

GPU is already a part of a sync boost group.

**DCGM\_ST\_GPU\_NOT\_IN\_SYNC\_BOOST\_GROUP = -28**

GPU is not a part of a sync boost group.

**DCGM\_ST\_REQUIRES\_ROOT = -29**

This operation cannot be performed when the host engine is running as non-root.

**DCGM\_ST\_NVVS\_ERROR = -30**

DCGM GPU Diagnostic was successfully executed, but reported an error.

**DCGM\_ST\_INSUFFICIENT\_SIZE = -31**

An input argument is not large enough.

**DCGM\_ST\_FIELD\_UNSUPPORTED\_BY\_API = -32**

The given field ID is not supported by the API being called.

**DCGM\_ST\_MODULE\_NOT\_LOADED = -33**

This request is serviced by a module of DCGM that is not currently loaded.

**DCGM\_ST\_IN\_USE = -34**

The requested operation could not be completed because the affected resource is in use

**DCGM\_ST\_GROUP\_IS\_EMPTY = -35**

This group is empty and the requested operation is not valid on an empty group.

**DCGM\_ST\_PROFILING\_NOT\_SUPPORTED = -36**

Profiling is not supported for this group of GPUs or GPU.

**DCGM\_ST\_PROFILING\_LIBRARY\_ERROR = -37**

The third-party Profiling module returned an unrecoverable error.

**DCGM\_ST\_PROFILING\_MULTI\_PASS = -38**

The requested profiling metrics cannot be collected in a single pass.

**DCGM\_ST\_DIAG\_ALREADY\_RUNNING = -39**

A diag instance is already running, cannot run a new diag until the current one finishes.

**DCGM\_ST\_DIAG\_BAD\_JSON = -40**

The DCGM GPU Diagnostic returned JSON that cannot be parsed.

**DCGM\_ST\_DIAG\_BAD\_LAUNCH = -41**

Error while launching the DCGM GPU Diagnostic.

**DCGM\_ST\_DIAG\_VARIANCE = -42**

There is too much variance while training the diagnostic.

**DCGM\_ST\_DIAG\_THRESHOLD\_EXCEEDED = -43**

A field value met or exceeded the error threshold.

**DCGM\_ST\_INSUFFICIENT\_DRIVER\_VERSION = -44**

The installed driver version is insufficient for this API.

**DCGM\_ST\_INSTANCE\_NOT\_FOUND = -45**

The specified GPU instance does not exist.

**DCGM\_ST\_COMPUTE\_INSTANCE\_NOT\_FOUND = -46**

The specified GPU compute instance does not exist.

**DCGM\_ST\_CHILD\_NOT\_KILLED = -47**

Couldn't kill a child process within the retries.

**DCGM\_ST\_3RD\_PARTY\_LIBRARY\_ERROR = -48**

Detected an error in a 3rd-party library.

**DCGM\_ST\_INSUFFICIENT\_RESOURCES = -49**

Not enough resources available.

**DCGM\_ST\_PLUGIN\_EXCEPTION = -50**

Exception thrown from a diagnostic plugin.



**DCGM\_ST\_NVVS\_ISOLATE\_ERROR = -51**

The diagnostic returned an error that indicates the need for isolation.

## enum dcgmGroupType\_t

Type of GPU groups

### Values

**DCGM\_GROUP\_DEFAULT = 0**

All the GPUs on the node are added to the group.

**DCGM\_GROUP\_EMPTY = 1**

Creates an empty group.

**DCGM\_GROUP\_DEFAULT\_NVSWITCHES = 2**

All NvSwitches of the node are added to the group.

**DCGM\_GROUP\_DEFAULT\_INSTANCES = 3**

All GPU instances of the node are added to the group.

**DCGM\_GROUP\_DEFAULT\_COMPUTE\_INSTANCES = 4**

All compute instances of the node are added to the group.

**DCGM\_GROUP\_DEFAULT\_EVERYTHING = 5**

All entities are added to this default group.

## enum dcgmChipArchitecture\_t

Simplified chip architecture. Note that these are made to match `nvmlChipArchitecture_t` and thus do not start at 0.

### Values

**DCGM\_CHIP\_ARCH\_OLDER = 1**

All GPUs older than Kepler.

**DCGM\_CHIP\_ARCH\_KEPLER = 2**

All Kepler-architecture parts.

**DCGM\_CHIP\_ARCH\_MAXWELL = 3**

All Maxwell-architecture parts.

**DCGM\_CHIP\_ARCH\_PASCAL = 4**

All Pascal-architecture parts.

**DCGM\_CHIP\_ARCH\_VOLTA = 5**

All Volta-architecture parts.

**DCGM\_CHIP\_ARCH\_TURING = 6**

All Turing-architecture parts.

**DCGM\_CHIP\_ARCH\_AMPERE = 7**

All Ampere-architecture parts.

**DCGM\_CHIP\_ARCH\_COUNT**

Keep this second to last, exclude unknown.

**DCGM\_CHIP\_ARCH\_UNKNOWN = 0xffffffff**

Anything else, presumably something newer.

## enum dcgmConfigType\_t

Represents the type of configuration to be fetched from the GPUs

### Values

**DCGM\_CONFIG\_TARGET\_STATE = 0**

The target configuration values to be applied.

**DCGM\_CONFIG\_CURRENT\_STATE = 1**

The current configuration state.

## enum dcgmConfigPowerLimitType\_t

Represents the power cap for each member of the group.

### Values

**DCGM\_CONFIG\_POWER\_CAP\_INDIVIDUAL = 0**

Represents the power cap to be applied for each member of the group.

**DCGM\_CONFIG\_POWER\_BUDGET\_GROUP = 1**

Represents the power budget for the entire group.

```
#define MAKE_DCGM_VERSION (unsigned int)  
(sizeof(typeName) | ((unsigned long)(ver) << 24U))
```

Creates a unique version number for each struct

```
#define DCGM_INT32_BLANK 0x7fffffff0
```

Represents value of the field which can be returned by Host Engine in case the operation is not successful Base value for 32 bits integer blank. can be used as an unspecified blank

```
#define DCGM_INT64_BLANK 0x7fffffffffffffff0
```

Base value for 64 bits integer blank. can be used as an unspecified blank

```
#define DCGM_FP64_BLANK 140737488355328.0
```

Base value for double blank.  $2^{47}$ . FP 64 has 52 bits of mantissa, so 47 bits can still increment by 1 and represent each value from 0-15

```
#define DCGM_STR_BLANK "<<<NULL>>>"
```

Base value for string blank.

```
#define DCGM_INT32_NOT_FOUND (DCGM_INT32_BLANK + 1)
```

Represents an error where INT32 data was not found

```
#define DCGM_INT64_NOT_FOUND (DCGM_INT64_BLANK + 1)
```

Represents an error where INT64 data was not found

```
#define DCGM_FP64_NOT_FOUND (DCGM_FP64_BLANK + 1.0)
```

Represents an error where FP64 data was not found

```
#define DCGM_STR_NOT_FOUND "<<<NOT_FOUND>>>"
```

Represents an error where STR data was not found

```
#define DCGM_INT32_NOT_SUPPORTED  
(DCGM_INT32_BLANK + 2)
```

Represents an error where fetching the INT32 value is not supported

```
#define DCGM_INT64_NOT_SUPPORTED  
(DCGM_INT64_BLANK + 2)
```

Represents an error where fetching the INT64 value is not supported

```
#define DCGM_FP64_NOT_SUPPORTED  
(DCGM_FP64_BLANK + 2.0)
```

Represents an error where fetching the FP64 value is not supported

```
#define DCGM_STR_NOT_SUPPORTED  
<<<NOT_SUPPORTED>>>"
```

Represents an error where fetching the STR value is not supported

```
#define DCGM_INT32_NOT_PERMISSIONED
(DCGM_INT32_BLANK + 3)
```

Represents and error where fetching the INT32 value is not allowed with our current credentials

```
#define DCGM_INT64_NOT_PERMISSIONED
(DCGM_INT64_BLANK + 3)
```

Represents and error where fetching the INT64 value is not allowed with our current credentials

```
#define DCGM_FP64_NOT_PERMISSIONED
(DCGM_FP64_BLANK + 3.0)
```

Represents and error where fetching the FP64 value is not allowed with our current credentials

```
#define DCGM_STR_NOT_PERMISSIONED
"<<<NOT_PERM>>>"
```

Represents and error where fetching the STR value is not allowed with our current credentials

```
#define DCGM_INT32_IS_BLANK (((val) >=
DCGM_INT32_BLANK) ? 1 : 0)
```

Macro to check if a INT32 value is blank or not

```
#define DCGM_INT64_IS_BLANK (((val) >=
DCGM_INT64_BLANK) ? 1 : 0)
```

Macro to check if a INT64 value is blank or not

```
#define DCGM_FP64_IS_BLANK (((val) >=
DCGM_FP64_BLANK ? 1 : 0))
```

Macro to check if a FP64 value is blank or not

```
#define DCGM_STR_IS_BLANK (val == strstr(val, "<<<")  
&& strstr(val, ">>>"))
```

Macro to check if a STR value is blank or not Works on (char \*). Looks for <<< at first position and >>> inside string

```
#define DCGM_MAX_NUM_DEVICES 32
```

Max number of GPUs supported by DCGM

```
#define DCGM_NVLINK_MAX_LINKS_PER_GPU 12
```

Number of NvLink links per GPU supported by DCGM This is 12 for Ampere, 6 for Volta, and 4 for Pascal

```
#define DCGM_NVLINK_MAX_LINKS_PER_GPU_LEGACY1 6
```

Maximum NvLink links pre-Ampere

```
#define DCGM_MAX_NUM_SWITCHES 12
```

Max number of NvSwitches supported by DCGM

```
#define DCGM_NVLINK_MAX_LINKS_PER_NVSWITCH 36
```

Number of NvLink links per NvSwitch supported by DCGM

```
#define DCGM_MAX_VGPU_INSTANCES_PER_PGPU 32
```

Maximum number of vGPU instances per physical GPU

```
#define DCGM_MAX_STR_LENGTH 256
```

Max length of the DCGM string field

```
#define DCGM_MAX_CLOCKS 256
```

Max number of clocks supported for a device

```
#define DCGM_MAX_NUM_GROUPS 64
```

Max limit on the number of groups supported by DCGM

```
#define DCGM_MAX_FBC_SESSIONS 256
```

Max number of active FBC sessions

```
#define DCGM_VGPU_NAME_BUFFER_SIZE 64
```

Represents the size of a buffer that holds a vGPU type Name or vGPU class type or name of process running on vGPU instance.

```
#define DCGM_GRID_LICENSE_BUFFER_SIZE 128
```

Represents the size of a buffer that holds a vGPU license string

```
#define DCGM_CONFIG_COMPUTEMODE_DEFAULT 0
```

Default compute mode -- multiple contexts per device

```
#define DCGM_CONFIG_COMPUTEMODE_PROHIBITED 1
```

Compute-prohibited mode -- no contexts per device

```
#define
```

```
DCGM_CONFIG_COMPUTEMODE_EXCLUSIVE_PROCESS 2
```

Compute-exclusive-process mode -- only one context per device, usable from multiple threads at a time

```
#define DCGM_HE_PORT_NUMBER 5555
```

Default Port Number for DCGM Host Engine

```
#define DCGM_GROUP_ALL_GPUS 0x7fffffff
```

Identifies for special DCGM groups

```
#define DCGM_GROUP_MAX_ENTITIES 64
```

Maximum number of entities per entity group

## 1.16. Field Types

Field Types are a single byte.

```
#define DCGM_FT_BINARY 'b'
```

Blob of binary data representing a structure

```
#define DCGM_FT_DOUBLE 'd'
```

8-byte double precision

```
#define DCGM_FT_INT64 'i'
```

8-byte signed integer

```
#define DCGM_FT_STRING 's'
```

Null-terminated ASCII Character string

```
#define DCGM_FT_TIMESTAMP 't'
```

8-byte signed integer usec since 1970

## 1.17. Field Scope

Represents field association with entity scope or global scope.

```
#define DCGM_FS_GLOBAL 0
```

Field is global (ex: driver version)

```
#define DCGM_FS_ENTITY 1
```

Field is associated with an entity (GPU, VGPU...etc)

```
#define DCGM_FS_DEVICE DCGM_FS_ENTITY
```

Field is associated with a device. Deprecated. Use DCGM\_FS\_ENTITY

## 1.18. Field Constants

Constants that represent contents of individual field values.

## enum dcgmGpuVirtualizationMode\_t

GPU virtualization mode types for DCGM\_FI\_DEV\_VIRTUAL\_MODE

### Values

**DCGM\_GPU\_VIRTUALIZATION\_MODE\_NONE = 0**

Represents Bare Metal GPU.

**DCGM\_GPU\_VIRTUALIZATION\_MODE\_PASSTHROUGH = 1**

Device is associated with GPU-Passthrough.

**DCGM\_GPU\_VIRTUALIZATION\_MODE\_VGPU = 2**

Device is associated with vGPU inside virtual machine.

**DCGM\_GPU\_VIRTUALIZATION\_MODE\_HOST\_VGPU = 3**

Device is associated with VGX hypervisor in vGPU mode.

**DCGM\_GPU\_VIRTUALIZATION\_MODE\_HOST\_VSGA = 4**

Device is associated with VGX hypervisor in vSGA mode.

```
#define DCGM_CUDA_COMPUTE_CAPABILITY_MAJOR
((uint64_t)(x)&0xFFFF0000)
```

DCGM\_FI\_DEV\_CUDA\_COMPUTE\_CAPABILITY is 16 bits of major version followed by 16 bits of the minor version. These macros separate the two.

```
#define DCGM_CLOCKS_THROTTLE_REASON_GPU_IDLE
0x00000000000000000000000000000001LL
```

DCGM\_FI\_DEV\_CLOCK\_THROTTLE\_REASONS is a bitmap of why the clock is throttled. These macros are masks for relevant throttling, and are a 1:1 map to the NVML reasons documented in nvml.h. The notes for the header are copied below: Nothing is running on the GPU and the clocks are dropping to Idle state



This limiter may be removed in a later release

```
#define
DCGM_CLOCKS_THROTTLE_REASON_CLOCKS_SETTING
0x00000000000000000000000000000002LL
```

GPU clocks are limited by current setting of applications clocks



```
#define
DCGM_CLOCKS_THROTTLE_REASON_SW_POWER_CAP
0x0000000000000004LL
```

SW Power Scaling algorithm is reducing the clocks below requested clocks

```
#define
DCGM_CLOCKS_THROTTLE_REASON_HW_SLOWDOWN
0x0000000000000008LL
```

HW Slowdown (reducing the core clocks by a factor of 2 or more) is engaged

This is an indicator of:

- ▶ temperature being too high
- ▶ External Power Brake Assertion is triggered (e.g. by the system power supply)
- ▶ Power draw is too high and Fast Trigger protection is reducing the clocks
- ▶ May be also reported during PState or clock change
- ▶ This behavior may be removed in a later release.

```
#define
DCGM_CLOCKS_THROTTLE_REASON_SYNC_BOOST
0x0000000000000010LL
```

Sync Boost

This GPU has been added to a Sync boost group with nvidia-smi or DCGM in order to maximize performance per watt. All GPUs in the sync boost group will boost to the minimum possible clocks across the entire group. Look at the throttle reasons for other GPUs in the system to see why those GPUs are holding this one at lower clocks.

```
#define
DCGM_CLOCKS_THROTTLE_REASON_SW_THERMAL
0x0000000000000020LL
```

SW Thermal Slowdown

This is an indicator of one or more of the following:

- ▶ Current GPU temperature above the GPU Max Operating Temperature
- ▶ Current memory temperature above the Memory Max Operating Temperature

```
#define
DCGM_CLOCKS_THROTTLE_REASON_HW_THERMAL
0x0000000000000040LL
```

HW Thermal Slowdown (reducing the core clocks by a factor of 2 or more) is engaged

This is an indicator of:

- ▶ temperature being too high

```
#define
DCGM_CLOCKS_THROTTLE_REASON_HW_POWER_BRAKE
0x0000000000000080LL
```

HW Power Brake Slowdown (reducing the core clocks by a factor of 2 or more) is engaged

This is an indicator of:

- ▶ External Power Brake Assertion being triggered (e.g. by the system power supply)

```
#define
DCGM_CLOCKS_THROTTLE_REASON_DISPLAY_CLOCKS
0x0000000000000100LL
```

GPU clocks are limited by current setting of Display clocks

## 1.19. Field Entity

Represents field association with a particular entity

```
enum dcgm_field_entity_group_t
```

Enum of possible field entity groups

### Values

```
DCGM_FE_NONE = 0
```

Field is not associated with an entity. Field scope should be DCGM\_FS\_GLOBAL

```
DCGM_FE_GPU
```

Field is associated with a GPU entity

```
DCGM_FE_VGPU
```

Field is associated with a VGPU entity

**DCGM\_FE\_SWITCH**

Field is associated with a Switch entity

**DCGM\_FE\_GPU\_I**

Field is associated with a GPU Instance entity

**DCGM\_FE\_GPU\_CI**

Field is associated with a GPU Compute Instance entity

**DCGM\_FE\_COUNT**

Number of elements in this enumeration. Keep this entry last

**typedef unsigned int dcgm\_field\_eid\_t**

Represents an identifier for an entity within a field entity. For instance, this is the `gpuId` for `DCGM_FE_GPU`.

## 1.20. Field Identifiers

Field Identifiers

### DcgmFieldGetById (unsigned short fieldId)

**Parameters****fieldId**

IN: One of the field IDs (`DCGM_FI_?`)

**Returns**

0 On Failure >0 Pointer to field metadata structure if found.

**Description**

Get a pointer to the metadata for a field by its field ID. See `DCGM_FI_?` for a list of field IDs.

### DcgmFieldGetByTag (char \*tag)

**Parameters****tag**

IN: Tag for the field of interest

**Returns**

0 On failure or not found >0 Pointer to field metadata structure if found

**Description**

Get a pointer to the metadata for a field by its field tag.

## DcgmFieldsInit (void)

**Returns**

0 On success <0 On error

**Description**

Initialize the DcgmFields module. Call this once from inside your program

## DcgmFieldsTerm (void)

**Returns**

0 On success <0 On error

**Description**

Terminates the DcgmFields module. Call this once from inside your program

## const char \*DcgmFieldsGetEntityGroupString (dcgm\_field\_entity\_group\_t entityGroupId)

**Returns**

- ▶ Pointer to a string like GPU/NvSwitch..etc
- ▶ Null on error

**Description**

Get the string version of a entityGroupId

## #define DCGM\_FI\_UNKNOWN 0

NULL field

## #define DCGM\_FI\_DRIVER\_VERSION 1

Driver Version

```
#define DCGM_FI_DEV_COUNT 4
```

Number of Devices on the node

```
#define DCGM_FI_CUDA_DRIVER_VERSION 5
```

Cuda Driver Version Retrieves a number with the major value in the thousands place and the minor value in the hundreds place. CUDA 11.1 = 11100

```
#define DCGM_FI_DEV_NAME 50
```

Name of the GPU device

```
#define DCGM_FI_DEV_BRAND 51
```

Device Brand

```
#define DCGM_FI_DEV_NVML_INDEX 52
```

NVML index of this GPU

```
#define DCGM_FI_DEV_SERIAL 53
```

Device Serial Number

```
#define DCGM_FI_DEV_UUID 54
```

UUID corresponding to the device

```
#define DCGM_FI_DEV_MINOR_NUMBER 55
```

Device node minor number `/dev/nvidia#`

```
#define DCGM_FI_DEV_OEM_INFOROM_VER 56
```

OEM inforom version

```
#define DCGM_FI_DEV_PCI_BUSID 57
```

PCI attributes for the device

```
#define DCGM_FI_DEV_PCI_COMBINED_ID 58
```

The combined 16-bit device id and 16-bit vendor id

```
#define DCGM_FI_DEV_PCI_SUBSYS_ID 59
```

The 32-bit Sub System Device ID

```
#define DCGM_FI_GPU_TOPOLOGY_PCI 60
```

Topology of all GPUs on the system via PCI (static)

```
#define DCGM_FI_GPU_TOPOLOGY_NVLINK 61
```

Topology of all GPUs on the system via NVLINK (static)

```
#define DCGM_FI_GPU_TOPOLOGY_AFFINITY 62
```

Affinity of all GPUs on the system (static)

```
#define DCGM_FI_DEV_CUDA_COMPUTE_CAPABILITY 63
```

Cuda compute capability for the device. The major version is the upper 32 bits and the minor version is the lower 32 bits.

```
#define DCGM_FI_DEV_COMPUTE_MODE 65
```

Compute mode for the device

```
#define DCGM_FI_DEV_PERSISTENCE_MODE 66
```

Persistence mode for the device Boolean: 0 is disabled, 1 is enabled

```
#define DCGM_FI_DEV_MIG_MODE 67
```

MIG mode for the device Boolean: 0 is disabled, 1 is enabled

```
#define DCGM_FI_DEV_CUDA_VISIBLE_DEVICES_STR 68
```

The string that CUDA\_VISIBLE\_DEVICES should be set to for this entity (including MIG)

```
#define DCGM_FI_DEV_MIG_MAX_SLICES 69
```

The maximum number of MIG slices supported by this GPU

```
#define DCGM_FI_DEV_CPU_AFFINITY_0 70
```

Device CPU affinity. part 1/8 = cpus 0 - 63

**#define DCGM\_FI\_DEV\_CPU\_AFFINITY\_1 71**

Device CPU affinity. part 1/8 = cpus 64 - 127

**#define DCGM\_FI\_DEV\_CPU\_AFFINITY\_2 72**

Device CPU affinity. part 2/8 = cpus 128 - 191

**#define DCGM\_FI\_DEV\_CPU\_AFFINITY\_3 73**

Device CPU affinity. part 3/8 = cpus 192 - 255

**#define DCGM\_FI\_DEV\_ECC\_INFOROM\_VER 80**

ECC inforom version

**#define DCGM\_FI\_DEV\_POWER\_INFOROM\_VER 81**

Power management object inforom version

**#define DCGM\_FI\_DEV\_INFOROM\_IMAGE\_VER 82**

Inforom image version

**#define DCGM\_FI\_DEV\_INFOROM\_CONFIG\_CHECK 83**

Inforom configuration checksum

**#define DCGM\_FI\_DEV\_INFOROM\_CONFIG\_VALID 84**

Reads the infoROM from the flash and verifies the checksums

**#define DCGM\_FI\_DEV\_VBIOS\_VERSION 85**

VBIOS version of the device

**#define DCGM\_FI\_DEV\_BAR1\_TOTAL 90**

Total BAR1 of the GPU in MB

**#define DCGM\_FI\_SYNC\_BOOST 91**

Deprecated - Sync boost settings on the node

```
#define DCGM_FI_DEV_BAR1_USED 92
```

Used BAR1 of the GPU in MB

```
#define DCGM_FI_DEV_BAR1_FREE 93
```

Free BAR1 of the GPU in MB

```
#define DCGM_FI_DEV_SM_CLOCK 100
```

SM clock for the device

```
#define DCGM_FI_DEV_MEM_CLOCK 101
```

Memory clock for the device

```
#define DCGM_FI_DEV_VIDEO_CLOCK 102
```

Video encoder/decoder clock for the device

```
#define DCGM_FI_DEV_APP_SM_CLOCK 110
```

SM Application clocks

```
#define DCGM_FI_DEV_APP_MEM_CLOCK 111
```

Memory Application clocks

```
#define DCGM_FI_DEV_CLOCK_THROTTLE_REASONS 112
```

Current clock throttle reasons (bitmask of DCGM\_CLOCKS\_THROTTLE\_REASON\_\*)

```
#define DCGM_FI_DEV_MAX_SM_CLOCK 113
```

Maximum supported SM clock for the device

```
#define DCGM_FI_DEV_MAX_MEM_CLOCK 114
```

Maximum supported Memory clock for the device

```
#define DCGM_FI_DEV_MAX_VIDEO_CLOCK 115
```

Maximum supported Video encoder/decoder clock for the device



```
#define DCGM_FI_DEV_AUTOBOOST 120
```

Auto-boost for the device (1 = enabled. 0 = disabled)

```
#define DCGM_FI_DEV_SUPPORTED_CLOCKS 130
```

Supported clocks for the device

```
#define DCGM_FI_DEV_MEMORY_TEMP 140
```

Memory temperature for the device

```
#define DCGM_FI_DEV_GPU_TEMP 150
```

Current temperature readings for the device, in degrees C

```
#define DCGM_FI_DEV_MEM_MAX_OP_TEMP 151
```

Maximum operating temperature for the memory of this GPU

```
#define DCGM_FI_DEV_GPU_MAX_OP_TEMP 152
```

Maximum operating temperature for this GPU

```
#define DCGM_FI_DEV_POWER_USAGE 155
```

Power usage for the device in Watts

```
#define DCGM_FI_DEV_TOTAL_ENERGY_CONSUMPTION  
156
```

Total energy consumption for the GPU in mJ since the driver was last reloaded

```
#define DCGM_FI_DEV_SLOWDOWN_TEMP 158
```

Slowdown temperature for the device

```
#define DCGM_FI_DEV_SHUTDOWN_TEMP 159
```

Shutdown temperature for the device

```
#define DCGM_FI_DEV_POWER_MGMT_LIMIT 160
```

Current Power limit for the device

```
#define DCGM_FI_DEV_POWER_MGMT_LIMIT_MIN 161
```

Minimum power management limit for the device

```
#define DCGM_FI_DEV_POWER_MGMT_LIMIT_MAX 162
```

Maximum power management limit for the device

```
#define DCGM_FI_DEV_POWER_MGMT_LIMIT_DEF 163
```

Default power management limit for the device

```
#define DCGM_FI_DEV_ENFORCED_POWER_LIMIT 164
```

Effective power limit that the driver enforces after taking into account all limiters

```
#define DCGM_FI_DEV_PSTATE 190
```

Performance state (P-State) 0-15. 0=highest

```
#define DCGM_FI_DEV_FAN_SPEED 191
```

Fan speed for the device in percent 0-100

```
#define DCGM_FI_DEV_PCIE_TX_THROUGHPUT 200
```

PCIe Tx utilization information

Deprecated: Use DCGM\_FI\_PROF\_PCIE\_TX\_BYTES instead.

```
#define DCGM_FI_DEV_PCIE_RX_THROUGHPUT 201
```

PCIe Rx utilization information

Deprecated: Use DCGM\_FI\_PROF\_PCIE\_RX\_BYTES instead.

```
#define DCGM_FI_DEV_PCIE_REPLAY_COUNTER 202
```

PCIe replay counter

```
#define DCGM_FI_DEV_GPU_UTIL 203
```

GPU Utilization

## **#define DCGM\_FI\_DEV\_MEM\_COPY\_UTIL 204**

Memory Utilization

## **#define DCGM\_FI\_DEV\_ACCOUNTING\_DATA 205**

Process accounting stats.

This field is only supported when the host engine is running as root unless you enable accounting ahead of time. Accounting mode can be enabled by running "nvidia-smi -am 1" as root on the same node the host engine is running on.

## **#define DCGM\_FI\_DEV\_ENC\_UTIL 206**

Encoder Utilization

## **#define DCGM\_FI\_DEV\_DEC\_UTIL 207**

Decoder Utilization

## **#define DCGM\_FI\_DEV\_MEM\_COPY\_UTIL\_SAMPLES 210**

Memory utilization samples

## **#define DCGM\_FI\_DEV\_GRAPHICS\_PIDS 220**

Graphics processes running on the GPU.

## **#define DCGM\_FI\_DEV\_COMPUTE\_PIDS 221**

Compute processes running on the GPU.

## **#define DCGM\_FI\_DEV\_XID\_ERRORS 230**

XID errors. The value is the specific XID error

## **#define DCGM\_FI\_DEV\_PCIE\_MAX\_LINK\_GEN 235**

PCIe Max Link Generation

## **#define DCGM\_FI\_DEV\_PCIE\_MAX\_LINK\_WIDTH 236**

PCIe Max Link Width

```
#define DCGM_FI_DEV_PCIE_LINK_GEN 237
```

PCIe Current Link Generation

```
#define DCGM_FI_DEV_PCIE_LINK_WIDTH 238
```

PCIe Current Link Width

```
#define DCGM_FI_DEV_POWER_VIOLATION 240
```

Power Violation time in usec

```
#define DCGM_FI_DEV_THERMAL_VIOLATION 241
```

Thermal Violation time in usec

```
#define DCGM_FI_DEV_SYNC_BOOST_VIOLATION 242
```

Sync Boost Violation time in usec

```
#define DCGM_FI_DEV_BOARD_LIMIT_VIOLATION 243
```

Board violation limit.

```
#define DCGM_FI_DEV_LOW_UTIL_VIOLATION 244
```

Low utilisation violation limit.

```
#define DCGM_FI_DEV_RELIABILITY_VIOLATION 245
```

Reliability violation limit.

```
#define DCGM_FI_DEV_TOTAL_APP_CLOCKS_VIOLATION  
246
```

App clock violation limit.

```
#define DCGM_FI_DEV_TOTAL_BASE_CLOCKS_VIOLATION  
247
```

Base clock violation limit.

```
#define DCGM_FI_DEV_FB_TOTAL 250
```

Total Frame Buffer of the GPU in MB

```
#define DCGM_FI_DEV_FB_FREE 251
```

Free Frame Buffer in MB

```
#define DCGM_FI_DEV_FB_USED 252
```

Used Frame Buffer in MB

```
#define DCGM_FI_DEV_ECC_CURRENT 300
```

Current ECC mode for the device

```
#define DCGM_FI_DEV_ECC_PENDING 301
```

Pending ECC mode for the device

```
#define DCGM_FI_DEV_ECC_SBE_VOL_TOTAL 310
```

Total single bit volatile ECC errors

```
#define DCGM_FI_DEV_ECC_DBE_VOL_TOTAL 311
```

Total double bit volatile ECC errors

```
#define DCGM_FI_DEV_ECC_SBE_AGG_TOTAL 312
```

Total single bit aggregate (persistent) ECC errors Note: monotonically increasing

```
#define DCGM_FI_DEV_ECC_DBE_AGG_TOTAL 313
```

Total double bit aggregate (persistent) ECC errors Note: monotonically increasing

```
#define DCGM_FI_DEV_ECC_SBE_VOL_L1 314
```

L1 cache single bit volatile ECC errors

```
#define DCGM_FI_DEV_ECC_DBE_VOL_L1 315
```

L1 cache double bit volatile ECC errors

```
#define DCGM_FI_DEV_ECC_SBE_VOL_L2 316
```

L2 cache single bit volatile ECC errors

```
#define DCGM_FI_DEV_ECC_DBE_VOL_L2 317
```

L2 cache double bit volatile ECC errors

```
#define DCGM_FI_DEV_ECC_SBE_VOL_DEV 318
```

Device memory single bit volatile ECC errors

```
#define DCGM_FI_DEV_ECC_DBE_VOL_DEV 319
```

Device memory double bit volatile ECC errors

```
#define DCGM_FI_DEV_ECC_SBE_VOL_REG 320
```

Register file single bit volatile ECC errors

```
#define DCGM_FI_DEV_ECC_DBE_VOL_REG 321
```

Register file double bit volatile ECC errors

```
#define DCGM_FI_DEV_ECC_SBE_VOL_TEX 322
```

Texture memory single bit volatile ECC errors

```
#define DCGM_FI_DEV_ECC_DBE_VOL_TEX 323
```

Texture memory double bit volatile ECC errors

```
#define DCGM_FI_DEV_ECC_SBE_AGG_L1 324
```

L1 cache single bit aggregate (persistent) ECC errors Note: monotonically increasing

```
#define DCGM_FI_DEV_ECC_DBE_AGG_L1 325
```

L1 cache double bit aggregate (persistent) ECC errors Note: monotonically increasing

```
#define DCGM_FI_DEV_ECC_SBE_AGG_L2 326
```

L2 cache single bit aggregate (persistent) ECC errors Note: monotonically increasing

**#define DCGM\_FI\_DEV\_ECC\_DBE\_AGG\_L2 327**

L2 cache double bit aggregate (persistent) ECC errors Note: monotonically increasing

**#define DCGM\_FI\_DEV\_ECC\_SBE\_AGG\_DEV 328**

Device memory single bit aggregate (persistent) ECC errors Note: monotonically increasing

**#define DCGM\_FI\_DEV\_ECC\_DBE\_AGG\_DEV 329**

Device memory double bit aggregate (persistent) ECC errors Note: monotonically increasing

**#define DCGM\_FI\_DEV\_ECC\_SBE\_AGG\_REG 330**

Register File single bit aggregate (persistent) ECC errors Note: monotonically increasing

**#define DCGM\_FI\_DEV\_ECC\_DBE\_AGG\_REG 331**

Register File double bit aggregate (persistent) ECC errors Note: monotonically increasing

**#define DCGM\_FI\_DEV\_ECC\_SBE\_AGG\_TEX 332**

Texture memory single bit aggregate (persistent) ECC errors Note: monotonically increasing

**#define DCGM\_FI\_DEV\_ECC\_DBE\_AGG\_TEX 333**

Texture memory double bit aggregate (persistent) ECC errors Note: monotonically increasing

**#define DCGM\_FI\_DEV\_RETIRED\_SBE 390**

Number of retired pages because of single bit errors Note: monotonically increasing

**#define DCGM\_FI\_DEV\_RETIRED\_DBE 391**

Number of retired pages because of double bit errors Note: monotonically increasing

**#define DCGM\_FI\_DEV\_RETIRED\_PENDING 392**

Number of pages pending retirement

```
#define  
DCGM_FI_DEV_UNCORRECTABLE_REMAPPED_ROWS 393
```

Number of remapped rows for uncorrectable errors

```
#define DCGM_FI_DEV_CORRECTABLE_REMAPPED_ROWS  
394
```

Number of remapped rows for correctable errors

```
#define DCGM_FI_DEV_ROW_REMAP_FAILURE 395
```

Whether remapping of rows has failed

```
#define DCGM_FI_DEV_VIRTUAL_MODE 500
```

Virtualization Mode corresponding to the GPU.

One of DCGM\_GPU\_VIRTUALIZATION\_MODE\_\* constants.

```
#define DCGM_FI_DEV_SUPPORTED_TYPE_INFO 501
```

Includes Count and Static info of vGPU types supported on a device

```
#define DCGM_FI_DEV_CREATABLE_VGPU_TYPE_IDS 502
```

Includes Count and currently Creatable vGPU types on a device

```
#define DCGM_FI_DEV_VGPU_INSTANCE_IDS 503
```

Includes Count and currently Active vGPU Instances on a device

```
#define DCGM_FI_DEV_VGPU_UTILIZATIONS 504
```

Utilization values for vGPUs running on the device

```
#define  
DCGM_FI_DEV_VGPU_PER_PROCESS_UTILIZATION 505
```

Utilization values for processes running within vGPU VMs using the device

```
#define DCGM_FI_DEV_ENC_STATS 506
```

Current encoder statistics for a given device



**#define DCGM\_FI\_DEV\_FBC\_STATS 507**

Statistics of current active frame buffer capture sessions on a given device

**#define DCGM\_FI\_DEV\_FBC\_SESSIONS\_INFO 508**

Information about active frame buffer capture sessions on a target device

**#define DCGM\_FI\_DEV\_VGPU\_VM\_ID 520**

VM ID of the vGPU instance

**#define DCGM\_FI\_DEV\_VGPU\_VM\_NAME 521**

VM name of the vGPU instance

**#define DCGM\_FI\_DEV\_VGPU\_TYPE 522**

vGPU type of the vGPU instance

**#define DCGM\_FI\_DEV\_VGPU\_UUID 523**

UUID of the vGPU instance

**#define DCGM\_FI\_DEV\_VGPU\_DRIVER\_VERSION 524**

Driver version of the vGPU instance

**#define DCGM\_FI\_DEV\_VGPU\_MEMORY\_USAGE 525**

Memory usage of the vGPU instance

**#define DCGM\_FI\_DEV\_VGPU\_LICENSE\_STATUS 526**

License status of the vGPU instance

**#define DCGM\_FI\_DEV\_VGPU\_FRAME\_RATE\_LIMIT 527**

Frame rate limit of the vGPU instance

**#define DCGM\_FI\_DEV\_VGPU\_ENC\_STATS 528**

Current encoder statistics of the vGPU instance

```
#define DCGM_FI_DEV_VGPU_ENC_SESSIONS_INFO 529
```

Information about all active encoder sessions on the vGPU instance

```
#define DCGM_FI_DEV_VGPU_FBC_STATS 530
```

Statistics of current active frame buffer capture sessions on the vGPU instance

```
#define DCGM_FI_DEV_VGPU_FBC_SESSIONS_INFO 531
```

Information about active frame buffer capture sessions on the vGPU instance

```
#define DCGM_FI_DEV_VGPU_LICENSE_INSTANCE_STATUS  
532
```

License status of the vGPU host

```
#define DCGM_FI_FIRST_VGPU_FIELD_ID 520
```

Starting field ID of the vGPU instance

```
#define DCGM_FI_LAST_VGPU_FIELD_ID 570
```

Last field ID of the vGPU instance

```
#define DCGM_FI_MAX_VGPU_FIELDS  
DCGM_FI_LAST_VGPU_FIELD_ID -  
DCGM_FI_FIRST_VGPU_FIELD_ID
```

For now max vGPU field Ids taken as difference of DCGM\_FI\_LAST\_VGPU\_FIELD\_ID and DCGM\_FI\_FIRST\_VGPU\_FIELD\_ID i.e. 50

```
#define DCGM_FI_INTERNAL_FIELDS_0_START 600
```

Starting ID for all the internal fields

```
#define DCGM_FI_INTERNAL_FIELDS_0_END 699
```

Last ID for all the internal fields

NVSwitch entity field IDs start here.

NVSwitch latency bins for port 0

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P00  
700
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P00  
701
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P00  
702
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P00  
703
```

Max latency bin

NVSwitch latency bins for port 1

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P01  
704
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P01  
705
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P01  
706
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P01  
707
```

Max latency bin

NVSwitch latency bins for port 2

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P02  
708
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P02  
709
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P02  
710
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P02  
711
```

Max latency bin

NVSwitch latency bins for port 3

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P03  
712
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P03  
713
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P03  
714
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P03  
715
```

Max latency bin

NVSwitch latency bins for port 4

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P04  
716
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P04  
717
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P04  
718
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P04  
719
```

Max latency bin

NVSwitch latency bins for port 5

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P05  
720
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P05  
721
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P05  
722
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P05  
723
```

Max latency bin

NVSwitch latency bins for port 6

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P06  
724
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P06  
725
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P06  
726
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P06  
727
```

Max latency bin

NVSwitch latency bins for port 7

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P07  
728
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P07  
729
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P07  
730
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P07  
731
```

Max latency bin

NVSwitch latency bins for port 8

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P08  
732
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P08  
733
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P08  
734
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P08  
735
```

Max latency bin

NVSwitch latency bins for port 9

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P09  
736
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P09  
737
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P09  
738
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P09  
739
```

Max latency bin

NVSwitch latency bins for port 10

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P10  
740
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P10  
741
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P10  
742
```

High latency bin



```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P10  
743
```

Max latency bin

NVSwitch latency bins for port 11

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P11  
744
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P11  
745
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P11  
746
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P11  
747
```

Max latency bin

NVSwitch latency bins for port 12

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P12  
748
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P12  
749
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P12  
750
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P12  
751
```

Max latency bin

NVSwitch latency bins for port 13

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P13  
752
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P13  
753
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P13  
754
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P13  
755
```

Max latency bin

NVSwitch latency bins for port 14

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P14  
756
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P14  
757
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P14  
758
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P14  
759
```

Max latency bin

NVSwitch latency bins for port 15

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P15  
760
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P15  
761
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P15  
762
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P15  
763
```

Max latency bin

NVSwitch latency bins for port 16

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P16  
764
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P16  
765
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P16  
766
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P16  
767
```

Max latency bin

NVSwitch latency bins for port 17

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P17  
768
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P17  
769
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P17  
770
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P17  
771
```

Max latency bin

NVSwitch Tx and Rx Counter 0 for each port

By default, Counter 0 counts bytes.

**#define**

**DCGM\_FI\_DEV\_NVSWITCH\_BANDWIDTH\_TX\_0\_P00 780**

NVSwitch Tx Bandwidth Counter 0 for port 0

**#define**

**DCGM\_FI\_DEV\_NVSWITCH\_BANDWIDTH\_RX\_0\_P00 781**

NVSwitch Rx Bandwidth Counter 0 for port 0

**#define**

**DCGM\_FI\_DEV\_NVSWITCH\_BANDWIDTH\_TX\_0\_P01 782**

NVSwitch Tx Bandwidth Counter 0 for port 1

**#define**

**DCGM\_FI\_DEV\_NVSWITCH\_BANDWIDTH\_RX\_0\_P01 783**

NVSwitch Rx Bandwidth Counter 0 for port 1

**#define**

**DCGM\_FI\_DEV\_NVSWITCH\_BANDWIDTH\_TX\_0\_P02 784**

NVSwitch Tx Bandwidth Counter 0 for port 2

**#define**

**DCGM\_FI\_DEV\_NVSWITCH\_BANDWIDTH\_RX\_0\_P02 785**

NVSwitch Rx Bandwidth Counter 0 for port 2

**#define**

**DCGM\_FI\_DEV\_NVSWITCH\_BANDWIDTH\_TX\_0\_P03 786**

NVSwitch Tx Bandwidth Counter 0 for port 3

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P03 787
```

NVSwitch Rx Bandwidth Counter 0 for port 3

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P04 788
```

NVSwitch Tx Bandwidth Counter 0 for port 4

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P04 789
```

NVSwitch Rx Bandwidth Counter 0 for port 4

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P05 790
```

NVSwitch Tx Bandwidth Counter 0 for port 5

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P05 791
```

NVSwitch Rx Bandwidth Counter 0 for port 5

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P06 792
```

NVSwitch Tx Bandwidth Counter 0 for port 6

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P06 793
```

NVSwitch Rx Bandwidth Counter 0 for port 6

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P07 794
```

NVSwitch Tx Bandwidth Counter 0 for port 7

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P07 795
```

NVSwitch Rx Bandwidth Counter 0 for port 7

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P08 796
```

NVSwitch Tx Bandwidth Counter 0 for port 8

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P08 797
```

NVSwitch Rx Bandwidth Counter 0 for port 8

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P09 798
```

NVSwitch Tx Bandwidth Counter 0 for port 9

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P09 799
```

NVSwitch Rx Bandwidth Counter 0 for port 9

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P10 800
```

NVSwitch Tx Bandwidth Counter 0 for port 10

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P10 801
```

NVSwitch Rx Bandwidth Counter 0 for port 10

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P11 802
```

NVSwitch Tx Bandwidth Counter 0 for port 11

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P11 803
```

NVSwitch Rx Bandwidth Counter 0 for port 11

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P12 804
```

NVSwitch Tx Bandwidth Counter 0 for port 12

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P12 805
```

NVSwitch Rx Bandwidth Counter 0 for port 12

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P13 806
```

NVSwitch Tx Bandwidth Counter 0 for port 13

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P13 807
```

NVSwitch Rx Bandwidth Counter 0 for port 13

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P14 808
```

NVSwitch Tx Bandwidth Counter 0 for port 14

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P14 809
```

NVSwitch Rx Bandwidth Counter 0 for port 14

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P15 810
```

NVSwitch Tx Bandwidth Counter 0 for port 15



```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P15 811
```

NVSwitch Rx Bandwidth Counter 0 for port 15

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P16 812
```

NVSwitch Tx Bandwidth Counter 0 for port 16

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P16 813
```

NVSwitch Rx Bandwidth Counter 0 for port 16

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P17 814
```

NVSwitch Tx Bandwidth Counter 0 for port 17

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P17 815
```

NVSwitch Rx Bandwidth Counter 0 for port 17

NVSwitch Tx and RX Bandwidth Counter 1 for each port

By default, Counter 1 counts packets.

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P00 820
```

NVSwitch Tx Bandwidth Counter 1 for port 0

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P00 821
```

NVSwitch Rx Bandwidth Counter 1 for port 0

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P01 822
```

NVSwitch Tx Bandwidth Counter 1 for port 1

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P01 823
```

NVSwitch Rx Bandwidth Counter 1 for port 1

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P02 824
```

NVSwitch Tx Bandwidth Counter 1 for port 2

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P02 825
```

NVSwitch Rx Bandwidth Counter 1 for port 2

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P03 826
```

NVSwitch Tx Bandwidth Counter 1 for port 3

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P03 827
```

NVSwitch Rx Bandwidth Counter 1 for port 3

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P04 828
```

NVSwitch Tx Bandwidth Counter 1 for port 4

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P04 829
```

NVSwitch Rx Bandwidth Counter 1 for port 4

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P05 830
```

NVSwitch Tx Bandwidth Counter 1 for port 5

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P05 831
```

NVSwitch Rx Bandwidth Counter 1 for port 5

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P06 832
```

NVSwitch Tx Bandwidth Counter 1 for port 6

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P06 833
```

NVSwitch Rx Bandwidth Counter 1 for port 6

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P07 834
```

NVSwitch Tx Bandwidth Counter 1 for port 7

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P07 835
```

NVSwitch Rx Bandwidth Counter 1 for port 7

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P08 836
```

NVSwitch Tx Bandwidth Counter 1 for port 8

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P08 837
```

NVSwitch Rx Bandwidth Counter 1 for port 8

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P09 838
```

NVSwitch Tx Bandwidth Counter 1 for port 9

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P09 839
```

NVSwitch Rx Bandwidth Counter 1 for port 9

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P10 840
```

NVSwitch Tx Bandwidth Counter 0 for port 10

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P10 841
```

NVSwitch Rx Bandwidth Counter 1 for port 10

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P11 842
```

NVSwitch Tx Bandwidth Counter 1 for port 11

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P11 843
```

NVSwitch Rx Bandwidth Counter 1 for port 11

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P12 844
```

NVSwitch Tx Bandwidth Counter 1 for port 12

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P12 845
```

NVSwitch Rx Bandwidth Counter 1 for port 12

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P13 846
```

NVSwitch Tx Bandwidth Counter 0 for port 13

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P13 847
```

NVSwitch Rx Bandwidth Counter 1 for port 13

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P14 848
```

NVSwitch Tx Bandwidth Counter 1 for port 14

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P14 849
```

NVSwitch Rx Bandwidth Counter 1 for port 14

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P15 850
```

NVSwitch Tx Bandwidth Counter 1 for port 15

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P15 851
```

NVSwitch Rx Bandwidth Counter 1 for port 15

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P16 852
```

NVSwitch Tx Bandwidth Counter 1 for port 16

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P16 853
```

NVSwitch Rx Bandwidth Counter 1 for port 16

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P17 854
```

NVSwitch Tx Bandwidth Counter 1 for port 17

```
#define  
DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P17 855
```

NVSwitch Rx Bandwidth Counter 1 for port 17

NVSwitch error counters

```
#define DCGM_FI_DEV_NVSWITCH_FATAL_ERRORS 856
```

NVSwitch fatal error information. Note: value field indicates the specific SXid reported

```
#define DCGM_FI_DEV_NVSWITCH_NON_FATAL_ERRORS  
857
```

NVSwitch non fatal error information. Note: value field indicates the specific SXid reported

```
#define DCGM_FI_FIRST_NVSWITCH_FIELD_ID 700
```

Starting field ID of the NVSwitch instance

```
#define DCGM_FI_LAST_NVSWITCH_FIELD_ID 860
```

Last field ID of the NVSwitch instance

```
#define DCGM_FI_MAX_NVSWITCH_FIELDS  
DCGM_FI_LAST_NVSWITCH_FIELD_ID -  
DCGM_FI_FIRST_NVSWITCH_FIELD_ID + 1
```

For now max NVSwitch field Ids taken as difference of DCGM\_FI\_LAST\_NVSWITCH\_FIELD\_ID and DCGM\_FI\_FIRST\_NVSWITCH\_FIELD\_ID + 1 i.e. 200

**#define DCGM\_FI\_PROF\_GR\_ENGINE\_ACTIVE 1001**

Profiling Fields. These all start with DCGM\_FI\_PROF\_\* Ratio of time the graphics engine is active. The graphics engine is active if a graphics/compute context is bound and the graphics pipe or compute pipe is busy.

**#define DCGM\_FI\_PROF\_SM\_ACTIVE 1002**

The ratio of cycles an SM has at least 1 warp assigned (computed from the number of cycles and elapsed cycles)

**#define DCGM\_FI\_PROF\_SM\_OCCUPANCY 1003**

The ratio of number of warps resident on an SM. (number of resident as a ratio of the theoretical maximum number of warps per elapsed cycle)

**#define DCGM\_FI\_PROF\_PIPE\_TENSOR\_ACTIVE 1004**

The ratio of cycles the tensor (HMMA) pipe is active (off the peak sustained elapsed cycles)

**#define DCGM\_FI\_PROF\_DRAM\_ACTIVE 1005**

The ratio of cycles the device memory interface is active sending or receiving data.

**#define DCGM\_FI\_PROF\_PIPE\_FP64\_ACTIVE 1006**

Ratio of cycles the fp64 pipe is active.

**#define DCGM\_FI\_PROF\_PIPE\_FP32\_ACTIVE 1007**

Ratio of cycles the fp32 pipe is active.

**#define DCGM\_FI\_PROF\_PIPE\_FP16\_ACTIVE 1008**

Ratio of cycles the fp16 pipe is active. This does not include HMMA.

**#define DCGM\_FI\_PROF\_PCIE\_TX\_BYTES 1009**

The number of bytes of active PCIe tx (transmit) data including both header and payload.

Note that this is from the perspective of the GPU, so copying data from device to host (DtoH) would be reflected in this metric.

## #define DCGM\_FI\_PROF\_PCIE\_RX\_BYTES 1010

The number of bytes of active PCIe rx (read) data including both header and payload.

Note that this is from the perspective of the GPU, so copying data from host to device (HtoD) would be reflected in this metric.

## #define DCGM\_FI\_PROF\_NVLINK\_TX\_BYTES 1011

The number of bytes of active NvLink tx (transmit) data including both header and payload.

## #define DCGM\_FI\_PROF\_NVLINK\_RX\_BYTES 1012

The number of bytes of active NvLink rx (read) data including both header and payload.

## #define DCGM\_FI\_MAX\_FIELDS 1013

1 greater than maximum fields above. This is the 1 greater than the maximum field id that could be allocated

## 1.21. DCGMAPI\_Admin\_ExecCtrl

### dcgmReturn\_t dcgmUpdateAllFields (dcgmHandle\_t pDcgmHandle, int waitForUpdate)

#### Parameters

##### pDcgmHandle

IN: DCGM Handle

##### waitForUpdate

IN: Whether or not to wait for the update loop to complete before returning to the caller 1=wait. 0=do not wait.

#### Returns

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if waitForUpdate is invalid
- ▶ DCGM\_ST\_GENERIC\_ERROR if an unspecified DCGM error occurs

#### Description

This method is used to tell the DCGM module to update all the fields being watched.



Note: If the if the operation mode was set to manual mode (DCGM\_OPERATION\_MODE\_MANUAL) during initialization (`dcgmInit`), this method must be called periodically to allow field value watches the opportunity to gather samples.

## dcgmReturn\_t dcgmPolicyTrigger (dcgmHandle\_t pDcgmHandle)

### Parameters

#### pDcgmHandle

IN: DCGM Handle

### Returns

- ▶ DCGM\_ST\_OK If the call was successful
- ▶ DCGM\_ST\_GENERIC\_ERROR The policy manager was unable to perform another iteration.

### Description

Inform the policy manager loop to perform an iteration and trigger the callbacks of any registered functions. Callback functions will be called from a separate thread as the calling function.

Note: The GPU monitoring and management agent must call this method periodically if the operation mode is set to manual mode (DCGM\_OPERATION\_MODE\_MANUAL) during initialization (`dcgmInit`).

## 1.15. Modules

Here is a list of all modules:

- ▶ Administrative
  - ▶ Init and Shutdown
  - ▶ Auxilary information about DCGM engine.
- ▶ System
  - ▶ Discovery
  - ▶ Grouping
  - ▶ Field Grouping
  - ▶ Status handling
- ▶ Configuration

- ▶ Setup and management
- ▶ Manual Invocation
- ▶ Field APIs
- ▶ Process Statistics
- ▶ Job Statistics
- ▶ Health Monitor
- ▶ Policies
  - ▶ Setup and Management
  - ▶ Manual Invocation
- ▶ Topology
- ▶ Metadata
- ▶ Topology
- ▶ Modules
- ▶ Profiling
- ▶ Enums and Macros
- ▶ Structure definitions
- ▶ Field Types
- ▶ Field Scope
- ▶ Field Constants
- ▶ Field Entity
- ▶ Field Identifiers
- ▶ DCGMAPI\_Admin\_ExecCtrl

## 1.1. Administrative

This chapter describes the administration interfaces for DCGM. It is the user's responsibility to call `dcgmInit()` before calling any other methods, and `dcgmShutdown()` once DCGM is no longer being used. The APIs in Administrative module can be broken down into following categories:

### Init and Shutdown

Auxiliary information about DCGM engine.

#### 1.1.1. Init and Shutdown

Administrative

Describes APIs to Initialize and Shutdown the DCGM Engine.

## dcgmReturn\_t dcgmInit (void)

### Returns

- ▶ DCGM\_ST\_OK if DCGM has been properly initialized
- ▶ DCGM\_ST\_INIT\_ERROR if there was an error initializing the library

### Description

This method is used to initialize DCGM within this process. This must be called before `dcgmStartEmbedded()` or `dcgmConnect()`

\*

## dcgmReturn\_t dcgmShutdown (void)

### Returns

- ▶ DCGM\_ST\_OK if DCGM has been properly shut down
- ▶ DCGM\_ST\_UNINITIALIZED if the library was not shut down properly

### Description

This method is used to shut down DCGM. Any embedded host engines or remote connections will automatically be shut down as well.

## dcgmReturn\_t dcgmStartEmbedded (dcgmOperationMode\_t opMode, dcgmHandle\_t \*pDcgmHandle)

### Parameters

#### opMode

IN: Collect data automatically or manually when asked by the user.

#### pDcgmHandle

OUT: DCGM Handle to use for API calls

### Returns

- ▶ DCGM\_ST\_OK if DCGM was started successfully within our process
- ▶ DCGM\_ST\_UNINITIALIZED if DCGM has not been initialized with `dcgmInit` yet

### Description

Start an embedded host engine agent within this process.

The agent is loaded as a shared library. This mode is provided to avoid any extra jitter associated with an additional autonomous agent needs to be managed. In this mode, the

user has to periodically call APIs such as [dcgmPolicyTrigger](#) and [dcgmUpdateAllFields](#) which tells DCGM to wake up and perform data collection and operations needed for policy management.

## `dcgmReturn_t dcgmStartEmbedded_v2 (dcgmStartEmbeddedV2Params_v1 *params[])`

### Parameters

#### `params`

IN/OUT: See [dcgmStartEmbeddedV2Params\\_v1](#) for details.

### Returns

- ▶ `DCGM_ST_OK` if DCGM was started successfully within our process
- ▶ `DCGM_ST_UNINITIALIZED` if DCGM has not been initialized with `dcgmInit` yet

### Description

Start an embedded host engine agent within this process.

The agent is loaded as a shared library. This mode is provided to avoid any extra jitter associated with an additional autonomous agent needs to be managed. In this mode, the user has to periodically call APIs such as [dcgmPolicyTrigger](#) and [dcgmUpdateAllFields](#) which tells DCGM to wake up and perform data collection and operations needed for policy management.

## `dcgmReturn_t dcgmStopEmbedded (dcgmHandle_t pDcgmHandle)`

### Parameters

#### `pDcgmHandle`

IN : DCGM Handle of the embedded host engine that came from `dcgmStartEmbedded`

### Returns

- ▶ `DCGM_ST_OK` if DCGM was stopped successfully within our process
- ▶ `DCGM_ST_UNINITIALIZED` if DCGM has not been initialized with `dcgmInit` or the embedded host engine was not running.
- ▶ `DCGM_ST_BADPARAM` if an invalid parameter was provided
- ▶ `DCGM_ST_INIT_ERROR` if an error occurred while trying to start the host engine.

**Description**

Stop the embedded host engine within this process that was started with `dcgmStartEmbedded`

`dcgmReturn_t dcgmConnect (char *ipAddress, dcgmHandle_t *pDcgmHandle)`

**Parameters****ipAddress**

IN: Valid IP address for the remote host engine to connect to. If `ipAddress` is specified as `x.x.x.x` it will attempt to connect to the default port specified by `DCGM_HE_PORT_NUMBER`. If `ipAddress` is specified as `x.x.x.x:yyyy` it will attempt to connect to the port specified by `yyyy`.

**pDcgmHandle**

OUT: DCGM Handle of the remote host engine

**Returns**

- ▶ `DCGM_ST_OK` if we successfully connected to the remote host engine
- ▶ `DCGM_ST_CONNECTION_NOT_VALID` if the remote host engine could not be reached
- ▶ `DCGM_ST_UNINITIALIZED` if DCGM has not been initialized with `dcgmInit`.
- ▶ `DCGM_ST_BADPARAM` if `pDcgmHandle` is `NULL` or `ipAddress` is invalid
- ▶ `DCGM_ST_INIT_ERROR` if DCGM encountered an error while initializing the remote client library
- ▶ `DCGM_ST_UNINITIALIZED` if DCGM has not been initialized with `dcgmInit`

**Description**

This method is used to connect to a stand-alone host engine process. Remote host engines are started by running the `nv-hostengine` command.

NOTE: `dcgmConnect_v2` provides additional connection options.

`dcgmReturn_t dcgmConnect_v2 (char *ipAddress, dcgmConnectV2Params_t *connectParams, dcgmHandle_t *pDcgmHandle)`

**Parameters****ipAddress**

IN: Valid IP address for the remote host engine to connect to. If `ipAddress` is specified as `x.x.x.x` it will attempt to connect to the default port specified by

DCGM\_HE\_PORT\_NUMBER. If ipAddress is specified as x.x.x.x:yyyy it will attempt to connect to the port specified by yyyy

#### **connectParams**

IN: Additional connection parameters. See [dcmgConnectV2Params\\_t](#) for details.

#### **pDcgmHandle**

OUT: DCGM Handle of the remote host engine

#### **Returns**

- ▶ DCGM\_ST\_OK if we successfully connected to the remote host engine
- ▶ DCGM\_ST\_CONNECTION\_NOT\_VALID if the remote host engine could not be reached
- ▶ DCGM\_ST\_UNINITIALIZED if DCGM has not been initialized with `dcmgInit`.
- ▶ DCGM\_ST\_BADPARAM if `pDcgmHandle` is NULL or `ipAddress` is invalid
- ▶ DCGM\_ST\_INIT\_ERROR if DCGM encountered an error while initializing the remote client library
- ▶ DCGM\_ST\_UNINITIALIZED if DCGM has not been initialized with `dcmgInit`

#### **Description**

This method is used to connect to a stand-alone host engine process. Remote host engines are started by running the `nv-hostengine` command.

### **dcmgReturn\_t dcmgDisconnect (dcmgHandle\_t pDcgmHandle)**

#### **Parameters**

#### **pDcgmHandle**

IN: DCGM Handle that came from `dcmgConnect`

#### **Returns**

- ▶ DCGM\_ST\_OK if we successfully disconnected from the host engine
- ▶ DCGM\_ST\_UNINITIALIZED if DCGM has not been initialized with `dcmgInit`
- ▶ DCGM\_ST\_BADPARAM if `pDcgmHandle` is not a valid DCGM handle
- ▶ DCGM\_ST\_GENERIC\_ERROR if an unspecified internal error occurred

#### **Description**

This method is used to disconnect from a stand-alone host engine process.

## **1.1.2. Auxiliary information about DCGM engine.**

Administrative

Describes APIs to get generic information about the DCGM Engine.

## `dcgmReturn_t dcgmVersionInfo (dcgmVersionInfo_t *pVersionInfo)`

### Parameters

#### **pVersionInfo**

OUT: Build environment information

### Returns

- ▶ DCGM\_ST\_OK if build information is successfully obtained
- ▶ DCGM\_ST\_BADPARAM if pVersionInfo is null
- ▶ DCGM\_ST\_VER\_MISMATCH if the expected and provided versions of dcgmVersionInfo\_t do not match

### Description

This method is used to return information about the build environment where DCGM was built.

## `dcgmReturn_t dcgmHostengineVersionInfo (dcgmHandle_t pDcgmHandle, dcgmVersionInfo_t *pVersionInfo)`

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle that came from dcgmConnect

#### **pVersionInfo**

OUT: Build environment information

### Returns

- ▶ DCGM\_ST\_OK if build information is successfully obtained
- ▶ DCGM\_ST\_BADPARAM if pVersionInfo is null
- ▶ DCGM\_ST\_VER\_MISMATCH if the expected and provided versions of dcgmVersionInfo\_t do not match

### Description

This method is used to return information about the build environment of the hostengine.

`dcgmReturn_t dcgmHostengineSetLoggingSeverity (dcgmHandle_t pDcgmHandle, dcgmSettingsSetLoggingSeverity_t *logging)`

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **logging**

IN: `dcgmSettingsSetLoggingSeverity_t` struct containing the target logger and severity

### Returns

- ▶ DCGM\_ST\_OK Severity successfully set
- ▶ DCGM\_ST\_BADPARAM Bad logger/severity string
- ▶ DCGM\_ST\_VER\_MISMATCH if the expected and provided versions of `dcgmSettingsSetLoggingSeverity_t` do not match

### Description

This method is used to set the logging severity on HostEngine for the specified logger

`dcgmReturn_t dcgmHostengineIsHealthy (dcgmHandle_t pDcgmHandle, dcgmHostengineHealth_t *heHealth)`

### Parameters

#### **pDcgmHandle**

- the handle to DCGM

#### **heHealth**

- struct describing the health of the hostengine. if `heHealth.hostengineHealth` is 0, then the hostengine is healthy. Non-zero indicates not healthy with error codes determining the cause.

### Returns

- ▶ DCGM\_ST\_OK Able to gauge health
- ▶ DCGM\_ST\_BADPARAM `isHealthy` is not a valid pointer

### Description

This function is used to return whether or not the host engine considers itself healthy



## 1.2. System

This chapter describes the APIs used to identify set of GPUs on the node, grouping functions to provide mechanism to operate on a group of GPUs, and status management APIs in order to get individual statuses for each operation. The APIs in System module can be broken down into following categories:

### Discovery

### Grouping

### Field Grouping

### Status handling

#### 1.2.1. Discovery

##### System

The following APIs are used to discover GPUs and their attributes on a Node.

```
dcgmReturn_t dcgmGetAllDevices (dcgmHandle_t pDcgmHandle, unsigned
int gpuIdList, int *count)
```

##### Parameters

###### **pDcgmHandle**

IN: DCGM Handle

###### **gpuIdList**

OUT: Array reference to fill GPU Ids present on the system.

###### **count**

OUT: Number of GPUs returned in gpuIdList.

##### Returns

- ▶ DCGM\_ST\_OK if the call was successful.
- ▶ DCGM\_ST\_BADPARAM if gpuIdList or count were not valid.

##### Description

This method is used to get identifiers corresponding to all the devices on the system. The identifier represents DCGM GPU Id corresponding to each GPU on the system and is immutable during the lifespan of the engine. The list should be queried again if the engine is restarted.

The GPUs returned from this function include gpuIds of GPUs that are not supported by DCGM. To only get gpuIds of GPUs that are supported by DCGM, use [dcgmGetAllSupportedDevices\(\)](#).

```
dcgmReturn_t dcgmGetAllSupportedDevices (dcgmHandle_t
pDcgmHandle, unsigned int gpuIdList, int *count)
```

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **gpuIdList**

OUT: Array reference to fill GPU Ids present on the system.

#### **count**

OUT: Number of GPUs returned in gpuIdList.

### Returns

- ▶ DCGM\_ST\_OK if the call was successful.
- ▶ DCGM\_ST\_BADPARAM if gpuIdList or count were not valid.

### Description

This method is used to get identifiers corresponding to all the DCGM-supported devices on the system. The identifier represents DCGM GPU Id corresponding to each GPU on the system and is immutable during the lifespan of the engine. The list should be queried again if the engine is restarted.

The GPUs returned from this function ONLY includes gpuIds of GPUs that are supported by DCGM. To get gpuIds of all GPUs in the system, use [dcgmGetAllDevices\(\)](#).

```
dcgmReturn_t dcgmGetDeviceAttributes (dcgmHandle_t pDcgmHandle,
unsigned int gpuId, dcgmDeviceAttributes_t *pDcgmAttr)
```

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **gpuId**

IN: GPU Id corresponding to which the attributes should be fetched

#### **pDcgmAttr**

IN/OUT: Device attributes corresponding to gpuId. pDcgmAttr->version should be set to [dcgmDeviceAttributes\\_version](#) before this call.

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful.
- ▶ DCGM\_ST\_VER\_MISMATCH if pDcgmAttr->version is not set or is invalid.

**Description**

Gets device attributes corresponding to the gpuId. If operation is not successful for any of the requested fields then the field is populated with one of DCGM\_BLANK\_VALUES defined in dcgm\_structs.h.

```
dcgmReturn_t dcgmGetEntityGroupEntities (dcgmHandle_t dcgmHandle,
dcgm_field_entity_group_t entityGroup, dcgm_field_eid_t *entities, int
*numEntities, unsigned int flags)
```

**Parameters****dcgmHandle**

IN: DCGM Handle

**entityGroup**

IN: Entity group to list entities of

**entities**

OUT: Array of entities for entityGroup

**numEntities**

IN/OUT: Upon calling, this should be the number of entities that entityList[] can hold. Upon return, this will contain the number of entities actually saved to entityList.

**flags**

IN: Flags to modify the behavior of this request. See DCGM\_GEGE\_FLAG\_\* defines in dcgm\_structs.h

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful.
- ▶ DCGM\_ST\_INSUFFICIENT\_SIZE if numEntities was not large enough to hold the number of entities in the entityGroup. numEntities will contain the capacity needed to complete this request successfully.
- ▶ DCGM\_ST\_NOT\_SUPPORTED if the given entityGroup does not support enumeration.
- ▶ DCGM\_ST\_BADPARAM if any parameter is invalid

**Description**

Gets the list of entities that exist for a given entity group. This API can be used in place of [dcgmGetAllDevices](#).

`dcgmReturn_t dcgmGetGpuInstanceHierarchy (dcgmHandle_t dcgmHandle, dcgmMigHierarchy_v2 *hierarchy)`

### Parameters

#### **dcgmHandle**

IN: DCGM Handle

#### **hierarchy**

### Returns

- ▶ DCGM\_ST\_OK if the call was successful.
- ▶ DCGM\_ST\_VER\_MISMATCH if the struct version is incorrect
- ▶ DCGM\_ST\_BADPARAM if any parameter is invalid

### Description

Gets the hierarchy of GPUs, GPU Instances, and Compute Instances by populating a list of each entity with a reference to their parent

`dcgmReturn_t dcgmGetNvLinkLinkStatus (dcgmHandle_t dcgmHandle, dcgmNvLinkStatus_v2 *linkStatus)`

### Parameters

#### **dcgmHandle**

IN: DCGM Handle

#### **linkStatus**

OUT: Structure in which to store NvLink link statuses. `.version` should be set to `dcgmNvLinkStatus_version1` before calling this.

### Returns

- ▶ DCGM\_ST\_OK if the call was successful.
- ▶ DCGM\_ST\_NOT\_SUPPORTED if the given entityGroup does not support enumeration.
- ▶ DCGM\_ST\_BADPARAM if any parameter is invalid

### Description

Get the NvLink link status for every NvLink in this system. This includes the NvLinks of both GPUs and NvSwitches. Note that only NvSwitches and GPUs that are visible to the current environment will be returned in this structure.

## 1.2.2. Grouping

### System

The following APIs are used for group management. The user can create a group of entities and perform an operation on a group of entities. If grouping is not needed and the user wishes to run commands on all GPUs seen by DCGM then the user can use DCGM\_GROUP\_ALL\_GPUS or DCGM\_GROUP\_ALL\_NVSWITCHES in place of group IDs when needed.

```
dcgmReturn_t dcgmGroupCreate (dcgmHandle_t pDcgmHandle,
dcgmGroupType_t type, char *groupName, dcgmGpuGrp_t *pDcgmGrpId)
```

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **type**

IN: Type of Entity Group to be formed

#### **groupName**

IN: Desired name of the GPU group specified as NULL terminated C string

#### **pDcgmGrpId**

OUT: Reference to group ID

### Returns

- ▶ DCGM\_ST\_OK if the group has been created
- ▶ DCGM\_ST\_BADPARAM if any of type, groupName, length or pDcgmGrpId is invalid
- ▶ DCGM\_ST\_MAX\_LIMIT if number of groups on the system has reached the max limit DCGM\_MAX\_NUM\_GROUPS
- ▶ DCGM\_ST\_INIT\_ERROR if the library has not been successfully initialized

### Description

Used to create a entity group handle which can store one or more entity Ids as an opaque handle returned in pDcgmGrpId. Instead of executing an operation separately for each entity, the DCGM group enables the user to execute same operation on all the entities present in the group as a single API call.

To create the group with all the entities present on the system, the type field should be specified as DCGM\_GROUP\_DEFAULT or DCGM\_GROUP\_ALL\_NVSWITCHES. To create an empty group, the type field should be specified as DCGM\_GROUP\_EMPTY. The empty group can be updated with the desired set of entities using the APIs [dcgmGroupAddDevice](#), [dcgmGroupAddEntity](#), [dcgmGroupRemoveDevice](#), and [dcgmGroupRemoveEntity](#).

`dcgmReturn_t dcgmGroupDestroy (dcgmHandle_t pDcgmHandle, dcgmGpuGrp_t groupId)`

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **groupId**

IN: Group ID

### Returns

- ▶ DCGM\_ST\_OK if the group has been destroyed
- ▶ DCGM\_ST\_BADPARAM if groupId is invalid
- ▶ DCGM\_ST\_INIT\_ERROR if the library has not been successfully initialized
- ▶ DCGM\_ST\_NOT\_CONFIGURED if entry corresponding to the group does not exist

### Description

Used to destroy a group represented by groupId. Since DCGM group is a logical grouping of entities, the properties applied on the group stay intact for the individual entities even after the group is destroyed.

`dcgmReturn_t dcgmGroupAddDevice (dcgmHandle_t pDcgmHandle, dcgmGpuGrp_t groupId, unsigned int gpuId)`

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **groupId**

IN: Group Id to which device should be added

#### **gpuId**

IN: DCGM GPU Id

### Returns

- ▶ DCGM\_ST\_OK if the GPU Id has been successfully added to the group
- ▶ DCGM\_ST\_INIT\_ERROR if the library has not been successfully initialized
- ▶ DCGM\_ST\_NOT\_CONFIGURED if entry corresponding to the group (groupId) does not exist
- ▶ DCGM\_ST\_BADPARAM if gpuId is invalid or already part of the specified group

**Description**

Used to add specified GPU Id to the group represented by groupId.

```
dcgmReturn_t dcgmGroupAddEntity (dcgmHandle_t pDcgmHandle,
dcgmGpuGrp_t groupId, dcgm_field_entity_group_t entityGroupId,
dcgm_field_eid_t entityId)
```

**Parameters****pDcgmHandle**

IN: DCGM Handle

**groupId**

IN: Group Id to which device should be added

**entityGroupId**

IN: Entity group that entityId belongs to

**entityId**

IN: DCGM entityId

**Returns**

- ▶ DCGM\_ST\_OK if the entity has been successfully added to the group
- ▶ DCGM\_ST\_INIT\_ERROR if the library has not been successfully initialized
- ▶ DCGM\_ST\_NOT\_CONFIGURED if entry corresponding to the group (groupId) does not exist
- ▶ DCGM\_ST\_BADPARAM if entityId is invalid or already part of the specified group

**Description**

Used to add specified entity to the group represented by groupId.

```
dcgmReturn_t dcgmGroupRemoveDevice (dcgmHandle_t pDcgmHandle,
dcgmGpuGrp_t groupId, unsigned int gpuld)
```

**Parameters****pDcgmHandle**

IN: DCGM Handle

**groupId**

IN: Group ID from which device should be removed

**gpuld**

IN: DCGM GPU Id

**Returns**

- ▶ DCGM\_ST\_OK if the GPU Id has been successfully removed from the group
- ▶ DCGM\_ST\_INIT\_ERROR if the library has not been successfully initialized
- ▶ DCGM\_ST\_NOT\_CONFIGURED if entry corresponding to the group (groupId) does not exist
- ▶ DCGM\_ST\_BADPARAM if groupId is invalid or not part of the specified group

**Description**

Used to remove specified GPU Id from the group represented by groupId.

```
dcgmReturn_t dcgmGroupRemoveEntity (dcgmHandle_t pDcgmHandle,
dcgmGpuGrp_t groupId, dcgm_field_entity_group_t entityGroupId,
dcgm_field_eid_t entityId)
```

**Parameters****pDcgmHandle**

IN: DCGM Handle

**groupId**

IN: Group ID from which device should be removed

**entityGroupId**

IN: Entity group that entityId belongs to

**entityId**

IN: DCGM entityId

**Returns**

- ▶ DCGM\_ST\_OK if the entity has been successfully removed from the group
- ▶ DCGM\_ST\_INIT\_ERROR if the library has not been successfully initialized
- ▶ DCGM\_ST\_NOT\_CONFIGURED if entry corresponding to the group (groupId) does not exist
- ▶ DCGM\_ST\_BADPARAM if entityId is invalid or not part of the specified group

**Description**

Used to remove specified entity from the group represented by groupId.

```
dcgmReturn_t dcgmGroupGetInfo (dcgmHandle_t pDcgmHandle,
dcgmGpuGrp_t groupId, dcgmGroupInfo_t *pDcgmGroupInfo)
```

**Parameters****pDcgmHandle**

IN: DCGM Handle



**groupId**

IN: Group ID for which information to be fetched

**pDcgmGroupInfo**

OUT: Group Information

**Returns**

- ▶ DCGM\_ST\_OK if the group info is successfully received.
- ▶ DCGM\_ST\_BADPARAM if any of groupId or pDcgmGroupInfo is invalid.
- ▶ DCGM\_ST\_INIT\_ERROR if the library has not been successfully initialized.
- ▶ DCGM\_ST\_MAX\_LIMIT if the group does not contain the GPU
- ▶ DCGM\_ST\_NOT\_CONFIGURED if entry corresponding to the group (groupId) does not exist

**Description**

Used to get information corresponding to the group represented by groupId. The information returned in pDcgmGroupInfo consists of group name, and the list of entities present in the group.

**dcgmReturn\_t dcgmGroupGetAllIds (dcgmHandle\_t pDcgmHandle, dcgmGpuGrp\_t groupIdList, unsigned int \*count)**

**Parameters****pDcgmHandle**

IN: DCGM Handle

**groupIdList**

OUT: List of Group Ids

**count**

OUT: The number of Group ids in the list

**Returns**

- ▶ DCGM\_ST\_OK if the ids of the groups were successfully retrieved
- ▶ DCGM\_ST\_BADPARAM if either of the groupIdList or count is null
- ▶ DCGM\_ST\_GENERIC\_ERROR if an unknown error has occurred

**Description**

Used to get the Ids of all groups of entities. The information returned is a list of group ids in groupIdList as well as a count of how many ids there are in count. Please allocate enough memory for groupIdList. Memory of size MAX\_NUM\_GROUPS should be allocated for groupIdList.

### 1.2.3. Field Grouping

System

The following APIs are used for field group management. The user can create a group of fields and perform an operation on a group of fields at once.

```
dcgmReturn_t dcgmFieldGroupCreate (dcgmHandle_t dcgmHandle,
int numFieldIds, unsigned short *fieldIds, char *fieldGroupName,
dcgmFieldGrp_t *dcgmFieldGroupId)
```

#### Parameters

##### **dcgmHandle**

IN: DCGM handle

##### **numFieldIds**

IN: Number of field IDs that are being provided in fieldIds[]. Must be between 1 and DCGM\_MAX\_FIELD\_IDS\_PER\_FIELD\_GROUP.

##### **fieldIds**

IN: Field IDs to be added to the newly-created field group

##### **fieldGroupName**

IN: Unique name for this group of fields. This must not be the same as any existing field groups.

##### **dcgmFieldGroupId**

OUT: Handle to the newly-created field group

#### Returns

- ▶ DCGM\_ST\_OK if the field group was successfully created.
- ▶ DCGM\_ST\_BADPARAM if any parameters were bad
- ▶ DCGM\_ST\_INIT\_ERROR if the library has not been successfully initialized.
- ▶ DCGM\_ST\_MAX\_LIMIT if too many field groups already exist

#### Description

Used to create a group of fields and return the handle in dcgmFieldGroupId

```
dcgmReturn_t dcgmFieldGroupDestroy (dcgmHandle_t dcgmHandle,
dcgmFieldGrp_t dcgmFieldGroupId)
```

#### Parameters

##### **dcgmHandle**

IN: DCGM handle

**dcgmFieldGroupId**

IN: Field group to remove

**Returns**

- ▶ DCGM\_ST\_OK if the field group was successfully removed
- ▶ DCGM\_ST\_BADPARAM if any parameters were bad
- ▶ DCGM\_ST\_INIT\_ERROR if the library has not been successfully initialized.

**Description**

Used to remove a field group that was created with [dcgmFieldGroupCreate](#)

```
dcgmReturn_t dcgmFieldGroupGetInfo (dcgmHandle_t dcgmHandle,
dcgmFieldGroupInfo_t *fieldGroupInfo)
```

**Parameters****dcgmHandle**

IN: DCGM handle

**fieldGroupInfo**

IN/OUT: Info about all of the field groups that exist. .version should be set to [dcgmFieldGroupInfo\\_version](#) before this call .fieldGroupId should contain the fieldGroupId you are interested in querying information for.

**Returns**

- ▶ DCGM\_ST\_OK if the field group info was returned successfully
- ▶ DCGM\_ST\_BADPARAM if any parameters were bad
- ▶ DCGM\_ST\_INIT\_ERROR if the library has not been successfully initialized.
- ▶ DCGM\_ST\_VER\_MISMATCH if .version is not set or is invalid.

**Description**

Used to get information about a field group that was created with [dcgmFieldGroupCreate](#).

```
dcgmReturn_t dcgmFieldGroupGetAll (dcgmHandle_t dcgmHandle,
dcgmAllFieldGroup_t *allGroupInfo)
```

**Parameters****dcgmHandle**

IN: DCGM handle

**allGroupInfo**

IN/OUT: Info about all of the field groups that exist. `.version` should be set to `dcgmAllFieldGroup_version` before this call.

**Returns**

- ▶ DCGM\_ST\_OK if the field group info was successfully returned
- ▶ DCGM\_ST\_BADPARAM if any parameters were bad
- ▶ DCGM\_ST\_INIT\_ERROR if the library has not been successfully initialized.
- ▶ DCGM\_ST\_VER\_MISMATCH if `.version` is not set or is invalid.

**Description**

Used to get information about all field groups in the system.

## 1.2.4. Status handling

System

The following APIs are used to manage statuses for multiple operations on one or more GPUs.

`dcgmReturn_t dcgmStatusCreate (dcgmStatus_t *statusHandle)`

**Parameters****statusHandle**

OUT: Reference to handle for list of statuses

**Returns**

- ▶ DCGM\_ST\_OK if the status handle is successfully created
- ▶ DCGM\_ST\_BADPARAM if statusHandle is invalid

**Description**

Creates reference to DCGM status handler which can be used to get the statuses for multiple operations on one or more devices.

The multiple statuses are useful when the operations are performed at group level. The status handle provides a mechanism to access error attributes for the failed operations.

The number of errors stored behind the opaque handle can be accessed using the the API `dcgmStatusGetCount`. The errors are accessed from the opaque handle statusHandle using the API `dcgmStatusPopError`. The user can invoke `dcgmStatusPopError` for the number of errors or until all the errors are fetched.

When the status handle is not required any further then it should be deleted using the API `dcgmStatusDestroy`.

`dcgmReturn_t dcgmStatusDestroy (dcgmStatus_t statusHandle)`

### Parameters

#### `statusHandle`

IN: Handle to list of statuses

### Returns

- ▶ `DCGM_ST_OK` if the status handle is successfully created
- ▶ `DCGM_ST_BADPARAM` if `statusHandle` is invalid

### Description

Used to destroy status handle created using `dcgmStatusCreate`.

`dcgmReturn_t dcgmStatusGetCount (dcgmStatus_t statusHandle, unsigned int *count)`

### Parameters

#### `statusHandle`

IN: Handle to list of statuses

#### `count`

OUT: Number of error entries present in the list of statuses

### Returns

- ▶ `DCGM_ST_OK` if the error count is successfully received
- ▶ `DCGM_ST_BADPARAM` if any of `statusHandle` or `count` is invalid

### Description

Used to get count of error entries stored inside the opaque handle `statusHandle`.

`dcgmReturn_t dcgmStatusPopError (dcgmStatus_t statusHandle, dcgmErrorInfo_t *pDcgmErrorInfo)`

### Parameters

#### `statusHandle`

IN: Handle to list of statuses

**pDcgmErrorInfo**

OUT: First error from the list of statuses

**Returns**

- ▶ DCGM\_ST\_OK if the error entry is successfully fetched
- ▶ DCGM\_ST\_BADPARAM if any of statusHandle or pDcgmErrorInfo is invalid
- ▶ DCGM\_ST\_NO\_DATA if the status handle list is empty

**Description**

Used to iterate through the list of errors maintained behind statusHandle. The method pops the first error from the list of DCGM statuses. In order to iterate through all the errors, the user can invoke this API for the number of errors or until all the errors are fetched.

**dcgmReturn\_t dcgmStatusClear (dcgmStatus\_t statusHandle)****Parameters****statusHandle**

IN: Handle to list of statuses

**Returns**

- ▶ DCGM\_ST\_OK if the errors are successfully cleared
- ▶ DCGM\_ST\_BADPARAM if statusHandle is invalid

**Description**

Used to clear all the errors in the status handle created by the API [dcgmStatusCreate](#). After one set of operation, the statusHandle can be cleared and reused for the next set of operation.

## 1.3. Configuration

This chapter describes the methods that handle device configuration retrieval and default settings. The APIs in Configuration module can be broken down into following categories:

### Setup and management

### Manual Invocation

### 1.3.1. Setup and management

#### Configuration

Describes APIs to Get/Set configuration on the group of GPUs.

`dcgmReturn_t dcgmConfigSet (dcgmHandle_t pDcgmHandle, dcgmGpuGrp_t groupId, dcgmConfig_t *pDeviceConfig, dcgmStatus_t statusHandle)`

#### Parameters

##### **pDcgmHandle**

IN: DCGM Handle

##### **groupId**

IN: Group ID representing collection of one or more GPUs. Look at [dcgmGroupCreate](#) for details on creating the group.

##### **pDeviceConfig**

IN: Pointer to memory to hold desired configuration to be applied for all the GPU in the group represented by groupId. The caller must populate the version field of pDeviceConfig.

##### **statusHandle**

IN/OUT: Resulting error status for multiple operations. Pass it as NULL if the detailed error information is not needed. Look at [dcgmStatusCreate](#) for details on creating status handle.

#### Returns

- ▶ DCGM\_ST\_OK if the configuration has been successfully set.
- ▶ DCGM\_ST\_BADPARAM if any of groupId or pDeviceConfig is invalid.
- ▶ DCGM\_ST\_VER\_MISMATCH if pDeviceConfig has the incorrect version.
- ▶ DCGM\_ST\_GENERIC\_ERROR if an unknown error has occurred.

#### Description

Used to set configuration for the group of one or more GPUs identified by groupId.

The configuration settings specified in pDeviceConfig are applied to all the GPUs in the group. Since DCGM group is a logical grouping of GPUs, the configuration settings stays intact for the individual GPUs even after the group is destroyed.

If the user wishes to ignore the configuration of one or more properties in the input pDeviceConfig then the property should be specified as one of DCGM\_INT32\_BLANK, DCGM\_INT64\_BLANK, DCGM\_FP64\_BLANK or DCGM\_STR\_BLANK based on the data type of the property to be ignored.

If any of the properties fail to be configured for any of the GPUs in the group then the API returns an error. The status handle `statusHandle` should be further evaluated to access error attributes for the failed operations. Please refer to status management APIs at [Status handling](#) to access the error attributes.

To find out valid supported clock values that can be passed to `dcgmConfigSet`, look at the device attributes of a GPU in the group using the API `dcgmGetDeviceAttributes`.

```
dcgmReturn_t dcgmConfigGet (dcgmHandle_t pDcgmHandle,
dcgmGpuGrp_t groupId, dcgmConfigType_t type, int count, dcgmConfig_t
deviceConfigList, dcgmStatus_t statusHandle)
```

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **groupId**

IN: Group ID representing collection of one or more GPUs. Look at [dcgmGroupCreate](#) for details on creating the group.

#### **type**

IN: Type of configuration values to be fetched.

#### **count**

IN: The number of entries that `deviceConfigList` array can store.

#### **deviceConfigList**

OUT: Pointer to memory to hold requested configuration corresponding to all the GPUs in the group (`groupId`). The size of the memory must be greater than or equal to hold output information for the number of GPUs present in the group (`groupId`).

#### **statusHandle**

IN/OUT: Resulting error status for multiple operations. Pass it as NULL if the detailed error information is not needed. Look at [dcgmStatusCreate](#) for details on creating status handle.

### Returns

- ▶ DCGM\_ST\_OK if the configuration has been successfully fetched.
- ▶ DCGM\_ST\_BADPARAM if any of `groupId`, `type`, `count`, or `deviceConfigList` is invalid.
- ▶ DCGM\_ST\_NOT\_CONFIGURED if the target configuration is not already set.
- ▶ DCGM\_ST\_VER\_MISMATCH if `deviceConfigList` has the incorrect version.
- ▶ DCGM\_ST\_GENERIC\_ERROR if an unknown error has occurred.

### Description

Used to get configuration for all the GPUs present in the group.



This API can get the most recent target or desired configuration set by [dcgmConfigSet](#). Set type as `DCGM_CONFIG_TARGET_STATE` to get target configuration. The target configuration properties are maintained by DCGM and are automatically enforced after a GPU reset or reinitialization is completed.

The method can also be used to get the actual configuration state for the GPUs in the group. Set type as `DCGM_CONFIG_CURRENT_STATE` to get the actual configuration state. Ideally, the actual configuration state will be exact same as the target configuration state.

If any of the property in the target configuration is unknown then the property value in the output is populated as one of `DCGM_INT32_BLANK`, `DCGM_INT64_BLANK`, `DCGM_FP64_BLANK` or `DCGM_STR_BLANK` based on the data type of the property.

If any of the property in the current configuration state is not supported then the property value in the output is populated as one of `DCGM_INT32_NOT_SUPPORTED`, `DCGM_INT64_NOT_SUPPORTED`, `DCGM_FP64_NOT_SUPPORTED` or `DCGM_STR_NOT_SUPPORTED` based on the data type of the property.

If any of the properties can't be fetched for any of the GPUs in the group then the API returns an error. The status handle `statusHandle` should be further evaluated to access error attributes for the failed operations. Please refer to status management APIs at [Status handling](#) to access the error attributes.

### 1.3.2. Manual Invocation

#### Configuration

Describes APIs used to manually enforce the desired configuration on a group of GPUs.

```
dcgmReturn_t dcgmConfigEnforce (dcgmHandle_t pDcgmHandle,
dcgmGpuGrp_t groupId, dcgmStatus_t statusHandle)
```

#### Parameters

##### **pDcgmHandle**

IN: DCGM Handle

##### **groupId**

IN: Group ID representing collection of one or more GPUs. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as `DCGM_GROUP_ALL_GPUS` to perform operation on all the GPUs.

##### **statusHandle**

IN/OUT: Resulting error status for multiple operations. Pass it as NULL if the detailed error information is not needed. Look at [dcgmStatusCreate](#) for details on creating status handle.

**Returns**

- ▶ DCGM\_ST\_OK if the configuration has been successfully enforced.
- ▶ DCGM\_ST\_BADPARAM if groupId is invalid.
- ▶ DCGM\_ST\_NOT\_CONFIGURED if the target configuration is not already set.
- ▶ DCGM\_ST\_GENERIC\_ERROR if an unknown error has occurred.

**Description**

Used to enforce previously set configuration for all the GPUs present in the group.

This API provides a mechanism to the users to manually enforce the configuration at any point of time. The configuration can only be enforced if it's already configured using the API [dcgmConfigSet](#).

If any of the properties can't be enforced for any of the GPUs in the group then the API returns an error. The status handle statusHandle should be further evaluated to access error attributes for the failed operations. Please refer to status management APIs at [Status handling](#) to access the error attributes.

## 1.4. Field APIs

These APIs are responsible for watching, unwatching, and updating specific fields as defined by DCGM\_FI\_\*

**dcgmReturn\_t dcgmWatchFields (dcgmHandle\_t pDcgmHandle, dcgmGpuGrp\_t groupId, dcgmFieldGrp\_t fieldGroupId, long long updateFreq, double maxKeepAge, int maxKeepSamples)**

**Parameters****pDcgmHandle**

IN: DCGM Handle

**groupId**

IN: Group ID representing collection of one or more entities. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs or DCGM\_GROUP\_ALL\_NVSWITCHES to to perform the operation on all NvSwitches.

**fieldGroupId**

IN: Fields to watch.

**updateFreq**

IN: How often to update this field in usec

**maxKeepAge**

IN: How long to keep data for this field in seconds

**maxKeepSamples**

IN: Maximum number of samples to keep. 0=no limit

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid

**Description**

Request that DCGM start recording updates for a given field collection.

Note that the first update of the field will not occur until the next field update cycle. To force a field update cycle, call `dcgmUpdateAllFields(1)`.

`dcgmReturn_t dcgmUnwatchFields (dcgmHandle_t pDcgmHandle, dcgmGpuGrp_t groupId, dcgmFieldGrp_t fieldGroupId)`

**Parameters****pDcgmHandle**

IN: DCGM Handle

**groupId**

IN: Group ID representing collection of one or more entities. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as `DCGM_GROUP_ALL_GPUS` to perform operation on all the GPUs or `DCGM_GROUP_ALL_NVSWITCHES` to to perform the operation on all NvSwitches.

**fieldGroupId**

IN: Fields to unwatch.

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid

**Description**

Request that DCGM stop recording updates for a given field collection.

`dcgmReturn_t dcgmGetValuesSince (dcgmHandle_t pDcgmHandle, dcgmGpuGrp_t groupId, dcgmFieldGrp_t fieldGroupId,`

```
long long sinceTimestamp, long long *nextSinceTimestamp,
dcmFieldValueEnumeration_f enumCB, void *userData)
```

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **groupId**

IN: Group ID representing collection of one or more GPUs. Look at [dcmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs.

#### **fieldGroupId**

IN: Fields to return data for

#### **sinceTimestamp**

IN: Timestamp to request values since in usec since 1970. This will be returned in nextSinceTimestamp for subsequent calls 0 = request all data

#### **nextSinceTimestamp**

OUT: Timestamp to use for sinceTimestamp on next call to this function

#### **enumCB**

IN: Callback to invoke for every field value update. Note that multiple updates can be returned in each invocation

#### **userData**

IN: User data pointer to pass to the userData field of enumCB.

### Returns

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_NOT\_SUPPORTED if one of the entities was from a non-GPU type
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid

### Description

Request updates for all field values that have updated since a given timestamp

This version only works with GPU entities. Use [dcmGetValuesSince\\_v2](#) for entity groups containing NvSwitches.

```
dcmReturn_t dcmGetValuesSince_v2 (dcmHandle_t
pDcmHandle, dcmGpuGrp_t groupId, dcmFieldGrp_t
fieldGroupId, long long sinceTimestamp, long long
```

`*nextSinceTimestamp, dcgmFieldValueEnumeration_f enumCB, void *userData)`

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **groupId**

IN: Group ID representing collection of one or more entities. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs or DCGM\_GROUP\_ALL\_NVSITCHES to perform the operation on all NvSwitches.

#### **fieldGroupId**

IN: Fields to return data for

#### **sinceTimestamp**

IN: Timestamp to request values since in usec since 1970. This will be returned in nextSinceTimestamp for subsequent calls 0 = request all data

#### **nextSinceTimestamp**

OUT: Timestamp to use for sinceTimestamp on next call to this function

#### **enumCB**

IN: Callback to invoke for every field value update. Note that multiple updates can be returned in each invocation

#### **userData**

IN: User data pointer to pass to the userData field of enumCB.

### Returns

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid

### Description

Request updates for all field values that have updated since a given timestamp

This version works with non-GPU entities like NvSwitches

`dcgmReturn_t dcgmGetLatestValues (dcgmHandle_t pDcgmHandle, dcgmGpuGrp_t groupId, dcgmFieldGrp_t fieldGroupId, dcgmFieldValueEnumeration_f enumCB, void *userData)`

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

**groupId**

IN: Group ID representing collection of one or more GPUs. Look at [dcmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs.

**fieldGroupId**

IN: Fields to return data for.

**enumCB**

IN: Callback to invoke for every field value update. Note that multiple updates can be returned in each invocation

**userData**

IN: User data pointer to pass to the userData field of enumCB.

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_NOT\_SUPPORTED if one of the entities was from a non-GPU type
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid

**Description**

Request latest cached field value for a field value collection

This version only works with GPU entities. Use [dcmGetLatestValues\\_v2](#) for entity groups containing NvSwitches.

```
dcmReturn_t dcmGetLatestValues_v2 (dcmHandle_t
pDcmHandle, dcmGpuGrp_t groupId, dcmFieldGrp_t
fieldGroupId, dcmFieldValueEnumeration_f enumCB, void
*userData)
```

**Parameters****pDcmHandle**

IN: DCGM Handle

**groupId**

IN: Group ID representing collection of one or more entities. Look at [dcmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs or DCGM\_GROUP\_ALL\_NVSWITCHES to perform the operation on all NvSwitches.

**fieldGroupId**

IN: Fields to return data for.

**enumCB**

IN: Callback to invoke for every field value update. Note that multiple updates can be returned in each invocation

**userData**

IN: User data pointer to pass to the userData field of enumCB.

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_NOT\_SUPPORTED if one of the entities was from a non-GPU type
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid

**Description**

Request latest cached field value for a field value collection

This version works with non-GPU entities like NvSwitches

`dcgmReturn_t dcgmGetLatestValuesForFields (dcgmHandle_t pDcgmHandle, int gpuId, unsigned short fields, unsigned int count, dcgmFieldValue_v1 values)`

**Parameters****pDcgmHandle**

IN: DCGM Handle

**gpuId**

IN: Gpu ID representing the GPU for which the fields are being requested.

**fields**

IN: Field IDs to return data for. See the definitions in `dcgm_fields.h` that start with `DCGM_FL_`.

**count**

IN: Number of field IDs in `fields[]` array.

**values**

OUT: Latest field values for the fields in `fields[]`.

**Description**

Request latest cached field value for a GPU

`dcgmReturn_t dcgmEntityGetLatestValues (dcgmHandle_t pDcgmHandle, dcgm_field_entity_group_t entityGroup, int entityId,`

unsigned short fields, unsigned int count, dcgmFieldValue\_v1 values)

#### Parameters

##### **pDcgmHandle**

IN: DCGM Handle

##### **entityGroup**

IN: entity\_group\_t (e.g. switch)

##### **entityId**

IN: entity ID representing the entity for which the fields are being requested.

##### **fields**

IN: Field IDs to return data for. See the definitions in dcgm\_fields.h that start with DCGM\_FI\_.

##### **count**

IN: Number of field IDs in fields[] array.

##### **values**

OUT: Latest field values for the fields in fields[].

#### Description

Request latest cached field value for a group of fields for a specific entity

**dcgmReturn\_t dcgmEntitiesGetLatestValues (dcgmHandle\_t pDcgmHandle, dcgmGroupEntityPair\_t entities, unsigned int entityCount, unsigned short fields, unsigned int fieldCount, unsigned int flags, dcgmFieldValue\_v2 values)**

#### Parameters

##### **pDcgmHandle**

IN: DCGM Handle

##### **entities**

IN: List of entities to get values for

##### **entityCount**

IN: Number of entries in entities[]

##### **fields**

IN: Field IDs to return data for. See the definitions in dcgm\_fields.h that start with DCGM\_FI\_.

##### **fieldCount**

IN: Number of field IDs in fields[] array.



**flags**

IN: Optional flags that affect how this request is processed. Pass

`DCGM_FV_FLAG_LIVE_DATA` here to retrieve a live driver value rather than a cached value. See that flag's documentation for caveats.

**values**

OUT: Latest field values for the fields requested. This must be able to hold `entityCount * fieldCount` field value records.

**Description**

Request the latest cached or live field value for a list of fields for a group of entities

Note: The returned entities are not guaranteed to be in any order. Reordering can occur internally in order to optimize calls to the NVIDIA driver.

`dcgmReturn_t dcgmGetFieldSummary (dcgmHandle_t pDcgmHandle, dcgmFieldSummaryRequest_t *request)`

**Parameters****pDcgmHandle**

IN: DCGM Handle

**request**

IN/OUT: a pointer to the struct detailing the request and containing the response

**Returns**

- ▶ `DCGM_ST_OK` if the call was successful
- ▶ `DCGM_ST_FIELD_UNSUPPORTED_BY_API` if the field is not int64 or double type

**Description**

Get a summary of the values for a field id over a period of time.

## 1.5. Process Statistics

Describes APIs to investigate statistics such as accounting, performance and errors during the lifetime of a GPU process

`dcgmReturn_t dcgmWatchPidFields (dcgmHandle_t pDcgmHandle, dcgmGpuGrp_t groupId, long long updateFreq, double maxKeepAge, int maxKeepSamples)`

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **groupId**

IN: Group ID representing collection of one or more GPUs. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs.

#### **updateFreq**

IN: How often to update this field in usec

#### **maxKeepAge**

IN: How long to keep data for this field in seconds

#### **maxKeepSamples**

IN: Maximum number of samples to keep. 0=no limit

### Returns

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid
- ▶ DCGM\_ST\_REQUIRES\_ROOT if the host engine is being run as non-root, and accounting mode could not be enabled (requires root). Run "nvidia-smi -am 1" as root on the node before starting DCGM to fix this.

### Description

Request that DCGM start recording stats for fields that can be queried with [dcgmGetPidInfo\(\)](#).

Note that the first update of the field will not occur until the next field update cycle. To force a field update cycle, call [dcgmUpdateAllFields\(1\)](#).

`dcgmReturn_t dcgmGetPidInfo (dcgmHandle_t pDcgmHandle, dcgmGpuGrp_t groupId, dcgmPidInfo_t *pidInfo)`

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

**groupId**

IN: Group ID representing collection of one or more GPUs. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs.

**pidInfo**

IN/OUT: Structure to return information about pid in. pidInfo->pid must be set to the pid in question. pidInfo->version should be set to dcgmPidInfo\_version.

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_NO\_DATA if the PID did not run on any GPU

**Description**

Get information about all GPUs while the provided pid was running

In order for this request to work, you must first call [dcgmWatchPidFields\(\)](#) to make sure that DCGM is watching the appropriate field IDs that will be populated in pidInfo

## 1.6. Job Statistics

The client can invoke DCGM APIs to start and stop collecting the stats at the process boundaries (during prologue and epilogue). This will enable DCGM to monitor all the PIDs while the job is in progress, and provide a summary of active processes and resource usage during the window of interest.

[dcgmReturn\\_t dcgmWatchJobFields \(dcgmHandle\\_t pDcgmHandle, dcgmGpuGrp\\_t groupId, long long updateFreq, double maxKeepAge, int maxKeepSamples\)](#)

**Parameters****pDcgmHandle**

IN: DCGM Handle

**groupId**

IN: Group ID representing collection of one or more GPUs. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs.

**updateFreq**

IN: How often to update this field in usec

**maxKeepAge**

IN: How long to keep data for this field in seconds

**maxKeepSamples**

IN: Maximum number of samples to keep. 0=no limit

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid
- ▶ DCGM\_ST\_REQUIRES\_ROOT if the host engine is being run as non-root, and accounting mode could not be enabled (requires root). Run "nvidia-smi -am 1" as root on the node before starting DCGM to fix this.

**Description**

Request that DCGM start recording stats for fields that are queried with [dcgmJobGetStats\(\)](#)

Note that the first update of the field will not occur until the next field update cycle. To force a field update cycle, call [dcgmUpdateAllFields\(1\)](#).

**dcgmReturn\_t dcgmJobStartStats (dcgmHandle\_t pDcgmHandle, dcgmGpuGrp\_t groupId, char jobId)**

**Parameters****pDcgmHandle**

IN: DCGM Handle

**groupId**

IN: Group ID representing collection of one or more GPUs. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs.

**jobId**

IN: User provided string to represent the job

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid
- ▶ DCGM\_ST\_DUPLICATE\_KEY if the specified jobId is already in use

**Description**

This API is used by the client to notify DCGM about the job to be started. Should be invoked as part of job prologue

`dcgmReturn_t dcgmJobStopStats (dcgmHandle_t pDcgmHandle, char jobId)`

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **jobId**

IN: User provided string to represent the job

### Returns

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid
- ▶ DCGM\_ST\_NO\_DATA if jobId is not a valid job identifier.

### Description

This API is used by the clients to notify DCGM to stop collecting stats for the job represented by job id. Should be invoked as part of job epilogue. The job Id remains available to view the stats at any point but cannot be used to start a new job. You must call `dcgmWatchJobFields()` before this call to enable watching of job

`dcgmReturn_t dcgmJobGetStats (dcgmHandle_t pDcgmHandle, char jobId, dcgmJobInfo_t *pJobInfo)`

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **jobId**

IN: User provided string to represent the job

#### **pJobInfo**

IN/OUT: Structure to return information about the job. `.version` should be set to `dcgmJobInfo_version` before this call.

### Returns

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid
- ▶ DCGM\_ST\_NO\_DATA if jobId is not a valid job identifier.
- ▶ DCGM\_ST\_VER\_MISMATCH if `.version` is not set or is invalid.

**Description**

Get stats for the job identified by DCGM generated job id. The stats can be retrieved at any point when the job is in process. If you want to reuse this jobId, call `dcgmJobRemove` after this call.

`dcgmReturn_t dcgmJobRemove (dcgmHandle_t pDcgmHandle, char jobId)`

**Parameters****pDcgmHandle**

IN: DCGM Handle

**jobId**

IN: User provided string to represent the job

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid
- ▶ DCGM\_ST\_NO\_DATA if jobId is not a valid job identifier.

**Description**

This API tells DCGM to stop tracking the job given by jobId. After this call, you will no longer be able to call `dcgmJobGetStats()` on this jobId. However, you will be able to reuse jobId after this call.

`dcgmReturn_t dcgmJobRemoveAll (dcgmHandle_t pDcgmHandle)`

**Parameters****pDcgmHandle**

IN: DCGM Handle

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid

**Description**

This API tells DCGM to stop tracking all jobs. After this call, you will no longer be able to call `dcgmJobGetStats()` any jobs until you call `dcgmJobStartStats` again. You will be able to reuse any previously-used jobIds after this call.

## 1.7. Health Monitor

This chapter describes the methods that handle the GPU health monitor.

`dcgmReturn_t dcgmHealthSet (dcgmHandle_t pDcgmHandle, dcgmGpuGrp_t groupId, dcgmHealthSystems_t systems)`

### Parameters

#### `pDcgmHandle`

IN: DCGM Handle

#### `groupId`

IN: Group ID representing collection of one or more entities. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as `DCGM_GROUP_ALL_GPUS` to perform operation on all the GPUs or `DCGM_GROUP_ALL_NVSWITCHES` to perform operation on all the NvSwitches.

#### `systems`

IN: An enum representing systems that should be enabled for health checks logically OR'd together. Refer to [dcgmHealthSystems\\_t](#) for details.

### Returns

- ▶ `DCGM_ST_OK` if the call was successful
- ▶ `DCGM_ST_BADPARAM` if a parameter is invalid

### Description

Enable the DCGM health check system for the given systems defined in [dcgmHealthSystems\\_t](#)

`dcgmReturn_t dcgmHealthSet_v2 (dcgmHandle_t pDcgmHandle, dcgmHealthSetParams_v2 *params[])`

### Parameters

#### `pDcgmHandle`

IN: DCGM Handle

#### `params`

### Returns

- ▶ `DCGM_ST_OK` if the call was successful
- ▶ `DCGM_ST_BADPARAM` if a parameter is invalid

**Description**

Enable the DCGM health check system for the given systems defined in [dcgmHealthSystems\\_t](#)

Since DCGM 2.0

**dcgmReturn\_t dcgmHealthGet (dcgmHandle\_t pDcgmHandle, dcgmGpuGrp\_t groupId, dcgmHealthSystems\_t \*systems)**

**Parameters****pDcgmHandle**

IN: DCGM Handle

**groupId**

IN: Group ID representing collection of one or more entities. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs or DCGM\_GROUP\_ALL\_NVSITCHES to perform operation on all the NvSwitches.

**systems**

OUT: An integer representing the enabled systems for the given group Refer to [dcgmHealthSystems\\_t](#) for details.

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid

**Description**

Retrieve the current state of the DCGM health check system

**dcgmReturn\_t dcgmHealthCheck (dcgmHandle\_t pDcgmHandle, dcgmGpuGrp\_t groupId, dcgmHealthResponse\_t \*results)**

**Parameters****pDcgmHandle**

IN: DCGM Handle

**groupId**

IN: Group ID representing a collection of one or more entities. Refer to [dcgmGroupCreate](#) for details on creating a group

**results**

OUT: A reference to the dcgmHealthResponse\_t structure to populate. results->version must be set to dcgmHealthResponse\_version.



**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid
- ▶ DCGM\_ST\_VER\_MISMATCH if results->version is not dcgmHealthResponse\_version

**Description**

Check the configured watches for any errors/failures/warnings that have occurred since the last time this check was invoked. On the first call, stateful information about all of the enabled watches within a group is created but no error results are provided. On subsequent calls, any error information will be returned.

## 1.8. Policies

This chapter describes the methods that handle system policy management and violation settings. The APIs in Policies module can be broken down into following categories:

### Setup and Management

#### Manual Invocation

##### 1.8.1. Setup and Management

**Policies**

Describes APIs for setting up policies and registering callbacks to receive notification in case specific policy condition has been violated.

```
dcgmReturn_t dcgmPolicySet (dcgmHandle_t pDcgmHandle,
dcgmGpuGrp_t groupId, dcgmPolicy_t *policy, dcgmStatus_t
statusHandle)
```

**Parameters****pDcgmHandle**

IN: DCGM Handle

**groupId**

IN: Group ID representing collection of one or more GPUs. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs.

**policy**

IN: A reference to [dcgmPolicy\\_t](#) that will be applied to all GPUs in the group.

**statusHandle**

IN/OUT: Resulting status for the operation. Pass it as NULL if the detailed error information is not needed. Refer to [dcgmStatusCreate](#) for details on creating a status handle.

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if groupId or policy is invalid
- ▶ DCGM\_ST\_NOT\_SUPPORTED if any unsupported GPUs are part of the GPU group specified in groupId
- ▶ DCGM\_ST\_\* a different error has occurred and is stored in statusHandle. Refer to [dcgmReturn\\_t](#)

**Description**

Set the current violation policy inside the policy manager. Given the conditions within the [dcgmPolicy\\_t](#) structure, if a violation has occurred, subsequent action(s) may be performed to either report or contain the failure.

[dcgmReturn\\_t](#) [dcgmPolicyGet](#) ([dcgmHandle\\_t](#) pDcgmHandle, [dcgmGpuGrp\\_t](#) groupId, int count, [dcgmPolicy\\_t](#) \*policy, [dcgmStatus\\_t](#) statusHandle)

**Parameters****pDcgmHandle**

IN: DCGM Handle

**groupId**

IN: Group ID representing collection of one or more GPUs. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs.

**count**

IN: The size of the policy array. This is the maximum number of policies that will be retrieved and ultimately should correspond to the number of GPUs specified in the group.

**policy**

OUT: A reference to [dcgmPolicy\\_t](#) that will be used as storage for the current policies applied to each GPU in the group.

**statusHandle**

IN/OUT: Resulting status for the operation. Pass it as NULL if the detailed error information for the operation is not needed. Refer to [dcgmStatusCreate](#) for details on creating a status handle.

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if groupId or policy is invalid
- ▶ DCGM\_ST\_\* a different error has occurred and is stored in statusHandle. Refer to `dcgmReturn_t`

**Description**

Get the current violation policy inside the policy manager. Given a groupId, a number of policy structures are retrieved.

```
dcgmReturn_t dcgmPolicyRegister (dcgmHandle_t pDcgmHandle,
dcgmGpuGrp_t groupId, dcgmPolicyCondition_t condition, fpRecvUpdates
beginCallback, fpRecvUpdates finishCallback)
```

**Parameters****pDcgmHandle**

IN: DCGM Handle

**groupId**

IN: Group ID representing collection of one or more GPUs. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs.

**condition**

IN: The set of conditions specified as an OR'd list (see [dcgmPolicyCondition\\_t](#)) for which to register a callback function

**beginCallback**

IN: A reference to a function that should be called should a violation occur. This function will be called prior to any actions specified by the policy are taken.

**finishCallback**

IN: A reference to a function that should be called should a violation occur. This function will be called after any action specified by the policy are completed.

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if groupId, condition, is invalid, beginCallback, or finishCallback is NULL
- ▶ DCGM\_ST\_NOT\_SUPPORTED if any unsupported GPUs are part of the GPU group specified in groupId

## Description

Register a function to be called when a specific policy condition (see [dcgmPolicyCondition\\_t](#)) has been violated. This callback(s) will be called automatically when in DCGM\_OPERATION\_MODE\_AUTO mode and only after `dcgmPolicyTrigger` when in DCGM\_OPERATION\_MODE\_MANUAL mode. All callbacks are made within a separate thread.

`dcgmReturn_t dcgmPolicyUnregister (dcgmHandle_t pDcgmHandle, dcgmGpuGrp_t groupId, dcgmPolicyCondition_t condition)`

## Parameters

### `pDcgmHandle`

IN: DCGM Handle

### `groupId`

IN: Group ID representing collection of one or more GPUs. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs.

### `condition`

IN: The set of conditions specified as an OR'd list (see [dcgmPolicyCondition\\_t](#)) for which to unregister a callback function

## Returns

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if `groupId`, `condition`, is invalid or callback is NULL

## Description

Unregister a function to be called for a specific policy condition (see [dcgmPolicyCondition\\_t](#)). This function will unregister all callbacks for a given condition and handle.

## 1.8.2. Manual Invocation

### Policies

Describes APIs which can be used to perform direct actions (e.g. Perform GPU Reset, Run Health Diagnostics) on a group of GPUs.

`dcgmReturn_t dcgmActionValidate (dcgmHandle_t pDcgmHandle, dcgmGpuGrp_t groupId, dcgmPolicyValidation_t validate, dcgmDiagResponse_t *response)`

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **groupId**

IN: Group ID representing collection of one or more GPUs. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs.

#### **validate**

IN: The validation to perform after the action.

#### **response**

OUT: Result of the validation process. Refer to [dcgmDiagResponse\\_t](#) for details.

### Returns

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_NOT\_SUPPORTED if running the specified validate is not supported. This is usually due to the Tesla recommended driver not being installed on the system.
- ▶ DCGM\_ST\_BADPARAM if groupId, validate, or statusHandle is invalid
- ▶ DCGM\_ST\_GENERIC\_ERROR an internal error has occurred
- ▶ DCGM\_ST\_GROUP\_INCOMPATIBLE if groupId refers to a group of non-homogeneous GPUs. This is currently not allowed.

### Description

Inform the action manager to perform a manual validation of a group of GPUs on the system

\*\*\*\*\* DEPRECATED \*\*\*\*\*

`dcgmReturn_t dcgmActionValidate_v2 (dcgmHandle_t pDcgmHandle, dcgmRunDiag_v7 *drd, dcgmDiagResponse_t *response)`

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

**drd**

IN: Contains the group id, test names, test parameters, struct version, and the validation that should be performed. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs.

**response**

OUT: Result of the validation process. Refer to [dcgmDiagResponse\\_t](#) for details.

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_NOT\_SUPPORTED if running the specified validate is not supported. This is usually due to the Tesla recommended driver not being installed on the system.
- ▶ DCGM\_ST\_BADPARAM if groupId, validate, or statusHandle is invalid
- ▶ DCGM\_ST\_GENERIC\_ERROR an internal error has occurred
- ▶ DCGM\_ST\_GROUP\_INCOMPATIBLE if groupId refers to a group of non-homogeneous GPUs. This is currently not allowed.

**Description**

Inform the action manager to perform a manual validation of a group of GPUs on the system

```
dcgmReturn_t dcgmRunDiagnostic (dcgmHandle_t pDcgmHandle,
dcgmGpuGrp_t groupId, dcgmDiagnosticLevel_t diagLevel,
dcgmDiagResponse_t *diagResponse)
```

**Parameters****pDcgmHandle**

IN: DCGM Handle

**groupId**

IN: Group ID representing collection of one or more GPUs. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs.

**diagLevel**

IN: Diagnostic level to run

**diagResponse**

IN/OUT: Result of running the DCGM diagnostic. .version should be set to [dcgmDiagResponse\\_version](#) before this call.

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful

- ▶ DCGM\_ST\_NOT\_SUPPORTED if running the diagnostic is not supported. This is usually due to the Tesla recommended driver not being installed on the system.
- ▶ DCGM\_ST\_BADPARAM if a provided parameter is invalid or missing
- ▶ DCGM\_ST\_GENERIC\_ERROR an internal error has occurred
- ▶ DCGM\_ST\_GROUP\_INCOMPATIBLE if groupId refers to a group of non-homogeneous GPUs. This is currently not allowed.
- ▶ DCGM\_ST\_VER\_MISMATCH if .version is not set or is invalid.

### Description

Run a diagnostic on a group of GPUs

## 1.9. Topology

`dcgmReturn_t dcgmGetDeviceTopology (dcgmHandle_t pDcgmHandle, unsigned int gpuId, dcgmDeviceTopology_t *pDcgmDeviceTopology)`

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **gpuId**

IN: GPU Id corresponding to which topology information should be fetched

#### **pDcgmDeviceTopology**

IN/OUT: Topology information corresponding to gpuId. pDcgmDeviceTopology->version must be set to dcgmDeviceTopology\_version before this call.

### Returns

- ▶ DCGM\_ST\_OK if the call was successful.
- ▶ DCGM\_ST\_BADPARAM if gpuId or pDcgmDeviceTopology were not valid.
- ▶ DCGM\_ST\_VER\_MISMATCH if pDcgmDeviceTopology->version was not set to dcgmDeviceTopology\_version.

### Description

Gets device topology corresponding to the gpuId.

```
dcgmReturn_t dcgmGetGroupTopology (dcgmHandle_t
pDcgmHandle, dcgmGpuGrp_t groupId, dcgmGroupTopology_t
*pDcgmGroupTopology)
```

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **groupId**

IN: GroupId corresponding to which topology information should be fetched

#### **pDcgmGroupTopology**

IN/OUT: Topology information corresponding to groupId. pDcgmgroupTopology->version must be set to dcgmGroupTopology\_version.

### Returns

- ▶ DCGM\_ST\_OK if the call was successful.
- ▶ DCGM\_ST\_BADPARAM if groupId or pDcgmGroupTopology were not valid.
- ▶ DCGM\_ST\_VER\_MISMATCH if pDcgmgroupTopology->version was not set to dcgmGroupTopology\_version.

### Description

Gets group topology corresponding to the groupId.

## 1.10. Metadata

This chapter describes the methods that query for DCGM metadata.

```
dcgmReturn_t dcgmIntrospectToggleState (dcgmHandle_t
pDcgmHandle, dcgmIntrospectState_t enabledState)
```

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **enabledState**

IN: The state to set gathering of introspection data to

### Returns

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM enabledState is an invalid state for metadata gathering



**Description**

Toggle the state of introspection metadata gathering in DCGM. Metadata gathering will increase the memory usage of DCGM so that it can store the metadata it gathers.

```
dcgmReturn_t dcgmIntrospectGetFieldsMemoryUsage
(dcgmHandle_t pDcgmHandle, dcgmIntrospectContext_t *context,
dcgmIntrospectFullMemory_t *memoryInfo, int waitIfNoData)
```

**Parameters****pDcgmHandle**

IN: DCGM Handle

**context**

IN: see [dcgmIntrospectContext\\_t](#). This identifies the level of fields to do introspection for (ex: all fields, field groups) context->version must be set to dcgmIntrospectContext\_version prior to this call.

**memoryInfo**

IN/OUT: see [dcgmIntrospectFullMemory\\_t](#). memoryInfo->version must be set to dcgmIntrospectFullMemory\_version prior to this call.

**waitIfNoData**

IN: if no metadata has been gathered, should this call block until data has been gathered (1), or should this call just return DCGM\_ST\_NO\_DATA (0).

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_NOT\_CONFIGURED if metadata gathering state is DCGM\_INTROSPECT\_STATE\_DISABLED
- ▶ DCGM\_ST\_NO\_DATA if waitIfNoData is false and metadata has not been gathered yet
- ▶ DCGM\_ST\_VER\_MISMATCH if context->version or memoryInfo->version is 0 or invalid.

**Description**

Get the current amount of memory used to store the given field collection.

```
dcgmReturn_t dcgmIntrospectGetHostengineMemoryUsage
(dcgmHandle_t pDcgmHandle, dcgmIntrospectMemory_t
*memoryInfo, int waitIfNoData)
```

### Parameters

#### pDcgmHandle

IN: DCGM Handle

#### memoryInfo

IN/OUT: see [dcgmIntrospectMemory\\_t](#). memoryInfo->version must be set to dcgmIntrospectMemory\_version prior to this call.

#### waitIfNoData

IN: if no metadata is gathered wait till this occurs (!0) or return DCGM\_ST\_NO\_DATA (0)

### Returns

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_NOT\_CONFIGURED if metadata gathering state is DCGM\_INTROSPECT\_STATE\_DISABLED
- ▶ DCGM\_ST\_NO\_DATA if waitIfNoData is false and metadata has not been gathered yet
- ▶ DCGM\_ST\_VER\_MISMATCH if memoryInfo->version is 0 or invalid.

### Description

Retrieve the total amount of memory that the hostengine process is currently using. This measurement represents both the resident set size (what is currently in RAM) and the swapped memory that belongs to the process.

```
dcgmReturn_t dcgmIntrospectGetFieldsExecTime (dcgmHandle_t
pDcgmHandle, dcgmIntrospectContext_t *context,
dcgmIntrospectFullFieldsExecTime_t *execTime, int waitIfNoData)
```

### Parameters

#### pDcgmHandle

IN: DCGM Handle

#### context

IN: see [dcgmIntrospectContext\\_t](#). This identifies the level of fields to do introspection for (ex: all fields, field group ) context->version must be set to dcgmIntrospectContext\_version prior to this call.

**execTime**

IN/OUT: see `dcmIntropectFullFieldsExecTime_t`. `execTime->version` must be set to `dcmIntropectFullFieldsExecTime_version` prior to this call.

**waitIfNoData**

IN: if no metadata is gathered, wait until data has been gathered (1) or return `DCGM_ST_NO_DATA` (0)

**Returns**

- ▶ `DCGM_ST_OK` if the call was successful
- ▶ `DCGM_ST_NOT_CONFIGURED` if metadata gathering state is `DCGM_INTROSPECT_STATE_DISABLED`
- ▶ `DCGM_ST_NO_DATA` if `waitIfNoData` is false and metadata has not been gathered yet
- ▶ `DCGM_ST_VER_MISMATCH` if `context->version` or `execTime->version` is 0 or invalid.

**Description**

Get introspection info relating to execution time needed to update the fields identified by context.

`dcmReturn_t dcmIntropectGetHostengineCpuUtilization`  
(`dcmHandle_t pDcmHandle`, `dcmIntropectCpuUtil_t *cpuUtil`, `int waitIfNoData`)

**Parameters****pDcmHandle**

IN: DCGM Handle

**cpuUtil**

IN/OUT: see `dcmIntropectCpuUtil_t`. `cpuUtil->version` must be set to `dcmIntropectCpuUtil_version` prior to this call.

**waitIfNoData**

IN: if no metadata is gathered wait till this occurs (!0) or return `DCGM_ST_NO_DATA` (0)

**Returns**

- ▶ `DCGM_ST_OK` if the call was successful
- ▶ `DCGM_ST_NOT_CONFIGURED` if metadata gathering state is `DCGM_INTROSPECT_STATE_DISABLED`
- ▶ `DCGM_ST_NO_DATA` if `waitIfNoData` is false and metadata has not been gathered yet

- ▶ DCGM\_ST\_VER\_MISMATCH if cpuUtil->version or execTime->version is 0 or invalid.

### Description

Retrieve the CPU utilization of the DCGM hostengine process.

**dcgmReturn\_t dcgmIntrospectUpdateAll (dcgmHandle\_t pDcgmHandle, int waitForUpdate)**

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **waitForUpdate**

IN: Whether or not to wait for the update loop to complete before returning to the caller

### Returns

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if waitForUpdate is invalid

### Description

This method is used to manually tell the the introspection module to update all DCGM introspection data. This is normally performed automatically on an interval of 1 second.

## 1.11. Topology

This chapter describes the methods that query for DCGM topology information.

**dcgmReturn\_t dcgmSelectGpusByTopology (dcgmHandle\_t pDcgmHandle, uint64\_t inputGpuIds, uint32\_t numGpus, uint64\_t \*outputGpuIds, uint64\_t hintFlags)**

### Parameters

#### **pDcgmHandle**

IN: DCGM Handle

#### **inputGpuIds**

IN: a bitmask of which GPUs DCGM should consider. If some of the GPUs on the system are already in use, they shouldn't be included in the bitmask. 0 means that all of the GPUs in the system should be considered.

**numGpus**

IN: the number of GPUs that are desired from inputGpuIds. If this number is greater than the number of healthy GPUs in inputGpuIds, then less than numGpus gpus will be specified in outputGpuIds.

**outputGpuIds**

OUT: a bitmask of numGpus or fewer GPUs from inputGpuIds that represent the best placement available from inputGpuIds.

**hintFlags**

IN: a bitmask of DCGM\_TOPO\_HINT\_F\_ defines of hints that should be taken into account when assigning outputGpuIds.

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful

**Description**

Get the best group of gpus from the specified bitmask according to topological proximity: cpuAffinity, NUMA node, and NVLink.

## 1.12. Modules

This chapter describes the methods that query and configure DCGM modules.

### **dcgmReturn\_t dcgmModuleBlacklist (dcgmHandle\_t pDcgmHandle, dcgmModuleId\_t moduleId)**

**Parameters****pDcgmHandle**

IN: DCGM Handle

**moduleId**

IN: ID of the module to blacklist. Use [dcgmModuleGetStatuses](#) to get a list of valid module IDs.

**Returns**

- ▶ DCGM\_ST\_OK if the module has been blacklisted.
- ▶ DCGM\_ST\_IN\_USE if the module has already been loaded and cannot be blacklisted.
- ▶ DCGM\_ST\_BADPARAM if a parameter is missing or bad.

**Description**

Set a module to be blacklisted. This module will be prevented from being loaded if it hasn't been loaded already. Modules are lazy-loaded as they are used by DCGM APIs, so it's important to call this API soon after the host engine has been started. You can also pass `--blacklist-modules` to the `nv-hostengine` binary to make sure modules get blacklisted immediately after the host engine starts up.

`dcgmReturn_t dcgmModuleGetStatuses (dcgmHandle_t pDcgmHandle, dcgmModuleGetStatuses_t *moduleStatuses)`

**Parameters****pDcgmHandle**

IN: DCGM Handle

**moduleStatuses**

OUT: Module statuses. `.version` should be set to `dcgmModuleStatuses_version` upon calling.

**Returns**

- ▶ DCGM\_ST\_OK if the request succeeds.
- ▶ DCGM\_ST\_BADPARAM if a parameter is missing or bad.

**Description**

Get the status of all of the DCGM modules.

## 1.13. Profiling

This chapter describes the methods that watch profiling fields from within DCGM.

`dcgmReturn_t dcgmProfGetSupportedMetricGroups (dcgmHandle_t pDcgmHandle, dcgmProfGetMetricGroups_t *metricGroups)`

**Parameters****pDcgmHandle**

IN: DCGM Handle

**metricGroups**

IN/OUT: Metric groups supported for `metricGroups->groupId`. `metricGroups->version` should be set to `dcgmProfGetMetricGroups_version` upon calling.

## Returns

- ▶ DCGM\_ST\_OK if the request succeeds.
- ▶ DCGM\_ST\_BADPARAM if a parameter is missing or bad.
- ▶ DCGM\_ST\_GROUP\_INCOMPATIBLE if metricGroups->groupId's GPUs are not identical GPUs.
- ▶ DCGM\_ST\_NOT\_SUPPORTED if profiling metrics are not supported for the given GPU group.

## Description

Get all of the profiling metric groups for a given GPU group.

Profiling metrics are watched in groups of fields that are all watched together. For instance, if you want to watch DCGM\_FI\_PROF\_GR\_ENGINE\_ACTIVITY, this might also be in the same group as DCGM\_FI\_PROF\_SM\_EFFICIENCY. Watching this group would result in DCGM storing values for both of these metrics.

Some groups cannot be watched concurrently as others as they utilize the same hardware resource. For instance, you may not be able to watch DCGM\_FI\_PROF\_TENSOR\_OP\_UTIL at the same time as DCGM\_FI\_PROF\_GR\_ENGINE\_ACTIVITY on your hardware. At the same time, you may be able to watch DCGM\_FI\_PROF\_TENSOR\_OP\_UTIL at the same time as DCGM\_FI\_PROF\_NVLINK\_TX\_DATA.

Metrics that can be watched concurrently will have different .majorId fields in their [dcmProfMetricGroupInfo\\_t](#)

See [dcmGroupCreate](#) for details on creating a GPU group See [dcmProfWatchFields](#) to actually watch a metric group

**dcmReturn\_t dcmProfWatchFields (dcmHandle\_t pDcmHandle, dcmProfWatchFields\_t \*watchFields)**

## Parameters

### pDcmHandle

IN: DCGM Handle

### watchFields

IN: Details of which metric groups to watch for which GPUs. See [dcmProfWatchFields\\_v1](#) for details of what should be put in each struct member. watchFields->version should be set to dcmProfWatchFields\_version upon calling.

## Returns

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid

- ▶ DCGM\_ST\_NOT\_SUPPORTED if profiling metric group metricGroupTag is not supported for the given GPU group.
- ▶ DCGM\_ST\_GROUP\_INCOMPATIBLE if groupId's GPUs are not identical GPUs. Profiling metrics are only support for homogenous groups of GPUs.
- ▶ DCGM\_ST\_PROFILING\_MULTI\_PASS if any of the metric groups could not be watched concurrently due to requiring the hardware to gather them with multiple passes

### Description

Request that DCGM start recording updates for a given list of profiling field IDs.

Once metrics have been watched by this API, any of the normal DCGM field-value retrieval APIs can be used on the underlying fieldIds of this metric group. See [dcgmGetLatestValues\\_v2](#), [dcgmGetLatestValuesForFields](#), [dcgmEntityGetLatestValues](#), and [dcgmEntitiesGetLatestValues](#).

**dcgmReturn\_t dcgmProfUnwatchFields (dcgmHandle\_t pDcgmHandle, dcgmProfUnwatchFields\_t \*unwatchFields)**

### Parameters

#### pDcgmHandle

IN: DCGM Handle

#### unwatchFields

IN: Details of which metric groups to unwatch for which GPUs. See [dcgmProfUnwatchFields\\_v1](#) for details of what should be put in each struct member. unwatchFields->version should be set to dcgmProfUnwatchFields\_version upon calling.

### Returns

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid

### Description

Request that DCGM stop recording updates for all profiling field IDs for all GPUs

**dcgmReturn\_t dcgmProfPause (dcgmHandle\_t pDcgmHandle)**

### Parameters

#### pDcgmHandle

IN: DCGM Handle



**Returns**

- ▶ DCGM\_ST\_OK If the call was successful.
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid.

**Description**

Pause profiling activities in DCGM. This should be used when you are monitoring profiling fields from DCGM but want to be able to still run developer tools like nvprof, nsight systems, and nsight compute. Profiling fields start with DCGM\_PROF\_ and are in the field ID range 1001-1012.

Call this API before you launch one of those tools and `dcmProfResume()` after the tool has completed.

DCGM will save BLANK values while profiling is paused.

Calling this while profiling activities are already paused is fine and will be treated as a no-op.

**dcmReturn\_t dcmProfResume (dcmHandle\_t pDcmHandle)****Parameters****pDcmHandle**

IN: DCGM Handle

**Returns**

- ▶ DCGM\_ST\_OK If the call was successful.
- ▶ DCGM\_ST\_BADPARAM if a parameter is invalid.

**Description**

Resume profiling activities in DCGM that were previously paused with `dcmProfPause()`.

Call this API after you have completed running other NVIDIA developer tools to reenables DCGM profiling metrics.

DCGM will save BLANK values while profiling is paused.

Calling this while profiling activities have already been resumed is fine and will be treated as a no-op.

## 1.14. Enums and Macros

## enum dcgmOperationMode\_t

Operation mode for DCGM

DCGM can run in auto-mode where it runs additional threads in the background to collect any metrics of interest and auto manages any operations needed for policy management.

DCGM can also operate in manual-mode where it's execution is controlled by the user. In this mode, the user has to periodically call APIs such as [dcgmPolicyTrigger](#) and [dcgmUpdateAllFields](#) which tells DCGM to wake up and perform data collection and operations needed for policy management.

### Values

**DCGM\_OPERATION\_MODE\_AUTO = 1**

**DCGM\_OPERATION\_MODE\_MANUAL = 2**

## enum dcgmOrder\_t

When more than one value is returned from a query, which order should it be returned in?

### Values

**DCGM\_ORDER\_ASCENDING = 1**

Data with earliest (lowest) timestamps returned first.

**DCGM\_ORDER\_DESCENDING = 2**

Data with latest (highest) timestamps returned first.

## enum dcgmReturn\_t

Return values for DCGM API calls.

### Values

**DCGM\_ST\_OK = 0**

Success.

**DCGM\_ST\_BADPARAM = -1**

A bad parameter was passed to a function.

**DCGM\_ST\_GENERIC\_ERROR = -3**

A generic, unspecified error.

**DCGM\_ST\_MEMORY = -4**

An out of memory error occurred.

**DCGM\_ST\_NOT\_CONFIGURED = -5**

Setting not configured.

**DCGM\_ST\_NOT\_SUPPORTED = -6**

Feature not supported.

**DCGM\_ST\_INIT\_ERROR = -7**

DCGM Init error.

**DCGM\_ST\_NVML\_ERROR = -8**

When NVML returns error.

**DCGM\_ST\_PENDING = -9**

Object is in pending state of something else.

**DCGM\_ST\_UNINITIALIZED = -10**

Object is in undefined state.

**DCGM\_ST\_TIMEOUT = -11**

Requested operation timed out.

**DCGM\_ST\_VER\_MISMATCH = -12**

Version mismatch between received and understood API.

**DCGM\_ST\_UNKNOWN\_FIELD = -13**

Unknown field id.

**DCGM\_ST\_NO\_DATA = -14**

No data is available.

**DCGM\_ST\_STALE\_DATA = -15**

Data is considered stale.

**DCGM\_ST\_NOT\_WATCHED = -16**

The given field id is not being updated by the cache manager.

**DCGM\_ST\_NO\_PERMISSION = -17**

Do not have permission to perform the desired action.

**DCGM\_ST\_GPU\_IS\_LOST = -18**

GPU is no longer reachable.

**DCGM\_ST\_RESET\_REQUIRED = -19**

GPU requires a reset.

**DCGM\_ST\_FUNCTION\_NOT\_FOUND = -20**

The function that was requested was not found (bindings only error).

**DCGM\_ST\_CONNECTION\_NOT\_VALID = -21**

The connection to the host engine is not valid any longer.

**DCGM\_ST\_GPU\_NOT\_SUPPORTED = -22**

This GPU is not supported by DCGM.

**DCGM\_ST\_GROUP\_INCOMPATIBLE = -23**

The GPUs of the provided group are not compatible with each other for the requested operation

**DCGM\_ST\_MAX\_LIMIT = -24**

Max limit reached for the object.

**DCGM\_ST\_LIBRARY\_NOT\_FOUND = -25**

DCGM library could not be found.

**DCGM\_ST\_DUPLICATE\_KEY = -26**

Duplicate key passed to a function.

**DCGM\_ST\_GPU\_IN\_SYNC\_BOOST\_GROUP = -27**

GPU is already a part of a sync boost group.

**DCGM\_ST\_GPU\_NOT\_IN\_SYNC\_BOOST\_GROUP = -28**

GPU is not a part of a sync boost group.

**DCGM\_ST\_REQUIRES\_ROOT = -29**

This operation cannot be performed when the host engine is running as non-root.

**DCGM\_ST\_NVVS\_ERROR = -30**

DCGM GPU Diagnostic was successfully executed, but reported an error.

**DCGM\_ST\_INSUFFICIENT\_SIZE = -31**

An input argument is not large enough.

**DCGM\_ST\_FIELD\_UNSUPPORTED\_BY\_API = -32**

The given field ID is not supported by the API being called.

**DCGM\_ST\_MODULE\_NOT\_LOADED = -33**

This request is serviced by a module of DCGM that is not currently loaded.

**DCGM\_ST\_IN\_USE = -34**

The requested operation could not be completed because the affected resource is in use

**DCGM\_ST\_GROUP\_IS\_EMPTY = -35**

This group is empty and the requested operation is not valid on an empty group.

**DCGM\_ST\_PROFILING\_NOT\_SUPPORTED = -36**

Profiling is not supported for this group of GPUs or GPU.

**DCGM\_ST\_PROFILING\_LIBRARY\_ERROR = -37**

The third-party Profiling module returned an unrecoverable error.

**DCGM\_ST\_PROFILING\_MULTI\_PASS = -38**

The requested profiling metrics cannot be collected in a single pass.

**DCGM\_ST\_DIAG\_ALREADY\_RUNNING = -39**

A diag instance is already running, cannot run a new diag until the current one finishes.

**DCGM\_ST\_DIAG\_BAD\_JSON = -40**

The DCGM GPU Diagnostic returned JSON that cannot be parsed.

**DCGM\_ST\_DIAG\_BAD\_LAUNCH = -41**

Error while launching the DCGM GPU Diagnostic.

**DCGM\_ST\_DIAG\_VARIANCE = -42**

There is too much variance while training the diagnostic.

**DCGM\_ST\_DIAG\_THRESHOLD\_EXCEEDED = -43**

A field value met or exceeded the error threshold.

**DCGM\_ST\_INSUFFICIENT\_DRIVER\_VERSION = -44**

The installed driver version is insufficient for this API.

**DCGM\_ST\_INSTANCE\_NOT\_FOUND = -45**

The specified GPU instance does not exist.

**DCGM\_ST\_COMPUTE\_INSTANCE\_NOT\_FOUND = -46**

The specified GPU compute instance does not exist.

**DCGM\_ST\_CHILD\_NOT\_KILLED = -47**

Couldn't kill a child process within the retries.

**DCGM\_ST\_3RD\_PARTY\_LIBRARY\_ERROR = -48**

Detected an error in a 3rd-party library.

**DCGM\_ST\_INSUFFICIENT\_RESOURCES = -49**

Not enough resources available.

**DCGM\_ST\_PLUGIN\_EXCEPTION = -50**

Exception thrown from a diagnostic plugin.

**DCGM\_ST\_NVVS\_ISOLATE\_ERROR = -51**

The diagnostic returned an error that indicates the need for isolation.

## enum dcfgmGroupType\_t

Type of GPU groups

### Values

**DCGM\_GROUP\_DEFAULT = 0**

All the GPUs on the node are added to the group.

**DCGM\_GROUP\_EMPTY = 1**

Creates an empty group.

**DCGM\_GROUP\_DEFAULT\_NVSWITCHES = 2**

All NvSwitches of the node are added to the group.

**DCGM\_GROUP\_DEFAULT\_INSTANCES = 3**

All GPU instances of the node are added to the group.

**DCGM\_GROUP\_DEFAULT\_COMPUTE\_INSTANCES = 4**

All compute instances of the node are added to the group.

**DCGM\_GROUP\_DEFAULT\_EVERYTHING = 5**

All entities are added to this default group.

## enum dcfgmChipArchitecture\_t

Simplified chip architecture. Note that these are made to match `nvmlChipArchitecture_t` and thus do not start at 0.

### Values

**DCGM\_CHIP\_ARCH\_OLDER = 1**

All GPUs older than Kepler.

**DCGM\_CHIP\_ARCH\_KEPLER = 2**

All Kepler-architecture parts.

**DCGM\_CHIP\_ARCH\_MAXWELL = 3**

All Maxwell-architecture parts.

**DCGM\_CHIP\_ARCH\_PASCAL = 4**

All Pascal-architecture parts.

**DCGM\_CHIP\_ARCH\_VOLTA = 5**

All Volta-architecture parts.

**DCGM\_CHIP\_ARCH\_TURING = 6**

All Turing-architecture parts.

**DCGM\_CHIP\_ARCH\_AMPERE = 7**

All Ampere-architecture parts.

**DCGM\_CHIP\_ARCH\_COUNT**

Keep this second to last, exclude unknown.

**DCGM\_CHIP\_ARCH\_UNKNOWN = 0xffffffff**

Anything else, presumably something newer.

## enum dcmgConfigType\_t

Represents the type of configuration to be fetched from the GPUs

### Values

**DCGM\_CONFIG\_TARGET\_STATE = 0**

The target configuration values to be applied.

**DCGM\_CONFIG\_CURRENT\_STATE = 1**

The current configuration state.

## enum dcmgConfigPowerLimitType\_t

Represents the power cap for each member of the group.

### Values

**DCGM\_CONFIG\_POWER\_CAP\_INDIVIDUAL = 0**

Represents the power cap to be applied for each member of the group.

**DCGM\_CONFIG\_POWER\_BUDGET\_GROUP = 1**

Represents the power budget for the entire group.

```
#define MAKE_DCGM_VERSION (unsigned int)(sizeof(typeName) |
((unsigned long)(ver) << 24U))
```

Creates a unique version number for each struct

```
#define DCGM_INT32_BLANK 0x7fffffff0
```

Represents value of the field which can be returned by Host Engine in case the operation is not successful Base value for 32 bits integer blank. can be used as an unspecified blank

```
#define DCGM_INT64_BLANK 0x7fffffffffffffff0
```

Base value for 64 bits integer blank. can be used as an unspecified blank

```
#define DCGM_FP64_BLANK 140737488355328.0
```

Base value for double blank.  $2^{47}$ . FP 64 has 52 bits of mantissa, so 47 bits can still increment by 1 and represent each value from 0-15

```
#define DCGM_STR_BLANK "<<<NULL>>>"
```

Base value for string blank.

```
#define DCGM_INT32_NOT_FOUND (DCGM_INT32_BLANK + 1)
```

Represents an error where INT32 data was not found

```
#define DCGM_INT64_NOT_FOUND (DCGM_INT64_BLANK + 1)
```

Represents an error where INT64 data was not found

```
#define DCGM_FP64_NOT_FOUND (DCGM_FP64_BLANK + 1.0)
```

Represents an error where FP64 data was not found

```
#define DCGM_STR_NOT_FOUND "<<<NOT_FOUND>>>"
```

Represents an error where STR data was not found

```
#define DCGM_INT32_NOT_SUPPORTED (DCGM_INT32_BLANK + 2)
```

Represents an error where fetching the INT32 value is not supported

```
#define DCGM_INT64_NOT_SUPPORTED (DCGM_INT64_BLANK + 2)
```

Represents an error where fetching the INT64 value is not supported

```
#define DCGM_FP64_NOT_SUPPORTED (DCGM_FP64_BLANK + 2.0)
```

Represents an error where fetching the FP64 value is not supported

```
#define DCGM_STR_NOT_SUPPORTED "<<<NOT_SUPPORTED>>>"
```

Represents an error where fetching the STR value is not supported

```
#define DCGM_INT32_NOT_PERMISSIONED (DCGM_INT32_BLANK + 3)
```

Represents an error where fetching the INT32 value is not allowed with our current credentials

```
#define DCGM_INT64_NOT_PERMISSIONED (DCGM_INT64_BLANK + 3)
```

Represents and error where fetching the INT64 value is not allowed with our current credentials

```
#define DCGM_FP64_NOT_PERMISSIONED (DCGM_FP64_BLANK + 3.0)
```

Represents and error where fetching the FP64 value is not allowed with our current credentials

```
#define DCGM_STR_NOT_PERMISSIONED "<<<NOT_PERM>>>"
```

Represents and error where fetching the STR value is not allowed with our current credentials

```
#define DCGM_INT32_IS_BLANK (((val) >= DCGM_INT32_BLANK) ? 1 : 0)
```

Macro to check if a INT32 value is blank or not

```
#define DCGM_INT64_IS_BLANK (((val) >= DCGM_INT64_BLANK) ? 1 : 0)
```

Macro to check if a INT64 value is blank or not

```
#define DCGM_FP64_IS_BLANK (((val) >= DCGM_FP64_BLANK ? 1 : 0))
```

Macro to check if a FP64 value is blank or not

```
#define DCGM_STR_IS_BLANK (val == strstr(val, "<<<") && strstr(val, ">>>"))
```

Macro to check if a STR value is blank or not Works on (char \*). Looks for <<< at first position and >>> inside string

```
#define DCGM_MAX_NUM_DEVICES 32
```

Max number of GPUs supported by DCGM

```
#define DCGM_NVLINK_MAX_LINKS_PER_GPU 12
```

Number of NvLink links per GPU supported by DCGM This is 12 for Ampere, 6 for Volta, and 4 for Pascal

```
#define DCGM_NVLINK_MAX_LINKS_PER_GPU_LEGACY1 6
```

Maximum NvLink links pre-Ampere



```
#define DCGM_MAX_NUM_SWITCHES 12
```

Max number of NvSwitches supported by DCGM

```
#define DCGM_NVLINK_MAX_LINKS_PER_NVSWITCH 36
```

Number of NvLink links per NvSwitch supported by DCGM

```
#define DCGM_MAX_VGPU_INSTANCES_PER_PGPU 32
```

Maximum number of vGPU instances per physical GPU

```
#define DCGM_MAX_STR_LENGTH 256
```

Max length of the DCGM string field

```
#define DCGM_MAX_CLOCKS 256
```

Max number of clocks supported for a device

```
#define DCGM_MAX_NUM_GROUPS 64
```

Max limit on the number of groups supported by DCGM

```
#define DCGM_MAX_FBC_SESSIONS 256
```

Max number of active FBC sessions

```
#define DCGM_VGPU_NAME_BUFFER_SIZE 64
```

Represents the size of a buffer that holds a vGPU type Name or vGPU class type or name of process running on vGPU instance.

```
#define DCGM_GRID_LICENSE_BUFFER_SIZE 128
```

Represents the size of a buffer that holds a vGPU license string

```
#define DCGM_CONFIG_COMPUTEMODE_DEFAULT 0
```

Default compute mode -- multiple contexts per device

```
#define DCGM_CONFIG_COMPUTEMODE_PROHIBITED 1
```

Compute-prohibited mode -- no contexts per device

```
#define DCGM_CONFIG_COMPUTEMODE_EXCLUSIVE_PROCESS 2
```

Compute-exclusive-process mode -- only one context per device, usable from multiple threads at a time

```
#define DCGM_HE_PORT_NUMBER 5555
```

Default Port Number for DCGM Host Engine

```
#define DCGM_GROUP_ALL_GPUS 0x7fffffff
```

Identifies for special DCGM groups

```
#define DCGM_GROUP_MAX_ENTITIES 64
```

Maximum number of entities per entity group

## 1.16. Field Types

Field Types are a single byte.

```
#define DCGM_FT_BINARY 'b'
```

Blob of binary data representing a structure

```
#define DCGM_FT_DOUBLE 'd'
```

8-byte double precision

```
#define DCGM_FT_INT64 'i'
```

8-byte signed integer

```
#define DCGM_FT_STRING 's'
```

Null-terminated ASCII Character string

```
#define DCGM_FT_TIMESTAMP 't'
```

8-byte signed integer usec since 1970

## 1.17. Field Scope

Represents field association with entity scope or global scope.

```
#define DCGM_FS_GLOBAL 0
```

Field is global (ex: driver version)

```
#define DCGM_FS_ENTITY 1
```

Field is associated with an entity (GPU, vGPU...etc)

```
#define DCGM_FS_DEVICE DCGM_FS_ENTITY
```

Field is associated with a device. Deprecated. Use DCGM\_FS\_ENTITY

## 1.18. Field Constants

Constants that represent contents of individual field values.

```
enum dcmGpuVirtualizationMode_t
```

GPU virtualization mode types for DCGM\_FI\_DEV\_VIRTUAL\_MODE

### Values

```
DCGM_GPU_VIRTUALIZATION_MODE_NONE = 0
```

Represents Bare Metal GPU.

```
DCGM_GPU_VIRTUALIZATION_MODE_PASSTHROUGH = 1
```

Device is associated with GPU-Passthrough.

```
DCGM_GPU_VIRTUALIZATION_MODE_VGPU = 2
```

Device is associated with vGPU inside virtual machine.

```
DCGM_GPU_VIRTUALIZATION_MODE_HOST_VGPU = 3
```

Device is associated with VGX hypervisor in vGPU mode.

```
DCGM_GPU_VIRTUALIZATION_MODE_HOST_VSGA = 4
```

Device is associated with VGX hypervisor in vSGA mode.

```
#define DCGM_CUDA_COMPUTE_CAPABILITY_MAJOR ((uint64_t)
(x)&0xFFFF0000)
```

DCGM\_FI\_DEV\_CUDA\_COMPUTE\_CAPABILITY is 16 bits of major version followed by 16 bits of the minor version. These macros separate the two.

```
#define DCGM_CLOCKS_THROTTLE_REASON_GPU_IDLE
0x00000000000000001LL
```

DCGM\_FI\_DEV\_CLOCK\_THROTTLE\_REASONS is a bitmap of why the clock is throttled. These macros are masks for relevant throttling, and are a 1:1 map to the NVML reasons documented in nvml.h. The notes for the header are copied below: Nothing is running on the GPU and the clocks are dropping to Idle state



This limiter may be removed in a later release

```
#define DCGM_CLOCKS_THROTTLE_REASON_CLOCKS_SETTING
0x0000000000000002LL
```

GPU clocks are limited by current setting of applications clocks

```
#define DCGM_CLOCKS_THROTTLE_REASON_SW_POWER_CAP
0x0000000000000004LL
```

SW Power Scaling algorithm is reducing the clocks below requested clocks

```
#define DCGM_CLOCKS_THROTTLE_REASON_HW_SLOWDOWN
0x0000000000000008LL
```

HW Slowdown (reducing the core clocks by a factor of 2 or more) is engaged

This is an indicator of:

- ▶ temperature being too high
- ▶ External Power Brake Assertion is triggered (e.g. by the system power supply)
- ▶ Power draw is too high and Fast Trigger protection is reducing the clocks
- ▶ May be also reported during PState or clock change
- ▶ This behavior may be removed in a later release.

```
#define DCGM_CLOCKS_THROTTLE_REASON_SYNC_BOOST
0x0000000000000010LL
```

Sync Boost

This GPU has been added to a Sync boost group with nvidia-smi or DCGM in order to maximize performance per watt. All GPUs in the sync boost group will boost to the minimum possible clocks across the entire group. Look at the throttle reasons for other GPUs in the system to see why those GPUs are holding this one at lower clocks.

```
#define DCGM_CLOCKS_THROTTLE_REASON_SW_THERMAL
0x0000000000000020LL
```

SW Thermal Slowdown

This is an indicator of one or more of the following:

- ▶ Current GPU temperature above the GPU Max Operating Temperature
- ▶ Current memory temperature above the Memory Max Operating Temperature

```
#define DCGM_CLOCKS_THROTTLE_REASON_HW_THERMAL
0x0000000000000040LL
```

HW Thermal Slowdown (reducing the core clocks by a factor of 2 or more) is engaged

This is an indicator of:

- ▶ temperature being too high

```
#define DCGM_CLOCKS_THROTTLE_REASON_HW_POWER_BRAKE
0x0000000000000080LL
```

HW Power Brake Slowdown (reducing the core clocks by a factor of 2 or more) is engaged

This is an indicator of:

- ▶ External Power Brake Assertion being triggered (e.g. by the system power supply)

```
#define DCGM_CLOCKS_THROTTLE_REASON_DISPLAY_CLOCKS
0x0000000000000100LL
```

GPU clocks are limited by current setting of Display clocks

## 1.19. Field Entity

Represents field association with a particular entity

```
enum dcgm_field_entity_group_t
```

Enum of possible field entity groups

### Values

**DCGM\_FE\_NONE = 0**

Field is not associated with an entity. Field scope should be DCGM\_FS\_GLOBAL

**DCGM\_FE\_GPU**

Field is associated with a GPU entity

**DCGM\_FE\_VGPU**

Field is associated with a VGPU entity

**DCGM\_FE\_SWITCH**

Field is associated with a Switch entity

**DCGM\_FE\_GPU\_I**

Field is associated with a GPU Instance entity

**DCGM\_FE\_GPU\_CI**

Field is associated with a GPU Compute Instance entity

**DCGM\_FE\_COUNT**

Number of elements in this enumeration. Keep this entry last

## typedef unsigned int dcgm\_field\_eid\_t

Represents an identifier for an entity within a field entity. For instance, this is the `gpuId` for `DCGM_FE_GPU`.

## 1.20. Field Identifiers

Field Identifiers

### DcgmFieldGetById (unsigned short fieldId)

#### Parameters

##### fieldId

IN: One of the field IDs (`DCGM_FI_?`)

#### Returns

0 On Failure >0 Pointer to field metadata structure if found.

#### Description

Get a pointer to the metadata for a field by its field ID. See `DCGM_FI_?` for a list of field IDs.

### DcgmFieldGetByTag (char \*tag)

#### Parameters

##### tag

IN: Tag for the field of interest

#### Returns

0 On failure or not found >0 Pointer to field metadata structure if found

#### Description

Get a pointer to the metadata for a field by its field tag.

### DcgmFieldsInit (void)

#### Returns

0 On success <0 On error

**Description**

Initialize the DcgmFields module. Call this once from inside your program

**DcgmFieldsTerm (void)****Returns**

0 On success <0 On error

**Description**

Terminates the DcgmFields module. Call this once from inside your program

**const char \*DcgmFieldsGetEntityGroupString  
(dcgm\_field\_entity\_group\_t entityGroupId)**
**Returns**

- ▶ Pointer to a string like GPU/NvSwitch..etc
- ▶ Null on error

**Description**

Get the string version of a entityGroupId

**#define DCGM\_FI\_UNKNOWN 0**

NULL field

**#define DCGM\_FI\_DRIVER\_VERSION 1**

Driver Version

**#define DCGM\_FI\_DEV\_COUNT 4**

Number of Devices on the node

**#define DCGM\_FI\_CUDA\_DRIVER\_VERSION 5**

Cuda Driver Version Retrieves a number with the major value in the thousands place and the minor value in the hundreds place. CUDA 11.1 = 11100

**#define DCGM\_FI\_DEV\_NAME 50**

Name of the GPU device

```
#define DCGM_FI_DEV_BRAND 51
```

Device Brand

```
#define DCGM_FI_DEV_NVML_INDEX 52
```

NVML index of this GPU

```
#define DCGM_FI_DEV_SERIAL 53
```

Device Serial Number

```
#define DCGM_FI_DEV_UUID 54
```

UUID corresponding to the device

```
#define DCGM_FI_DEV_MINOR_NUMBER 55
```

Device node minor number /dev/nvidia#

```
#define DCGM_FI_DEV_OEM_INFOROM_VER 56
```

OEM inforom version

```
#define DCGM_FI_DEV_PCI_BUSID 57
```

PCI attributes for the device

```
#define DCGM_FI_DEV_PCI_COMBINED_ID 58
```

The combined 16-bit device id and 16-bit vendor id

```
#define DCGM_FI_DEV_PCI_SUBSYS_ID 59
```

The 32-bit Sub System Device ID

```
#define DCGM_FI_GPU_TOPOLOGY_PCI 60
```

Topology of all GPUs on the system via PCI (static)

```
#define DCGM_FI_GPU_TOPOLOGY_NVLINK 61
```

Topology of all GPUs on the system via NVLINK (static)

```
#define DCGM_FI_GPU_TOPOLOGY_AFFINITY 62
```

Affinity of all GPUs on the system (static)



**#define DCGM\_FI\_DEV\_CUDA\_COMPUTE\_CAPABILITY 63**

Cuda compute capability for the device. The major version is the upper 32 bits and the minor version is the lower 32 bits.

**#define DCGM\_FI\_DEV\_COMPUTE\_MODE 65**

Compute mode for the device

**#define DCGM\_FI\_DEV\_PERSISTENCE\_MODE 66**

Persistence mode for the device Boolean: 0 is disabled, 1 is enabled

**#define DCGM\_FI\_DEV\_MIG\_MODE 67**

MIG mode for the device Boolean: 0 is disabled, 1 is enabled

**#define DCGM\_FI\_DEV\_CUDA\_VISIBLE\_DEVICES\_STR 68**

The string that CUDA\_VISIBLE\_DEVICES should be set to for this entity (including MIG)

**#define DCGM\_FI\_DEV\_MIG\_MAX\_SLICES 69**

The maximum number of MIG slices supported by this GPU

**#define DCGM\_FI\_DEV\_CPU\_AFFINITY\_0 70**

Device CPU affinity. part 1/8 = cpus 0 - 63

**#define DCGM\_FI\_DEV\_CPU\_AFFINITY\_1 71**

Device CPU affinity. part 1/8 = cpus 64 - 127

**#define DCGM\_FI\_DEV\_CPU\_AFFINITY\_2 72**

Device CPU affinity. part 2/8 = cpus 128 - 191

**#define DCGM\_FI\_DEV\_CPU\_AFFINITY\_3 73**

Device CPU affinity. part 3/8 = cpus 192 - 255

**#define DCGM\_FI\_DEV\_ECC\_INFOTM\_VER 80**

ECC infotm version

```
#define DCGM_FI_DEV_POWER_INFOTM_VER 81
```

Power management object infotm version

```
#define DCGM_FI_DEV_INFOTM_IMAGE_VER 82
```

Infotm image version

```
#define DCGM_FI_DEV_INFOTM_CONFIG_CHECK 83
```

Infotm configuration checksum

```
#define DCGM_FI_DEV_INFOTM_CONFIG_VALID 84
```

Reads the infotm from the flash and verifies the checksums

```
#define DCGM_FI_DEV_VBIOS_VERSION 85
```

VBIOS version of the device

```
#define DCGM_FI_DEV_BAR1_TOTAL 90
```

Total BAR1 of the GPU in MB

```
#define DCGM_FI_SYNC_BOOST 91
```

Deprecated - Sync boost settings on the node

```
#define DCGM_FI_DEV_BAR1_USED 92
```

Used BAR1 of the GPU in MB

```
#define DCGM_FI_DEV_BAR1_FREE 93
```

Free BAR1 of the GPU in MB

```
#define DCGM_FI_DEV_SM_CLOCK 100
```

SM clock for the device

```
#define DCGM_FI_DEV_MEM_CLOCK 101
```

Memory clock for the device

```
#define DCGM_FI_DEV_VIDEO_CLOCK 102
```

Video encoder/decoder clock for the device

```
#define DCGM_FI_DEV_APP_SM_CLOCK 110
```

SM Application clocks

```
#define DCGM_FI_DEV_APP_MEM_CLOCK 111
```

Memory Application clocks

```
#define DCGM_FI_DEV_CLOCK_THROTTLE_REASONS 112
```

Current clock throttle reasons (bitmask of DCGM\_CLOCKS\_THROTTLE\_REASON\_\*)

```
#define DCGM_FI_DEV_MAX_SM_CLOCK 113
```

Maximum supported SM clock for the device

```
#define DCGM_FI_DEV_MAX_MEM_CLOCK 114
```

Maximum supported Memory clock for the device

```
#define DCGM_FI_DEV_MAX_VIDEO_CLOCK 115
```

Maximum supported Video encoder/decoder clock for the device

```
#define DCGM_FI_DEV_AUTOBOOST 120
```

Auto-boost for the device (1 = enabled. 0 = disabled)

```
#define DCGM_FI_DEV_SUPPORTED_CLOCKS 130
```

Supported clocks for the device

```
#define DCGM_FI_DEV_MEMORY_TEMP 140
```

Memory temperature for the device

```
#define DCGM_FI_DEV_GPU_TEMP 150
```

Current temperature readings for the device, in degrees C

```
#define DCGM_FI_DEV_MEM_MAX_OP_TEMP 151
```

Maximum operating temperature for the memory of this GPU

```
#define DCGM_FI_DEV_GPU_MAX_OP_TEMP 152
```

Maximum operating temperature for this GPU

**#define DCGM\_FI\_DEV\_POWER\_USAGE 155**

Power usage for the device in Watts

**#define DCGM\_FI\_DEV\_TOTAL\_ENERGY\_CONSUMPTION 156**

Total energy consumption for the GPU in mJ since the driver was last reloaded

**#define DCGM\_FI\_DEV\_SLOWDOWN\_TEMP 158**

Slowdown temperature for the device

**#define DCGM\_FI\_DEV\_SHUTDOWN\_TEMP 159**

Shutdown temperature for the device

**#define DCGM\_FI\_DEV\_POWER\_MGMT\_LIMIT 160**

Current Power limit for the device

**#define DCGM\_FI\_DEV\_POWER\_MGMT\_LIMIT\_MIN 161**

Minimum power management limit for the device

**#define DCGM\_FI\_DEV\_POWER\_MGMT\_LIMIT\_MAX 162**

Maximum power management limit for the device

**#define DCGM\_FI\_DEV\_POWER\_MGMT\_LIMIT\_DEF 163**

Default power management limit for the device

**#define DCGM\_FI\_DEV\_ENFORCED\_POWER\_LIMIT 164**

Effective power limit that the driver enforces after taking into account all limiters

**#define DCGM\_FI\_DEV\_PSTATE 190**

Performance state (P-State) 0-15. 0=highest

**#define DCGM\_FI\_DEV\_FAN\_SPEED 191**

Fan speed for the device in percent 0-100

**#define DCGM\_FI\_DEV\_PCIE\_TX\_THROUGHPUT 200**

PCIe Tx utilization information

Deprecated: Use DCGM\_FI\_PROF\_PCIE\_TX\_BYTES instead.

**#define DCGM\_FI\_DEV\_PCIE\_RX\_THROUGHPUT 201**

PCIe Rx utilization information

Deprecated: Use DCGM\_FI\_PROF\_PCIE\_RX\_BYTES instead.

**#define DCGM\_FI\_DEV\_PCIE\_REPLAY\_COUNTER 202**

PCIe replay counter

**#define DCGM\_FI\_DEV\_GPU\_UTIL 203**

GPU Utilization

**#define DCGM\_FI\_DEV\_MEM\_COPY\_UTIL 204**

Memory Utilization

**#define DCGM\_FI\_DEV\_ACCOUNTING\_DATA 205**

Process accounting stats.

This field is only supported when the host engine is running as root unless you enable accounting ahead of time. Accounting mode can be enabled by running "nvidia-smi -am 1" as root on the same node the host engine is running on.

**#define DCGM\_FI\_DEV\_ENC\_UTIL 206**

Encoder Utilization

**#define DCGM\_FI\_DEV\_DEC\_UTIL 207**

Decoder Utilization

**#define DCGM\_FI\_DEV\_MEM\_COPY\_UTIL\_SAMPLES 210**

Memory utilization samples

**#define DCGM\_FI\_DEV\_GRAPHICS\_PIDS 220**

Graphics processes running on the GPU.

**#define DCGM\_FI\_DEV\_COMPUTE\_PIDS 221**

Compute processes running on the GPU.

**#define DCGM\_FI\_DEV\_XID\_ERRORS 230**

XID errors. The value is the specific XID error

**#define DCGM\_FI\_DEV\_PCIE\_MAX\_LINK\_GEN 235**

PCIe Max Link Generation

**#define DCGM\_FI\_DEV\_PCIE\_MAX\_LINK\_WIDTH 236**

PCIe Max Link Width

**#define DCGM\_FI\_DEV\_PCIE\_LINK\_GEN 237**

PCIe Current Link Generation

**#define DCGM\_FI\_DEV\_PCIE\_LINK\_WIDTH 238**

PCIe Current Link Width

**#define DCGM\_FI\_DEV\_POWER\_VIOLATION 240**

Power Violation time in usec

**#define DCGM\_FI\_DEV\_THERMAL\_VIOLATION 241**

Thermal Violation time in usec

**#define DCGM\_FI\_DEV\_SYNC\_BOOST\_VIOLATION 242**

Sync Boost Violation time in usec

**#define DCGM\_FI\_DEV\_BOARD\_LIMIT\_VIOLATION 243**

Board violation limit.

**#define DCGM\_FI\_DEV\_LOW\_UTIL\_VIOLATION 244**

Low utilisation violation limit.

**#define DCGM\_FI\_DEV\_RELIABILITY\_VIOLATION 245**

Reliability violation limit.

**#define DCGM\_FI\_DEV\_TOTAL\_APP\_CLOCKS\_VIOLATION 246**

App clock violation limit.

**#define DCGM\_FI\_DEV\_TOTAL\_BASE\_CLOCKS\_VIOLATION 247**

Base clock violation limit.

```
#define DCGM_FI_DEV_FB_TOTAL 250
```

Total Frame Buffer of the GPU in MB

```
#define DCGM_FI_DEV_FB_FREE 251
```

Free Frame Buffer in MB

```
#define DCGM_FI_DEV_FB_USED 252
```

Used Frame Buffer in MB

```
#define DCGM_FI_DEV_ECC_CURRENT 300
```

Current ECC mode for the device

```
#define DCGM_FI_DEV_ECC_PENDING 301
```

Pending ECC mode for the device

```
#define DCGM_FI_DEV_ECC_SBE_VOL_TOTAL 310
```

Total single bit volatile ECC errors

```
#define DCGM_FI_DEV_ECC_DBE_VOL_TOTAL 311
```

Total double bit volatile ECC errors

```
#define DCGM_FI_DEV_ECC_SBE_AGG_TOTAL 312
```

Total single bit aggregate (persistent) ECC errors Note: monotonically increasing

```
#define DCGM_FI_DEV_ECC_DBE_AGG_TOTAL 313
```

Total double bit aggregate (persistent) ECC errors Note: monotonically increasing

```
#define DCGM_FI_DEV_ECC_SBE_VOL_L1 314
```

L1 cache single bit volatile ECC errors

```
#define DCGM_FI_DEV_ECC_DBE_VOL_L1 315
```

L1 cache double bit volatile ECC errors

```
#define DCGM_FI_DEV_ECC_SBE_VOL_L2 316
```

L2 cache single bit volatile ECC errors

```
#define DCGM_FI_DEV_ECC_DBE_VOL_L2 317
```

L2 cache double bit volatile ECC errors

```
#define DCGM_FI_DEV_ECC_SBE_VOL_DEV 318
```

Device memory single bit volatile ECC errors

```
#define DCGM_FI_DEV_ECC_DBE_VOL_DEV 319
```

Device memory double bit volatile ECC errors

```
#define DCGM_FI_DEV_ECC_SBE_VOL_REG 320
```

Register file single bit volatile ECC errors

```
#define DCGM_FI_DEV_ECC_DBE_VOL_REG 321
```

Register file double bit volatile ECC errors

```
#define DCGM_FI_DEV_ECC_SBE_VOL_TEX 322
```

Texture memory single bit volatile ECC errors

```
#define DCGM_FI_DEV_ECC_DBE_VOL_TEX 323
```

Texture memory double bit volatile ECC errors

```
#define DCGM_FI_DEV_ECC_SBE_AGG_L1 324
```

L1 cache single bit aggregate (persistent) ECC errors Note: monotonically increasing

```
#define DCGM_FI_DEV_ECC_DBE_AGG_L1 325
```

L1 cache double bit aggregate (persistent) ECC errors Note: monotonically increasing

```
#define DCGM_FI_DEV_ECC_SBE_AGG_L2 326
```

L2 cache single bit aggregate (persistent) ECC errors Note: monotonically increasing

```
#define DCGM_FI_DEV_ECC_DBE_AGG_L2 327
```

L2 cache double bit aggregate (persistent) ECC errors Note: monotonically increasing

```
#define DCGM_FI_DEV_ECC_SBE_AGG_DEV 328
```

Device memory single bit aggregate (persistent) ECC errors Note: monotonically increasing



**#define DCGM\_FI\_DEV\_ECC\_DBE\_AGG\_DEV 329**

Device memory double bit aggregate (persistent) ECC errors Note: monotonically increasing

**#define DCGM\_FI\_DEV\_ECC\_SBE\_AGG\_REG 330**

Register File single bit aggregate (persistent) ECC errors Note: monotonically increasing

**#define DCGM\_FI\_DEV\_ECC\_DBE\_AGG\_REG 331**

Register File double bit aggregate (persistent) ECC errors Note: monotonically increasing

**#define DCGM\_FI\_DEV\_ECC\_SBE\_AGG\_TEX 332**

Texture memory single bit aggregate (persistent) ECC errors Note: monotonically increasing

**#define DCGM\_FI\_DEV\_ECC\_DBE\_AGG\_TEX 333**

Texture memory double bit aggregate (persistent) ECC errors Note: monotonically increasing

**#define DCGM\_FI\_DEV\_RETIRED\_SBE 390**

Number of retired pages because of single bit errors Note: monotonically increasing

**#define DCGM\_FI\_DEV\_RETIRED\_DBE 391**

Number of retired pages because of double bit errors Note: monotonically increasing

**#define DCGM\_FI\_DEV\_RETIRED\_PENDING 392**

Number of pages pending retirement

**#define DCGM\_FI\_DEV\_UNCORRECTABLE\_REMAPPED\_ROWS 393**

Number of remapped rows for uncorrectable errors

**#define DCGM\_FI\_DEV\_CORRECTABLE\_REMAPPED\_ROWS 394**

Number of remapped rows for correctable errors

**#define DCGM\_FI\_DEV\_ROW\_REMAP\_FAILURE 395**

Whether remapping of rows has failed

**#define DCGM\_FI\_DEV\_VIRTUAL\_MODE 500**

Virtualization Mode corresponding to the GPU.

One of DCGM\_GPU\_VIRTUALIZATION\_MODE\_\* constants.

**#define DCGM\_FI\_DEV\_SUPPORTED\_TYPE\_INFO 501**

Includes Count and Static info of vGPU types supported on a device

**#define DCGM\_FI\_DEV\_CREATABLE\_VGPU\_TYPE\_IDS 502**

Includes Count and currently Creatable vGPU types on a device

**#define DCGM\_FI\_DEV\_VGPU\_INSTANCE\_IDS 503**

Includes Count and currently Active vGPU Instances on a device

**#define DCGM\_FI\_DEV\_VGPU\_UTILIZATIONS 504**

Utilization values for vGPUs running on the device

**#define DCGM\_FI\_DEV\_VGPU\_PER\_PROCESS\_UTILIZATION 505**

Utilization values for processes running within vGPU VMs using the device

**#define DCGM\_FI\_DEV\_ENC\_STATS 506**

Current encoder statistics for a given device

**#define DCGM\_FI\_DEV\_FBC\_STATS 507**

Statistics of current active frame buffer capture sessions on a given device

**#define DCGM\_FI\_DEV\_FBC\_SESSIONS\_INFO 508**

Information about active frame buffer capture sessions on a target device

**#define DCGM\_FI\_DEV\_VGPU\_VM\_ID 520**

VM ID of the vGPU instance

**#define DCGM\_FI\_DEV\_VGPU\_VM\_NAME 521**

VM name of the vGPU instance

**#define DCGM\_FI\_DEV\_VGPU\_TYPE 522**

vGPU type of the vGPU instance

**#define DCGM\_FI\_DEV\_VGPU\_UUID 523**

UUID of the vGPU instance

**#define DCGM\_FI\_DEV\_VGPU\_DRIVER\_VERSION 524**

Driver version of the vGPU instance

**#define DCGM\_FI\_DEV\_VGPU\_MEMORY\_USAGE 525**

Memory usage of the vGPU instance

**#define DCGM\_FI\_DEV\_VGPU\_LICENSE\_STATUS 526**

License status of the vGPU instance

**#define DCGM\_FI\_DEV\_VGPU\_FRAME\_RATE\_LIMIT 527**

Frame rate limit of the vGPU instance

**#define DCGM\_FI\_DEV\_VGPU\_ENC\_STATS 528**

Current encoder statistics of the vGPU instance

**#define DCGM\_FI\_DEV\_VGPU\_ENC\_SESSIONS\_INFO 529**

Information about all active encoder sessions on the vGPU instance

**#define DCGM\_FI\_DEV\_VGPU\_FBC\_STATS 530**

Statistics of current active frame buffer capture sessions on the vGPU instance

**#define DCGM\_FI\_DEV\_VGPU\_FBC\_SESSIONS\_INFO 531**

Information about active frame buffer capture sessions on the vGPU instance

**#define DCGM\_FI\_DEV\_VGPU\_LICENSE\_INSTANCE\_STATUS 532**

License status of the vGPU host

**#define DCGM\_FI\_FIRST\_VGPU\_FIELD\_ID 520**

Starting field ID of the vGPU instance

**#define DCGM\_FI\_LAST\_VGPU\_FIELD\_ID 570**

Last field ID of the vGPU instance

```
#define DCGM_FI_MAX_VGPU_FIELDS DCGM_FI_LAST_VGPU_FIELD_ID  
- DCGM_FI_FIRST_VGPU_FIELD_ID
```

For now max vGPU field Ids taken as difference of DCGM\_FI\_LAST\_VGPU\_FIELD\_ID and DCGM\_FI\_FIRST\_VGPU\_FIELD\_ID i.e. 50

```
#define DCGM_FI_INTERNAL_FIELDS_0_START 600
```

Starting ID for all the internal fields

```
#define DCGM_FI_INTERNAL_FIELDS_0_END 699
```

Last ID for all the internal fields

NVSwitch entity field IDs start here.

NVSwitch latency bins for port 0

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P00 700
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P00 701
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P00 702
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P00 703
```

Max latency bin

NVSwitch latency bins for port 1

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P01 704
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P01 705
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P01 706
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P01 707
```

Max latency bin

NVSwitch latency bins for port 2

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P02 708
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P02 709
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P02 710
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P02 711
```

Max latency bin

NVSwitch latency bins for port 3

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P03 712
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P03 713
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P03 714
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P03 715
```

Max latency bin

NVSwitch latency bins for port 4

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P04 716
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P04 717
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P04 718
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P04 719
```

Max latency bin

NVSwitch latency bins for port 5

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P05 720
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P05 721
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P05 722
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P05 723
```

Max latency bin

NVSwitch latency bins for port 6

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P06 724
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P06 725
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P06 726
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P06 727
```

Max latency bin

NVSwitch latency bins for port 7

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P07 728
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P07 729
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P07 730
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P07 731
```

Max latency bin

NVSwitch latency bins for port 8

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P08 732
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P08 733
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P08 734
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P08 735
```

Max latency bin

NVSwitch latency bins for port 9

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P09 736
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P09 737
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P09 738
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P09 739
```

Max latency bin

NVSwitch latency bins for port 10

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P10 740
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P10 741
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P10 742
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P10 743
```

Max latency bin

NVSwitch latency bins for port 11

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P11 744
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P11 745
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P11 746
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P11 747
```

Max latency bin



NVSwitch latency bins for port 12

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P12 748
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P12 749
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P12 750
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P12 751
```

Max latency bin

NVSwitch latency bins for port 13

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P13 752
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P13 753
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P13 754
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P13 755
```

Max latency bin

NVSwitch latency bins for port 14

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P14 756
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P14 757
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P14 758
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P14 759
```

Max latency bin

NVSwitch latency bins for port 15

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P15 760
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P15 761
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P15 762
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P15 763
```

Max latency bin

NVSwitch latency bins for port 16

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P16 764
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P16 765
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P16 766
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P16 767
```

Max latency bin

NVSwitch latency bins for port 17

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_LOW_P17 768
```

Low latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MED_P17 769
```

Medium latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_HIGH_P17 770
```

High latency bin

```
#define DCGM_FI_DEV_NVSWITCH_LATENCY_MAX_P17 771
```

Max latency bin

NVSwitch Tx and Rx Counter 0 for each port

By default, Counter 0 counts bytes.

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P00 780
```

NVSwitch Tx Bandwidth Counter 0 for port 0

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P00 781
```

NVSwitch Rx Bandwidth Counter 0 for port 0

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P01 782
```

NVSwitch Tx Bandwidth Counter 0 for port 1

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P01 783
```

NVSwitch Rx Bandwidth Counter 0 for port 1

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P02 784
```

NVSwitch Tx Bandwidth Counter 0 for port 2

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P02 785
```

NVSwitch Rx Bandwidth Counter 0 for port 2

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P03 786
```

NVSwitch Tx Bandwidth Counter 0 for port 3

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P03 787
```

NVSwitch Rx Bandwidth Counter 0 for port 3

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P04 788
```

NVSwitch Tx Bandwidth Counter 0 for port 4

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P04 789
```

NVSwitch Rx Bandwidth Counter 0 for port 4

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P05 790
```

NVSwitch Tx Bandwidth Counter 0 for port 5

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P05 791
```

NVSwitch Rx Bandwidth Counter 0 for port 5

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P06 792
```

NVSwitch Tx Bandwidth Counter 0 for port 6

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P06 793
```

NVSwitch Rx Bandwidth Counter 0 for port 6

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P07 794
```

NVSwitch Tx Bandwidth Counter 0 for port 7

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P07 795
```

NVSwitch Rx Bandwidth Counter 0 for port 7

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P08 796
```

NVSwitch Tx Bandwidth Counter 0 for port 8

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P08 797
```

NVSwitch Rx Bandwidth Counter 0 for port 8

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P09 798
```

NVSwitch Tx Bandwidth Counter 0 for port 9

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P09 799
```

NVSwitch Rx Bandwidth Counter 0 for port 9

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P10 800
```

NVSwitch Tx Bandwidth Counter 0 for port 10

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P10 801
```

NVSwitch Rx Bandwidth Counter 0 for port 10

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P11 802
```

NVSwitch Tx Bandwidth Counter 0 for port 11

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P11 803
```

NVSwitch Rx Bandwidth Counter 0 for port 11

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P12 804
```

NVSwitch Tx Bandwidth Counter 0 for port 12

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P12 805
```

NVSwitch Rx Bandwidth Counter 0 for port 12

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P13 806
```

NVSwitch Tx Bandwidth Counter 0 for port 13

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P13 807
```

NVSwitch Rx Bandwidth Counter 0 for port 13

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P14 808
```

NVSwitch Tx Bandwidth Counter 0 for port 14

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P14 809
```

NVSwitch Rx Bandwidth Counter 0 for port 14

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P15 810
```

NVSwitch Tx Bandwidth Counter 0 for port 15

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P15 811
```

NVSwitch Rx Bandwidth Counter 0 for port 15

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P16 812
```

NVSwitch Tx Bandwidth Counter 0 for port 16

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P16 813
```

NVSwitch Rx Bandwidth Counter 0 for port 16

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_0_P17 814
```

NVSwitch Tx Bandwidth Counter 0 for port 17

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_0_P17 815
```

NVSwitch Rx Bandwidth Counter 0 for port 17

NVSwitch Tx and RX Bandwidth Counter 1 for each port

By default, Counter 1 counts packets.

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P00 820
```

NVSwitch Tx Bandwidth Counter 1 for port 0

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P00 821
```

NVSwitch Rx Bandwidth Counter 1 for port 0

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P01 822
```

NVSwitch Tx Bandwidth Counter 1 for port 1

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P01 823
```

NVSwitch Rx Bandwidth Counter 1 for port 1

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P02 824
```

NVSwitch Tx Bandwidth Counter 1 for port 2

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P02 825
```

NVSwitch Rx Bandwidth Counter 1 for port 2

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P03 826
```

NVSwitch Tx Bandwidth Counter 1 for port 3

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P03 827
```

NVSwitch Rx Bandwidth Counter 1 for port 3

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P04 828
```

NVSwitch Tx Bandwidth Counter 1 for port 4

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P04 829
```

NVSwitch Rx Bandwidth Counter 1 for port 4

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P05 830
```

NVSwitch Tx Bandwidth Counter 1 for port 5

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P05 831
```

NVSwitch Rx Bandwidth Counter 1 for port 5

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P06 832
```

NVSwitch Tx Bandwidth Counter 1 for port 6

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P06 833
```

NVSwitch Rx Bandwidth Counter 1 for port 6

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P07 834
```

NVSwitch Tx Bandwidth Counter 1 for port 7

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P07 835
```

NVSwitch Rx Bandwidth Counter 1 for port 7

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P08 836
```

NVSwitch Tx Bandwidth Counter 1 for port 8

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P08 837
```

NVSwitch Rx Bandwidth Counter 1 for port 8

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P09 838
```

NVSwitch Tx Bandwidth Counter 1 for port 9

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P09 839
```

NVSwitch Rx Bandwidth Counter 1 for port 9

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P10 840
```

NVSwitch Tx Bandwidth Counter 0 for port 10

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P10 841
```

NVSwitch Rx Bandwidth Counter 1 for port 10

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P11 842
```

NVSwitch Tx Bandwidth Counter 1 for port 11

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P11 843
```

NVSwitch Rx Bandwidth Counter 1 for port 11

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P12 844
```

NVSwitch Tx Bandwidth Counter 1 for port 12

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P12 845
```

NVSwitch Rx Bandwidth Counter 1 for port 12

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P13 846
```

NVSwitch Tx Bandwidth Counter 0 for port 13

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P13 847
```

NVSwitch Rx Bandwidth Counter 1 for port 13

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P14 848
```

NVSwitch Tx Bandwidth Counter 1 for port 14

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P14 849
```

NVSwitch Rx Bandwidth Counter 1 for port 14



```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P15 850
```

NVSwitch Tx Bandwidth Counter 1 for port 15

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P15 851
```

NVSwitch Rx Bandwidth Counter 1 for port 15

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P16 852
```

NVSwitch Tx Bandwidth Counter 1 for port 16

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P16 853
```

NVSwitch Rx Bandwidth Counter 1 for port 16

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_TX_1_P17 854
```

NVSwitch Tx Bandwidth Counter 1 for port 17

```
#define DCGM_FI_DEV_NVSWITCH_BANDWIDTH_RX_1_P17 855
```

NVSwitch Rx Bandwidth Counter 1 for port 17

NVSwitch error counters

```
#define DCGM_FI_DEV_NVSWITCH_FATAL_ERRORS 856
```

NVSwitch fatal error information. Note: value field indicates the specific SXid reported

```
#define DCGM_FI_DEV_NVSWITCH_NON_FATAL_ERRORS 857
```

NVSwitch non fatal error information. Note: value field indicates the specific SXid reported

```
#define DCGM_FI_FIRST_NVSWITCH_FIELD_ID 700
```

Starting field ID of the NVSwitch instance

```
#define DCGM_FI_LAST_NVSWITCH_FIELD_ID 860
```

Last field ID of the NVSwitch instance

```
#define DCGM_FI_MAX_NVSWITCH_FIELDS  
DCGM_FI_LAST_NVSWITCH_FIELD_ID -  
DCGM_FI_FIRST_NVSWITCH_FIELD_ID + 1
```

For now max NVSwitch field Ids taken as difference of DCGM\_FI\_LAST\_NVSWITCH\_FIELD\_ID and DCGM\_FI\_FIRST\_NVSWITCH\_FIELD\_ID + 1 i.e. 200

```
#define DCGM_FI_PROF_GR_ENGINE_ACTIVE 1001
```

Profiling Fields. These all start with DCGM\_FI\_PROF\_\* Ratio of time the graphics engine is active. The graphics engine is active if a graphics/compute context is bound and the graphics pipe or compute pipe is busy.

```
#define DCGM_FI_PROF_SM_ACTIVE 1002
```

The ratio of cycles an SM has at least 1 warp assigned (computed from the number of cycles and elapsed cycles)

```
#define DCGM_FI_PROF_SM_OCCUPANCY 1003
```

The ratio of number of warps resident on an SM. (number of resident as a ratio of the theoretical maximum number of warps per elapsed cycle)

```
#define DCGM_FI_PROF_PIPE_TENSOR_ACTIVE 1004
```

The ratio of cycles the tensor (HMMA) pipe is active (off the peak sustained elapsed cycles)

```
#define DCGM_FI_PROF_DRAM_ACTIVE 1005
```

The ratio of cycles the device memory interface is active sending or receiving data.

```
#define DCGM_FI_PROF_PIPE_FP64_ACTIVE 1006
```

Ratio of cycles the fp64 pipe is active.

```
#define DCGM_FI_PROF_PIPE_FP32_ACTIVE 1007
```

Ratio of cycles the fp32 pipe is active.

```
#define DCGM_FI_PROF_PIPE_FP16_ACTIVE 1008
```

Ratio of cycles the fp16 pipe is active. This does not include HMMA.

**#define DCGM\_FI\_PROF\_PCIE\_TX\_BYTES 1009**

The number of bytes of active PCIe tx (transmit) data including both header and payload.

Note that this is from the perspective of the GPU, so copying data from device to host (DtoH) would be reflected in this metric.

**#define DCGM\_FI\_PROF\_PCIE\_RX\_BYTES 1010**

The number of bytes of active PCIe rx (read) data including both header and payload.

Note that this is from the perspective of the GPU, so copying data from host to device (HtoD) would be reflected in this metric.

**#define DCGM\_FI\_PROF\_NVLINK\_TX\_BYTES 1011**

The number of bytes of active NvLink tx (transmit) data including both header and payload.

**#define DCGM\_FI\_PROF\_NVLINK\_RX\_BYTES 1012**

The number of bytes of active NvLink rx (read) data including both header and payload.

**#define DCGM\_FI\_MAX\_FIELDS 1013**

1 greater than maximum fields above. This is the 1 greater than the maximum field id that could be allocated

## 1.21. DCGMAPI\_Admin\_ExecCtrl

**dcgmReturn\_t dcgmUpdateAllFields (dcgmHandle\_t pDcgmHandle, int waitForUpdate)**

**Parameters****pDcgmHandle**

IN: DCGM Handle

**waitForUpdate**

IN: Whether or not to wait for the update loop to complete before returning to the caller 1=wait. 0=do not wait.

**Returns**

- ▶ DCGM\_ST\_OK if the call was successful
- ▶ DCGM\_ST\_BADPARAM if waitForUpdate is invalid

- ▶ `DCGM_ST_GENERIC_ERROR` if an unspecified DCGM error occurs

### Description

This method is used to tell the DCGM module to update all the fields being watched.

Note: If the operation mode was set to manual mode (`DCGM_OPERATION_MODE_MANUAL`) during initialization (`dcgmInit`), this method must be called periodically to allow field value watches the opportunity to gather samples.

## `dcgmReturn_t dcgmPolicyTrigger (dcgmHandle_t pDcgmHandle)`

### Parameters

#### `pDcgmHandle`

IN: DCGM Handle

### Returns

- ▶ `DCGM_ST_OK` If the call was successful
- ▶ `DCGM_ST_GENERIC_ERROR` The policy manager was unable to perform another iteration.

### Description

Inform the policy manager loop to perform an iteration and trigger the callbacks of any registered functions. Callback functions will be called from a separate thread as the calling function.

Note: The GPU monitoring and management agent must call this method periodically if the operation mode is set to manual mode (`DCGM_OPERATION_MODE_MANUAL`) during initialization (`dcgmInit`).

# Chapter 2.

## DATA STRUCTURES

Here are the data structures with brief descriptions:

- `dcgm_field_meta_t`
- `dcgm_field_output_format_t`
- `dcgmClockSet_v1`
- `dcgmConfig_v1`
- `dcgmConfigPerfStateSettings_t`
- `dcgmConfigPowerLimit_t`
- `dcgmConnectV2Params_v1`
- `dcgmConnectV2Params_v2`
- `dcgmDeviceAttributes_v1`
- `dcgmDeviceEncStats_v1`
- `dcgmDeviceFbcSessionInfo_v1`
- `dcgmDeviceFbcSessions_v1`
- `dcgmDeviceFbcStats_v1`
- `dcgmDeviceIdentifiers_v1`
- `dcgmDeviceMemoryUsage_v1`
- `dcgmDevicePidAccountingStats_v1`
- `dcgmDevicePowerLimits_v1`
- `dcgmDeviceSupportedClockSets_v1`
- `dcgmDeviceThermals_v1`
- `dcgmDeviceTopology_v1`
- `dcgmDeviceVgpuEncSessions_v1`
- `dcgmDeviceVgpuProcessUtilInfo_v1`
- `dcgmDeviceVgpuTypeInfo_v1`
- `dcgmDeviceVgpuUtilInfo_v1`
- `dcgmDiagResponse_v6`
- `dcgmDiagResponsePerGpu_v2`
- `dcgmErrorInfo_t`
- `dcgmFieldGroupInfo_v1`
- `dcgmFieldValue_v1`

dcgmFieldValue\_v2  
dcgmGpuUsageInfo\_t  
dcgmGroupEntityPair\_t  
dcgmGroupInfo\_v2  
dcgmGroupTopology\_v1  
dcgmHealthResponse\_v4  
dcgmHealthSetParams\_v2  
dcgmHostengineHealth\_v1  
dcgmIntrospectContext\_v1  
dcgmIntrospectCpuUtil\_v1  
dcgmIntrospectFieldsExecTime\_v1  
dcgmIntrospectFullFieldsExecTime\_v2  
dcgmIntrospectFullMemory\_v1  
dcgmIntrospectMemory\_v1  
dcgmJobInfo\_v3  
dcgmMigEntityInfo\_t  
dcgmMigHierarchy\_v1  
dcgmMigHierarchyInfo\_t  
dcgmModuleGetStatusesModule\_t  
dcgmNvLinkGpuLinkStatus\_v1  
dcgmNvLinkNvSwitchLinkStatus\_t  
dcgmNvLinkStatus\_v1  
dcgmPidInfo\_v2  
dcgmPidSingleInfo\_t  
dcgmPolicy\_v1  
dcgmPolicyCallbackResponse\_v1  
dcgmPolicyConditionDbe\_t  
dcgmPolicyConditionMpr\_t  
dcgmPolicyConditionNvlink\_t  
dcgmPolicyConditionParams\_t  
dcgmPolicyConditionPci\_t  
dcgmPolicyConditionPower\_t  
dcgmPolicyConditionThermal\_t  
dcgmPolicyConditionXID\_t  
dcgmPolicyViolationNotify\_t  
dcgmProcessUtilInfo\_t  
dcgmProcessUtilSample\_t  
dcgmProfMetricGroupInfo\_t  
dcgmProfUnwatchFields\_v1  
dcgmProfWatchFields\_v1  
dcgmRunningProcess\_v1  
dcgmSettingsSetLoggingSeverity\_v1  
dcgmStartEmbeddedV2Params\_v1

[dcgmStatSummaryFp64\\_t](#)  
[dcgmStatSummaryInt32\\_t](#)  
[dcgmStatSummaryInt64\\_t](#)  
[dcgmVersionInfo\\_v2](#)

## 2.1. dcgm\_field\_meta\_t Struct Reference

Structure to store meta data for the field

**unsigned short dcgm\_field\_meta\_t::fieldId**

Field identifier. DCGM\_FI\_? define

**char dcgm\_field\_meta\_t::fieldType**

Field type. DCGM\_FT\_? define

**unsigned char dcgm\_field\_meta\_t::size**

field size in bytes (raw value size). 0=variable (like DCGM\_FT\_STRING)

**char dcgm\_field\_meta\_t::tag**

Tag for this field for serialization like 'device\_temperature'

**int dcgm\_field\_meta\_t::scope**

Field scope. DCGM\_FS\_? define of this field's association

**int dcgm\_field\_meta\_t::nvmlFieldId**

Optional NVML field this DCGM field maps to. 0 = no mapping. Otherwise, this should be a NVML\_FI\_? define from nvml.h

**dcgm\_field\_entity\_group\_t**

**dcgm\_field\_meta\_t::entityLevel**

Field entity level. DCGM\_FE\_? specifying at what level the field is queryable

**struct dcgm\_field\_output\_format\_p**

**dcgm\_field\_meta\_t::valueFormat**

pointer to the structure that holds the formatting the values for fields

## 2.2. `dcgm_field_output_format_t` Struct Reference

Structure for formatting the output for dmon. Used as a member in `dcgm_field_meta_p`

### `char dcgm_field_output_format_t::shortName`

Short name corresponding to field. This short name is used to identify columns in dmon output.

### `char dcgm_field_output_format_t::unit`

The unit of value. Eg: C(elsius), W(att), MB/s

### `short dcgm_field_output_format_t::width`

Maximum width/number of digits that a value for field can have.

## 2.3. `dcgmClockSet_v1` Struct Reference

Represents a set of memory, SM, and video clocks for a device. This can be current values or a target values based on context

### `int dcgmClockSet_v1::version`

Version Number (`dcgmClockSet_version`).

### `unsigned int dcgmClockSet_v1::memClock`

Memory Clock (Memory Clock value OR `DCGM_INT32_BLANK` to Ignore/Use compatible value with `smClk`)

### `unsigned int dcgmClockSet_v1::smClock`

SM Clock (SM Clock value OR `DCGM_INT32_BLANK` to Ignore/Use compatible value with `memClk`).

## 2.4. `dcgmConfig_v1` Struct Reference

Structure to represent default and target configuration for a device



**unsigned int dcgmConfig\_v1::version**

Version number (dcgmConfig\_version).

**unsigned int dcgmConfig\_v1::gpuld**

GPU ID.

**unsigned int dcgmConfig\_v1::eccMode**

ECC Mode (0: Disabled, 1 : Enabled, DCGM\_INT32\_BLANK : Ignored).

**unsigned int dcgmConfig\_v1::computeMode**

Compute Mode (One of DCGM\_CONFIG\_COMPUTEMODE\_? OR DCGM\_INT32\_BLANK to Ignore).

**struct dcgmConfigPerfStateSettings\_t  
dcgmConfig\_v1::perfState**

Performance State Settings (clocks / boost mode).

**struct dcgmConfigPowerLimit\_t  
dcgmConfig\_v1::powerLimit**

Power Limits.

## 2.5. dcgmConfigPerfStateSettings\_t Struct Reference

Used to represent Performance state settings

**unsigned int dcgmConfigPerfStateSettings\_t::syncBoost**

Sync Boost Mode (0: Disabled, 1 : Enabled, DCGM\_INT32\_BLANK : Ignored). Note that using this setting may result in lower clocks than targetClocks

**struct dcgmClockSet\_t  
dcgmConfigPerfStateSettings\_t::targetClocks**

Target clocks. Set smClock and memClock to DCGM\_INT32\_BLANK to ignore/use compatible values. For GPUs > Maxwell, setting this implies autoBoost=0

## 2.6. `dcgmConfigPowerLimit_t` Struct Reference

Used to represents the power capping limit for each GPU in the group or to represent the power budget for the entire group

`dcgmConfigPowerLimitType_t`  
`dcgmConfigPowerLimit_t::type`

Flag to represent power cap for each GPU or power budget for the group of GPUs.

`unsigned int dcgmConfigPowerLimit_t::val`

Power Limit in Watts (Set a value OR `DCGM_INT32_BLANK` to Ignore).

## 2.7. `dcgmConnectV2Params_v1` Struct Reference

Connection options for `dcgmConnect_v2` (v1)

NOTE: This version is deprecated. use `dcgmConnectV2Params_v2`

`unsigned int dcgmConnectV2Params_v1::version`

Version number. Use `dcgmConnectV2Params_version`

`unsigned int`  
`dcgmConnectV2Params_v1::persistAfterDisconnect`

Whether to persist DCGM state modified by this connection once the connection is terminated. Normally, all field watches created by a connection are removed once a connection goes away. 1 = do not clean up after this connection. 0 = clean up after this connection

## 2.8. `dcgmConnectV2Params_v2` Struct Reference

Connection options for `dcgmConnect_v2` (v2)

`unsigned int dcgmConnectV2Params_v2::version`

Version number. Use `dcgmConnectV2Params_version`

unsigned int

### `dcgmConnectV2Params_v2::persistAfterDisconnect`

Whether to persist DCGM state modified by this connection once the connection is terminated. Normally, all field watches created by a connection are removed once a connection goes away. 1 = do not clean up after this connection. 0 = clean up after this connection

### `unsigned int dcgmConnectV2Params_v2::timeoutMs`

When attempting to connect to the specified host engine, how long should we wait in milliseconds before giving up

unsigned int

### `dcgmConnectV2Params_v2::addressIsUnixSocket`

Whether or not the passed-in address is a unix socket filename (1) or a TCP/IP address (0)

## 2.9. `dcgmDeviceAttributes_v1` Struct Reference

Represents attributes corresponding to a device

**unsigned int dcgmDeviceAttributes\_v1::version**

Version number (dcgmDeviceAttributes\_version).

**struct dcgmDeviceSupportedClockSets\_t**

**dcgmDeviceAttributes\_v1::clockSets**

Supported clocks for the device.

**struct dcgmDeviceThermals\_t**

**dcgmDeviceAttributes\_v1::thermalSettings**

Thermal settings for the device.

**struct dcgmDevicePowerLimits\_t**

**dcgmDeviceAttributes\_v1::powerLimits**

Various power limits for the device.

**struct dcgmDeviceIdentifiers\_t**

**dcgmDeviceAttributes\_v1::identifiers**

Identifiers for the device.

**struct dcgmDeviceMemoryUsage\_t**

**dcgmDeviceAttributes\_v1::memoryUsage**

Memory usage info for the device.

**char dcgmDeviceAttributes\_v1::unused**

Unused Space. Set to 0 for now.

## 2.10. dcgmDeviceEncStats\_v1 Struct Reference

Represents current encoder statistics for the given device/vGPU instance

**unsigned int dcgmDeviceEncStats\_v1::version**

Version Number (dcgmDeviceEncStats\_version).

**unsigned int dcgmDeviceEncStats\_v1::sessionCount**

Count of active encoder sessions.

**unsigned int dcgmDeviceEncStats\_v1::averageFps**

Trailing average FPS of all active sessions.

**unsigned int dcgmDeviceEncStats\_v1::averageLatency**

Encode latency in milliseconds.

## 2.11. dcgmDeviceFbcSessionInfo\_v1 Struct Reference

Represents information about active FBC session on the given device/vGPU instance

**unsigned int dcgmDeviceFbcSessionInfo\_v1::version**

Version Number (dcgmDeviceFbcSessionInfo\_version).

**unsigned int dcgmDeviceFbcSessionInfo\_v1::sessionId**

Unique session ID.

**unsigned int dcgmDeviceFbcSessionInfo\_v1::pid**

Owning process ID.

**unsigned int dcgmDeviceFbcSessionInfo\_v1::vgpuId**

vGPU instance ID (only valid on vGPU hosts, otherwise zero)

**unsigned int**

**dcgmDeviceFbcSessionInfo\_v1::displayOrdinal**

Display identifier.

**dcgmFBCSessionType\_t**

**dcgmDeviceFbcSessionInfo\_v1::sessionType**

Type of frame buffer capture session.

**unsigned int dcgmDeviceFbcSessionInfo\_v1::sessionFlags**

Session flags.

**unsigned int**

**dcgmDeviceFbcSessionInfo\_v1::hMaxResolution**

Max horizontal resolution supported by the capture session.

**unsigned int**

**dcgmDeviceFbcSessionInfo\_v1::vMaxResolution**

Max vertical resolution supported by the capture session.

**unsigned int dcgmDeviceFbcSessionInfo\_v1::hResolution**

Horizontal resolution requested by caller in capture call.

**unsigned int dcgmDeviceFbcSessionInfo\_v1::vResolution**

Vertical resolution requested by caller in capture call.

**unsigned int dcgmDeviceFbcSessionInfo\_v1::averageFps**

Moving average new frames captured per second.

**unsigned int**

**dcgmDeviceFbcSessionInfo\_v1::averageLatency**

Moving average new frame capture latency in microseconds.

## 2.12. dcgmDeviceFbcSessions\_v1 Struct Reference

Represents all the active FBC sessions on the given device/vGPU instance

**unsigned int dcgmDeviceFbcSessions\_v1::version**

Version Number (dcgmDeviceFbcSessions\_version).

**unsigned int dcgmDeviceFbcSessions\_v1::sessionCount**

Count of active FBC sessions.

**struct dcgmDeviceFbcSessionInfo\_t**

**dcgmDeviceFbcSessions\_v1::sessionInfo**

Info about the active FBC session.

## 2.13. dcgmDeviceFbcStats\_v1 Struct Reference

Represents current frame buffer capture sessions statistics for the given device/vGPU instance

### **unsigned int dcgmDeviceFbcStats\_v1::version**

Version Number (dcgmDeviceFbcStats\_version).

### **unsigned int dcgmDeviceFbcStats\_v1::sessionCount**

Count of active FBC sessions.

### **unsigned int dcgmDeviceFbcStats\_v1::averageFps**

Moving average new frames captured per second.

### **unsigned int dcgmDeviceFbcStats\_v1::averageLatency**

Moving average new frame capture latency in microseconds.

## **2.14. dcgmDeviceIdentifiers\_v1 Struct Reference**

Represents device identifiers



**unsigned int dcgmDeviceIdentifiers\_v1::version**

Version Number (dcgmDeviceIdentifiers\_version).

**char dcgmDeviceIdentifiers\_v1::brandName**

Brand Name.

**char dcgmDeviceIdentifiers\_v1::deviceName**

Name of the device.

**char dcgmDeviceIdentifiers\_v1::pciBusId**

PCI Bus ID.

**char dcgmDeviceIdentifiers\_v1::serial**

Serial for the device.

**char dcgmDeviceIdentifiers\_v1::uuid**

UUID for the device.

**char dcgmDeviceIdentifiers\_v1::vbios**

VBIOS version.

**char dcgmDeviceIdentifiers\_v1::inforomImageVersion**

Inforom Image version.

**unsigned int dcgmDeviceIdentifiers\_v1::pciDeviceId**

The combined 16-bit device id and 16-bit vendor id.

**unsigned int dcgmDeviceIdentifiers\_v1::pciSubSystemId**

The 32-bit Sub System Device ID.

**char dcgmDeviceIdentifiers\_v1::driverVersion**

Driver Version.

**unsigned int**

**dcgmDeviceIdentifiers\_v1::virtualizationMode**

Virtualization Mode.

## 2.15. dcgmDeviceMemoryUsage\_v1 Struct Reference

Represents device memory and usage

**unsigned int dcgmDeviceMemoryUsage\_v1::version**

Version Number (dcgmDeviceMemoryUsage\_version).

**unsigned int dcgmDeviceMemoryUsage\_v1::bar1Total**

Total BAR1 size in megabytes.

**unsigned int dcgmDeviceMemoryUsage\_v1::fbTotal**

Total framebuffer memory in megabytes.

**unsigned int dcgmDeviceMemoryUsage\_v1::fbUsed**

Used framebuffer memory in megabytes.

**unsigned int dcgmDeviceMemoryUsage\_v1::fbFree**

Free framebuffer memory in megabytes.

## 2.16. dcgmDevicePidAccountingStats\_v1 Struct Reference

Represents accounting data for one process

**unsigned int dcgmDevicePidAccountingStats\_v1::version**

Version Number. Should match dcgmDevicePidAccountingStats\_version.

**unsigned int dcgmDevicePidAccountingStats\_v1::pid**

Process id of the process these stats are for.

**unsigned int**

**dcgmDevicePidAccountingStats\_v1::gpuUtilization**

Percent of time over the process's lifetime during which one or more kernels was executing on the GPU. Set to DCGM\_INT32\_NOT\_SUPPORTED if is not supported

unsigned int

`dcgmDevicePidAccountingStats_v1::memoryUtilization`

Percent of time over the process's lifetime during which global (device) memory was being read or written. Set to `DCGM_INT32_NOT_SUPPORTED` if is not supported

unsigned long long

`dcgmDevicePidAccountingStats_v1::maxMemoryUsage`

Maximum total memory in bytes that was ever allocated by the process. Set to `DCGM_INT64_NOT_SUPPORTED` if is not supported

unsigned long long

`dcgmDevicePidAccountingStats_v1::startTimestamp`

CPU Timestamp in usec representing start time for the process.

unsigned long long

`dcgmDevicePidAccountingStats_v1::activeTimeUsec`

Amount of time in usec during which the compute context was active. Note that this does not mean the context was being used. `endTimeStamp` can be computed as `startTimestamp + activeTime`

## 2.17. `dcgmDevicePowerLimits_v1` Struct Reference

Represents various power limits

**unsigned int dcgmDevicePowerLimits\_v1::version**

Version Number.

**unsigned int dcgmDevicePowerLimits\_v1::curPowerLimit**

Power management limit associated with this device (in W).

**unsigned int**

**dcgmDevicePowerLimits\_v1::defaultPowerLimit**

Power management limit effective at device boot (in W).

**unsigned int**

**dcgmDevicePowerLimits\_v1::enforcedPowerLimit**

Effective power limit that the driver enforces after taking into account all limiters (in W)

**unsigned int**

**dcgmDevicePowerLimits\_v1::minPowerLimit**

Minimum power management limit (in W).

**unsigned int**

**dcgmDevicePowerLimits\_v1::maxPowerLimit**

Maximum power management limit (in W).

## 2.18. dcgmDeviceSupportedClockSets\_v1 Struct Reference

Represents list of supported clock sets for a device

**unsigned int dcgmDeviceSupportedClockSets\_v1::version**

Version Number (dcgmDeviceSupportedClockSets\_version).

**unsigned int dcgmDeviceSupportedClockSets\_v1::count**

Number of supported clocks.

**struct dcgmClockSet\_t**

**dcgmDeviceSupportedClockSets\_v1::clockSet**

Valid clock sets for the device. Upto count entries are filled.

## 2.19. dcgmDeviceThermals\_v1 Struct Reference

Represents thermal information

**unsigned int dcgmDeviceThermals\_v1::version**

Version Number.

**unsigned int dcgmDeviceThermals\_v1::slowdownTemp**

Slowdown temperature.

**unsigned int dcgmDeviceThermals\_v1::shutdownTemp**

Shutdown temperature.

## 2.20. dcgmDeviceTopology\_v1 Struct Reference

Device topology information

**unsigned int dcgmDeviceTopology\_v1::version**

version number (dcgmDeviceTopology\_version)

**unsignedlong dcgmDeviceTopology\_v1::cpuAffinityMask**

affinity mask for the specified GPU a 1 represents affinity to the CPU in that bit position supports up to 256 cores

## `unsigned int dcgmDeviceTopology_v1::numGpus`

number of valid entries in `gpuPaths`

## `unsigned int dcgmDeviceTopology_v1::gpuId`

`gpuId` to which the path represents

## `dcgmGpuTopologyLevel_t dcgmDeviceTopology_v1::path`

path to the `gpuId` from this GPU. Note that this is a bit-mask of `DCGM_TOPOLOGY_*` values and can contain both PCIe topology and NvLink topology where applicable. For instance: `0x210 = DCGM_TOPOLOGY_CPU | DCGM_TOPOLOGY_NVLINK2` Use the macros `DCGM_TOPOLOGY_PATH_NVLINK` and `DCGM_TOPOLOGY_PATH_PCI` to mask the NvLink and PCI paths, respectively.

## `unsigned int dcgmDeviceTopology_v1::localNvLinkIds`

bits representing the local links connected to `gpuId` e.g. if this field == 3, links 0 and 1 are connected, field is only valid if `NVLINKS` actually exist between GPUs

## 2.21. `dcgmDeviceVgpuEncSessions_v1` Struct Reference

Represents information about active encoder sessions on the given vGPU instance

**unsigned int dcgmDeviceVgpuEncSessions\_v1::version**

Version Number (dcgmDeviceVgpuEncSessions\_version).

**unsigned int dcgmDeviceVgpuEncSessions\_v1::vgpuId**

vGPU instance ID

**unsigned int dcgmDeviceVgpuEncSessions\_v1::sessionId**

Unique session ID.

**unsigned int dcgmDeviceVgpuEncSessions\_v1::pid**

Process ID.

**dcgmEncoderType\_t**

**dcgmDeviceVgpuEncSessions\_v1::codecType**

Video encoder type.

**unsigned int**

**dcgmDeviceVgpuEncSessions\_v1::hResolution**

Current encode horizontal resolution.

**unsigned int**

**dcgmDeviceVgpuEncSessions\_v1::vResolution**

Current encode vertical resolution.

**unsigned int**

**dcgmDeviceVgpuEncSessions\_v1::averageFps**

Moving average encode frames per second.

**unsigned int**

**dcgmDeviceVgpuEncSessions\_v1::averageLatency**

Moving average encode latency in milliseconds.

## 2.22. dcgmDeviceVgpuProcessUtilInfo\_v1 Struct Reference

Represents utilization values for processes running in vGPU VMs using the device

**unsigned int dcgmDeviceVgpuProcessUtilInfo\_v1::version**

Version Number (dcgmDeviceVgpuProcessUtilInfo\_version).

**unsigned int dcgmDeviceVgpuProcessUtilInfo\_v1::vgpuld**

vGPU instance ID

**unsigned int**

**dcgmDeviceVgpuProcessUtilInfo\_v1::vgpuProcessSamplesCount**

Count of processes running in the vGPU VM, for which utilization rates are being reported in this cycle.

**unsigned int dcgmDeviceVgpuProcessUtilInfo\_v1::pid**

Process ID of the process running in the vGPU VM.

**char dcgmDeviceVgpuProcessUtilInfo\_v1::processName**

Process Name of process running in the vGPU VM.

**unsigned int dcgmDeviceVgpuProcessUtilInfo\_v1::smUtil**

GPU utilization of process running in the vGPU VM.

**unsigned int**

**dcgmDeviceVgpuProcessUtilInfo\_v1::memUtil**

Memory utilization of process running in the vGPU VM.

**unsigned int**

**dcgmDeviceVgpuProcessUtilInfo\_v1::encUtil**

Encoder utilization of process running in the vGPU VM.

**unsigned int**

**dcgmDeviceVgpuProcessUtilInfo\_v1::decUtil**

Decoder utilization of process running in the vGPU VM.

## 2.23. dcgmDeviceVgpuTypeInfo\_v1 Struct Reference

Represents static info related to vGPUs supported on the device.



**unsigned int dcgmDeviceVgpuTypeInfo\_v1::version**

Version number (dcgmDeviceVgpuTypeIdStaticInfo\_version).

**dcgmDeviceVgpuTypeInfo\_v1::@2**

**dcgmDeviceVgpuTypeInfo\_v1::vgpuTypeInfo**

vGPU type ID and Supported vGPU type count

**char dcgmDeviceVgpuTypeInfo\_v1::vgpuTypeName**

vGPU type Name

**char dcgmDeviceVgpuTypeInfo\_v1::vgpuTypeClass**

Class of vGPU type.

**char dcgmDeviceVgpuTypeInfo\_v1::vgpuTypeLicense**

license of vGPU type

**int dcgmDeviceVgpuTypeInfo\_v1::deviceId**

device ID of vGPU type

**int dcgmDeviceVgpuTypeInfo\_v1::subsystemId**

Subsystem ID of vGPU type.

**int dcgmDeviceVgpuTypeInfo\_v1::numDisplayHeads**

Count of vGPU's supported display heads.

**int dcgmDeviceVgpuTypeInfo\_v1::maxInstances**

maximum number of vGPU instances creatable on a device for given vGPU type

**int dcgmDeviceVgpuTypeInfo\_v1::frameRateLimit**

Frame rate limit value of the vGPU type.

**int dcgmDeviceVgpuTypeInfo\_v1::maxResolutionX**

vGPU display head's maximum supported resolution in X dimension

**int dcgmDeviceVgpuTypeInfo\_v1::maxResolutionY**

vGPU display head's maximum supported resolution in Y dimension

**int dcgmDeviceVgpuTypeInfo\_v1::fbTotal**

vGPU Total framebuffer size in megabytes

## 2.24. dcgmDeviceVgpuUtilInfo\_v1 Struct Reference

Represents utilization values for vGPUs running on the device

**unsigned int dcgmDeviceVgpuUtilInfo\_v1::version**

Version Number (dcgmDeviceVgpuUtilInfo\_version).

**unsigned int dcgmDeviceVgpuUtilInfo\_v1::vgpuld**

vGPU instance ID

**unsigned int dcgmDeviceVgpuUtilInfo\_v1::smUtil**

GPU utilization for vGPU.

**unsigned int dcgmDeviceVgpuUtilInfo\_v1::memUtil**

Memory utilization for vGPU.

**unsigned int dcgmDeviceVgpuUtilInfo\_v1::encUtil**

Encoder utilization for vGPU.

**unsigned int dcgmDeviceVgpuUtilInfo\_v1::decUtil**

Decoder utilization for vGPU.

## 2.25. dcgmDiagResponse\_v6 Struct Reference

Global diagnostics result structure v6

Since DCGM 2.0

**unsigned int dcgmDiagResponse\_v6::version**

version number (dcgmDiagResult\_version)

**unsigned int dcgmDiagResponse\_v6::gpuCount**

number of valid per GPU results

**unsigned int dcgmDiagResponse\_v6::levelOneTestCount**

number of valid levelOne results

**dcgmDiagTestResult\_v2**

**dcgmDiagResponse\_v6::levelOneResults**

Basic, system-wide test results.

**struct dcgmDiagResponsePerGpu\_v2**

**dcgmDiagResponse\_v6::perGpuResponses**

per GPU test results

**dcgmDiagErrorDetail\_t**

**dcgmDiagResponse\_v6::systemError**

System-wide error reported from NVVS.

**char dcgmDiagResponse\_v6::trainingMsg**

Training Message.

## 2.26. dcgmDiagResponsePerGpu\_v2 Struct Reference

Per GPU diagnostics result structure

**unsigned int dcgmDiagResponsePerGpu\_v2::gpuld**

ID for the GPU this information pertains.

**unsigned int**

**dcgmDiagResponsePerGpu\_v2::hwDiagnosticReturn**

Per GPU hardware diagnostic test return code.

**dcgmDiagTestResult\_v2**

**dcgmDiagResponsePerGpu\_v2::results**

Array with a result for each per-gpu test.

## 2.27. dcgmErrorInfo\_t Struct Reference

Structure to represent error attributes

**unsigned int dcgmErrorInfo\_t::gpuld**

Represents GPU ID.

**short dcgmErrorInfo\_t::fieldId**

One of DCGM\_FI\_?

**int dcgmErrorInfo\_t::status**

One of DCGM\_ST\_?

## 2.28. dcgmFieldGroupInfo\_v1 Struct Reference

Structure to represent information about a field group

**unsigned int dcgmFieldGroupInfo\_v1::version**

Version number (dcgmFieldGroupInfo\_version).

**unsigned int dcgmFieldGroupInfo\_v1::numFieldIds**

Number of entries in fieldIds[] that are valid.

**dcgmFieldGrp\_t dcgmFieldGroupInfo\_v1::fieldGroupId**

ID of this field group.

**char dcgmFieldGroupInfo\_v1::fieldGroupName**

Field Group Name.

**unsigned short dcgmFieldGroupInfo\_v1::fieldIds**

Field ids that belong to this group.

## 2.29. dcgmFieldValue\_v1 Struct Reference

This structure is used to represent value for the field to be queried.

**unsigned int dcgmFieldValue\_v1::version**

version number (dcgmFieldValue\_version1)

**unsigned short dcgmFieldValue\_v1::fieldId**

One of DCGM\_FI\_?

**unsigned short dcgmFieldValue\_v1::fieldType**

One of DCGM\_FT\_?

**int dcgmFieldValue\_v1::status**

Status for the querying the field. DCGM\_ST\_OK or one of DCGM\_ST\_?

**int64\_t dcgmFieldValue\_v1::ts**

Timestamp in usec since 1970.

**int64\_t dcgmFieldValue\_v1::i64**

Int64 value.

**double dcgmFieldValue\_v1::dbl**

Double value.

**char dcgmFieldValue\_v1::str**

NULL terminated string.

**char dcgmFieldValue\_v1::blob**

Binary blob.

**dcgmFieldValue\_v1::@7 dcgmFieldValue\_v1::value**

Value.

## 2.30. dcgmFieldValue\_v2 Struct Reference

This structure is used to represent value for the field to be queried.

**unsigned int dcgmFieldValue\_v2::version**

version number (dcgmFieldValue\_version2)

**dcgm\_field\_entity\_group\_t  
dcgmFieldValue\_v2::entityGroupId**

Entity group this field value's entity belongs to.

**dcgm\_field\_eid\_t dcgmFieldValue\_v2::entityId**

Entity this field value belongs to.

**unsigned short dcgmFieldValue\_v2::fieldId**

One of DCGM\_FI\_?

**unsigned short dcgmFieldValue\_v2::fieldType**

One of DCGM\_FT\_?

**int dcgmFieldValue\_v2::status**

Status for the querying the field. DCGM\_ST\_OK or one of DCGM\_ST\_?

**unsigned int dcgmFieldValue\_v2::unused**

Unused for now to align ts to an 8-byte boundary.

**int64\_t dcgmFieldValue\_v2::ts**

Timestamp in usec since 1970.

**int64\_t dcgmFieldValue\_v2::i64**

Int64 value.

**double dcgmFieldValue\_v2::dbl**

Double value.

**char dcgmFieldValue\_v2::str**

NULL terminated string.

**char dcgmFieldValue\_v2::blob**

Binary blob.

**dcgmFieldValue\_v2::@8 dcgmFieldValue\_v2::value**

Value.

## 2.31. dcgmGpuUsageInfo\_t Struct Reference

Info corresponding to the job on a GPU



**unsigned int dcmGpuUsageInfo\_t::gpuld**

ID of the GPU this pertains to. GPU\_ID\_INVALID = summary information for multiple GPUs.

**long long dcmGpuUsageInfo\_t::energyConsumed**

Energy consumed in milli-watt/seconds.

**struct dcmStatSummaryFp64\_t  
dcmGpuUsageInfo\_t::powerUsage**

Power usage Min/Max/Avg in watts.

**struct dcmStatSummaryInt64\_t  
dcmGpuUsageInfo\_t::pcieRxBandwidth**

PCI-E bytes read from the GPU.

**struct dcmStatSummaryInt64\_t  
dcmGpuUsageInfo\_t::pcieTxBandwidth**

PCI-E bytes written to the GPU.

**long long dcmGpuUsageInfo\_t::pcieReplays**

Count of PCI-E replays that occurred.

**long long dcmGpuUsageInfo\_t::startTime**

User provided job start time in microseconds since 1970.

**long long dcmGpuUsageInfo\_t::endTime**

User provided job end time in microseconds since 1970.

**struct dcmStatSummaryInt32\_t  
dcmGpuUsageInfo\_t::smUtilization**

GPU SM Utilization in percent.

**struct dcmStatSummaryInt32\_t  
dcmGpuUsageInfo\_t::memoryUtilization**

GPU Memory Utilization in percent.

**unsigned int dcmGpuUsageInfo\_t::eccSingleBit**

Deprecated - Count of ECC single bit errors that occurred.

**unsigned int dcgmGpuUsageInfo\_t::eccDoubleBit**

Count of ECC double bit errors that occurred.

**struct dcgmStatSummaryInt32\_t  
dcgmGpuUsageInfo\_t::memoryClock**

Memory clock in MHz.

**struct dcgmStatSummaryInt32\_t  
dcgmGpuUsageInfo\_t::smClock**

SM clock in MHz.

**int dcgmGpuUsageInfo\_t::numXidCriticalErrors**

Number of valid entries in xidCriticalErrorsTs.

**long long dcgmGpuUsageInfo\_t::xidCriticalErrorsTs**

Timestamps of the critical XID errors that occurred.

**int dcgmGpuUsageInfo\_t::numComputePids**

Count of computePids entries that are valid.

**struct dcgmProcessUtilInfo\_t  
dcgmGpuUsageInfo\_t::computePidInfo**

List of compute processes that ran during the job 0=no process

**int dcgmGpuUsageInfo\_t::numGraphicsPids**

Count of graphicsPids entries that are valid.

**struct dcgmProcessUtilInfo\_t  
dcgmGpuUsageInfo\_t::graphicsPidInfo**

List of compute processes that ran during the job 0=no process

**long long dcgmGpuUsageInfo\_t::maxGpuMemoryUsed**

Maximum amount of GPU memory that was used in bytes.

**long long dcgmGpuUsageInfo\_t::powerViolationTime**

Number of microseconds we were at reduced clocks due to power violation.

**long long dcgmGpuUsageInfo\_t::thermalViolationTime**

Number of microseconds we were at reduced clocks due to thermal violation.

**long long dcgmGpuUsageInfo\_t::reliabilityViolationTime**

Amount of microseconds we were at reduced clocks due to the reliability limit

**long long**

**dcgmGpuUsageInfo\_t::boardLimitViolationTime**

Amount of microseconds we were at reduced clocks due to being at the board's max voltage

## `long long dcgmGpuUsageInfo_t::lowUtilizationTime`

Amount of microseconds we were at reduced clocks due to low utilization.

## `long long dcgmGpuUsageInfo_t::syncBoostTime`

Amount of microseconds we were at reduced clocks due to sync boost.

## `dcgmHealthWatchResults_t` `dcgmGpuUsageInfo_t::overallHealth`

The overall health of the system. `dcgmHealthWatchResults_t`.

## `dcgmHealthSystems_t dcgmGpuUsageInfo_t::system`

system to which this information belongs

## `dcgmHealthWatchResults_t` `dcgmGpuUsageInfo_t::health`

health of the specified system on this GPU

## 2.32. `dcgmGroupEntityPair_t` Struct Reference

Represents a `entityGroupId` + `entityId` pair to uniquely identify a given `entityId` inside a group of entities

Added in DCGM 1.5.0

### `dcgm_field_entity_group_t` `dcgmGroupEntityPair_t::entityGroupId`

Entity Group ID entity belongs to.

### `dcgm_field_eid_t dcgmGroupEntityPair_t::entityId`

Entity ID of the entity.

## 2.33. `dcgmGroupInfo_v2` Struct Reference

Structure to store information for DCGM group

Added in DCGM 1.5.0

**unsigned int dcgmGroupInfo\_v2::version**

Version Number (use dcgmGroupInfo\_version2).

**unsigned int dcgmGroupInfo\_v2::count**

count of entityIds returned in entityList

**char dcgmGroupInfo\_v2::groupName**

Group Name.

**struct dcgmGroupEntityPair\_t  
dcgmGroupInfo\_v2::entityList**

List of the entities that are in this group.

## 2.34. dcgmGroupTopology\_v1 Struct Reference

Group topology information

**unsigned int dcgmGroupTopology\_v1::version**

version number (dcgmGroupTopology\_version)

**unsigned long**

**dcgmGroupTopology\_v1::groupCpuAffinityMask**

the CPU affinity mask for all GPUs in the group a 1 represents affinity to the CPU in that bit position supports up to 256 cores

**unsigned int dcgmGroupTopology\_v1::numaOptimalFlag**

a zero value indicates that 1 or more GPUs in the group have a different CPU affinity and thus may not be optimal for certain algorithms

**dcgmGpuTopologyLevel\_t**  
**dcgmGroupTopology\_v1::slowestPath**  
 the slowest path amongst GPUs in the group

## 2.35. dcgmHealthResponse\_v4 Struct Reference

Health response structure version 4 - Simply list the incidents instead of reporting by entity

Since DCGM 2.0

**unsigned int dcgmHealthResponse\_v4::version**  
 The version number of this struct.

**dcgmHealthWatchResults\_t**  
**dcgmHealthResponse\_v4::overallHealth**  
 The overall health of this entire host.

**unsigned int dcgmHealthResponse\_v4::incidentCount**  
 The number of health incidents reported in this struct.

**dcgmIncidentInfo\_t dcgmHealthResponse\_v4::incidents**  
 Report of the errors detected.

## 2.36. dcgmHealthSetParams\_v2 Struct Reference

Structure used to set health watches via the dcgmHealthSet\_v2 API

**unsigned int dcgmHealthSetParams\_v2::version**  
 Version of this struct. Should be dcgmHealthSet\_version2

**dcgmGpuGrp\_t dcgmHealthSetParams\_v2::groupId**

Group ID representing collection of one or more entities. Look at [dcgmGroupCreate](#) for details on creating the group. Alternatively, pass in the group id as DCGM\_GROUP\_ALL\_GPUS to perform operation on all the GPUs or DCGM\_GROUP\_ALL\_NVSITCHES to perform operation on all the NvSwitches.

## `dcgmHealthSystems_t` `dcgmHealthSetParams_v2::systems`

An enum representing systems that should be enabled for health checks logically OR'd together. Refer to `dcgmHealthSystems_t` for details.

## `long long dcgmHealthSetParams_v2::updateInterval`

How often to query the underlying health information from the NVIDIA driver in usec. This should be the same as how often you call `dcgmHealthCheck`

## `double dcgmHealthSetParams_v2::maxKeepAge`

How long to keep data cached for this field in seconds. This should be at least your maximum time between calling `dcgmHealthCheck`

## 2.37. `dcgmHostengineHealth_v1` Struct Reference

Typedef for `dcgmHostengineHealth_v1`

### `unsigned int dcgmHostengineHealth_v1::version`

The version of this request.

### `unsigned int dcgmHostengineHealth_v1::overallHealth`

0 to indicate healthy, or a code to indicate the error

## 2.38. `dcgmIntrospectContext_v1` Struct Reference

Identifies the retrieval context for introspection API calls.

**unsigned int dcgmIntrospectContext\_v1::version**

version number (dcgmIntrospectContext\_version)

**dcgmIntrospectLevel\_t**

**dcgmIntrospectContext\_v1::introspectLvl**

Introspect Level dcgmIntrospectLevel\_t.

**dcgmGpuGrp\_t dcgmIntrospectContext\_v1::fieldGroupId**

Only needed if introspectLvl is DCGM\_INTROSPECT\_LVL\_FIELD\_GROUP.

**unsigned short dcgmIntrospectContext\_v1::fieldId**

Only needed if introspectLvl is DCGM\_INTROSPECT\_LVL\_FIELD.

**unsigned long long**

**dcgmIntrospectContext\_v1::contextId**

Overloaded way to access both fieldGroupId and fieldId.

## 2.39. dcgmIntrospectCpuUtil\_v1 Struct Reference

DCGM CPU Utilization information. Multiply values by 100 to get them in %.

**unsigned int dcgmIntrospectCpuUtil\_v1::version**

version number (dcgmMetadataCpuUtil\_version)

**double dcgmIntrospectCpuUtil\_v1::total**

fraction of device's CPU resources that were used

**double dcgmIntrospectCpuUtil\_v1::kernel**

fraction of device's CPU resources that were used in kernel mode

**double dcgmIntrospectCpuUtil\_v1::user**

fraction of device's CPU resources that were used in user mode

## 2.40. dcgmIntrospectFieldsExecTime\_v1 Struct Reference

DCGM Execution time info for a set of fields



**unsigned int** `dcgmIntrospectFieldsExecTime_v1::version`

version number (`dcgmIntrospectFieldsExecTime_version`)

**long long**

**`dcgmIntrospectFieldsExecTime_v1::meanUpdateFreqUsec`**

the mean update frequency of all fields

**double**

**`dcgmIntrospectFieldsExecTime_v1::recentUpdateUsec`**

the sum of every field's most recent execution time after they have been normalized to `meanUpdateFreqUsec`". This is roughly how long it takes to update fields every `meanUpdateFreqUsec`

**long long**

**`dcgmIntrospectFieldsExecTime_v1::totalEverUpdateUsec`**

The total amount of time, ever, that has been spent updating all the fields.

## 2.41. `dcgmIntrospectFullFieldsExecTime_v2` Struct Reference

Full introspection info for field execution time

Since DCGM 2.0

unsigned int

`dcgmIntrospectFullFieldsExecTime_v2::version`

version number (`dcgmIntrospectFullFieldsExecTime_version`)

struct `dcgmIntrospectFieldsExecTime_v1`

`dcgmIntrospectFullFieldsExecTime_v2::aggregateInfo`

info that includes global and device scope

int `dcgmIntrospectFullFieldsExecTime_v2::hasGlobalInfo`

0 means `globalInfo` is populated, !0 means it's not

struct `dcgmIntrospectFieldsExecTime_v1`

`dcgmIntrospectFullFieldsExecTime_v2::globalInfo`

info that only includes global field scope

unsigned short

`dcgmIntrospectFullFieldsExecTime_v2::gpuInfoCount`

count of how many entries in `gpuInfo` are populated

unsigned int

`dcgmIntrospectFullFieldsExecTime_v2::gpusForGpuInfo`

the GPU ID at a given index identifies which gpu the corresponding entry in `gpuInfo` is from

struct `dcgmIntrospectFieldsExecTime_v1`

`dcgmIntrospectFullFieldsExecTime_v2::gpuInfo`

info that is separated by the GPU ID that the watches were for

## 2.42. `dcgmIntrospectFullMemory_v1` Struct Reference

Full introspection info for field memory

**unsigned int dcgmIntrospectFullMemory\_v1::version**

version number (dcgmIntrospectFullMemory\_version)

**struct dcgmIntrospectMemory\_v1**

**dcgmIntrospectFullMemory\_v1::aggregateInfo**

info that includes global and device scope

**int dcgmIntrospectFullMemory\_v1::hasGlobalInfo**

0 means globalInfo is populated, !0 means it's not

**struct dcgmIntrospectMemory\_v1**

**dcgmIntrospectFullMemory\_v1::globalInfo**

info that only includes global field scope

**unsigned short**

**dcgmIntrospectFullMemory\_v1::gpuInfoCount**

count of how many entries in gpuInfo are populated

**unsigned int**

**dcgmIntrospectFullMemory\_v1::gpuIdsForGpuInfo**

the GPU ID at a given index identifies which gpu the corresponding entry in `gpuInfo` is from

**struct dcgmIntrospectMemory\_v1**

**dcgmIntrospectFullMemory\_v1::gpuInfo**

info that is divided by the GPU ID that the watches were for

## 2.43. dcgmIntrospectMemory\_v1 Struct Reference

DCGM Memory usage information

**unsigned int dcgmIntrospectMemory\_v1::version**

version number (dcgmIntrospectMemory\_version)

**long long dcgmIntrospectMemory\_v1::bytesUsed**

number of bytes

## 2.44. dcgmJobInfo\_v3 Struct Reference

To store job statistics The following fields are not applicable in the summary info:

- ▶ pcieRxBandwidth (Min/Max)
- ▶ pcieTxBandwidth (Min/Max)
- ▶ smUtilization (Min/Max)
- ▶ memoryUtilization (Min/Max)
- ▶ memoryClock (Min/Max)
- ▶ smClock (Min/Max)
- ▶ processSamples

The average value in the above fields (in the summary) is the average of the averages of respective fields from all GPUs

**unsigned int dcgmJobInfo\_v3::version**

Version of this message (dcgmPidInfo\_version).

**int dcgmJobInfo\_v3::numGpus**

Number of GPUs that are valid in gpus[].

**struct dcgmGpuUsageInfo\_t dcgmJobInfo\_v3::summary**

Summary information for all GPUs listed in gpus[].

**struct dcgmGpuUsageInfo\_t dcgmJobInfo\_v3::gpus**

Per-GPU information for this PID.

## 2.45. dcgmMigEntityInfo\_t Struct Reference

Provides additional information about location of MIG entities.

**char dcgmMigEntityInfo\_t::gpuUuid**

GPU UUID

## unsigned int dcgmMigEntityInfo\_t::nvmlGpuIndex

GPU index from NVML

## unsigned int dcgmMigEntityInfo\_t::nvmlInstancelId

GPU instance index within GPU. 0 to N. -1 for GPU entities

## unsigned int dcgmMigEntityInfo\_t::nvmlComputeInstancelId

GPU Compute instance index within GPU instance. 0 to N. -1 for GPU Instance and GPU entities

## unsigned int dcgmMigEntityInfo\_t::nvmlMigProfileId

Unique profile ID for GPU or Compute instances. -1 GPU entities

**See also:**

`nvmlComputeInstanceProfileInfo_st`

`nvmlGpuInstanceProfileInfo_st`

## unsigned int dcgmMigEntityInfo\_t::nvmlProfileSlices

Number of slices in the MIG profile

## 2.46. dcgmMigHierarchy\_v1 Struct Reference

Structure to store the GPU hierarchy for a system

Added in DCGM 2.0

## 2.47. dcgmMigHierarchyInfo\_t Struct Reference

Represents a pair of entity pairings to uniquely identify an entity and its place in the hierarchy.

```
struct dcgmGroupEntityPair_t
dcgmMigHierarchyInfo_t::entity
```

Entity id and type for the entity in question.

```
struct dcgmGroupEntityPair_t
dcgmMigHierarchyInfo_t::parent
```

Entity id and type for the parent of the entity in question.

```
dcgmMigProfile_t dcgmMigHierarchyInfo_t::sliceProfile
```

Entity MIG profile identifier.

## 2.48. dcgmModuleGetStatusesModule\_t Struct Reference

Status of all of the modules of the host engine

```
dcgmModuleId_t dcgmModuleGetStatusesModule_t::id
```

ID of this module.

```
dcgmModuleStatus_t
dcgmModuleGetStatusesModule_t::status
```

Status of this module.

## 2.49. dcgmNvLinkGpuLinkStatus\_v1 Struct Reference

State of NvLink links for a GPU

`dcgm_field_eid_t`  
`dcgmNvLinkGpuLinkStatus_v1::entityId`  
Entity ID of the GPU (`gpuId`).

`dcgmNvLinkLinkState_t`  
`dcgmNvLinkGpuLinkStatus_v1::linkState`  
Per-GPU link states.

## 2.50. `dcgmNvLinkNvSwitchLinkStatus_t` Struct Reference

State of `NvLink` links for a `NvSwitch`

`dcgm_field_eid_t`  
`dcgmNvLinkNvSwitchLinkStatus_t::entityId`  
Entity ID of the `NvSwitch` (`physicalId`).

`dcgmNvLinkLinkState_t`  
`dcgmNvLinkNvSwitchLinkStatus_t::linkState`  
Per-`NvSwitch` link states.

## 2.51. `dcgmNvLinkStatus_v1` Struct Reference

Status of all of the `NvLinks` in a given system

**unsigned int dcgmNvLinkStatus\_v1::version**

Version of this request. Should be dcgmNvLinkStatus\_version1.

**unsigned int dcgmNvLinkStatus\_v1::numGpus**

Number of entries in gpus[] that are populated.

**struct dcgmNvLinkGpuLinkStatus\_v1  
dcgmNvLinkStatus\_v1::gpus**

Per-GPU NvLink link statuses.

**unsigned int dcgmNvLinkStatus\_v1::numNvSwitches**

Number of entries in nvSwitches[] that are populated.

**struct dcgmNvLinkNvSwitchLinkStatus\_t  
dcgmNvLinkStatus\_v1::nvSwitches**

Per-NvSwitch link statuses.

## 2.52. dcgmPidInfo\_v2 Struct Reference

To store process statistics



**unsigned int dcgmPidInfo\_v2::version**

Version of this message (dcgmPidInfo\_version).

**unsigned int dcgmPidInfo\_v2::pid**

PID of the process.

**int dcgmPidInfo\_v2::numGpus**

Number of GPUs that are valid in GPUs.

**struct dcgmPidSingleInfo\_t dcgmPidInfo\_v2::summary**

Summary information for all GPUs listed in gpus[].

**struct dcgmPidSingleInfo\_t dcgmPidInfo\_v2::gpus**

Per-GPU information for this PID.

## 2.53. dcgmPidSingleInfo\_t Struct Reference

Info corresponding to single PID

**unsigned int dcgmPidSingleInfo\_t::gpuld**

ID of the GPU this pertains to. GPU\_ID\_INVALID = summary information for multiple GPUs.

**long long dcgmPidSingleInfo\_t::energyConsumed**

Energy consumed by the gpu in milli-watt/seconds.

**struct dcgmStatSummaryInt64\_t  
dcgmPidSingleInfo\_t::pcieRxBandwidth**

PCI-E bytes read from the GPU.

**struct dcgmStatSummaryInt64\_t  
dcgmPidSingleInfo\_t::pcieTxBandwidth**

PCI-E bytes written to the GPU.

**long long dcgmPidSingleInfo\_t::pcieReplays**

Count of PCI-E replays that occurred.

**long long dcgmPidSingleInfo\_t::startTime**

Process start time in microseconds since 1970.

**long long dcgmPidSingleInfo\_t::endTime**

Process end time in microseconds since 1970 or reported as 0 if the process is not completed.

**struct dcgmProcessUtilInfo\_t  
dcgmPidSingleInfo\_t::processUtilization**

Process SM and Memory Utilization (in percent).

**struct dcgmStatSummaryInt32\_t  
dcgmPidSingleInfo\_t::smUtilization**

GPU SM Utilization in percent.

**struct dcgmStatSummaryInt32\_t  
dcgmPidSingleInfo\_t::memoryUtilization**

GPU Memory Utilization in percent.

**unsigned int dcgmPidSingleInfo\_t::eccSingleBit**

Deprecated - Count of ECC single bit errors that occurred.

**unsigned int dcgmPidSingleInfo\_t::eccDoubleBit**

Count of ECC double bit errors that occurred.

**struct dcgmStatSummaryInt32\_t  
dcgmPidSingleInfo\_t::memoryClock**

Memory clock in MHz.

**struct dcgmStatSummaryInt32\_t  
dcgmPidSingleInfo\_t::smClock**

SM clock in MHz.

**int dcgmPidSingleInfo\_t::numXidCriticalErrors**

Number of valid entries in xidCriticalErrorsTs.

**long long dcgmPidSingleInfo\_t::xidCriticalErrorsTs**

Timestamps of the critical XID errors that occurred.

**int dcgmPidSingleInfo\_t::numOtherComputePids**

Count of otherComputePids entries that are valid.

**unsigned int dcgmPidSingleInfo\_t::otherComputePids**

Other compute processes that ran. 0=no process.

**int dcgmPidSingleInfo\_t::numOtherGraphicsPids**

Count of otherGraphicsPids entries that are valid.

**unsigned int dcgmPidSingleInfo\_t::otherGraphicsPids**

Other graphics processes that ran. 0=no process.

**long long dcgmPidSingleInfo\_t::maxGpuMemoryUsed**

Maximum amount of GPU memory that was used in bytes.

**long long dcgmPidSingleInfo\_t::powerViolationTime**

Number of microseconds we were at reduced clocks due to power violation.

**long long dcgmPidSingleInfo\_t::thermalViolationTime**

Number of microseconds we were at reduced clocks due to thermal violation.

**long long dcgmPidSingleInfo\_t::reliabilityViolationTime**

Amount of microseconds we were at reduced clocks due to the reliability limit

**long long dcgmPidSingleInfo\_t::boardLimitViolationTime**

Amount of microseconds we were at reduced clocks due to being at the board's max voltage

**long long dcgmPidSingleInfo\_t::lowUtilizationTime**

Amount of microseconds we were at reduced clocks due to low utilization.

**long long dcgmPidSingleInfo\_t::syncBoostTime**

Amount of microseconds we were at reduced clocks due to sync boost.

**dcgmHealthWatchResults\_t****dcgmPidSingleInfo\_t::overallHealth**

The overall health of the system. dcgmHealthWatchResults\_t.

**dcgmHealthSystems\_t dcgmPidSingleInfo\_t::system**

system to which this information belongs

**dcgmHealthWatchResults\_t dcgmPidSingleInfo\_t::health**

health of the specified system on this GPU

## 2.54. dcgmPolicy\_v1 Struct Reference

Define the structure that specifies a policy to be enforced for a GPU

**unsigned int dcgmPolicy\_v1::version**

version number (dcgmPolicy\_version)

**dcgmPolicyCondition\_t dcgmPolicy\_v1::condition**

Condition(s) to access dcgmPolicyCondition\_t.

**dcgmPolicyMode\_t dcgmPolicy\_v1::mode**

Mode of operation dcgmPolicyMode\_t.

**dcgmPolicyIsolation\_t dcgmPolicy\_v1::isolation**

Isolation level after a policy violation dcgmPolicyIsolation\_t.

**dcgmPolicyAction\_t dcgmPolicy\_v1::action**

Action to perform after a policy violation dcgmPolicyAction\_t action.

**dcgmPolicyValidation\_t dcgmPolicy\_v1::validation**

Validation to perform after action is taken dcgmPolicyValidation\_t.

**dcgmPolicyFailureResp\_t dcgmPolicy\_v1::response**

Failure to validation response dcgmPolicyFailureResp\_t.

**struct dcgmPolicyConditionParams\_t****dcgmPolicy\_v1::parms**

Parameters for the condition fields.

## 2.55. dcgmPolicyCallbackResponse\_v1 Struct Reference

Define the structure that is given to the callback function

**unsigned int dcgmPolicyCallbackResponse\_v1::version**

version number (dcgmPolicyCallbackResponse\_version)

**dcgmPolicyCondition\_t**

**dcgmPolicyCallbackResponse\_v1::condition**

Condition that was violated.

**struct dcgmPolicyConditionDbe\_t**

**dcgmPolicyCallbackResponse\_v1::dbe**

ECC DBE return structure.

**struct dcgmPolicyConditionPci\_t**

**dcgmPolicyCallbackResponse\_v1::pci**

PCI replay error return structure.

**struct dcgmPolicyConditionMpr\_t**

**dcgmPolicyCallbackResponse\_v1::mpr**

Max retired pages limit return structure.

**struct dcgmPolicyConditionThermal\_t**

**dcgmPolicyCallbackResponse\_v1::thermal**

Thermal policy violations return structure.

**struct dcgmPolicyConditionPower\_t**

**dcgmPolicyCallbackResponse\_v1::power**

Power policy violations return structure.

**struct dcgmPolicyConditionNvlink\_t**

**dcgmPolicyCallbackResponse\_v1::nvlink**

Nvlink policy violations return structure.

**struct dcgmPolicyConditionXID\_t**

**dcgmPolicyCallbackResponse\_v1::xid**

XID policy violations return structure.

## 2.56. dcgmPolicyConditionDbe\_t Struct Reference

Define the ECC DBE return structure

**long long dcgmPolicyConditionDbe\_t::timestamp**

timestamp of the error

**enum dcgmPolicyConditionDbe\_t::@5**

**dcgmPolicyConditionDbe\_t::location**

location of the error

**unsigned int dcgmPolicyConditionDbe\_t::numerrors**

number of errors

## 2.57. dcgmPolicyConditionMpr\_t Struct Reference

Define the maximum pending retired pages limit return structure

**long long dcgmPolicyConditionMpr\_t::timestamp**

timestamp of the error

**unsigned int dcgmPolicyConditionMpr\_t::sbepages**

number of pending pages due to SBE

**unsigned int dcgmPolicyConditionMpr\_t::dbepages**

number of pending pages due to DBE

## 2.58. dcgmPolicyConditionNvlink\_t Struct Reference

Define the nvlink policy violations return structure

**long long dcgmPolicyConditionNvlink\_t::timestamp**

timestamp of the error

**unsigned short dcgmPolicyConditionNvlink\_t::fieldId**

Nvlink counter field ID that violated policy.

**unsigned int dcgmPolicyConditionNvlink\_t::counter**

Nvlink counter value that violated policy.

## 2.59. dcgmPolicyConditionParams\_t Struct Reference

Structure for policy condition parameters. This structure contains a tag that represents the type of the value being passed as well as a "val" which is a union of the possible value types. For example, to pass a true boolean: tag = BOOL, val.boolean = 1.

## 2.60. dcgmPolicyConditionPci\_t Struct Reference

Define the PCI replay error return structure

**long long dcgmPolicyConditionPci\_t::timestamp**

timestamp of the error

**unsigned int dcgmPolicyConditionPci\_t::counter**

value of the PCIe replay counter

## 2.61. dcgmPolicyConditionPower\_t Struct Reference

Define the power policy violations return structure



**long long dcgmPolicyConditionPower\_t::timestamp**

timestamp of the error

**unsigned int**

**dcgmPolicyConditionPower\_t::powerViolation**

Power value reached that violated policy.

## 2.62. dcgmPolicyConditionThermal\_t Struct Reference

Define the thermal policy violations return structure

**long long dcgmPolicyConditionThermal\_t::timestamp**

timestamp of the error

**unsigned int**

**dcgmPolicyConditionThermal\_t::thermalViolation**

Temperature reached that violated policy.

## 2.63. dcgmPolicyConditionXID\_t Struct Reference

Define the xid policy violations return structure

**long long dcgmPolicyConditionXID\_t::timestamp**

Timestamp of the error.

**unsigned int dcgmPolicyConditionXID\_t::errnum**

The XID error number.

## 2.64. dcgmPolicyViolationNotify\_t Struct Reference

Structure to fill when a user queries for policy violations

**unsigned int dcgmPolicyViolationNotify\_t::gpuld**

gpu ID

**unsigned int**

**dcgmPolicyViolationNotify\_t::violationOccurred**

a violation based on the bit values in dcgmPolicyCondition\_t

## 2.65. dcgmProcessUtilInfo\_t Struct Reference

per process utilization rates

## 2.66. dcgmProcessUtilSample\_t Struct Reference

Internal structure used to get the PID and the corresponding utilization rate

## 2.67. dcgmProfMetricGroupInfo\_t Struct Reference

Structure to return all of the profiling metric groups that are available for the given groupId.

**unsigned short dcgmProfMetricGroupInfo\_t::majorId**

Major ID of this metric group. Metric groups with the same majorId cannot be watched concurrently with other metric groups with the same majorId

**unsigned short dcgmProfMetricGroupInfo\_t::minorId**

Minor ID of this metric group. This distinguishes metric groups within the same major metric group from each other

**unsigned int dcgmProfMetricGroupInfo\_t::numFieldIds**

Number of field IDs that are populated in fieldIds[].

**unsigned short dcgmProfMetricGroupInfo\_t::fieldIds**

DCGM Field IDs that are part of this profiling group. See DCGM\_FI\_PROF\_\* definitions in dcgm\_fields.h for details.

## 2.68. dcgmProfUnwatchFields\_v1 Struct Reference

Structure to pass to `dcgmProfUnwatchFields` when unwatching profiling metrics

### `unsigned int dcgmProfUnwatchFields_v1::version`

Version of this request. Should be `dcgmProfUnwatchFields_version`.

### `dcgmGpuGrp_t dcgmProfUnwatchFields_v1::groupid`

Group ID representing collection of one or more GPUs. Look at `dcgmGroupCreate` for details on creating the group. Alternatively, pass in the group id as `DCGM_GROUP_ALL_GPUS` to perform operation on all the GPUs. The GPUs of the group must all be identical or `DCGM_ST_GROUP_INCOMPATIBLE` will be returned by this API.

### `unsigned int dcgmProfUnwatchFields_v1::flags`

For future use. Set to 0 for now.

## 2.69. dcgmProfWatchFields\_v1 Struct Reference

Structure to pass to `dcgmProfWatchFields()` when watching profiling metrics

### `unsigned int dcgmProfWatchFields_v1::version`

Version of this request. Should be `dcgmProfWatchFields_version`.

### `dcgmGpuGrp_t dcgmProfWatchFields_v1::groupid`

Group ID representing collection of one or more GPUs. Look at `dcgmGroupCreate` for details on creating the group. Alternatively, pass in the group id as `DCGM_GROUP_ALL_GPUS` to perform operation on all the GPUs. The GPUs of the group must all be identical or `DCGM_ST_GROUP_INCOMPATIBLE` will be returned by this API.

**unsigned int dcgmProfWatchFields\_v1::numFieldIds**

Number of field IDs that are being passed in fieldIds[].

**unsigned short dcgmProfWatchFields\_v1::fieldIds**

DCGM\_FI\_PROF\_? field IDs to watch.

**long long dcgmProfWatchFields\_v1::updateFreq**

How often to update this field in usec. Note that profiling metrics may need to be sampled more frequently than this value. See `dcgmProfMetricGroupInfo_t.minUpdateFreqUsec` of the metric group matching `metricGroupTag` to see what this minimum is. If `minUpdateFreqUsec < updateFreq` then samples will be aggregated to `updateFreq` intervals in DCGM's internal cache.

**double dcgmProfWatchFields\_v1::maxKeepAge**

How long to keep data for every fieldId in seconds.

**int dcgmProfWatchFields\_v1::maxKeepSamples**

Maximum number of samples to keep for each fieldId. 0=no limit.

**unsigned int dcgmProfWatchFields\_v1::flags**

For future use. Set to 0 for now.

**2.70. dcgmRunningProcess\_v1 Struct Reference**

Running process information for a compute or graphics process

**unsigned int dcgRunningProcess\_v1::version**

Version of this message (dcgRunningProcess\_version).

**unsigned int dcgRunningProcess\_v1::pid**

PID of the process.

**unsigned long long**

**dcgRunningProcess\_v1::memoryUsed**

GPU memory used by this process in bytes.

## 2.71. dcgSettingsSetLoggingSeverity\_v1 Struct Reference

Version 1 of dcgSettingsSetLoggingSeverity\_t

## 2.72. dcgStartEmbeddedV2Params\_v1 Struct Reference

Options for dcgStartEmbedded\_v2

Added in DCGM 2.0.0

**unsigned int dcgStartEmbeddedV2Params\_v1::version**

Version number. Use dcgStartEmbeddedV2Params\_version1

**dcgOperationMode\_t**

**dcgStartEmbeddedV2Params\_v1::opMode**

IN: Collect data automatically or manually when asked by the user.

**dcgHandle\_t**

**dcgStartEmbeddedV2Params\_v1::dcgHandle**

OUT: DCGM Handle to use for API calls

**const char \*dcgStartEmbeddedV2Params\_v1::logFile**

IN: File that DCGM should log to. NULL = do not log. '-' = stdout

## `DcgmLoggingSeverity_t` `dcgmStartEmbeddedV2Params_v1::severity`

IN: Severity at which DCGM should log to logFile

## `unsigned int` `dcgmStartEmbeddedV2Params_v1::blackListCount`

IN: Number of modules that to be blacklisted in blackList[]

## `unsigned int dcgmStartEmbeddedV2Params_v1::unused`

IN: Unused. Set to 0. Aligns structure to 8-bytes

## 2.73. `dcgmStatSummaryFp64_t` Struct Reference

Summary of time series data in double-precision format. Each value will either be set or be a BLANK value. Check for blank with the `DCGM_FP64_IS_BLANK()` macro.

### See also:

See `dcgmvalue.h` for the actual values of BLANK values

## `double dcgmStatSummaryFp64_t::minValue`

Minimum value of the samples looked at.

## `double dcgmStatSummaryFp64_t::maxValue`

Maximum value of the samples looked at.

## `double dcgmStatSummaryFp64_t::average`

Simple average of the samples looked at. Blank values are ignored for this calculation.

## 2.74. `dcgmStatSummaryInt32_t` Struct Reference

Same as `dcgmStatSummaryInt64_t`, but with 32-bit integer values

## `int dcgmStatSummaryInt32_t::minValue`

Minimum value of the samples looked at.

## `int dcgmStatSummaryInt32_t::maxValue`

Maximum value of the samples looked at.

## `int dcgmStatSummaryInt32_t::average`

Simple average of the samples looked at. Blank values are ignored for this calculation.

## 2.75. `dcgmStatSummaryInt64_t` Struct Reference

Summary of time series data in int64 format.

Each value will either be set or be a BLANK value. Check for blank with the `DCGM_INT64_IS_BLANK()` macro.

### **See also:**

See `dcgmvalue.h` for the actual values of BLANK values

## `long long dcgmStatSummaryInt64_t::minValue`

Minimum value of the samples looked at.

## `long long dcgmStatSummaryInt64_t::maxValue`

Maximum value of the samples looked at.

## `long long dcgmStatSummaryInt64_t::average`

Simple average of the samples looked at. Blank values are ignored for this calculation.

## 2.76. `dcgmVersionInfo_v2` Struct Reference

Structure to describe the DCGM build environment ver 2.0

## `char dcgmVersionInfo_v2::rawBuildInfoString`

Raw form of the DCGM build info. There may be multiple kv-pairs separated by semicolon (;). Every pair is separated by a colon char (:). Only the very first colon is considered as a separation. Values can contain colon chars. Values and Keys cannot contain semicolon chars. Usually defined keys are:

version : DCGM Version. arch : Target DCGM Architecture. buildid : Build ID. Usually a sequential number. commit : Commit ID (Usually a git commit hash). author : Author of the commit above. branch : Branch (Usually a git branch that was used for the build). buildtype : Build Type. builddate : Date of the build. buildplatform : Platform where the build was made. Any or all keys may be absent. This values are for reference only are not supposed to participate in some complicated logic.



# Chapter 3.

## DATA FIELDS

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

### A

#### **action**

[dcmPolicy\\_v1](#)

#### **activeTimeUsec**

[dcmDevicePidAccountingStats\\_v1](#)

#### **addressIsUnixSocket**

[dcmConnectV2Params\\_v2](#)

#### **aggregateInfo**

[dcmIntrospectFullFieldsExecTime\\_v2](#)

[dcmIntrospectFullMemory\\_v1](#)

#### **average**

[dcmStatSummaryFp64\\_t](#)

[dcmStatSummaryInt64\\_t](#)

[dcmStatSummaryInt32\\_t](#)

#### **averageFps**

[dcmDeviceEncStats\\_v1](#)

[dcmDeviceFbcStats\\_v1](#)

[dcmDeviceFbcSessionInfo\\_v1](#)

[dcmDeviceVgpuEncSessions\\_v1](#)

#### **averageLatency**

[dcmDeviceEncStats\\_v1](#)

[dcmDeviceFbcSessionInfo\\_v1](#)

[dcmDeviceFbcStats\\_v1](#)

[dcmDeviceVgpuEncSessions\\_v1](#)

**B****bar1Total**

dcmDeviceMemoryUsage\_v1

**blackListCount**

dcmStartEmbeddedV2Params\_v1

**blob**

dcmFieldValue\_v2

dcmFieldValue\_v1

**boardLimitViolationTime**

dcmPidSingleInfo\_t

dcmGpuUsageInfo\_t

**brandName**

dcmDeviceIdentifiers\_v1

**bytesUsed**

dcmIntrospectMemory\_v1

**C****clockSet**

dcmDeviceSupportedClockSets\_v1

**clockSets**

dcmDeviceAttributes\_v1

**codecType**

dcmDeviceVgpuEncSessions\_v1

**computeMode**

dcmConfig\_v1

**computePidInfo**

dcmGpuUsageInfo\_t

**condition**

dcmPolicy\_v1

dcmPolicyCallbackResponse\_v1

**contextId**

dcmIntrospectContext\_v1

**count**

dcmGroupInfo\_v2

dcmDeviceSupportedClockSets\_v1

**counter**

dcmPolicyConditionNvlink\_t

dcmPolicyConditionPci\_t

**cpuAffinityMask**

dcmDeviceTopology\_v1

**curPowerLimit**

dcmDevicePowerLimits\_v1

**D****dbe**`dcgmPolicyCallbackResponse_v1`**dbepages**`dcgmPolicyConditionMpr_t`**dbl**`dcgmFieldValue_v2``dcgmFieldValue_v1`**dcgmHandle**`dcgmStartEmbeddedV2Params_v1`**decUtil**`dcgmDeviceVgpuUtilInfo_v1``dcgmDeviceVgpuProcessUtilInfo_v1`**defaultPowerLimit**`dcgmDevicePowerLimits_v1`**deviceId**`dcgmDeviceVgpuTypeInfo_v1`**deviceName**`dcgmDeviceIdentifiers_v1`**displayOrdinal**`dcgmDeviceFbcSessionInfo_v1`**driverVersion**`dcgmDeviceIdentifiers_v1`**E****eccDoubleBit**`dcgmPidSingleInfo_t``dcgmGpuUsageInfo_t`**eccMode**`dcgmConfig_v1`**eccSingleBit**`dcgmPidSingleInfo_t``dcgmGpuUsageInfo_t`**encUtil**`dcgmDeviceVgpuUtilInfo_v1``dcgmDeviceVgpuProcessUtilInfo_v1`**endTime**`dcgmPidSingleInfo_t``dcgmGpuUsageInfo_t`**energyConsumed**`dcgmPidSingleInfo_t``dcgmGpuUsageInfo_t`

**enforcedPowerLimit**

dcmDevicePowerLimits\_v1

**entity**

dcmMigHierarchyInfo\_t

**entityGroupId**

dcmFieldValue\_v2

dcmGroupEntityPair\_t

**entityId**

dcmGroupEntityPair\_t

dcmNvLinkNvSwitchLinkStatus\_t

dcmFieldValue\_v2

dcmNvLinkGpuLinkStatus\_v1

**entityLevel**

dcm\_field\_meta\_t

**entityList**

dcmGroupInfo\_v2

**errnum**

dcmPolicyConditionXID\_t

**F****fbFree**

dcmDeviceMemoryUsage\_v1

**fbTotal**

dcmDeviceMemoryUsage\_v1

dcmDeviceVgpuTypeInfo\_v1

**fbUsed**

dcmDeviceMemoryUsage\_v1

**fieldGroupId**

dcmFieldGroupInfo\_v1

dcmIntrospectContext\_v1

**fieldGroupName**

dcmFieldGroupInfo\_v1

**fieldId**

dcmErrorInfo\_t

dcmPolicyConditionNvlink\_t

dcmFieldValue\_v1

dcmFieldValue\_v2

dcmIntrospectContext\_v1

dcm\_field\_meta\_t

**fieldIds**

dcmProfWatchFields\_v1

dcmProfMetricGroupInfo\_t

dcmFieldGroupInfo\_v1

**fieldType**

dcgm\_field\_meta\_t  
 dcgmFieldValue\_v1  
 dcgmFieldValue\_v2

**flags**

dcgmProfUnwatchFields\_v1  
 dcgmProfWatchFields\_v1

**frameRateLimit**

dcgmDeviceVgpuTypeInfo\_v1

**G****globalInfo**

dcgmIntrospectFullFieldsExecTime\_v2  
 dcgmIntrospectFullMemory\_v1

**gpuCount**

dcgmDiagResponse\_v6

**gpuId**

dcgmErrorInfo\_t  
 dcgmGpuUsageInfo\_t  
 dcgmDiagResponsePerGpu\_v2  
 dcgmConfig\_v1  
 dcgmDeviceTopology\_v1  
 dcgmPolicyViolationNotify\_t  
 dcgmPidSingleInfo\_t

**gpuIdsForGpuInfo**

dcgmIntrospectFullFieldsExecTime\_v2  
 dcgmIntrospectFullMemory\_v1

**gpuInfo**

dcgmIntrospectFullFieldsExecTime\_v2  
 dcgmIntrospectFullMemory\_v1

**gpuInfoCount**

dcgmIntrospectFullMemory\_v1  
 dcgmIntrospectFullFieldsExecTime\_v2

**gpus**

dcgmPidInfo\_v2  
 dcgmJobInfo\_v3  
 dcgmNvLinkStatus\_v1

**gpuUtilization**

dcgmDevicePidAccountingStats\_v1

**gpuUuid**

dcgmMigEntityInfo\_t

**graphicsPidInfo**

dcgmGpuUsageInfo\_t

**groupCpuAffinityMask**

dcmGroupTopology\_v1

**groupId**

dcmProfWatchFields\_v1

dcmProfUnwatchFields\_v1

dcmHealthSetParams\_v2

**groupName**

dcmGroupInfo\_v2

**H****hasGlobalInfo**

dcmIntrospectFullFieldsExecTime\_v2

dcmIntrospectFullMemory\_v1

**health**

dcmGpuUsageInfo\_t

dcmPidSingleInfo\_t

**hMaxResolution**

dcmDeviceFbcSessionInfo\_v1

**hResolution**

dcmDeviceVgpuEncSessions\_v1

dcmDeviceFbcSessionInfo\_v1

**hwDiagnosticReturn**

dcmDiagResponsePerGpu\_v2

**I****i64**

dcmFieldValue\_v1

dcmFieldValue\_v2

**id**

dcmModuleGetStatusesModule\_t

**identifiers**

dcmDeviceAttributes\_v1

**incidentCount**

dcmHealthResponse\_v4

**incidents**

dcmHealthResponse\_v4

**inforomImageVersion**

dcmDeviceIdentifiers\_v1

**introspectLvl**

dcmIntrospectContext\_v1

**isolation**

dcmPolicy\_v1

**K****kernel**

dcgmIntrospectCpuUtil\_v1

**L****levelOneResults**

dcgmDiagResponse\_v6

**levelOneTestCount**

dcgmDiagResponse\_v6

**linkState**

dcgmNvLinkNvSwitchLinkStatus\_t

dcgmNvLinkGpuLinkStatus\_v1

**localNvLinkIds**

dcgmDeviceTopology\_v1

**location**

dcgmPolicyConditionDbe\_t

**logFile**

dcgmStartEmbeddedV2Params\_v1

**lowUtilizationTime**

dcgmPidSingleInfo\_t

dcgmGpuUsageInfo\_t

**M****majorId**

dcgmProfMetricGroupInfo\_t

**maxGpuMemoryUsed**

dcgmPidSingleInfo\_t

dcgmGpuUsageInfo\_t

**maxInstances**

dcgmDeviceVgpuTypeInfo\_v1

**maxKeepAge**

dcgmHealthSetParams\_v2

dcgmProfWatchFields\_v1

**maxKeepSamples**

dcgmProfWatchFields\_v1

**maxMemoryUsage**

dcgmDevicePidAccountingStats\_v1

**maxPowerLimit**

dcgmDevicePowerLimits\_v1

**maxResolutionX**

dcgmDeviceVgpuTypeInfo\_v1

**maxResolutionY**

dcgmDeviceVgpuTypeInfo\_v1

**maxValue**

dcgmStatSummaryInt64\_t  
 dcgmStatSummaryInt32\_t  
 dcgmStatSummaryFp64\_t

**meanUpdateFreqUseC**

dcgmIntrospectFieldsExecTime\_v1

**memClock**

dcgmClockSet\_v1

**memoryClock**

dcgmPidSingleInfo\_t  
 dcgmGpuUsageInfo\_t

**memoryUsage**

dcgmDeviceAttributes\_v1

**memoryUsed**

dcgmRunningProcess\_v1

**memoryUtilization**

dcgmGpuUsageInfo\_t  
 dcgmDevicePidAccountingStats\_v1  
 dcgmPidSingleInfo\_t

**memUtil**

dcgmDeviceVgpuProcessUtilInfo\_v1  
 dcgmDeviceVgpuUtilInfo\_v1

**minorId**

dcgmProfMetricGroupInfo\_t

**minPowerLimit**

dcgmDevicePowerLimits\_v1

**minValue**

dcgmStatSummaryFp64\_t  
 dcgmStatSummaryInt64\_t  
 dcgmStatSummaryInt32\_t

**mode**

dcgmPolicy\_v1

**mpr**

dcgmPolicyCallbackResponse\_v1

**N****numaOptimalFlag**

dcgmGroupTopology\_v1

**numComputePids**

dcgmGpuUsageInfo\_t

**numDisplayHeads**

dcgmDeviceVgpuTypeInfo\_v1



**numerrors**

dcgmPolicyConditionDbe\_t

**numFieldIds**

dcgmFieldGroupInfo\_v1  
 dcgmProfMetricGroupInfo\_t  
 dcgmProfWatchFields\_v1

**numGpus**

dcgmPidInfo\_v2  
 dcgmJobInfo\_v3  
 dcgmDeviceTopology\_v1  
 dcgmNvLinkStatus\_v1

**numGraphicsPids**

dcgmGpuUsageInfo\_t

**numNvSwitches**

dcgmNvLinkStatus\_v1

**numOtherComputePids**

dcgmPidSingleInfo\_t

**numOtherGraphicsPids**

dcgmPidSingleInfo\_t

**numXidCriticalErrors**

dcgmPidSingleInfo\_t  
 dcgmGpuUsageInfo\_t

**nvlink**

dcgmPolicyCallbackResponse\_v1

**nvmlComputeInstanceId**

dcgmMigEntityInfo\_t

**nvmlFieldId**

dcgm\_field\_meta\_t

**nvmlGpuIndex**

dcgmMigEntityInfo\_t

**nvmlInstanceId**

dcgmMigEntityInfo\_t

**nvmlMigProfileId**

dcgmMigEntityInfo\_t

**nvmlProfileSlices**

dcgmMigEntityInfo\_t

**nvSwitches**

dcgmNvLinkStatus\_v1

**O****opMode**

dcgmStartEmbeddedV2Params\_v1

**otherComputePids**

dcbmPidSingleInfo\_t

**otherGraphicsPids**

dcbmPidSingleInfo\_t

**overallHealth**

dcbmHealthResponse\_v4

dcbmHostengineHealth\_v1

dcbmPidSingleInfo\_t

dcbmGpuUsageInfo\_t

**P****parent**

dcbmMigHierarchyInfo\_t

**parms**

dcbmPolicy\_v1

**path**

dcbmDeviceTopology\_v1

**pci**

dcbmPolicyCallbackResponse\_v1

**pciBusId**

dcbmDeviceIdentifiers\_v1

**pciDeviceId**

dcbmDeviceIdentifiers\_v1

**pcieReplays**

dcbmGpuUsageInfo\_t

dcbmPidSingleInfo\_t

**pcieRxBandwidth**

dcbmPidSingleInfo\_t

dcbmGpuUsageInfo\_t

**pcieTxBandwidth**

dcbmPidSingleInfo\_t

dcbmGpuUsageInfo\_t

**pciSubSystemId**

dcbmDeviceIdentifiers\_v1

**perfState**

dcbmConfig\_v1

**perGpuResponses**

dcbmDiagResponse\_v6

**persistAfterDisconnect**

dcbmConnectV2Params\_v1

dcbmConnectV2Params\_v2

**pid**

dcbmRunningProcess\_v1

dcmDevicePidAccountingStats\_v1  
 dcmDeviceVgpuEncSessions\_v1  
 dcmDeviceVgpuProcessUtilInfo\_v1  
 dcmPidInfo\_v2  
 dcmDeviceFbcSessionInfo\_v1

**power**

dcmPolicyCallbackResponse\_v1

**powerLimit**

dcmConfig\_v1

**powerLimits**

dcmDeviceAttributes\_v1

**powerUsage**

dcmGpuUsageInfo\_t

**powerViolation**

dcmPolicyConditionPower\_t

**powerViolationTime**

dcmPidSingleInfo\_t

dcmGpuUsageInfo\_t

**processName**

dcmDeviceVgpuProcessUtilInfo\_v1

**processUtilization**

dcmPidSingleInfo\_t

**R****rawBuildInfoString**

dcmVersionInfo\_v2

**recentUpdateUsec**

dcmIntrospectFieldsExecTime\_v1

**reliabilityViolationTime**

dcmGpuUsageInfo\_t

dcmPidSingleInfo\_t

**response**

dcmPolicy\_v1

**results**

dcmDiagResponsePerGpu\_v2

**S****sbepages**

dcmPolicyConditionMpr\_t

**scope**

dcm\_field\_meta\_t

**serial**

dcmDeviceIdentifiers\_v1

**sessionCount**

dcgmDeviceEncStats\_v1  
dcgmDeviceFbcStats\_v1  
dcgmDeviceFbcSessions\_v1

**sessionFlags**

dcgmDeviceFbcSessionInfo\_v1

**sessionId**

dcgmDeviceFbcSessionInfo\_v1  
dcgmDeviceVgpuEncSessions\_v1

**sessionInfo**

dcgmDeviceFbcSessions\_v1

**sessionType**

dcgmDeviceFbcSessionInfo\_v1

**severity**

dcgmStartEmbeddedV2Params\_v1

**shortName**

dcgm\_field\_output\_format\_t

**shutdownTemp**

dcgmDeviceThermals\_v1

**size**

dcgm\_field\_meta\_t

**sliceProfile**

dcgmMigHierarchyInfo\_t

**slowdownTemp**

dcgmDeviceThermals\_v1

**slowestPath**

dcgmGroupTopology\_v1

**smClock**

dcgmClockSet\_v1  
dcgmPidSingleInfo\_t  
dcgmGpuUsageInfo\_t

**smUtil**

dcgmDeviceVgpuUtilInfo\_v1  
dcgmDeviceVgpuProcessUtilInfo\_v1

**smUtilization**

dcgmPidSingleInfo\_t  
dcgmGpuUsageInfo\_t

**startTime**

dcgmGpuUsageInfo\_t  
dcgmPidSingleInfo\_t

**startTimestamp**

dcgmDevicePidAccountingStats\_v1

**status**

dcgmModuleGetStatusesModule\_t  
 dcgmFieldValue\_v2  
 dcgmErrorInfo\_t  
 dcgmFieldValue\_v1

**str**

dcgmFieldValue\_v1  
 dcgmFieldValue\_v2

**subsystemId**

dcgmDeviceVgpuTypeInfo\_v1

**summary**

dcgmJobInfo\_v3  
 dcgmPidInfo\_v2

**syncBoost**

dcgmConfigPerfStateSettings\_t

**syncBoostTime**

dcgmGpuUsageInfo\_t  
 dcgmPidSingleInfo\_t

**system**

dcgmGpuUsageInfo\_t  
 dcgmPidSingleInfo\_t

**systemError**

dcgmDiagResponse\_v6

**systems**

dcgmHealthSetParams\_v2

**T****tag**

dcgm\_field\_meta\_t

**targetClocks**

dcgmConfigPerfStateSettings\_t

**thermal**

dcgmPolicyCallbackResponse\_v1

**thermalSettings**

dcgmDeviceAttributes\_v1

**thermalViolation**

dcgmPolicyConditionThermal\_t

**thermalViolationTime**

dcgmPidSingleInfo\_t  
 dcgmGpuUsageInfo\_t

**timeoutMs**

dcgmConnectV2Params\_v2

**timestamp**

dcgmPolicyConditionXID\_t  
 dcgmPolicyConditionDbe\_t  
 dcgmPolicyConditionPci\_t  
 dcgmPolicyConditionMpr\_t  
 dcgmPolicyConditionThermal\_t  
 dcgmPolicyConditionPower\_t  
 dcgmPolicyConditionNvlink\_t

**total**

dcgmIntrospectCpuUtil\_v1

**totalEverUpdateUsec**

dcgmIntrospectFieldsExecTime\_v1

**trainingMsg**

dcgmDiagResponse\_v6

**ts**

dcgmFieldValue\_v1  
 dcgmFieldValue\_v2

**type**

dcgmConfigPowerLimit\_t

**U****unit**

dcgm\_field\_output\_format\_t

**unused**

dcgmDeviceAttributes\_v1  
 dcgmStartEmbeddedV2Params\_v1  
 dcgmFieldValue\_v2

**updateFreq**

dcgmProfWatchFields\_v1

**updateInterval**

dcgmHealthSetParams\_v2

**user**

dcgmIntrospectCpuUtil\_v1

**uuid**

dcgmDeviceIdentifiers\_v1

**V****val**

dcgmConfigPowerLimit\_t

**validation**

dcgmPolicy\_v1

**value**

dcgmFieldValue\_v2

dcgmFieldValue\_v1

**valueFormat**

dcgm\_field\_meta\_t

**vbios**

dcgmDeviceIdentifiers\_v1

**version**

dcgmConnectV2Params\_v2

dcgmDevicePowerLimits\_v1

dcgmFieldValue\_v2

dcgmHealthResponse\_v4

dcgmDeviceIdentifiers\_v1

dcgmHealthSetParams\_v2

dcgmPidInfo\_v2

dcgmHostengineHealth\_v1

dcgmDeviceMemoryUsage\_v1

dcgmProfUnwatchFields\_v1

dcgmJobInfo\_v3

dcgmRunningProcess\_v1

dcgmNvLinkStatus\_v1

dcgmIntrospectCpuUtil\_v1

dcgmIntrospectFullMemory\_v1

dcgmIntrospectMemory\_v1

dcgmIntrospectFullFieldsExecTime\_v2

dcgmIntrospectFieldsExecTime\_v1

dcgmIntrospectContext\_v1

dcgmDeviceVgpuUtilInfo\_v1

dcgmDiagResponse\_v6

dcgmDeviceTopology\_v1

dcgmGroupInfo\_v2

dcgmDeviceEncStats\_v1

dcgmGroupTopology\_v1

dcgmDeviceFbcStats\_v1

dcgmFieldGroupInfo\_v1

dcgmDeviceFbcSessionInfo\_v1

dcgmFieldValue\_v1

dcgmPolicyCallbackResponse\_v1

dcgmPolicy\_v1

dcgmConfig\_v1

dcgmDeviceAttributes\_v1

dcgmDeviceVgpuTypeInfo\_v1

dcgmDeviceVgpuProcessUtilInfo\_v1

dcgmProfWatchFields\_v1

dcgmDeviceFbcSessions\_v1

dcgmClockSet\_v1  
 dcgmDeviceVgpuEncSessions\_v1  
 dcgmStartEmbeddedV2Params\_v1  
 dcgmDeviceSupportedClockSets\_v1  
 dcgmConnectV2Params\_v1  
 dcgmDevicePidAccountingStats\_v1  
 dcgmDeviceThermals\_v1

**vgpuId**

dcgmDeviceFbcSessionInfo\_v1  
 dcgmDeviceVgpuProcessUtilInfo\_v1  
 dcgmDeviceVgpuEncSessions\_v1  
 dcgmDeviceVgpuUtilInfo\_v1

**vgpuProcessSamplesCount**

dcgmDeviceVgpuProcessUtilInfo\_v1

**vgpuTypeClass**

dcgmDeviceVgpuTypeInfo\_v1

**vgpuTypeInfo**

dcgmDeviceVgpuTypeInfo\_v1

**vgpuTypeLicense**

dcgmDeviceVgpuTypeInfo\_v1

**vgpuTypeName**

dcgmDeviceVgpuTypeInfo\_v1

**violationOccurred**

dcgmPolicyViolationNotify\_t

**virtualizationMode**

dcgmDeviceIdentifiers\_v1

**vMaxResolution**

dcgmDeviceFbcSessionInfo\_v1

**vResolution**

dcgmDeviceVgpuEncSessions\_v1  
 dcgmDeviceFbcSessionInfo\_v1

**W****width**

dcgm\_field\_output\_format\_t

**X****xid**

dcgmPolicyCallbackResponse\_v1

**xidCriticalErrorsTs**

dcgmGpuUsageInfo\_t  
 dcgmPidSingleInfo\_t



## Notice

THE INFORMATION IN THIS GUIDE AND ALL OTHER INFORMATION CONTAINED IN NVIDIA DOCUMENTATION REFERENCED IN THIS GUIDE IS PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE INFORMATION FOR THE PRODUCT, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA’s aggregate and cumulative liability towards customer for the product described in this guide shall be limited in accordance with the NVIDIA terms and conditions of sale for the product.

THE NVIDIA PRODUCT DESCRIBED IN THIS GUIDE IS NOT FAULT TOLERANT AND IS NOT DESIGNED, MANUFACTURED OR INTENDED FOR USE IN CONNECTION WITH THE DESIGN, CONSTRUCTION, MAINTENANCE, AND/OR OPERATION OF ANY SYSTEM WHERE THE USE OR A FAILURE OF SUCH SYSTEM COULD RESULT IN A SITUATION THAT THREATENS THE SAFETY OF HUMAN LIFE OR SEVERE PHYSICAL HARM OR PROPERTY DAMAGE (INCLUDING, FOR EXAMPLE, USE IN CONNECTION WITH ANY NUCLEAR, AVIONICS, LIFE SUPPORT OR OTHER LIFE CRITICAL APPLICATION). NVIDIA EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS FOR SUCH HIGH RISK USES. NVIDIA SHALL NOT BE LIABLE TO CUSTOMER OR ANY THIRD PARTY, IN WHOLE OR IN PART, FOR ANY CLAIMS OR DAMAGES ARISING FROM SUCH HIGH RISK USES.

NVIDIA makes no representation or warranty that the product described in this guide will be suitable for any specified use without further testing or modification. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to ensure the product is suitable and fit for the application planned by customer and to do the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this guide. NVIDIA does not accept any liability related to any default, damage, costs or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this guide, or (ii) customer product designs.

Other than the right for customer to use the information in this guide with the product, no other license, either expressed or implied, is hereby granted by NVIDIA under this guide. Reproduction of information in this guide is permissible only if reproduction is approved by NVIDIA in writing, is reproduced without alteration, and is accompanied by all associated conditions, limitations, and notices.

## Trademarks

NVIDIA and the NVIDIA logo are trademarks and/or registered trademarks of NVIDIA Corporation in the United States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

## Copyright

© 2013-2020 NVIDIA Corporation. All rights reserved.