

Process Management Interface for Exascale (PMIx) Standard

Version 4.0 (Draft)

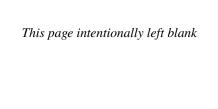
Created on July 13, 2020

This document describes the Process Management Interface for Exascale (PMIx) Standard, version 4.0 (Draft).

Comments: Please provide comments on the PMIx Standard by filing issues on the document repository https://github.com/pmix/pmix-standard/issues or by sending them to the PMIx Community mailing list at https://groups.google.com/forum/#!forum/pmix. Comments should include the version of the PMIx standard you are commenting about, and the page, section, and line numbers that you are referencing. Please note that messages sent to the mailing list from an unsubscribed e-mail address will be ignored.

Copyright © 2018-2019 PMIx Standard Review Board.

Permission to copy without fee all or part of this material is granted, provided the PMIx Standard Review Board copyright notice and the title of this document appear, and notice is given that copying is by permission of PMIx Standard Review Board.



Contents

1.	Intro	duction	1					
	1.1.	Charter	2					
	1.2.		2					
	1.2.	1.2.1. Who should use the standard?	2					
		1.2.2. What is defined in the standard?	3					
		1.2.3. What is <i>not</i> defined in the standard?	3					
		1.2.4. General Guidance for PMIx Users and Implementors	4					
	1.3.	PMIx Architecture Overview	4					
	1.3.	1.3.1. The PMIx Reference Implementation (PRI)	6					
		1.3.2. The PMIx Reference RunTime Environment (PRRTE)	7					
	1.4.	Organization of this document	7					
	1.5.	Version 1.0: June 12, 2015	8					
	1.6.	Version 2.0: Sept. 2018	9					
		Version 2.1: Dec. 2018	9					
	1.7. 1.8.							
			10					
	1.9.		11					
			11					
			12					
	1.12.	Version 4.0: June 2019	12					
2.	PMIx Terms and Conventions 14							
	2.1.	Notational Conventions	16					
	2.2.	Semantics	17					
	2.3.		18					
	2.4.	Procedure Conventions	18					
	2.5.	Standard vs Reference Implementation	19					
3.	Data	Structures and Types 2	20					
	3.1.	Constants	21					
		3.1.1. PMIx Error Constants	22					

	3.1.2.	Macros for use with PMIx constants	27
3.2.	Data Ty	pes	28
	3.2.1.	Key Structure	28
	3.2.2.	Namespace Structure	29
	3.2.3.	Rank Structure	31
	3.2.4.	Process Structure	31
	3.2.5.	Process structure support macros	31
	3.2.6.	Process State Structure	35
	3.2.7.	Job State Structure	36
	3.2.8.	Process Information Structure	36
	3.2.9.	Process Information Structure support macros	37
	3.2.10.	Scope of Put Data	38
	3.2.11.	Job State Structure	38
	3.2.12.	Range of Published Data	39
	3.2.13.	Data Persistence Structure	40
	3.2.14.	Data Array Structure	40
	3.2.15.	Data array structure support macros	40
	3.2.16.	Value Structure	42
	3.2.17.	Value structure support macros	43
	3.2.18.	Info Structure	46
	3.2.19.	Info structure support macros	47
	3.2.20.	Info Type Directives	49
	3.2.21.	Info Directive support macros	50
	3.2.22.	Job Allocation Directives	52
	3.2.23.	IO Forwarding Channels	52
	3.2.24.	Environmental Variable Structure	52
	3.2.25.	Environmental variable support macros	53
	3.2.26.	Lookup Returned Data Structure	54
	3.2.27.	Lookup data structure support macros	54
	3.2.28.	Application Structure	57
	3.2.29.	App structure support macros	58
	3.2.30.	Query Structure	60
	3.2.31.	Query structure support macros	60

	3.2.32.	Attribute registration structure	62
	3.2.33.	Attribute registration structure support macros	63
	3.2.34.	PMIx Group Directives	65
	3.2.35.	Byte Object Type	65
	3.2.36.	Byte object support macros	65
	3.2.37.	Data Array Structure	67
	3.2.38.	Data array support macros	67
	3.2.39.	Argument Array Macros	68
	3.2.40.	Set Environment Variable	72
3.3.	General	ized Data Types Used for Packing/Unpacking	73
3.4.	Reserve	d attributes	75
	3.4.1.	Initialization attributes	75
	3.4.2.	Identification attributes	76
	3.4.3.	Programming model attributes	77
	3.4.4.	UNIX socket rendezvous socket attributes	77
	3.4.5.	TCP connection attributes	78
	3.4.6.	Global Data Storage (GDS) attributes	78
	3.4.7.	General process-level attributes	79
	3.4.8.	Scratch directory attributes	79
	3.4.9.	Relative Rank Descriptive Attributes	79
	3.4.10.	Information retrieval attributes	81
	3.4.11.	Information storage attributes	82
	3.4.12.	Size information attributes	83
	3.4.13.	Memory information attributes	84
	3.4.14.	Topology information attributes	84
	3.4.15.	Request-related attributes	85
	3.4.16.	Server-to-PMIx library attributes	86
	3.4.17.	Server-to-Client attributes	87
	3.4.18.	Event handler registration and notification attributes	87
	3.4.19.	Fault tolerance attributes	88
	3.4.20.	Spawn attributes	89
	3.4.21.	Query attributes	91
	3.4.22.	Log attributes	93

	3.4.23.	Resource manager attributes	94
	3.4.24.	Environment variable attributes	94
	3.4.25.	Job Allocation attributes	95
	3.4.26.	Job control attributes	97
	3.4.27.	Monitoring attributes	98
	3.4.28.	Security attributes	99
	3.4.29.	IO Forwarding attributes	99
	3.4.30.	Application setup attributes	100
	3.4.31.	Attribute support level attributes	100
	3.4.32.	Descriptive attributes	100
	3.4.33.	Process group attributes	101
	3.4.34.	Storage-related attributes	101
3.5.	Callback	k Functions	102
	3.5.1.	Release Callback Function	102
	3.5.2.	Modex Callback Function	103
	3.5.3.	Spawn Callback Function	104
	3.5.4.	Op Callback Function	104
	3.5.5.	Lookup Callback Function	105
	3.5.6.	Value Callback Function	105
	3.5.7.	Info Callback Function	106
	3.5.8.	Event Handler Registration Callback Function	107
	3.5.9.	Notification Handler Completion Callback Function	107
	3.5.10.	Notification Function	108
	3.5.11.	Server Setup Application Callback Function	110
	3.5.12.	Server Direct Modex Response Callback Function	111
	3.5.13.	PMIx Client Connection Callback Function	112
	3.5.14.	PMIx Tool Connection Callback Function	112
	3.5.15.	Credential callback function	113
	3.5.16.	Credential validation callback function	114
	3.5.17.	IOF delivery function	115
	3.5.18.	IOF and Event registration function	116
3 6	Constan	at String Functions	117

4.	Initia	lization	and Finalization	121
	4.1.	Query .		121
		4.1.1.	PMIx_Initialized	121
		4.1.2.	PMIx_Get_version	122
	4.2.	Client In	nitialization and Finalization	122
		4.2.1.	PMIx_Init	122
		4.2.2.	PMIx_Finalize	125
	4.3.	Server I	nitialization and Finalization	125
		4.3.1.	PMIx_server_init	125
		4.3.2.	PMIx_server_finalize	128
5.	Key/	Value M	anagement	129
	5.1.	Setting	and Accessing Key/Value Pairs	129
		5.1.1.	PMIx_Put	129
		5.1.2.	PMIx_Get	130
		5.1.3.	PMIx_Get_nb	133
		5.1.4.	PMIx_Store_internal	136
		5.1.5.	Accessing information: examples	137
	5.2.	Exchang	ging Key/Value Pairs	141
		5.2.1.	PMIx_Commit	141
		5.2.2.	PMIx_Fence	142
		5.2.3.	PMIx_Fence_nb	143
	5.3.	Publish	and Lookup Data	146
		5.3.1.	PMIx_Publish	146
		5.3.2.	PMIx_Publish_nb	148
		5.3.3.	PMIx_Lookup	149
		5.3.4.	PMIx_Lookup_nb	152
		5.3.5.	PMIx_Unpublish	153
		5.3.6.	PMIx_Unpublish_nb	155
6.	Proc	ess Mar	nagement	157
	6.1.	Abort .		157
		611	PMTx Abort	157

	6.2.	Process	Creation	8
		6.2.1.	PMIx_Spawn 15	8
		6.2.2.	PMIx_Spawn_nb	3
	6.3.	Connect	ting and Disconnecting Processes	7
		6.3.1.	PMIx_Connect 16	8
		6.3.2.	PMIx_Connect_nb 17	0
		6.3.3.	PMIx_Disconnect	2
		6.3.4.	PMIx_Disconnect_nb	4
	6.4.	IO Forw	varding	6
		6.4.1.	PMIx_IOF_pull	7
		6.4.2.	PMIx_IOF_deregister	9
		6.4.3.	PMIx_IOF_push	0
7.	Job	Manage	ment and Reporting 18	3
	7.1.	•		3
		7.1.1.	PMIx_Resolve_peers	3
		7.1.2.	PMIx_Resolve_nodes	4
		7.1.3.	PMIx_Query_info	5
		7.1.4.	PMIx_Query_info_nb	0
	7.2.	Allocati	on Requests	5
		7.2.1.	PMIx_Allocation_request	6
		7.2.2.	PMIx_Allocation_request_nb	9
	7.3.	Job Con	trol	2
		7.3.1.	PMIx_Job_control	2
		7.3.2.	PMIx_Job_control_nb	5
	7.4.	Process	and Job Monitoring	8
		7.4.1.	PMIx_Process_monitor	8
		7.4.2.	PMIx_Process_monitor_nb	0
		7.4.3.	PMIx_Heartbeat	2
	7.5.	Logging	g	3
		7.5.1.	PMIx_Log	3
		752	PMTy Tog ph	

8.	Ever	nt Notific	cation	219
	8.1.	Notifica	tion and Management	219
		8.1.1.	PMIx_Register_event_handler	221
		8.1.2.	PMIx_Deregister_event_handler	224
		8.1.3.	PMIx_Notify_event	225
9.	Data	Packing	g and Unpacking	229
	9.1.	Data Bu	ıffer Type	229
	9.2.	Support	Macros	230
		9.2.1.	PMIX_DATA_BUFFER_CREATE	230
		9.2.2.	PMIX_DATA_BUFFER_RELEASE	230
		9.2.3.	PMIX_DATA_BUFFER_CONSTRUCT	230
		9.2.4.	PMIX_DATA_BUFFER_DESTRUCT	231
		9.2.5.	PMIX_DATA_BUFFER_LOAD	231
		9.2.6.	PMIX_DATA_BUFFER_UNLOAD	232
	9.3.	General	Routines	232
		9.3.1.	PMIx_Data_pack	232
		9.3.2.	PMIx_Data_unpack	234
		9.3.3.	PMIx_Data_copy	236
		9.3.4.	PMIx_Data_print	236
		9.3.5.	PMIx_Data_copy_payload	237
10	.Secı	ırity		239
	10.1.	Obtainii	ng Credentials	240
		10.1.1.	PMIx_Get_credential	240
		10.1.2.	PMIx_Get_credential_nb	241
	10.2.	Validati	ng Credentials	243
		10.2.1.	PMIx_Validate_credential	243
		10.2.2.	PMIx_Validate_credential_nb	245
11	.Serv	er-Spec	ific Interfaces	248
	11.1.	Server S	Support Functions	248
			PMIx_generate_regex	
			PMIx_generate_ppn	249
		11.1.3.		250

	11.1.4.	PMIx_server_deregister_nspace	266
	11.1.5.	PMIx_server_register_client	267
	11.1.6.	PMIx_server_deregister_client	268
	11.1.7.	PMIx_server_setup_fork	269
	11.1.8.	PMIx_server_dmodex_request	269
	11.1.9.	PMIx_server_setup_application	271
	11.1.10.	PMIx_Register_attributes	273
	11.1.11.	PMIx_server_setup_local_support	275
	11.1.12.	PMIx_server_IOF_deliver	276
	11.1.13.	PMIx_server_collect_inventory	277
	11.1.14.	PMIx_server_deliver_inventory	278
11.2.	Server F	unction Pointers	280
	11.2.1.	<pre>pmix_server_module_t Module</pre>	280
	11.2.2.	<pre>pmix_server_client_connected_fn_t</pre>	281
	11.2.3.	<pre>pmix_server_client_finalized_fn_t</pre>	283
	11.2.4.	<pre>pmix_server_abort_fn_t</pre>	284
	11.2.5.	<pre>pmix_server_fencenb_fn_t</pre>	285
	11.2.6.	<pre>pmix_server_dmodex_req_fn_t</pre>	289
	11.2.7.	<pre>pmix_server_publish_fn_t</pre>	290
	11.2.8.	pmix_server_lookup_fn_t	292
	11.2.9.	<pre>pmix_server_unpublish_fn_t</pre>	294
	11.2.10.	pmix_server_spawn_fn_t	296
	11.2.11.	pmix_server_connect_fn_t	301
	11.2.12.	pmix_server_disconnect_fn_t	303
	11.2.13.	<pre>pmix_server_register_events_fn_t</pre>	305
	11.2.14.	<pre>pmix_server_deregister_events_fn_t</pre>	307
	11.2.15.	pmix_server_notify_event_fn_t	309
	11.2.16.	pmix_server_listener_fn_t	310
	11.2.17.	pmix_server_query_fn_t	311
	11.2.18.	pmix_server_tool_connection_fn_t	314
	11.2.19.	pmix_server_log_fn_t	315
	11.2.20.	pmix_server_alloc_fn_t	317
	11.2.21	pmix server job control fn t	320

	11.2.22.	<pre>pmix_server_monitor_fn_t</pre>	323
	11.2.23.	<pre>pmix_server_get_cred_fn_t</pre>	326
	11.2.24.	<pre>pmix_server_validate_cred_fn_t</pre>	327
	11.2.25.	<pre>pmix_server_iof_fn_t</pre>	329
	11.2.26.	pmix_server_stdin_fn_t	332
	11.2.27.	pmix_server_grp_fn_t	333
	11.2.28.	<pre>pmix_server_fabric_fn_t</pre>	335
12. Fabri	ic Supp	ort Definitions	338
12.1.	Fabric S	Support Constants	338
12.2.	Fabric S	Support Datatypes	338
		Fabric Coordinate Structure	339
	12.2.2.	Fabric Coordinate Support Macros	340
	12.2.3.	Fabric Coordinate Views	341
	12.2.4.	Fabric Link State	341
	12.2.5.	Fabric Operation Constants	342
	12.2.6.	Fabric registration structure	342
12.3.	Fabric S	Support Attributes	346
12.4.	Fabric S	Support Functions	348
	12.4.1.	PMIx_Fabric_register	349
	12.4.2.	PMIx_Fabric_register_nb	350
	12.4.3.	PMIx_Fabric_update	351
	12.4.4.	PMIx_Fabric_update_nb	351
	12.4.5.	PMIx_Fabric_deregister	352
	12.4.6.	PMIx_Fabric_deregister_nb	353
	12.4.7.	PMIx_Fabric_get_vertex_info	353
	12.4.8.	PMIx_Fabric_get_vertex_info_nb	356
	12.4.9.	PMIx_Fabric_get_device_index	356
	12.4.10.	PMIx_Fabric_get_device_index_nb	357
13. Proc	ess Sets	s and Groups	359
13.1.	Process	Sets	359
13.2.	Process	Groups	360
	13.2.1.	Group Operation Constants	362

	13.2.2.	PMIx_Group_construct	363
	13.2.3.	PMIx_Group_construct_nb	366
	13.2.4.	PMIx_Group_destruct	369
	13.2.5.	PMIx_Group_destruct_nb	371
	13.2.6.	PMIx_Group_invite	373
	13.2.7.	PMIx_Group_invite_nb	377
	13.2.8.	PMIx_Group_join	379
	13.2.9.	PMIx_Group_join_nb	382
	13.2.10.	PMIx_Group_leave	384
	13.2.11.	PMIx_Group_leave_nb	385
14. Tools	and De	ebuggers	387
		ion Mechanisms	387
	14.1.1.	Rendezvousing with a local server	390
	14.1.2.	Connecting to a remote server	391
	14.1.3.	Attaching to running jobs	391
	14.1.4.	Tool initialization attributes	392
	14.1.5.	Tool connection attributes	392
14.2.	Launchi	ng Applications with Tools	393
	14.2.1.	Direct launch	393
	14.2.2.	Indirect launch	394
	14.2.3.	Tool spawn-related attributes	396
	14.2.4.	Tool spawn-related constants	396
14.3.	Debugge	er Support	397
	14.3.1.	Co-Spawn of Debugger Daemons	397
	14.3.2.	Debugger attributes	397
14.4.	IO Forw	rarding	398
	14.4.1.	Forwarding stdout/stderr	399
	14.4.2.	Forwarding stdin	400
	14.4.3.	IO Forwarding attributes	401
14.5.	Tool-Spe	ecific APIs	402
	14.5.1.	PMIx_tool_init	402
	14.5.2.	PMIx_tool_finalize	405
	14.5.3.	PMIx_tool_disconnect	405

		14.5.4.	PMIx_tool_connect_to_server	406
		14.5.5.	PMIx_IOF_pull	407
		14.5.6.	PMIx_IOF_deregister	409
		14.5.7.	PMIx_IOF_push	410
Α.	Pyth	on Bind	lings	413
	A.1.		Considerations	413
		A.1.1.	Error Codes vs Python Exceptions	413
		A.1.2.	Representation of Structured Data	413
	A.2.	Datatype	e Definitions	414
		A.2.1.	Example	419
	A.3.	Callback	k Function Definitions	419
		A.3.1.	IOF Delivery Function	419
		A.3.2.	Event Handler	420
		A.3.3.	Server Module Functions	421
	A.4.	PMIxCl	ient	434
		A.4.1.	Client.init	434
		A.4.2.	Client.initialized	434
		A.4.3.	Client.get_version	435
		A.4.4.	Client.finalize	435
		A.4.5.	Client.abort	435
		A.4.6.	Client.store_internal	436
		A.4.7.	Client.put	436
		A.4.8.	Client.commit	437
		A.4.9.	Client.fence	437
		A.4.10.	Client.get	438
		A.4.11.	Client.publish	438
		A.4.12.	Client.lookup	439
		A.4.13.	Client.unpublish	439
		A.4.14.	Client.spawn	440
		A.4.15.	Client.connect	440
		A.4.16.	Client.disconnect	441
		A.4.17.	Client.resolve_peers	441
		A.4.18.	Client resolve nodes	442

A.4.19.	Client.query	442
A.4.20.	Client.log	443
A.4.21.	Client.allocate	443
A.4.22.	Client.job_ctrl	444
A.4.23.	Client.monitor	444
A.4.24.	Client.get_credential	445
A.4.25.	Client.validate_credential	445
A.4.26.	Client.group_construct	446
A.4.27.	Client.group_invite	446
A.4.28.	Client.group_join	447
A.4.29.	Client.group_leave	448
A.4.30.	Client.group_destruct	448
A.4.31.	Client.register_event_handler	448
A.4.32.	Client.deregister_event_handler	449
A.4.33.	Client.notify_event	449
A.4.34.	Client.fabric_register	450
A.4.35.	Client.fabric_update	450
A.4.36.	Client.fabric_deregister	451
A.4.37.	Client.fabric_get_vertex_info	451
A.4.38.	Client.fabric_get_device_index	452
A.4.39.	Client.error_string	452
A.4.40.	Client.proc_state_string	452
A.4.41.	Client.scope_string	453
A.4.42.	Client.persistence_string	453
A.4.43.	Client.data_range_string	454
A.4.44.	Client.info_directives_string	454
A.4.45.	Client.data_type_string	454
A.4.46.	Client.alloc_directive_string	455
A.4.47.	Client.iof_channel_string	455
A.4.48.	Client.job_state_string	456
A.4.49.	Client.get_attribute_string	456
A.4.50.	Client.get_attribute_name	456
A 4 51	Client link state string	457

	A.5.	PMIxSe	rver	457		
		A.5.1.	Server.init	457		
		A.5.2.	Server.finalize	458		
		A.5.3.	Server.generate_regex	458		
		A.5.4.	Server.generate_ppn	459		
		A.5.5.	Server.register_nspace	459		
		A.5.6.	Server.deregister_nspace	460		
		A.5.7.	Server.register_client	460		
		A.5.8.	Server.deregister_client	461		
		A.5.9.	Server.setup_fork	461		
		A.5.10.	Server.dmodex_request	462		
		A.5.11.	Server.setup_application	462		
		A.5.12.	Server.register_attributes	463		
		A.5.13.	Server.setup_local_support	463		
		A.5.14.	Server.iof_deliver	464		
		A.5.15.	Server.collect_inventory	464		
		A.5.16.	Server.deliver_inventory	465		
	A.6.	PMIxTo	ol	465		
		A.6.1.	Tool.init	465		
		A.6.2.	Tool.finalize	466		
		A.6.3.	Tool.connect_to_server	466		
		A.6.4.	Tool.iof_pull	467		
		A.6.5.	Tool.iof_deregister	467		
		A.6.6.	Tool.iof_push	468		
	A.7.	Example	e Usage	468		
		A.7.1.	Python Client	469		
		A.7.2.	Python Server	471		
В.	Ackn	owledg	ements	475		
	B.1.	_	3.0	475		
	B.2.		2.0	476		
	B.3.		1.0	477		
Bibliography 478						

Index	479
Index of APIs	480
Index of Support Macros	486
Index of Data Structures	489
Index of Constants	491
Index of Attributes	498

CHAPTER 1

Introduction

The Process Management Interface (PMI) has been used for quite some time as a means of exchanging wireup information needed for inter-process communication. Two versions (PMI-1 and PMI-2) have been released as part of the MPICH effort, with PMI-2 demonstrating better scaling properties than its PMI-1 predecessor. However, two significant challenges face the High Performance Computing (HPC) community as it continues to move towards machines capable of exaflop and higher performance levels:

- the physical scale of the machines, and the corresponding number of total processes they support, is expected to reach levels approaching 1 million processes executing across 100 thousand nodes. Prior methods for initiating applications relied on exchanging communication endpoint information between the processes, either directly or in some form of hierarchical collective operation. Regardless of the specific mechanism employed, the exchange across such large applications would consume considerable time, with estimates running in excess of 5-10 minutes; and
- whether it be hybrid applications that combine OpenMP threading operations with MPI, or application-steered workflow computations, the HPC community is experiencing an unprecedented wave of new approaches for computing at exascale levels. One common thread across the proposed methods is an increasing need for orchestration between the application and the system management software stack (SMS) comprising the scheduler (a.k.a. the workload manager (WLM)), the resource manager (RM), global file system, fabric, and other subsystems. The lack of available support for application-to-SMS integration has forced researchers to develop "virtual" environments that hide the SMS behind a customized abstraction layer, but this results in considerable duplication of effort and a lack of portability.

Process Management Interface - Exascale (PMIx) represents an attempt to resolve these questions by providing an extended version of the PMI definitions specifically designed to support clusters up to exascale and larger sizes. The overall objective of the project is not to branch the existing definitions – in fact, PMIx fully supports both of the existing PMI-1 and PMI-2 Application Programming Interfaces (APIs) – but rather to:

- a) add flexibility to the existing APIs by adding an array of key-value "attribute" pairs to each API signature that allows implementers to customize the behavior of the API as future needs emerge without having to alter or create new variants of it;
- b) add new APIs that provide extended capabilities such as asynchronous event notification plus dynamic resource allocation and management;

- c) establish a collaboration between SMS subsystem providers including resource manager, fabric, 1 2 file system, and programming library developers to define integration points between the various subsystems as well as agreed upon definitions for associated APIs, attribute names, and 3 4 data types;
 - d) form a standards-like body for the definitions; and
 - e) provide a reference implementation of the PMIx standard.

Complete information about the PMIx standard and affiliated projects can be found at the PMIx web site: https://pmix.org

1.1 Charter

5

6

7

8

11

12

13 14

15

16

17 18

19

21

29

The charter of the PMIx community is to: 10

- Define a set of agnostic APIs (not affiliated with any specific programming model or code base) to support interactions between application processes and the SMS.
- Develop an open source (non-copy-left licensed) standalone "reference" library implementation to facilitate adoption of the PMIx standard.
- Retain transparent backward compatibility with the existing PMI-1 and PMI-2 definitions, any future PMI releases, and across all PMIx versions.
- Support the "Instant On" initiative for rapid startup of applications at exascale and beyond.
- Work with the HPC community to define and implement new APIs that support evolving programming model requirements for application interactions with the SMS.

20 Participation in the PMIx community is open to anyone, and not restricted to only code contributors to the reference implementation.

1.2 PMIx Standard Overview

23 The PMIx Standard defines and describes the interface developed by the PMIx Reference 24 Implementation (PRI). Much of this document is specific to the PMIx Reference 25 Implementation (PRI)'s design and implementation. Specifically the standard describes the functionality provided by the PRI, and what the PRI requires of the clients and resource 26 27 managers (RMs) that use it's interface.

1.2.1 Who should use the standard?

- The PMIx Standard informs PMIx clients and RMs of the syntax and semantics of the PMIx APIs.
- 30 PMIx clients (e.g., tools, Message Passing Environment (MPE) libraries) can use this standard to 31 understand the set of attributes provided by various APIs of the PRI and their intended behavior.

1 Additional information about the rationale for the selection of specific interfaces and attributes is also provided.

PMIx-enabled RMs can use this standard to understand the expected behavior required of them when they support various interfaces/attributes. In addition, optional features and suggestions on behavior are also included in the discussion to help guide RM design and implementation.

1.2.2 What is defined in the standard?

The PMIx Standard defines and describes the interface developed by the PMIx Reference Implementation (PRI). It defines the set of attributes that the PRI supports; the set of attributes that are required of a RM to support, for a given interface; and the set of optional attributes that an RM may choose to support, for a given interface.

1.2.3 What is *not* defined in the standard?

No standards body can require an implementer to support something in their standard, and PMIx is no different in that regard. While an implementer of the PMIx library itself must at least include the standard PMIx headers and instantiate each function, they are free to return "not supported" for any function they choose not to implement.

This also applies to the host environments. Resource managers and other system management stack components retain the right to decide on support of a particular function. The PMIx community continues to look at ways to assist SMS implementers in their decisions by highlighting functions that are critical to basic application execution (e.g., PMIx_Get), while leaving flexibility for tailoring a vendor's software for their target market segment.

One area where this can become more complicated is regarding the attributes that provide information to the client process and/or control the behavior of a PMIx standard API. For example, the **PMIX_TIMEOUT** attribute can be used to specify the time (in seconds) before the requested operation should time out. The intent of this attribute is to allow the client to avoid "hanging" in a request that takes longer than the client wishes to wait, or may never return (e.g., a **PMIx_Fence** that a blocked participant never enters).

If an application (for example) truly relies on the <code>PMIX_TIMEOUT</code> attribute in a call to <code>PMIx_Fence</code>, it should set the required flag in the <code>pmix_info_t</code> for that attribute. This informs the library and its SMS host that it must return an immediate error if this attribute is not supported. By not setting the flag, the library and SMS host are allowed to treat the attribute as optional, ignoring it if support is not available.

It is therefore critical that users and application implementers:

- a) consider whether or not a given attribute is required, marking it accordingly; and
- b) check the return status on all PMIx function calls to ensure support was present and that the request was accepted. Note that for non-blocking APIs, a return of **PMIX_SUCCESS** only indicates that the request had no obvious errors and is being processed the eventual callback will return the status of the requested operation itself.

While a PMIx library implementer, or an SMS component server, may choose to support a particular PMIx API, they are not required to support every attribute that might apply to it. This would pose a significant barrier to entry for an implementer as there can be a broad range of applicable attributes to a given API, at least some of which may rarely be used. The PMIx community is attempting to help differentiate the attributes by indicating those that are generally used (and therefore, of higher importance to support) vs those that a "complete implementation" would support.

Note that an environment that does not include support for a particular attribute/API pair is not "incomplete" or of lower quality than one that does include that support. Vendors must decide where to invest their time based on the needs of their target markets, and it is perfectly reasonable for them to perform cost/benefit decisions when considering what functions and attributes to support.

The flip side of that statement is also true: Users who find that their current vendor does not support a function or attribute they require may raise that concern with their vendor and request that the implementation be expanded. Alternatively, users may wish to utilize the PMIx-based Reference RunTime Environment (PRRTE) as a "shim" between their application and the host environment as it might provide the desired support until the vendor can respond. Finally, in the extreme, one can exploit the portability of PMIx-based applications to change vendors.

1.2.4 General Guidance for PMIx Users and Implementors

The PMIx Standard defines the behavior of the PMIx Reference Implementation (PRI). A complete system harnessing the PMIx interface requires an agreement between the PMIx client, be it a tool or library, and the PMIx-enabled RM. The PRI acts as an intermediary between these two entities by providing a standard API for the exchange of requests and responses. The degree to which the PMIx client and the PMIx-enabled RM may interact needs to be defined by those developer communities. The PMIx standard can be used to define the specifics of this interaction.

PMIx clients (e.g., tools, MPE libraries) may find that they depend only on a small subset of interfaces and attributes to work correctly. PMIx clients are strongly advised to define a document itemizing the PMIx interfaces and associated attributes that are required for correct operation, and are optional but recommended for full functionality. The PMIx standard cannot define this list for all given PMIx clients, but such a list is valuable to RMs desiring to support these clients.

PMIx-enabled RMs may choose to implement a subset of the PMIx standard and/or define attributes beyond those defined herein. PMIx-enabled RMs are strongly advised to define a document itemizing the PMIx interfaces and associated attributes they support, with any annotations about behavior limitations. The PMIx standard cannot define this list for all given PMIx-enabled RMs, but such a list is valuable to PMIx clients desiring to support a broad range of PMIx-enabled RMs.

1.3 PMIx Architecture Overview

This section presents a brief overview of the PMIx Architecture [1]. Note that this is a conceptual model solely used to help guide the standards process — it does not represent a design requirement

on any PMIx implementation. Instead, the model is used by the PMIx community as a sounding board for evaluating proposed interfaces and avoid unintentionally imposing constraints on implementers. Built into the model are two guiding principles also reflected in the standard. First, PMIx operates in the mode of a *messenger*, and not a *doer* — i.e., the role of PMIx is to provide communication between the various participants, relaying requests and returning responses. The intent of the standard is not to suggest that PMIx itself actually perform any of the defined operations — this is left to the various SMS elements and/or the application. Any exceptions to that intent are left to the discretion of the particular implementation.

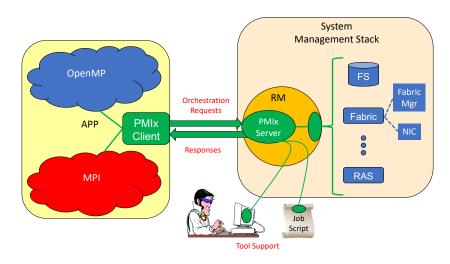


Figure 1.1.: PMIx-SMS Interactions

Thus, as the diagram in Fig. 1.1 shows, the application is built against a PMIx client library that contains the client-side APIs, attribute definitions, and communication support for interacting with the local PMIx server. Intra-process cross-library interactions are supported at the client level to avoid unnecessary burdens on the server. Orchestration requests are sent to the local PMIx server, which subsequently passes them to the host SMS (here represented by an RM daemon) using the PMIx server callback functions the host SMS registered during PMIx_server_init. The host SMS can indicate its lack of support for any operation by simply providing a *NULL* for the associated callback function, or can create a function entry that returns *not supported* when called.

The conceptual model places the burden of fulfilling the request on the host SMS. This includes performing any inter-node communications, or interacting with other SMS elements. Thus, a client request for a network traffic report does not go directly from the client to the Fabric Manager (FM), but instead is relayed to the PMIx server, and then passed to the host SMS for execution. This architecture reflects the second principle underlying the standard — namely, that connectivity is to be minimized by channeling all application interactions with the SMS through the local PMIx server.

Recognizing the burden this places on SMS vendors, the PMIx community has included interfaces

by which the host can request support from local SMS elements. Once the SMS has transferred the request to an appropriate location, a PMIx server interface can be used to pass the request between SMS subsystems. For example, a request for network traffic statistics can utilize the PMIx networking abstractions to retrieve the information from the FM. This reduces the portability and interoperability issues between the individual subsystems by transferring the burden of defining the interoperable interfaces from the SMS subsystems to the PMIx community, which continues to work with those providers to develop the necessary support.

Tools, whether standalone or embedded in job scripts, are an exception to the communication rule and can connect to any PMIx server providing they are given adequate rendezvous information. The PMIx conceptual model views the collection of PMIx servers as a cloud-like conglomerate — i.e., orchestration and information requests can be given to any server regardless of location. However, tools frequently execute on locations that may not house an operating PMIx server — e.g., a users notebook computer. Thus, tools need the ability to remotely connect to the PMIx server "cloud".

The scope of the PMIx standard therefore spans the range of these interactions, between client-and-SMS and between SMS subsystems. Note again that this does not impose a requirement on any given PMIx implementation to cover the entire range — implementers are free to return *not supported* from any PMIx function.

1.3.1 The PMIx Reference Implementation (PRI)

The PMIx community has committed to providing a complete, reference implementation of each version of the standard. Note that the definition of the PMIx Standard is not contingent upon use of the PMIx Reference Implementation (PRI) — any implementation that supports the defined APIs is a PMIx Standard compliant implementation. The PRI is provided solely for the following purposes:

- Validation of the standard.
 - No proposed change and/or extension to the PMIx standard is accepted without an accompanying prototype implementation in the PRI. This ensures that the proposal has undergone at least some minimal level of scrutiny and testing before being considered.
- Ease of adoption.
 - The PRI is designed to be particularly easy for resource managers (and the SMS in general) to adopt, thus facilitating a rapid uptake into that community for application portability. Both client and server PMIx libraries are included, along with examples of client usage and server-side integration. A list of supported environments and versions is maintained on the PMIx web site https://pmix.org/support/faq/what-apis-are-supported-on-my-rm/

The PRI does provide some internal implementations that lie outside the scope of the PMIx standard. This includes several convenience macros as well as support for consolidating collectives for optimization purposes (e.g., the PMIx server aggregates all local PMIx_Fence calls before passing them to the SMS for global execution). In a few additional cases, the PMIx community (in partnership with the SMS subsystem providers) have determined that a base level of support for a given operation can best be portably provided by including it in the PRI.

Instructions for downloading, and installing the PRI are available on the community's web site

https://pmix.org/code/getting-the-reference-implementation/. The PRI targets support for the Linux operating system. A reasonable effort is made to support all major, modern Linux distributions; however, validation is limited to the most recent 2-3 releases of RedHat Enterprise Linux (RHEL), Fedora, CentOS, and SUSE Linux Enterprise Server (SLES). In addition, development support is maintained for Mac OSX. Production support for vendor-specific operating systems is included as provided by the vendor.

1.3.2 The PMIx Reference RunTime Environment (PRRTE)

The PMIx community has also released PRRTE — i.e., a runtime environment containing the reference implementation and capable of operating within a host SMS. PRRTE provides an easy way of exploring PMIx capabilities and testing PMIx-based applications outside of a PMIx-enabled environment by providing a "shim" between the application and the host environment that includes full support for the PRI. The intent of PRRTE is not to replace any existing production environment, but rather to enable developers to work on systems that do not yet feature a PMIx-enabled host SMS or one that lacks a PMIx feature of interest. Instructions for downloading, installing, and using PRRTE are available on the community's web site https://pmix.org/code/getting-the-pmix-reference-server/

1.4 Organization of this document

9

10

11

12

13

14

15

16

17

20 21

22

23

24

25

26

27

28

29

30 31

32

19 The remainder of this document is structured as follows:

- Introduction and Overview in Chapter 1 on page 1
- Terms and Conventions in Chapter 2 on page 14
- Data Structures and Types in Chapter 3 on page 20
- PMIx Initialization and Finalization in Chapter 4 on page 121
- Key/Value Management in Chapter 5 on page 129
- Process Management in Chapter 6 on page 157
- Job Management in Chapter 7 on page 183
- Event Notification in Chapter 8 on page 219
 - Data Packing and Unpacking in Chapter 9 on page 229
- Security in Chapter 10 on page 239
 - PMIx Server Specific Interfaces in Chapter 11 on page 248
 - Scheduler-Specific Interface in Chapter ?? on page ??
 - Process Sets and Groups in Chapter 13 on page 359

- Network Coordinates in Chapter ?? on page ??
 - Python Bindings in Appendix A on page 413

2

1.5 Version 1.0: June 12, 2015

The PMIx version 1.0 ad hoc standard was defined in the PMIx Reference Implementation (PRI) 5 header files as part of the PRI v1.0.0 release prior to the creation of the formal PMIx 2.0 standard. 6 Below are a summary listing of the interfaces defined in the 1.0 headers. Client APIs 7 8 PMIx Init, PMIx Initialized, PMIx Abort, PMIx Finalize - PMIx Put, PMIx Commit, 9 10 - PMIx Fence, PMIx Fence nb 11 - PMIx_Get, PMIx_Get_nb 12 - PMIx Publish PMIx Publish nb 13 - PMIx_Lookup, PMIx_Lookup - PMIx_Unpublish, PMIx_Unpublish_nb 14 15 - PMIx_Spawn, PMIx_Spawn_nb - PMIx Connect, PMIx Connect nb 16 17 - PMIx Disconnect, PMIx Disconnect nb 18 - PMIx Resolve nodes, PMIx Resolve peers Server APIs 19 20 - PMIx server init, PMIx server finalize 21 - PMIx generate regex, PMIx generate ppn 22 - PMIx_server_register_nspace.PMIx_server_deregister_nspace

- Common APIs
 - PMIx_Get_version, PMIx_Store_internal, PMIx_Error_string

- PMIx_server_register_client, PMIx_server_deregister_client

- PMIx Register errhandler, PMIx Deregister errhandler, PMIx Notify error

- PMIx _server_setup_fork , PMIx_server_dmodex_request

The **PMIx** Init API was subsequently modified in the PRI release v1.1.0.

23

24

26

1 1.6 Version 2.0: Sept. 2018

The following APIs were introduced in v2.0 of the PMIx Standard: 2 3 Client APIs 4 - PMIx_Query_info_nb, PMIx_Log_nb 5 - PMIx_Allocation_request_nb, PMIx_Job_control_nb, 6 PMIx Process monitor nb. PMIx Heartbeat Server APIs PMIx_server_setup_application, PMIx_server_setup_local_support 9 Tool APIs - PMIx_tool_init, PMIx_tool_finalize 10 11 Common APIs 12 - PMIx Register event handler, PMIx Deregister event handler 13 - PMIx Notify event - PMIx_Proc_state_string, PMIx_Scope_string 14 - PMIx_Persistence_string, PMIx_Data_range_string 15 16 - PMIx_Info_directives_string, PMIx_Data_type_string 17 - PMIx_Alloc_directive_string 18 - PMIx_Data_pack, PMIx_Data_unpack, PMIx_Data_copy 19 - PMIx Data print, PMIx Data copy payload 20 The **PMIx** Init API was modified in v2.0 of the standard from its ad hoc v1.0 signature to

24 1.7 Version 2.1; Dec. 2018

APIs were replaced.

21

22

23

25 26

27 28

29

30

The v2.1 update includes clarifications and corrections from the v2.0 document, plus addition of examples:

include passing of a pmix info t array for flexibility and "future-proofing" of the API. In

addition, the PMIx Notify error, PMIx Register errhandler, and PMIx Deregister errhandler

- Clarify description of **PMIx_Connect** and **PMIx_Disconnect** APIs.
- Explain that values for the **PMIX_COLLECTIVE_ALGO** are environment-dependent
- Identify the namespace/rank values required for retrieving attribute-associated information using the PMIx_Get API

- Provide definitions for session, job, application, and other terms used throughout the
 document
 - Clarify definitions of PMIX_UNIV_SIZE versus PMIX_JOB_SIZE
 - Clarify server module function return values
 - Provide examples of the use of **PMIx Get** for retrieval of information
 - Clarify the use of PMIx Get versus PMIx Query info nb
 - Clarify return values for non-blocking APIs and emphasize that callback functions must not be invoked prior to return from the API
 - Provide detailed example for construction of the PMIx_server_register_nspace input information array
 - Define information levels (e.g., **session** vs **job**) and associated attributes for both storing and retrieving values
 - Clarify roles of PMIx server library and host environment for collective operations
 - Clarify definition of PMIX UNIV SIZE

15 1.8 Version 2.2: Jan 2019

- The v2.2 update includes the following clarifications and corrections from the v2.1 document:
- Direct modex upcall function (pmix_server_dmodex_req_fn_t) cannot complete
 atomically as the API cannot return the requested information except via the provided callback
 function
 - Add missing pmix data array t definition and support macros
 - Add a rule divider between implementer and host environment required attributes for clarity
 - Add PMIX_QUERY_QUALIFIERS_CREATE macro to simplify creation of pmix_query_t qualifiers
 - Add PMIX_APP_INFO_CREATE macro to simplify creation of pmix_app_t directives
 - Add flag and PMIX_INFO_IS_END macro for marking and detecting the end of a pmix_info_t array
- Clarify the allowed hierarchical nesting of the PMIX_SESSION_INFO_ARRAY ,
 PMIX JOB INFO ARRAY , and associated attributes

4 5

6 7

8

9 10

11

12 13

14

16 17

18

19 20

21

22

23 24

25

1.9 Version 3.0: Dec. 2018

- The following APIs were introduced in v3.0 of the PMIx Standard:
 Client APIs
- 4 PMIx_Log, PMIx_Job_control
- 5 PMIx_Allocation_request, PMIx_Process_monitor
- 6 PMIx_Get_credential, PMIx_Validate_credential
- For the second of the
- 8 PMIx server IOF deliver
- 9 PMIx_server_collect_inventory, PMIx_server_deliver_inventory
- Tool APIs
- 11 PMIx_IOF_pull, PMIx_IOF_push, PMIx_IOF_deregister
- 12 PMIx_tool_connect_to_server
- Common APIs

14

20

21

22

23

24

25 26

30

- PMIx_IOF_channel_string

The document added a chapter on security credentials, a new section for Input/Output (IO)
forwarding to the Process Management chapter, and a few blocking forms of previously-existing
non-blocking APIs. Attributes supporting the new APIs were introduced, as well as additional
attributes for a few existing functions.

19 1.10 Version 3.1: Jan. 2019

- The v3.1 update includes clarifications and corrections from the v3.0 document:
- Direct modex upcall function (pmix_server_dmodex_req_fn_t) cannot complete
 atomically as the API cannot return the requested information except via the provided callback
 function
 - Fix typo in name of **PMIX FWD STDDIAG** attribute
 - Correctly identify the information retrieval and storage attributes as "new" to v3 of the standard
 - Add missing pmix data array t definition and support macros
- Add a rule divider between implementer and host environment required attributes for clarity
- Add PMIX_QUERY_QUALIFIERS_CREATE macro to simplify creation of pmix_query_t qualifiers
 - Add PMIX_APP_INFO_CREATE macro to simplify creation of pmix_app_t directives

- Add new attributes to specify the level of information being requested where ambiguity may exist (see 3.4.10)
 - Add new attributes to assemble information by its level for storage where ambiguity may exist (see 3.4.11)
 - Add flag and PMIX_INFO_IS_END macro for marking and detecting the end of a pmix_info_t array
 - Clarify that PMIX_NUM_SLOTS is duplicative of (a) PMIX_UNIV_SIZE when used at the session level and (b) PMIX_MAX_PROCS when used at the job and application levels, but leave it in for backward compatibility.
 - Clarify difference between PMIX_JOB_SIZE and PMIX_MAX_PROCS
 - Clarify that PMIx_server_setup_application must be called per-job instead of per-application as the name implies. Unfortunately, this is a historical artifact. Note that both PMIX_NODE_MAP and PMIX_PROC_MAP must be included as input in the *info* array provided to that function. Further descriptive explanation of the "instant on" procedure will be provided in the next version of the PMIx Standard.
 - Clarify how the PMIx server expects data passed to the host by
 pmix_server_fencenb_fn_t should be aggregated across nodes, and provide a code
 snippet example

19 1.11 Version 3.2: Oct. 2019

- The v3.2 update includes clarifications and corrections from the v3.1 document:
- Correct an error in the PMIx_Allocation_request function signature, and clarify the allocation ID attributes
- Rename the PMIX_ALLOC_ID attribute to PMIX_ALLOC_REQ_ID to clarify that this is a string the user provides as a means to identify their request to query status
- Add a new PMIX_ALLOC_ID attribute that contains the identifier (provided by the host environment) for the resulting allocation which can later be used to reference the allocated resources in, for example, a call to PMIx_Spawn

8 1.12 Version 4.0: June 2019

- The following changes were introduced in v4.0 of the PMIx Standard:
- Clarified that the PMIx_Fence_nb operation can immediately return

 PMIX_OPERATION_SUCCEEDED in lieu of passing the request to a PMIx server if only the calling process is involved in the operation

3

4

5

6

7

8

9

10 11

12

13 14

15

16

17

18

20

21

22

23

24 25

26

27

- Added the **PMIx_Register_attributes** API by which a host environment can register the attributes it supports for each server-to-host operation
- Added the ability to query supported attributes from the PMIx tool, client and server libraries, as
 well as the host environment via the new pmix_regattr_t structure. Both human-readable
 and machine-parsable output is supported. New attributes to support this operation include:
 - PMIX_CLIENT_ATTRIBUTES, PMIX_SERVER_ATTRIBUTES,
 PMIX_TOOL_ATTRIBUTES, and PMIX_HOST_ATTRIBUTES to identify which library supports the attribute; and
 - PMIX_MAX_VALUE , PMIX_MIN_VALUE , and PMIX_ENUM_VALUE to provide machine-parsable description of accepted values
- Add PMIX APP WILDCARD to reference all applications within a given job

- Fix signature of blocking APIs PMIx_Allocation_request, PMIx_Job_control,
 PMIx_Process_monitor, PMIx_Get_credential, and
 PMIx Validate credential to allow return of results
- Update description to provide an option for blocking behavior of the
 PMIx_Register_event_handler, PMIx_Deregister_event_handler,

 PMIx_Notify_event, PMIx_IOF_pull, PMIx_IOF_deregister, and
 PMIx_IOF_push APIs. The need for blocking forms of these functions was not initially anticipated but has emerged over time. For these functions, the return value is sufficient to provide the caller with information otherwise returned via callback. Thus, use of a NULL value as the callback function parameter was deemed a minimal disruption method for providing the desired capability

CHAPTER 2

PMIx Terms and Conventions

The PMIx Standard has adopted the widespread use of key-value *attributes* to add flexibility to the functionality expressed in the existing APIs. Accordingly, the community has chosen to require that the definition of each standard API include the passing of an array of attributes. These provide a means of customizing the behavior of the API as future needs emerge without having to alter or create new variants of it. In addition, attributes provide a mechanism by which researchers can easily explore new approaches to a given operation without having to modify the API itself.

The PMIx community has further adopted a policy that modification of existing released APIs will only be permitted under extreme circumstances. In its effort to avoid introduction of any such backward incompatibility, the community has avoided the definitions of large numbers of APIs that each focus on a narrow scope of functionality, and instead relied on the definition of fewer generic APIs that include arrays of directives for "tuning" the function's behavior. Thus, modifications to the PMIx standard increasingly consist of the definition of new attributes along with a description of the APIs to which they relate and the expected behavior when used with those APIs.

One area where this can become more complicated relates to the attributes that provide directives to the client process and/or control the behavior of a PMIx standard API. For example, the **PMIX_TIMEOUT** attribute can be used to specify the time (in seconds) before the requested operation should time out. The intent of this attribute is to allow the client to avoid hanging in a request that takes longer than the client wishes to wait, or may never return (e.g., a **PMIx_Fence** that a blocked participant never enters).

If an application truly relies on the <code>PMIX_TIMEOUT</code> attribute in a call to <code>PMIx_Fence</code>, it should set the <code>required</code> flag in the <code>pmix_info_t</code> for that attribute. This informs the library and its SMS host that it must return an immediate error if this attribute is not supported. By not setting the flag, the library and SMS host are allowed to treat the attribute as optional, silently ignoring it if support is not available.

Advice to users -

It is critical that users and application developers consider whether or not a given attribute is required (marking it accordingly) and always check the return status on all PMIx function calls to ensure support was present and that the request was accepted. Note that for non-blocking APIs, a return of **PMIX_SUCCESS** only indicates that the request had no obvious errors and is being processed. The eventual callback will return the status of the requested operation itself.

 While a PMIx library implementer, or an SMS component server, may choose to support a particular PMIx API, they are not required to support every attribute that might apply to it. This would pose a significant barrier to entry for an implementer as there can be a broad range of applicable attributes to a given API, at least some of which may rarely be used in a specific market area. The PMIx community is attempting to help differentiate the attributes by indicating in the standard those that are generally used (and therefore, of higher importance to support) versus those that a "complete implementation" would support.

In addition, the document refers to the following entities and process stages when describing use-cases or operations involving PMIx:

- session refers to an allocated set of resources assigned to a particular user by the system WLM.
 Historically, HPC sessions have consisted of a static allocation of resources i.e., a block of
 resources are assigned to a user in response to a specific request and managed as a unified
 collection. However, this is changing in response to the growing use of dynamic programming
 models that require on-the-fly allocation and release of system resources. Accordingly, the term
 session in this document refers to the current block of assigned resources and is a potentially
 dynamic entity.
- *slot* refers to an allocated entry for a process. WLMs frequently allocate entire nodes to a *session*, but can also be configured to define the maximum number of processes that can simultaneously be executed on each node. This often corresponds to the number of hardware Processing Units (PUs) (typically cores, but can also be defined as hardware threads) on the node. However, the correlation between hardware PUs and slot allocations strictly depends upon system configuration.
- *job* refers to a set of one or more *applications* executed as a single invocation by the user within a session. For example, "*mpiexec -n 1 app1 : -n 2 app2*" is considered a single Multiple Program Multiple Data (MPMD) job containing two applications.
- namespace refers to a character string value assigned by the RM to a job. All applications executed as part of that job share the same namespace. The namespace assigned to each job must be unique within the scope of the governing RM.
- *application* refers to a single executable (binary, script, etc.) member of a *job*. Applications consist of one or more *processes*, either operating independently or in parallel at any given time during their execution.
- rank refers to the numerical location (starting from zero) of a process within the defined scope. Thus, global rank is the rank of a process within its job, while application rank is the rank of that process within its application.
- workflow refers to an orchestrated execution plan frequently spanning multiple jobs carried out under the control of a workflow manager process. An example workflow might first execute a computational job to generate the flow of liquid through a complex cavity, followed by a visualization job that takes the output of the first job as its input to produce an image output.

- *scheduler* refers to the component of the SMS responsible for scheduling of resource allocations. This is also generally referred to as the *system workflow manager* for the purposes of this document, the *WLM* acronym will be used interchangeably to refer to the scheduler.
 - resource manager is used in a generic sense to represent the subsystem that will host the PMIx server library. This could be a vendor's RM, a programming library's RunTime Environment (RTE), or some other agent.
 - *host environment* is used interchangeably with *resource manager* to refer to the process hosting the PMIx server library.
 - *fabric* is used in a generic sense to refer to the networks within the system regardless of speed or protocol. Any use of the term *network* in the document should be considered interchangeable with *fabric*.
- *fabric plane* refers to a collection of devices (Network Interface Cards (NICs)) and switches in a common logical or physical configuration. Fabric planes are often implemented in HPC clusters as separate overlay or physical networks controlled by a dedicated fabric manager.

This document borrows freely from other standards (most notably from the Message Passing Interface (MPI) and OpenMP standards) in its use of notation and conventions in an attempt to reduce confusion. The following sections provide an overview of the conventions used throughout the PMIx Standard document.

2.1 Notational Conventions

20 21	that applies only to programs for which the base language is C is shown as follows:			
	C			
22	C specific text			
23	int foo = 42;			
24 25	Some text is for information only, and is not part of the normative specification. These take several forms, described in their examples below:			
26	Note: General text			
	▼Rationale			
27	Throughout this document, the rationale for the design choices made in the interface specification is			

set off in this section. Some readers may wish to skip these sections, while readers interested in

interface design may want to read them carefully.

	Advice to users
1 2 3	Throughout this document, material aimed at users and that illustrates usage is set off in this section. Some readers may wish to skip these sections, while readers interested in programming with the PMIx API may want to read them carefully.
	Advice to PMIx library implementers
4	Throughout this document, material that is primarily commentary to PMIx library implementers is
5 6	set off in this section. Some readers may wish to skip these sections, while readers interested in PMIx implementations may want to read them carefully.
	Advice to PMIx server hosts
7	Throughout this document, material that is primarily commentary aimed at host environments (e.g.,
8	RMs and RTEs) providing support for the PMIx server library is set off in this section. Some
9	readers may wish to skip these sections, while readers interested in integrating PMIx servers into
10	their environment may want to read them carefully.
	lacklack

1 2.2 Semantics

13

14

15

- The following terms will be taken to mean:
 - *shall, must* and *will* indicate that the specified behavior is *required* of all conforming implementations
 - *should* and *may* indicate behaviors that a complete implementation would include, but are not required of all conforming implementations

2.3 Naming Conventions

- 2 The PMIx standard has adopted the following conventions:
 - PMIx constants and attributes are prefixed with PMIX.
 - Structures and type definitions are prefixed with pmix.
 - Underscores are used to separate words in a function or variable name.
 - Lowercase letters are used in PMIx client APIs except for the PMIx prefix (noted below) and the first letter of the word following it. For example, PMIx_Get_version.
 - PMIx server and tool APIs are all lower case letters following the prefix e.g., PMIx_server_register_nspace.
 - The **PMIx** prefix is used to denote functions.
 - The **pmix**_ prefix is used to denote function pointer and type definitions.

Users should not use the **PMIX**, **PMIX**, or **pmix** prefixes in their applications or libraries so as to avoid symbol conflicts with current and later versions of the PMIx standard and implementations such as the PRI.

5 2.4 Procedure Conventions

While the current PMIx Reference Implementation (PRI) is solely based on the C programming language, it is not the intent of the PMIx Standard to preclude the use of other languages. Accordingly, the procedure specifications in the PMIx Standard are written in a language-independent syntax with the arguments marked as IN, OUT, or INOUT. The meanings of these are:

- IN: The call may use the input value but does not update the argument from the perspective of the caller at any time during the calls execution,
- OUT: The call may update the argument but does not use its input value
- INOUT: The call may both use and update the argument.

Many PMIx interfaces, particularly nonblocking interfaces, use a **void***cbdata object passed to the function that is then passed to the associated callback. In a client-side API, the cbdata is a client-provided context (opaque object) that the client can pass to the nonblocking call (e.g., PMIx_Get_nb). When the nonblocking call (e.g., pmix_value_cbfunc_t) completes, the cbdata is passed back to the client without modification by the PMIx library, thus allowing the client to associate a context with that callback. This is useful if there are many outstanding nonblocking calls.

A similar model is used for the server module functions (see 11.2.1). In this case, the PMIx library is making an upcall into its host via the PMIx server module function and passing a specific cbfunc

and cbdata. The PMIx library expects the host to call the cbfunc with the necessary arguments and pass back the original cbdata upon completing the operation. This gives the server-side PMIx library the ability to associate a context with the call back (since multiple operations may be outstanding). The host has no visibility into the contents of the cbdata object, nor is permitted to alter it in any way.

2.5 Standard vs Reference Implementation

The *PMIx Standard* is implementation independent. The *PMIx Reference Implementation* (PRI) is one implementation of the Standard and the PMIx community strives to ensure that it fully implements the Standard. Given its role as the community's testbed and its widespread use, this document cites the attributes supported by the PRI for each API where relevant by marking them in red. This is not meant to imply nor confer any special role to the PRI with respect to the Standard itself, but instead to provide a convenience to users of the Standard and PRI.

Similarly, the *PMIx Reference RunTime Environment* (PRRTE) is provided by the community to enable users operating in non-PMIx environments to develop and execute PMIx-enabled applications and tools. Attributes supported by the PRRTE are marked in green.

CHAPTER 3

Data Structures and Types

This chapter defines PMIx standard data structures (along with macros for convenient use), types, and constants. These apply to all consumers of the PMIx interface. Where necessary for clarification, the description of, for example, an attribute may be copied from this chapter into a section where it is used.

A PMIx implementation may define additional attributes beyond those specified in this document.

Advice to PMIx library implementers —

Structures, types, and macros in the PMIx Standard are defined in terms of the C-programming language. Implementers wishing to support other languages should provide the equivalent definitions in a language-appropriate manner.

If a PMIx implementation chooses to define additional attributes they should avoid using the **PMIX** prefix in their name or starting the attribute string with a *pmix* prefix. This helps the end user distinguish between what is defined by the PMIx standard and what is specific to that PMIx implementation, and avoids potential conflicts with attributes defined by the standard.

Advice to users —

Use of increment/decrement operations on indices inside PMIx macros is discouraged due to unpredictable behavior. For example, the following sequence:

```
PMIX_INFO_LOAD(&array[n++], "mykey", &mystring, PMIX_STRING);
PMIX_INFO_LOAD(&array[n++], "mykey2", &myint, PMIX_INT);
```

will load the given key-values into incorrect locations if the macro is implemented as:

```
define PMIX_INFO_LOAD(m, k, v, t)
    do {
        if (NULL != (k)) {
            pmix_strncpy((m)->key, (k), PMIX_MAX_KEYLEN);
        }
        (m)->flags = 0;
        pmix_value_load(&((m)->value), (v), (t));
    } while (0)
```

since the index is cited more than once in the macro. The PMIx standard only governs the existence and syntax of macros - it does not specify their implementation. Given the freedom of implementation, a safer call sequence might be as follows:

3.1 Constants

PMIx defines a few values that are used throughout the standard to set the size of fixed arrays or as a means of identifying values with special meaning. The community makes every attempt to minimize the number of such definitions. The constants defined in this section may be used before calling any PMIx library initialization routine. Additional constants associated with specific data structures or types are defined in the section describing that data structure or type.

PMIX_MAX_NSLEN Maximum namespace string length as an integer.

Advice to PMIx library implementers -

PMIX_MAX_NSLEN should have a minimum value of 63 characters. Namespace arrays in PMIx defined structures must reserve a space of size **PMIX_MAX_NSLEN** +1 to allow room for the **NULL** terminator

PMIX_MAX_KEYLEN Maximum key string length as an integer.

PMIX_APP_WILDCARD A value to indicate that the user wants the data for the given key from every application that posted that key, or that the given value applies to all applications within the given nspace.

Advice to PMIx library implementers -

PMIX_MAX_KEYLEN should have a minimum value of 63 characters. Key arrays in PMIx defined structures must reserve a space of size **PMIX_MAX_KEYLEN** +1 to allow room for the **NULL** terminator

3.1.1 PMIx Error Constants

3

4

5 6

7

8

9

10 11

12

13

14 15

16

17

18

19

20

21 22

23

24

25

26 27

28

29

30

31

32

33

34

The pmix_status_t structure is an int type for return status.

The tables shown in this section define the possible values for <code>pmix_status_t</code>. PMIx errors are required to always be negative, with 0 reserved for <code>PMIX_SUCCESS</code>. Values in the list that were deprecated in later standards are denoted as such. Values added to the list in this version of the standard are shown in <code>magenta</code>.

Advice to PMIx library implementers

A PMIx implementation must define all of the constants defined in this section, even if they will never return the specific value to the caller.

Advice to users -

Other than PMIX_SUCCESS (which is required to be zero), the actual value of any PMIx error constant is left to the PMIx library implementer. Thus, users are advised to always refer to constant by name, and not a specific implementation's value, for portability between implementations and compatibility across library versions.

3.1.1.1 General Error Constants

These are general constants originally defined in versions 1 and 2 of the PMIx Standard.

PMIX SUCCESS Success

PMIX ERROR General Error

PMIX ERR SILENT Silent error

PMIX_ERR_DEBUGGER_RELEASE Error in debugger release

PMIX_ERR_PROC_RESTART Fault tolerance: Error in process restart

PMIX_ERR_PROC_CHECKPOINT Fault tolerance: Error in process checkpoint

PMIX ERR PROC MIGRATE Fault tolerance: Error in process migration

PMIX ERR PROC ABORTED Process was aborted

PMIX_ERR_PROC_REQUESTED_ABORT Process is already requested to abort

PMIX_ERR_PROC_ABORTING Process is being aborted

PMIX ERR SERVER FAILED REQUEST Failed to connect to the server

PMIX_EXISTS Requested operation would overwrite an existing value

PMIX_ERR_INVALID_CRED Invalid security credentials

PMIX_ERR_HANDSHAKE_FAILED Connection handshake failed

PMIX ERR READY FOR HANDSHAKE Ready for handshake

PMIX ERR WOULD BLOCK Operation would block

PMIX ERR UNKNOWN DATA TYPE Unknown data type

PMIX ERR PROC ENTRY NOT FOUND Process not found

PMIX_ERR_TYPE_MISMATCH Invalid type

PMIX_ERR_UNPACK_INADEQUATE_SPACE Inadequate space to unpack data

1	PMIX_ERR_UNPACK_FAILURE Unpack failed
2	PMIX_ERR_PACK_FAILURE Pack failed
3	PMIX_ERR_PACK_MISMATCH Pack mismatch
4	PMIX_ERR_NO_PERMISSIONS No permissions
5	PMIX_ERR_TIMEOUT Timeout expired
6	PMIX_ERR_UNREACH Unreachable
7	PMIX_ERR_IN_ERRNO Error defined in errno
8	PMIX_ERR_BAD_PARAM Bad parameter
9	PMIX_ERR_RESOURCE_BUSY Resource busy
10	PMIX_ERR_OUT_OF_RESOURCE Resource exhausted
11	PMIX_ERR_DATA_VALUE_NOT_FOUND Data value not found
12	PMIX_ERR_INIT Error during initialization
13	PMIX_ERR_NOMEM Out of memory
14	PMIX_ERR_INVALID_ARG Invalid argument
15	PMIX_ERR_INVALID_KEY Invalid key
16	PMIX_ERR_INVALID_KEY_LENGTH Invalid key length
17	PMIX_ERR_INVALID_VAL Invalid value
18	PMIX_ERR_INVALID_VAL_LENGTH Invalid value length
19	PMIX_ERR_INVALID_LENGTH Invalid argument length
20	PMIX_ERR_INVALID_NUM_ARGS Invalid number of arguments
21	PMIX_ERR_INVALID_ARGS Invalid arguments
22	PMIX_ERR_INVALID_NUM_PARSED Invalid number parsed
23	PMIX_ERR_INVALID_KEYVALP Invalid key/value pair
24	PMIX_ERR_INVALID_SIZE Invalid size
25	PMIX_ERR_INVALID_NAMESPACE Invalid namespace
26	PMIX_ERR_SERVER_NOT_AVAIL Server is not available
27	PMIX_ERR_NOT_FOUND Not found
28	PMIX_ERR_NOT_SUPPORTED Not supported
29	PMIX_ERR_NOT_IMPLEMENTED Not implemented
30	PMIX_ERR_COMM_FAILURE Communication failure
31	PMIX_ERR_UNPACK_READ_PAST_END_OF_BUFFER Unpacking past the end of the buffer
32	provided
33	PMIX_ERR_CONFLICTING_CLEANUP_DIRECTIVES Conflicting directives given for
34	job/process cleanup
35	PMIX_ERR_LOST_CONNECTION_TO_SERVER Lost connection to server
36	PMIX_ERR_LOST_PEER_CONNECTION Lost connection to peer
37	PMIX_ERR_LOST_CONNECTION_TO_CLIENT Lost connection to client
38	PMIX_QUERY_PARTIAL_SUCCESS Query partial success (used by query system)
39	PMIX_NOTIFY_ALLOC_COMPLETE Notify that allocation is complete
40	PMIX_JCTRL_CHECKPOINT Job control: Monitored by PMIx client to trigger checkpoint
41	operation
42	PMIX_JCTRL_CHECKPOINT_COMPLETE Job control: Sent by PMIx client and monitored
43	by PMIx server to notify that requested checkpoint operation has completed.

1		PMIX_JCTRL_PREEMPT_ALERT Job control: Monitored by PMIx client to detect an RM
2		intending to preempt the job.
3		PMIX_MONITOR_HEARTBEAT_ALERT Job monitoring: Heartbeat alert
4		PMIX_MONITOR_FILE_ALERT Job monitoring: File alert
5		PMIX_PROC_TERMINATED Process terminated - can be either normal or abnormal
6		termination
7		PMIX_ERR_INVALID_TERMINATION Process terminated without calling
8		PMIx_Finalize, or was a member of an assemblage formed via PMIx_Connect and
9		terminated or called PMIx_Finalize without first calling PMIx_Disconnect (or its
0		non-blocking form) from that assemblage.
11	3.1.1.2	Job-Related Error Constants
12		PMIX_ERR_JOB_FAILED_TO_START At least one process in the job failed to start
13		PMIX_ERR_JOB_APP_NOT_EXECUTABLE A specified application executable is does not
4		have execution permissions
15		PMIX_ERR_JOB_NO_EXE_SPECIFIED No executable was specified in a spawn request
16		PMIX_ERR_JOB_FAILED_TO_MAP The RM was unable to map (i.e., assign locations) the
17		processes in the job
8		PMIX_ERR_JOB_CANCELLED The submitted job was canceled prior to launch
19		PMIX_ERR_JOB_FAILED_TO_LAUNCH The job failed to launch
20		PMIX_ERR_JOB_ABORTED The job was aborted due to at least one process calling
21		PMIx_Abort
22		PMIX_ERR_JOB_KILLED_BY_CMD The job was killed in response to a user command
23		PMIX_ERR_JOB_ABORTED_BY_SIG The job was killed in response to a signal (e.g.,
24		SIGTERM or SIGKILL)
25		PMIX_ERR_JOB_TERM_WO_SYNC The job was killed due to at least one process terminating
26		without first calling PMIx_Finalize
27		PMIX_ERR_JOB_SENSOR_BOUND_EXCEEDED The job was killed due to at least one
28		process exceeding a monitor threshold
29		PMIX_ERR_JOB_NEVER_LAUNCHED The job was never launched
30		PMIX_ERR_JOB_NON_ZERO_TERM The job was killed due to at least one process exiting
31		with a non-zero status
32		PMIX_ERR_JOB_ALLOC_FAILED The job was unable to obtain an allocation
33		PMIX_ERR_JOB_CANNOT_LAUNCH The resources required by the job within the given
34		allocation are busy, thus preventing the job from being launched
35	3.1.1.3	Operational Error Constants
36		PMIX_ERR_EVENT_REGISTRATION Error in event registration
37		PMIX_ERR_JOB_TERMINATED (Deprecated in PMIx C) hanged to
38		PMIX_EVENT_JOB_END
39		PMIX_ERR_UPDATE_ENDPOINTS Error updating endpoints
10		PMIX_MODEL_DECLARED Model declared
11		PMIX CDS ACTION COMPLETE. The global data storage (GDS) action has completed

- **PMIX_ERR_INVALID_OPERATION** The requested operation is supported by the implementation and host environment, but fails to meet a requirement (e.g., requesting to *disconnect* from processes without first *connecting* to them).
- PMIX_PROC_HAS_CONNECTED A tool or client has connected to the PMIx server
- PMIX_CONNECT_REQUESTED Connection has been requested by a PMIx-based tool
- PMIX_MODEL_RESOURCES Resource usage by a programming model has changed
- PMIX_OPENMP_PARALLEL_ENTERED An OpenMP parallel code region has been entered
- PMIX_OPENMP_PARALLEL_EXITED An OpenMP parallel code region has completed
- PMIX_OPERATION_IN_PROGRESS A requested operation is already in progress
- **PMIX_OPERATION_SUCCEEDED** The requested operation was performed atomically no callback function will be executed
- **PMIX_ERR_PARTIAL_SUCCESS** The operation is considered successful but not all elements of the operation were concluded (e.g., some members of a group construct operation chose not to participate)
- PMIX_ERR_DUPLICATE_KEY The provided key has already been published on a different data range
- **PMIX_ERR_INVALID_OPERATION** The requested operation is not valid this can possibly indicate the inclusion of conflicting directives or a request to perform an operation that conflicts with an ongoing one.
- **PMIX_GROUP_INVITED** The process has been invited to join a PMIx Group the identifier of the group and the ID's of other invited (or already joined) members will be included in the notification
- **PMIX_GROUP_LEFT** A process has asynchronously left a PMIx Group the process identifier of the departing process will in included in the notification
- **PMIX_GROUP_MEMBER_FAILED** A member of a PMIx Group has abnormally terminated (i.e., without formally leaving the group prior to termination) the process identifier of the failed process will in included in the notification
- **PMIX_GROUP_INVITE_ACCEPTED** A process has accepted an invitation to join a PMIx Group the identifier of the group being joined will be included in the notification
- **PMIX_GROUP_INVITE_DECLINED** A process has declined an invitation to join a PMIx Group the identifier of the declined group will be included in the notification
- **PMIX_GROUP_INVITE_FAILED** An invited process failed or terminated prior to responding to the invitation the identifier of the failed process will be included in the notification.
- **PMIX_GROUP_MEMBERSHIP_UPDATE** The membership of a PMIx group has changed the identifiers of the revised membership will be included in the notification.
- PMIX_GROUP_CONSTRUCT_ABORT Any participant in a PMIx group construct operation that returns PMIX_GROUP_CONSTRUCT_ABORT from the leader failed event handler will cause all participants to receive an event notifying them of that status. Similarly, the leader may elect to abort the procedure by either returning this error code from the handler assigned to the PMIX_GROUP_INVITE_ACCEPTED or PMIX_GROUP_INVITE_DECLINED codes, or by generating an event for the abort code. Abort events will be sent to all invited or existing members of the group.

1 PMIX_GROUP_CONSTRUCT_COMPLETE The group construct operation has completed - the 2 final membership will be included in the notification. PMIX GROUP LEADER FAILED The current leader of a group including this process has 3 4 abnormally terminated - the group identifier will be included in the notification. PMIX GROUP LEADER SELECTED A new *leader* of a group including this process has been 5 6 selected - the identifier of the new leader will be included in the notification 7 PMIX GROUP CONTEXT ID ASSIGNED A new Process Group Context IDentifier (PGCID) has been assigned by the host environment to a group that includes this 8 9 process - the group identifier will be included in the notification. PMIX_ERR_REPEAT_ATTR_REGISTRATION The attributes for an identical function have 10 already been registered at the specified level (host, server, or client) 11 An IO forwarding operation failed - the affected channel will be 12 PMIX ERR IOF FAILURE included in the notification 13 14 PMIX ERR IOF COMPLETE IO forwarding of the standard input for this process has completed - i.e., the stdin file descriptor has closed 15 16 PMIX ERR GET MALLOC REOD The data returned by **PMIx Get** contains values that required dynamic memory allocations (i.e., "malloc"), despite a request for static pointers to 17 the values in the key-value store. User is responsible for releasing the memory when done 18 19 with the information. 3.1.1.4 **Job-related constants** 20 21 The following constants indicate that a session or job has started and normally completed: 22 PMIX_EVENT_JOB_START The job has started 23 PMIX EVENT JOB END The job has normally terminated 24 PMIX EVENT SESSION START The session has started 25 PMIX EVENT SESSION END The session has normally terminated 26 The next set of constants are used to indicate that a job has abnormally terminated and to convey 27 some information as to the cause: 28 PMIX_ERR_JOB_APP_NOT_EXECUTABLE The specified application executable either 29 could not be found, or lacks execution privileges. PMIX ERR JOB NO EXE SPECIFIED The job request did not specify an executable. 30 31 PMIX ERR JOB FAILED TO MAP The launcher was unable to map the processes for the 32 specified job request. PMIX ERR JOB CANCELED The job was canceled by the host environment 33 PMIX_ERR_JOB_FAILED_TO LAUNCH One or more processes in the job request failed to 34 35 launch

One or more processes in the job called abort, causing the job to

The job was aborted due to receipt of an error signal

The job was killed by user command

PMIX ERR JOB ABORTED

PMIX_ERR_JOB_KILLED_BY_CMD

PMIX ERR JOB ABORTED BY SIG

be terminated

(e.g., SIGKILL)

36

37

38

39

The job was terminated due to one or more processes 1 PMIX_ERR_JOB_TERM_WO_SYNC 2 exiting without first calling PMIx Finalize 3 PMIX ERR JOB SENSOR BOUND EXCEEDED The job was terminated due to one or more 4 processes exceeding a specified sensor limit 5 PMIX_ERR_JOB_NON_ZERO_TERM The job was terminated due to one or more processes 6 exiting with a non-zero status 7 PMIX ERR JOB ALLOC FAILED The job request could not be executed due to failure to 8 obtain the specified allocation 9 PMIX_ERR_JOB_ABORTED_BY_SYS_EVENT The job was aborted due to receipt of a 10 system event 3.1.1.5 System error constants 11 12 PMIX ERR SYS BASE Mark the beginning of a dedicated range of constants for system event 13 reporting. 14 PMIX ERR NODE DOWN A node has gone down - the identifier of the affected node will be 15 included in the notification PMIX ERR NODE OFFLINE A node has been marked as offline - the identifier of the affected 16 17 node will be included in the notification PMIX_ERR_SYS_OTHER Mark the end of a dedicated range of constants for system event 18 reporting. 19 20 3.1.1.6 **Event handler error constants** 21 Event handler: No action taken PMIX EVENT NO ACTION TAKEN PMIX_EVENT_PARTIAL_ACTION_TAKEN Event handler: Partial action taken 22 23 PMIX EVENT ACTION DEFERRED Event handler: Action deferred PMIX EVENT ACTION COMPLETE Event handler: Action complete 24 3.1.1.7 **User-Defined Error Constants** 25 26 PMIx establishes an error code boundary for constants defined in the PMIx standard. Negative 27 values larger than this (and any positive values greater than zero) are guaranteed not to conflict with PMIx values. 28 29 PMIX_EXTERNAL_ERR_BASE A starting point for user-level defined error constants. 30 Negative values lower than this are guaranteed not to conflict with PMIx values. Definitions 31 should always be based on the PMIX EXTERNAL ERR BASE constant and not a specific 32 value as the value of the constant may change.

3.1.2 Macros for use with PMIx constants

4 3.1.2.1 Detect system event constant

Test a given error constant to see if it falls within the dedicated range of constants for system event reporting.

PMIx v2.2

35

1 PMIX SYSTEM EVENT(a) IN 2 3 Error constant to be checked (pmix_status_t) Returns **true** if the provided values falls within the dedicated range of constants for system event 4 5 reporting 3.2 **Data Types** 7 This section defines various data types used by the PMIx APIs. The version of the standard in 8 which a particular data type was introduced is shown in the margin. **Key Structure** 3.2.1 The pmix_key_t structure is a statically defined character array of length PMIX_MAX_KEYLEN 10 +1, thus supporting keys of maximum length PMIX MAX KEYLEN while preserving space for a 11 12 mandatory **NULL** terminator. PMIx v2.0 13 typedef char pmix key t[PMIX MAX KEYLEN+1]; Characters in the key must be standard alphanumeric values supported by common utilities such as 14 15 strcmp. Advice to users -References to keys in PMIx v1 were defined simply as an array of characters of size 16 **PMIX MAX KEYLEN+1.** The **pmix key t** type definition was introduced in version 2 of the 17 18 standard. The two definitions are code-compatible and thus do not represent a break in backward compatibility. 19 20 Passing a pmix_key_t value to the standard size of utility can result in compiler warnings of 21 incorrect returned value. Users are advised to avoid using sizeof(pmix_key_t) and instead rely on 22 the PMIX MAX KEYLEN constant.

3.2.1.1 Check key macro 2 Compare the key in a **pmix_info_t** to a given value PMIx v3.0 3 PMIX_CHECK_KEY(a, b) IN 4 5 Pointer to the structure whose key is to be checked (pointer to pmix_info_t) 6 IN 7 String value to be compared against (char*) 8 Returns **true** if the key matches the given value 3.2.1.2 Load key macro 10 Load a key into a pmix_info_t PMIx v4.0 11 PMIX_LOAD_KEY(a, b) IN 12 13 Pointer to the structure whose key is to be loaded (pointer to pmix info t) 14 IN 15 String value to be loaded (char*) No return value. 16 3.2.2 **Namespace Structure** 18 The **pmix_nspace_t** structure is a statically defined character array of length PMIX_MAX_NSLEN +1, thus supporting namespaces of maximum length PMIX_MAX_NSLEN 19 20 while preserving space for a mandatory **NULL** terminator. PMIx v2.0typedef char pmix_nspace_t[PMIX_MAX_NSLEN+1]; 21 Characters in the namespace must be standard alphanumeric values supported by common utilities 22 23 such as stremp.

Advice to users -References to namespace values in PMIx v1 were defined simply as an array of characters of size 1 2 **PMIX_MAX_NSLEN+1.** The **pmix_nspace_t** type definition was introduced in version 2 of the standard. The two definitions are code-compatible and thus do not represent a break in backward 3 4 compatibility. 5 Passing a pmix_nspace_t value to the standard size of utility can result in compiler warnings of 6 incorrect returned value. Users are advised to avoid using sizeof(pmix_nspace_t) and instead rely 7 on the PMIX MAX NSLEN constant. Check namespace macro 3.2.2.1 Compare the string in a pmix_nspace_t to a given value 9 PMIx v3.0PMIX_CHECK_NSPACE(a, b) 10 IN 11 12 Pointer to the structure whose value is to be checked (pointer to pmix nspace t) IN 13 String value to be compared against (char*) 14 Returns **true** if the namespace matches the given value 15 3.2.2.2 Load namespace macro 17 Load a namespace into a pmix_nspace_t PMIx v4.018 PMIX_LOAD_NSPACE(a, b) IN 19 20 Pointer to the target structure (pointer to pmix nspace t) IN 21 22 String value to be loaded - if **NULL** is given, then the target structure will be initialized to

zero's (char*)

No return value.

23

3.2.3 Rank Structure

2 The pmix_rank_t structure is a uint32_t type for rank values. PMIx v1.0typedef uint32_t pmix_rank_t; 3 The following constants can be used to set a variable of the type pmix rank t. All definitions 4 were introduced in version 1 of the standard unless otherwise marked. Valid rank values start at 5 6 zero. 7 PMIX RANK UNDEF A value to request job-level data where the information itself is not 8 associated with any specific rank, or when passing a pmix_proc_t identifier to an 9 operation that only references the namespace field of that structure. A value to indicate that the user wants the data for the given key 10 PMIX RANK WILDCARD 11 from every rank that posted that key. PMIX RANK LOCAL NODE Special rank value used to define groups of ranks. This constant 12 defines the group of all ranks on a local node. 13 14 PMIX RANK LOCAL PEERS Special rank value used to define groups of rankss. This 15 constant defines the group of all ranks on a local node within the same namespace as the 16 current process. 17 An invalid rank value. PMIX RANK INVALID Define an upper boundary for valid rank values. 18 PMIX RANK VALID 3.2.4 Process Structure 20 The pmix proc t structure is used to identify a single process in the PMIx universe. It contains 21 a reference to the namespace and the **pmix rank t** within that namespace. PMIx v1.022 typedef struct pmix_proc { 23 pmix_nspace_t nspace; pmix rank t rank; 24

6 3.2.5 Process structure support macros

} pmix proc t;

25

27

The following macros are provided to support the **pmix_proc_t** structure.

1	3.2.5.1	Initialize the pmix_proc_t structure
2		Initialize the <pre>pmix_proc_t</pre> fields
	PMIx v1.0	C
3		PMIX_PROC_CONSTRUCT (m)
		C
4		IN m
5		Pointer to the structure to be initialized (pointer to pmix_proc_t)
6	3.2.5.2	Destruct the pmix_proc_t structure
7		Destruct the pmix_proc_t fields
		C
8		PMIX_PROC_DESTRUCT (m)
		C
9		IN m
10		Pointer to the structure to be destructed (pointer to pmix_proc_t)
11		There is nothing to release here as the fields in pmix_proc_t are all declared <i>static</i> . However,
12		the macro is provided for symmetry in the code and for future-proofing should some allocated field
13		be included some day.
14	3.2.5.3	Create a pmix_proc_t array
15		Allocate and initialize an array of pmix_proc_t structures
	PMIx v1.0	C
16		PMIX_PROC_CREATE(m, n)
		C
17		INOUT m
8		Address where the pointer to the array of pmix_proc_t structures shall be stored (handle)
19		IN n
20		Number of structures to be allocated (size_t)
21	3.2.5.4	Free a pmix_proc_t
22		Release a pmix_proc_t structure
	PMIx v4.0	C
23		PMIX_PROC_RELEASE (m)
		C
24		IN m
25		Pointer to a pmix proc t structure (handle)

```
3.2.5.5 Free a pmix_proc_t array
2
               Release an array of pmix_proc_t structures
   PMIx v1.0
 3
               PMIX_PROC_FREE(m, n)
               IN
4
5
                   Pointer to the array of pmix_proc_t structures (handle)
6
               IN
                   Number of structures in the array (size_t)
7
    3.2.5.6
              Load a pmix_proc_t structure
9
               Load values into a pmix_proc_t
   PMIx v2.0
10
               PMIX_PROC_LOAD(m, n, r)
11
               IN
12
                   Pointer to the structure to be loaded (pointer to pmix_proc_t)
               IN
13
14
                   Namespace to be loaded (pmix nspace t)
              IN
15
                   Rank to be assigned ( pmix_rank_t )
16
17
               No return value. Deprecated in favor of PMIX_LOAD_PROCID
              Compare identifiers
    3.2.5.7
18
19
               Compare two pmix proc t identifiers
   PMIx v3.0
20
               PMIX CHECK PROCID(a, b)
               IN
21
22
                   Pointer to a structure whose ID is to be compared (pointer to pmix_proc_t)
23
               IN
24
                   Pointer to a structure whose ID is to be compared (pointer to pmix_proc_t)
               Returns true if the two structures contain matching namespaces and:
25
               • the ranks are the same value
26
27
               • one of the ranks is PMIX RANK WILDCARD
```

```
3.2.5.8 Load a procID structure
               Load values into a pmix_proc_t
   PMIx v4.0
3
               PMIX LOAD PROCID (m, n, r)
               IN
4
                   Pointer to the structure to be loaded (pointer to pmix_proc_t)
 5
               IN
6
7
                   Namespace to be loaded ( pmix_nspace_t )
8
               IN
9
                   Rank to be assigned ( pmix_rank_t )
    3.2.5.9
              Construct a multi-cluster namespace
               Construct a multi-cluster identifier containing a cluster ID and a namespace
11
   PMIx v4.0
               PMIX MULTICLUSTER_NSPACE_CONSTRUCT(m, n, r)
12
13
               IN
                   pmix nspace t structure that will contain the multi-cluster identifier (
14
15
                   pmix nspace t)
               IN
16
17
                   Cluster identifier (char*)
               IN
18
19
                   Namespace to be loaded (pmix nspace t)
20
               Combined length of the cluster identifier and namespace must be less than PMIX MAX NSLEN -2.
    3.2.5.10 Parse a multi-cluster namespace
21
22
               Parse a multi-cluster identifier into its cluster ID and namespace parts
   PMIx v4.0
23
               PMIX MULTICLUSTER NSPACE PARSE (m, n, r)
               IN
24
                   pmix_nspace_t structure containing the multi-cluster identifier (pointer to
25
                   pmix_nspace_t)
26
               IN
27
                   Location where the cluster ID is to be stored ( pmix_nspace_t )
28
               IN
29
                   Location where the namespace is to be stored ( pmix_nspace_t )
30
```

1 3.2.6 Process State Structure

2 <i>PMIx v2.0</i> 3 4	The <code>pmix_proc_state_t</code> structure is a <code>uint8_t</code> type for process state values. The following constants can be used to set a variable of the type <code>pmix_proc_state_t</code> . All values were originally defined in version 2 of the standard unless otherwise marked.
	Advice to users
5 6	The fine-grained nature of the following constants may exceed the ability of an RM to provide updated process state values during the process lifetime. This is particularly true of states in the
7	launch process, and for short-lived processes.
8	PMIX_PROC_STATE_UNDEF Undefined process state
9	PMIX_PROC_STATE_PREPPED Process is ready to be launched
0	PMIX_PROC_STATE_LAUNCH_UNDERWAY Process launch is underway
1	PMIX_PROC_STATE_RESTART Process is ready for restart
2	PMIX_PROC_STATE_TERMINATE Process is marked for termination
3	PMIX_PROC_STATE_RUNNING Process has been locally fork'ed by the RM
4	PMIX_PROC_STATE_CONNECTED Process has connected to PMIx server
5	PMIX_PROC_STATE_UNTERMINATED Define a "boundary" between the terminated states
6	and PMIX_PROC_STATE_CONNECTED so users can easily and quickly determine if a
7	process is still running or not. Any value less than this constant means that the process has not
8	terminated.
9	PMIX_PROC_STATE_TERMINATED Process has terminated and is no longer running
0	PMIX_PROC_STATE_ERROR Define a boundary so users can easily and quickly determine if
:1	a process abnormally terminated. Any value above this constant means that the process has
2	terminated abnormally.
3	PMIX_PROC_STATE_KILLED_BY_CMD Process was killed by a command
4	PMIX_PROC_STATE_ABORTED Process was aborted by a call to PMIx_Abort
5	PMIX_PROC_STATE_FAILED_TO_START Process failed to start
6	PMIX_PROC_STATE_ABORTED_BY_SIG Process aborted by a signal
7	PMIX_PROC_STATE_TERM_WO_SYNC Process exited without calling PMIx_Finalize
8	PMIX_PROC_STATE_COMM_FAILED Process communication has failed
9	PMIX_PROC_STATE_SENSOR_BOUND_EXCEEDED Process exceeded a specified sensor
0	limit
1	PMIX_PROC_STATE_CALLED_ABORT Process called PMIx_Abort
2	PMIX_PROC_STATE_HEARTBEAT_FAILED Frocess failed to send heartbeat within
3	specified time limit
4	PMIX_PROC_STATE_MIGRATING Process failed and is waiting for resources before
5	restarting
6	PMIX_PROC_STATE_CANNOT_RESTART Process failed and cannot be restarted
7	PMIX_PROC_STATE_TERM_NON_ZERO Process exited with a non-zero status
ρ	DMTY DDOC STATE FATIED TO LAUNCH Unable to launch process

1 3.2.7 Job State Structure

The pmix_job_state_t structure is a uint8_t type for job state values. The following constants can be used to set a variable of the type pmix_job_state_t. All values were originally defined in version 4 of the standard unless otherwise marked.

Advice to users

The fine-grained nature of the following constants may exceed the ability of an RM to provide updated job state values during the job lifetime. This is particularly true of states in the launch process, and for short-lived jobs.

```
8
               PMIX_JOB_STATE_UNDEF
                                             Undefined job state
9
               PMIX JOB STATE PREPPED
                                               Job is ready to be launched
               PMIX_JOB_STATE_LAUNCH_UNDERWAY
                                                          Job launch procedure is in progress
10
                                               Job is running
11
               PMIX_JOB_STATE_RUNNING
12
               PMIX_JOB_STATE_SUSPENDED
                                                  Job has been suspended
               PMIX JOB STATE CONNECTED
                                                  All processes in the job have connected to their PMIx
13
14
                    server
15
               PMIX JOB STATE TERMINATED
                                                   The job has terminated and is no longer running -
16
                    typically will be accompanied by the job exit status in response to a query
17
               PMIX JOB STATE TERMINATED WITH ERROR
                                                                  The job has terminated and is no longer
18
                    running - typically will be accompanied by the job-related error code in response to a query
```

19 3.2.8 Process Information Structure

The **pmix_proc_info_t** structure defines a set of information about a specific process including it's name, location, and state.

```
PMIx v2.0
```

20

21

5

```
22
            typedef struct pmix_proc_info {
                /** Process structure */
23
24
                pmix_proc_t proc;
                /** Hostname where process resides */
25
26
                char *hostname;
27
                /** Name of the executable */
                char *executable name:
28
29
                /** Process ID on the host */
30
                pid t pid;
                /** Exit code of the process. Default: 0 */
31
32
                int exit_code;
33
                /** Current state of the process */
34
                pmix_proc_state_t state;
35
            } pmix_proc_info_t;
```

3.2.9 Process Information Structure support macros The following macros are provided to support the **pmix_proc_info_t** structure. 2 3.2.9.1 Initialize the pmix_proc_info_t structure Initialize the pmix_proc_info_t fields PMIx v2.0 5 PMIX PROC INFO CONSTRUCT (m) IN 6 7 Pointer to the structure to be initialized (pointer to **pmix_proc_info_t**) 3.2.9.2 Destruct the pmix_proc_info_t structure 9 Destruct the pmix proc info t fields PMIx v2.0 10 PMIX PROC INFO DESTRUCT (m) IN 11 12 Pointer to the structure to be destructed (pointer to pmix_proc_info_t) 3.2.9.3 13 Create a pmix_proc_info_t array 14 Allocate and initialize a **pmix_proc_info_t** array *PMIx v2.0* 15 PMIX PROC INFO CREATE (m, n) INOUT m 16 Address where the pointer to the array of pmix_proc_info_t structures shall be stored 17 18 (handle) IN 19 n Number of structures to be allocated (size_t) 20 3.2.9.4 Free a pmix proc info t Release a pmix_proc_info_t structure 22 PMIx v2.0PMIX PROC INFO RELEASE (m) 23 IN 24 25 Pointer to a pmix proc info t structure (handle)

3.2.9.5 Free a pmix_proc_info_t array 2 Release an array of pmix_proc_info_t structures PMIx v2.03 PMIX PROC INFO FREE (m, n) IN 4 5 Pointer to the array of **pmix proc info t** structures (handle) 6 IN 7 Number of structures in the array (size t) 3.2.10 Scope of Put Data PMIx v1.0The pmix_scope_t structure is a uint8_t type that defines the scope for data passed to 10 **PMIx_Put**. The following constants can be used to set a variable of the type **pmix_scope_t**. All definitions were introduced in version 1 of the standard unless otherwise marked. 11 12 Specific implementations may support different scope values, but all implementations must support 13 at least PMIX GLOBAL. If a scope value is not supported, then the PMIx Put call must return 14 PMIX_ERR_NOT_SUPPORTED. 15 PMIX SCOPE UNDEF Undefined scope PMIX LOCAL The data is intended only for other application processes on the same node. 16 17 Data marked in this way will not be included in data packages sent to remote requestors — 18 i.e., it is only available to processes on the local node. 19 PMIX REMOTE The data is intended solely for applications processes on remote nodes. Data 20 marked in this way will not be shared with other processes on the same node — i.e., it is only 21 available to processes on remote nodes. 22 PMIX GLOBAL The data is to be shared with all other requesting processes, regardless of 23 location. 24 PMIX INTERNAL The data is intended solely for this process and is not shared with other PMIx v2.025 processes. 3.2.11 Job State Structure The pmix_job_state_t structure is a uint8_t type for job state values. The following 27 *PMIx v4.0* constants can be used to set a variable of the type **pmix_job_state_t**. All values were 28 29 originally defined in version 4 of the standard unless otherwise marked. Advice to users -The fine-grained nature of the following constants may exceed the ability of an RM to provide 30 updated job state values during the job lifetime. This is particularly true of states in the launch 31 32 process, and for short-lived jobs.

1	PMIX_JOB_STATE_UNDEF Undefined job state
2	PMIX_JOB_STATE_PREPPED Job is ready to be launched
3	PMIX_JOB_STATE_LAUNCH_UNDERWAY Job launch is underway
4	PMIX_JOB_STATE_RUNNING All processes in the job have been spawned
5	PMIX_JOB_STATE_SUSPENDED All processes in the job have been suspended
6	PMIX_JOB_STATE_CONNECTED All processes in the job have connected to their PMIx
7	server
8	PMIX_JOB_STATE_UNTERMINATED Define a "boundary" between the terminated states
9	and PMIX_JOB_STATE_TERMINATED so users can easily and quickly determine if a job
10	is still running or not. Any value less than this constant means that the job has not terminated.
11	PMIX_JOB_STATE_TERMINATED All processes in the job have terminated and are no
12	longer running - typically will be accompanied by the job exit status in response to a query
13	PMIX_JOB_STATE_TERMINATED_WITH_ERROR Define a boundary so users can easily
14	and quickly determine if a job abnormally terminated - typically will be accompanied by a
15	job-related error code in response to a query Any value above this constant means that the job
16	terminated abnormally.

17 3.2.12 Range of Published Data

32

33

18 <i>PMIx v1.0</i>	The pmix_data_range_t structure is a uint8_t type that defines a range for data <i>published</i>
19	via functions other than PMIx_Put - e.g., the PMIx_Publish API. The following constants
20	can be used to set a variable of the type pmix_data_range_t . Several values were initially
21	defined in version 1 of the standard but subsequently renamed and other values added in version 2.
22	Thus, all values shown below are as they were defined in version 2 except where noted.
23	PMIX_RANGE_UNDEF Undefined range
24	PMIX_RANGE_RM Data is intended for the host resource manager.
25	PMIX_RANGE_LOCAL Data is only available to processes on the local node.
26	PMIX_RANGE_NAMESPACE Data is only available to processes in the same namespace.
27	PMIX_RANGE_SESSION Data is only available to all processes in the session.
28	PMIX_RANGE_GLOBAL Data is available to all processes.
29	PMIX_RANGE_CUSTOM Range is specified in the pmix_info_t associated with this call.
30	PMIX_RANGE_PROC_LOCAL Data is only available to this process.
31	PMIX_RANGE_INVALID Invalid value
	Advice to users

The names of the $\ensuremath{ \mbox{\tt pmix_data_range_t} }$ values changed between version 1 and version 2 of the

standard, thereby breaking backward compatibility

3.2.13 Data Persistence Structure

```
The pmix_persistence_t structure is a uint8_t type that defines the policy for data
  PMIx v1.0
 3
               published by clients via the PMIx_Publish API. The following constants can be used to set a
 4
               variable of the type pmix persistence t. All definitions were introduced in version 1 of the
               standard unless otherwise marked.
 5
6
               PMIX PERSIST INDEF
                                            Retain data until specifically deleted.
 7
               PMIX PERSIST FIRST READ
                                                   Retain data until the first access, then the data is deleted.
8
               PMIX PERSIST PROC
                                           Retain data until the publishing process terminates.
9
                                          Retain data until the application terminates.
               PMIX_PERSIST_APP
10
               PMIX PERSIST SESSION
                                               Retain data until the session/allocation terminates.
               PMIX PERSIST INVALID
                                               Invalid value
11
```

12 3.2.14 Data Array Structure

19

21

23

29

30

IN

```
13 typedef struct pmix_data_array
14 pmix_data_type_t type;
15 size_t size;
16 void *array;
17 pmix_data_array_t;

C

The pmix_data_array_t structure is used to pass arrays of related values. Any PMIx data
```

The **pmix_data_array_t** structure is used to pass arrays of related values. Any PMIx data type (including complex structures) can be included in the array.

3.2.15 Data array structure support macros

The following macros are provided to support the **pmix_data_array_t** structure.

2 3.2.15.1 Initialize the pmix_data_array_t structure

Initialize the pmix_data_array_t fields, allocating memory for the array itself.

```
PMIx v2.2

PMIX_DATA_ARRAY_CONSTRUCT (m, n, t)

IN m
Pointer to the structure to be initialized (pointer to pmix_data_array_t)

IN n
Number of elements in the array (size t)
```

3.2.15.2 Destruct the pmix_data_array_t structure Destruct the pmix_data_array_t fields, releasing the array's memory. PMIx v2.23 PMIX DATA ARRAY DESTRUCT (m) IN m 4 5 Pointer to the structure to be destructed (pointer to pmix_data_array_t) 3.2.15.3 Create and initialize a pmix data array t object Allocate and initialize a pmix_data_array_t structure and initialize it, allocating memory for 7 8 the array itself as well. PMIx v2.29 PMIX DATA ARRAY CREATE (m, n, t) INOUT m 10 Address where the pointer to the pmix data array t structure shall be stored (handle) 11 IN 12 13 Number of elements in the array (size t) 14 IN 15 PMIx data type for the array elements (pmix_data_type_t) 3.2.15.4 Free a pmix_data_array_t object 16 Release a pmix_data_array_t structure, including releasing the array's memory. 17 PMIx v2.218 PMIX DATA ARRAY FREE (m) IN 19 Pointer to the pmix_data_array_t structure (handle) 20

3.2.16 Value Structure

4

5

6 7

The pmix_value_t structure is used to represent the value passed to PMIx_Put and retrieved by PMIx_Get, as well as many of the other PMIx functions.

A collection of values may be specified under a single key by passing a **pmix_value_t** containing an array of type **pmix_data_array_t**, with each array element containing its own object. All members shown below were introduced in version 1 of the standard unless otherwise marked.

```
PMIx v1.0
8
            typedef struct pmix value {
9
                pmix_data_type_t type;
10
                union {
                    bool flag;
11
12
                    uint8 t byte;
                    char *string;
13
14
                    size_t size;
15
                    pid_t pid;
                    int integer;
16
17
                    int8 t int8;
                    int16_t int16;
18
19
                    int32_t int32;
20
                    int64_t int64;
21
                    unsigned int uint;
22
                    uint8 t uint8;
23
                    uint16 t uint16;
24
                    uint32 t uint32;
25
                    uint64 t uint64;
26
                    float fval;
27
                    double dval:
28
                    struct timeval tv;
29
                                                      // version 2.0
                    time_t time;
                                                      // version 2.0
30
                    pmix_status_t status;
                    pmix_rank_t rank;
                                                     // version 2.0
31
32
                    pmix_proc_t *proc;
                                                     // version 2.0
33
                    pmix_byte_object_t bo;
                    pmix_persistence_t persist; // version 2.0
34
                                                     // version 2.0
35
                    pmix_scope_t scope;
36
                    pmix_data_range_t range;
                                                     // version 2.0
                                                     // version 2.0
37
                    pmix_proc_state_t state;
38
                    pmix_proc_info_t *pinfo;
                                                      // version 2.0
```

39

40

41

void *ptr;

pmix_data_array_t *darray;

pmix alloc directive t adir;

// version 2.0

// version 2.0

// version 2.0

```
1
                   } data;
2
              } pmix value t;
    3.2.17
              Value structure support macros
              The following macros are provided to support the pmix_value_t structure.
4
    3.2.17.1
               Initialize the pmix_value_t structure
              Initialize the pmix value t fields
6
   PMIx v1.0
 7
              PMIX VALUE CONSTRUCT (m)
              IN
8
                   Pointer to the structure to be initialized (pointer to pmix_value_t)
9
               Destruct the pmix_value_t structure
    3.2.17.2
10
              Destruct the pmix_value_t fields
11
   PMIx v1.0
12
              PMIX_VALUE_DESTRUCT (m)
              IN
13
14
                   Pointer to the structure to be destructed (pointer to pmix value t)
   3.2.17.3 Create a pmix_value_t array
15
16
              Allocate and initialize an array of pmix_value_t structures
   PMIx v1.0
17
              PMIX VALUE CREATE (m, n)
18
              INOUT m
                   Address where the pointer to the array of pmix_value_t structures shall be stored (handle)
19
20
              IN
                   Number of structures to be allocated (size_t)
21
```

```
3.2.17.4 Free a pmix_value_t
2
               Release a pmix_value_t structure
   PMIx v4.0
3
               PMIX VALUE RELEASE (m)
               IN
4
                    m
5
                    Pointer to a pmix_value_t structure (handle)
    3.2.17.5 Free a pmix_value_t array
7
               Release an array of pmix_value_t structures
   PMIx v1.0
8
               PMIX VALUE FREE (m, n)
               IN
9
                    Pointer to the array of pmix value t structures (handle)
10
11
               IN
                   Number of structures in the array (size_t)
12
    3.2.17.6 Load a value structure
13
               Summarv
14
               Load data into a pmix value t structure.
15
   PMIx v2.0
16
               PMIX VALUE LOAD (v, d, t);
               IN
17
                    The pmix_value_t into which the data is to be loaded (pointer to pmix_value_t)
18
               IN
19
20
                    Pointer to the data value to be loaded (handle)
               IN
21
22
                    Type of the provided data value ( pmix_data_type_t )
               Description
23
               This macro simplifies the loading of data into a pmix_value_t by correctly assigning values to
24
               the structure's fields.
25
                                                Advice to users -
               The data will be copied into the pmix_value_t - thus, any data stored in the source value can be
26
               modified or free'd without affecting the copied data once the macro has completed.
27
```

```
3.2.17.7 Unload a pmix_value_t structure
 2
               Summary
               Unload data from a pmix_value_t structure.
 3
   PMIx v2.2
               PMIX_VALUE_UNLOAD(r, v, d, t);
 4
               OUT r
 5
 6
                   Status code indicating result of the operation pmix status t
 7
               IN
 8
                   The pmix_value_t from which the data is to be unloaded (pointer to pmix_value_t)
               INOUT a
 9
10
                   Pointer to the location where the data value is to be returned (handle)
               INOUT t
11
                   Pointer to return the data type of the unloaded value (handle)
12
               Description
13
14
               This macro simplifies the unloading of data from a pmix_value_t.

 Advice to users

               Memory will be allocated and the data will be in the pmix_value_t returned - the source
15
               pmix_value_t will not be altered.
16
    3.2.17.8 Transfer data between pmix_value_t structures
               Summary
18
19
               Transfer the data value between two pmix value t structures.
   PMIx v2.0
               PMIX_VALUE_XFER(r, d, s);
20
               OUT r
21
                   Status code indicating success or failure of the transfer ( pmix_status_t )
22
               IN
23
24
                   Pointer to the pmix_value_t destination (handle)
              IN
25
26
                   Pointer to the pmix value t source (handle)
```

```
Description
 1
 2
               This macro simplifies the transfer of data between two pmix value t structures, ensuring that
               all fields are properly copied.
 3

    Advice to users

               The data will be copied into the destination pmix_value_t - thus, any data stored in the source
 4
               value can be modified or free'd without affecting the copied data once the macro has completed.
 5
    3.2.17.9 Retrieve a numerical value from a pmix value t
               Retrieve a numerical value from a pmix_value_t structure
   PMIx v3.0
 8
               PMIX VALUE GET NUMBER(s, m, n, t)
               OUT s
 9
                   Status code for the request ( pmix_status_t )
10
               IN
11
12
                   Pointer to the pmix value t structure (handle)
13
               OUT n
                   Variable to be set to the value (match expected type)
14
               IN
15
16
                   Type of number expected in m (pmix data type t)
17
               Sets the provided variable equal to the numerical value contained in the given pmix value t,
               returning success if the data type of the value matches the expected type and
18
19
               PMIX ERR BAD PARAM if it doesn't
    3.2.18 Info Structure
21
               The pmix info t structure defines a key/value pair with associated directive. All fields were
22
               defined in version 1.0 unless otherwise marked.
   PMIx v1.0
23
               typedef struct pmix info t {
24
                    pmix_key_t key;
                    pmix_info_directives_t flags; // version 2.0
25
                    pmix_value_t value;
26
               } pmix_info_t;
27
```

```
3.2.19
               Info structure support macros
              The following macros are provided to support the pmix_info_t structure.
 2
    3.2.19.1 Initialize the pmix_info_t structure
              Initialize the pmix info t fields
 4
   PMIx v1.0
 5
              PMIX INFO CONSTRUCT (m)
              IN
 6
                   m
 7
                   Pointer to the structure to be initialized (pointer to pmix_info_t)
    3.2.19.2 Destruct the pmix_info_t structure
              Destruct the pmix info t fields
 9
   PMIx v1.0
              PMIX INFO DESTRUCT (m)
10
11
              IN
                   Pointer to the structure to be destructed (pointer to pmix info t)
12
    3.2.19.3 Create a pmix_info_t array
13
14
              Allocate and initialize an array of pmix_info_t structures
   PMIx v1.0
15
              PMIX INFO CREATE (m, n)
              INOUT m
16
                   Address where the pointer to the array of pmix_info_t structures shall be stored (handle)
17
              IN
18
                   Number of structures to be allocated (size t)
19
    3.2.19.4 Free a pmix_info_t array
21
              Release an array of pmix_info_t structures
   PMIx v1.0
22
              PMIX INFO FREE (m, n)
23
              IN
24
                   Pointer to the array of pmix_info_t structures (handle)
              IN
25
                   Number of structures in the array (size t)
26
```

3.2.19.5 Load key and value data into a pmix_info_t PMIx v1.0 2 PMIX_INFO_LOAD(v, k, d, t); IN 3 4 Pointer to the pmix info t into which the key and data are to be loaded (pointer to pmix info t) 5 IN k 6 7 String key to be loaded - must be less than or equal to PMIX MAX KEYLEN in length (handle) 8 IN 9 d 10 Pointer to the data value to be loaded (handle) 11 IN Type of the provided data value (pmix_data_type_t) 12 This macro simplifies the loading of key and data into a **pmix_info_t** by correctly assigning 13 values to the structure's fields. 14 Advice to users -15 Both key and data will be copied into the **pmix_info_t** - thus, the key and any data stored in the 16 source value can be modified or free'd without affecting the copied data once the macro has 17 completed. 3.2.19.6 Copy data between pmix_info_t structures 18 19 Copy all data (including key, value, and directives) between two pmix info t structures. PMIx v2.0 20 PMIX INFO XFER(d, s); IN d 21 Pointer to the destination pmix_info_t (pointer to pmix_info_t) 22 23 IN Pointer to the source pmix_info_t (pointer to pmix_info_t) 24 25 This macro simplifies the transfer of data between two pmix_info_t structures. Advice to users 26 All data (including key, value, and directives) will be copied into the destination pmix info t thus, the source pmix_info_t may be free'd without affecting the copied data once the macro 27 has completed. 28

3.2.19.7 Test a boolean pmix_info_t 2 A special macro for checking if a boolean pmix_info_t is true PMIx v2.03 PMIX INFO TRUE (m) IN 4 m 5 Pointer to a pmix_info_t structure (handle) A pmix_info_t structure is considered to be of type PMIX_BOOL and value true if: 6 7 • the structure reports a type of **PMIX_UNDEF**, or 8 • the structure reports a type of **PMIX BOOL** and the data flag is **true** 3.2.20 Info Type Directives 10 *PMIx v2.0* The pmix info directives t structure is a uint32 t type that defines the behavior of command directives via pmix info t arrays. By default, the values in the pmix info t 11 array passed to a PMIx are optional. 12 Advice to users -A PMIx implementation or PMIx-enabled RM may ignore any pmix_info_t value passed to a 13 14 PMIx API if it is not explicitly marked as PMIX_INFO_REQD. This is because the values specified default to optional, meaning they can be ignored. This may lead to unexpected behavior if 15 16 the user is relying on the behavior specified by the **pmix info t** value. If the user relies on the 17 behavior defined by the pmix_info_t then they must set the PMIX_INFO_REQD flag using the PMIX INFO REQUIRED macro. 18 — Advice to PMIx library implementers ——— 19 The top 16-bits of the **pmix_info_directives_t** are reserved for internal use by PMIx 20 library implementers - the PMIx standard will not specify their intent, leaving them for customized use by implementers. Implementers are advised to use the provided PMIX_INFO_IS_REQUIRED 21 22 macro for testing this flag, and must return PMIX ERR NOT SUPPORTED as soon as possible to 23 the caller if the required behavior is not supported.

1 2		The following constants were introduced in version 2.0 (unless otherwise marked) and can be used to set a variable of the type pmix_info_directives_t .
3 4 5 6 7 8		<pre>PMIX_INFO_REQD</pre>
9 10 11		Host environments are advised to use the provided PMIX_INFO_IS_REQUIRED macro for testing this flag and must return PMIX_ERR_NOT_SUPPORTED as soon as possible to the caller if the required behavior is not supported.
12	3.2.21	Info Directive support macros
13 14	3.2.21.1	The following macros are provided to support the setting and testing of <pre>pmix_info_t</pre> directives. Mark an info structure as required
15 16		Summary Set the PMIX_INFO_REQD flag in a pmix_info_t structure.
17	PMIx v2.0	PMIX_INFO_REQUIRED(info);
18 19		<pre>IN info Pointer to the pmix_info_t (pointer to pmix_info_t)</pre>
20 21	3.2.21.2	This macro simplifies the setting of the PMIX_INFO_REQD flag in pmix_info_t structures. Mark an info structure as optional
22 23		Summary Unsets the PMIX_INFO_REQD flag in a pmix_info_t structure.
24	PMIx v2.0	PMIX_INFO_OPTIONAL(info);
25 26		<pre>IN info Pointer to the pmix_info_t (pointer to pmix_info_t)</pre>
27		This macro simplifies marking a pmix_info_t structure as <i>optional</i> .

3.2.21.3 Test an info structure for required directive 2 Summary Test the PMIX_INFO_REQD flag in a pmix_info_t structure, returning true if the flag is set. 3 PMIx v2.0PMIX INFO IS REQUIRED (info); 4 5 IN info Pointer to the pmix_info_t (pointer to pmix_info_t) 6 7 This macro simplifies the testing of the required flag in **pmix info** t structures. 3.2.21.4 Test an info structure for *optional* directive Summary 9 Test a pmix info t structure, returning true if the structure is optional. 10 PMIx v2.0PMIX INFO IS OPTIONAL (info); 11 IN info 12 Pointer to the pmix_info_t (pointer to pmix_info_t) 13 14 Test the PMIX_INFO_REQD flag in a pmix_info_t structure, returning true if the flag is not 15 3.2.21.5 Test an info structure for end of array directive 16 17 Summary 18 Test a pmix info t structure, returning true if the structure is at the end of an array created by the PMIX INFO CREATE macro. 19 PMIx v2.2PMIX_INFO_IS_END(info); 20 IN 21 info 22 Pointer to the pmix_info_t (pointer to pmix_info_t) 23 This macro simplifies the testing of the end-of-array flag in **pmix_info_t** structures.

1 3.2.22 Job Allocation Directives

- The pmix_alloc_directive_t structure is a uint8_t type that defines the behavior of 2 PMIx v2.03 allocation requests. The following constants can be used to set a variable of the type 4 pmix_alloc_directive_t . All definitions were introduced in version 2 of the standard unless otherwise marked. 5 6 PMIX ALLOC NEW A new allocation is being requested. The resulting allocation will be 7 disjoint (i.e., not connected in a job sense) from the requesting allocation. 8 Extend the existing allocation, either in time or as additional PMIX ALLOC EXTEND 9 resources. 10 PMIX ALLOC RELEASE Release part of the existing allocation. Attributes in the 11 accompanying pmix info t array may be used to specify permanent release of the 12 identified resources, or "lending" of those resources for some period of time. 13 PMIX ALLOC REAQUIRE Reacquire resources that were previously "lent" back to the
- 14 scheduler.

 15 PMTY ALLOG EVERDNAL A value boundary above which implementers are free to defi
- 15 **PMIX_ALLOC_EXTERNAL** A value boundary above which implementers are free to define their own directive values.

17 3.2.23 IO Forwarding Channels

- The **pmix_iof_channel_t** structure is a **uint16_t** type that defines a set of bit-mask flags for specifying IO forwarding channels. These can be bitwise OR'd together to reference multiple channels.
- 21 **PMIX_FWD_NO_CHANNELS** Forward no channels
 22 **PMIX_FWD_STDIN_CHANNEL** Forward stdin
- 23 PMIX_FWD_STDOUT_CHANNEL Forward stdout
- 24 **PMIX_FWD_STDERR_CHANNEL** Forward stderr
- 25 **PMIX_FWD_STDDIAG_CHANNEL** Forward stddiag, if available
- 26 PMIX_FWD_ALL_CHANNELS Forward all available channels

27 3.2.24 Environmental Variable Structure

28 *PMIx v3.0* Define a structure for specifying environment variable modifications. Standard environment variables (e.g., **PATH**, **LD_LIBRARY_PATH**, and **LD_PRELOAD**) take multiple arguments separated by delimiters. Unfortunately, the delimiters depend upon the variable itself - some use semi-colons, some colons, etc. Thus, the operation requires not only the name of the variable to be modified and the value to be inserted, but also the separator to be used when composing the aggregate value.

```
1
              typedef struct
 2
                   char *envar;
 3
                   char *value;
 4
                   char separator;
 5
               pmix_envar_t;
   3.2.25
              Environmental variable support macros
7
              The following macros are provided to support the pmix_envar_t structure.
    3.2.25.1
               Initialize the pmix_envar_t structure
9
              Initialize the pmix_envar_t fields
   PMIx v3.0
10
              PMIX_ENVAR_CONSTRUCT (m)
11
              IN
12
                  Pointer to the structure to be initialized (pointer to pmix_envar_t)
               Destruct the pmix_envar_t structure
    3.2.25.2
13
14
              Clear the pmix envar t fields
   PMIx v3.0
15
              PMIX ENVAR DESTRUCT (m)
              IN
16
17
                  Pointer to the structure to be destructed (pointer to pmix_envar_t)
   3.2.25.3 Create a pmix_envar_t array
18
19
              Allocate and initialize an array of pmix_envar_t structures
   PMIx v3.0
20
              PMIX_ENVAR_CREATE(m, n)
              INOUT m
21
22
                   Address where the pointer to the array of pmix_envar_t structures shall be stored (handle)
              IN
23
24
                  Number of structures to be allocated (size t)
```

```
3.2.25.4 Free a pmix_envar_t array
2
              Release an array of pmix_envar_t structures
   PMIx v3.0
 3
              PMIX_ENVAR_FREE(m, n)
              IN
4
5
                  Pointer to the array of pmix_envar_t structures (handle)
6
              IN
7
                  Number of structures in the array (size_t)
   3.2.25.5 Load a pmix_envar_t structure
9
              Load values into a pmix_envar_t
   PMIx v2.0
10
              PMIX_ENVAR_LOAD(m, e, v, s)
11
              IN
12
                  Pointer to the structure to be loaded (pointer to pmix envar t)
              IN
13
14
                  Environmental variable name (char*)
              IN
15
                  Value of variable (char⋆)
16
17
              IN
                  Separator character (char)
18
   3.2.26
             Lookup Returned Data Structure
20
              The pmix_pdata_t structure is used by PMIx_Lookup to describe the data being accessed.
   PMIx v1.0
21
              typedef struct pmix_pdata {
22
                  pmix_proc_t proc;
23
                  pmix_key_t key;
                  pmix_value_t value;
24
25
              } pmix pdata t;
```

6 3.2.27 Lookup data structure support macros

The following macros are provided to support the **pmix pdata t** structure.



```
3.2.27.5 Free a pmix_pdata_t array
 2
               Release an array of pmix_pdata_t structures
   PMIx v1.0
 3
               PMIX PDATA FREE (m, n)
               IN
 4
 5
                    Pointer to the array of pmix_pdata_t structures (handle)
 6
               IN
 7
                    Number of structures in the array (size_t)
    3.2.27.6 Load a lookup data structure
               Summary
 9
               Load key, process identifier, and data value into a pmix_pdata_t structure.
10
   PMIx v1.0
               PMIX_PDATA_LOAD(m, p, k, d, t);
11
               IN
12
                    Pointer to the pmix pdata t structure into which the key and data are to be loaded
13
                    (pointer to pmix_pdata_t)
14
               IN
15
                    Pointer to the pmix proc t structure containing the identifier of the process being
16
17
                    referenced (pointer to pmix proc t)
               IN
18
                    String key to be loaded - must be less than or equal to PMIX MAX KEYLEN in length
19
20
                    (handle)
21
               IN
                    d
22
                    Pointer to the data value to be loaded (handle)
               IN
23
                    Type of the provided data value (pmix data type t)
24
               This macro simplifies the loading of key, process identifier, and data into a pmix_proc_t by
25
26
               correctly assigning values to the structure's fields.
                                        Advice to users -
27
               Key, process identifier, and data will all be copied into the pmix_pdata_t - thus, the source
               information can be modified or free'd without affecting the copied data once the macro has
28
29
               completed.
```

3.2.27.7 Transfer a lookup data structure

```
2
             Summary
             Transfer key, process identifier, and data value between two pmix_pdata_t structures.
 3
   PMIx v2.0
             PMIX PDATA XFER(d, s);
4
5
             IN
6
                  Pointer to the destination pmix_pdata_t (pointer to pmix_pdata_t)
             IN
 7
                  Pointer to the source pmix_pdata_t (pointer to pmix_pdata_t)
9
             This macro simplifies the transfer of key and data between two pmix_pdata_t structures.
                                Advice to users -
             Key, process identifier, and data will all be copied into the destination pmix_pdata_t - thus, the
10
11
             source pmix pdata t may free'd without affecting the copied data once the macro has
12
             completed.
   3.2.28 Application Structure
             The pmix app t structure describes the application context for the PMIx Spawn and
14
15
             PMIx Spawn nb operations.
   PMIx v1.0
16
             typedef struct pmix_app {
17
                  /** Executable */
18
                  char *cmd;
                  /** Argument set, NULL terminated */
19
20
                  char **argv;
                  /** Environment set, NULL terminated */
21
22
                  char **env;
                  /** Current working directory */
23
24
                  char *cwd;
                  /** Maximum processes with this profile */
25
                  int maxprocs;
26
                  /** Array of info keys describing this application*/
27
                  pmix_info_t *info;
28
                  /** Number of info keys in 'info' array */
29
                  size t ninfo;
30
31
              } pmix app t;
```

```
3.2.29 App structure support macros
              The following macros are provided to support the pmix app t structure.
 2
    3.2.29.1 Initialize the pmix_app_t structure
              Initialize the pmix_app_t fields
   PMIx v1.0
 5
              PMIX APP CONSTRUCT (m)
              IN
                   m
 6
 7
                   Pointer to the structure to be initialized (pointer to pmix_app_t)
    3.2.29.2 Destruct the pmix_app_t structure
 9
              Destruct the pmix app t fields
   PMIx v1.0
10
              PMIX APP DESTRUCT (m)
              IN
11
12
                   Pointer to the structure to be destructed (pointer to pmix_app_t)
    3.2.29.3 Create a pmix_app_t array
13
14
              Allocate and initialize an array of pmix_app_t structures
   PMIx v1.0
15
              PMIX APP CREATE (m, n)
              INOUT m
16
                   Address where the pointer to the array of pmix_app_t structures shall be stored (handle)
17
18
                   Number of structures to be allocated (size t)
19
    3.2.29.4 Free a pmix_app_t
20
21
              Release a pmix app t structure
   PMIx v4.0
22
              PMIX APP RELEASE (m)
              IN
23
                   Pointer to a pmix_app_t structure (handle)
24
```

```
3.2.29.5 Free a pmix_app_t array
2
               Release an array of pmix_app_t structures
   PMIx v1.0
 3
               PMIX_APP_FREE(m, n)
               IN
 4
5
                   Pointer to the array of pmix_app_t structures (handle)
6
               IN
7
                   Number of structures in the array (size_t)
               Create the pmix_info_t array of application directives
    3.2.29.6
               Create an array of pmix_info_t structures for passing application-level directives, updating the
9
               ninfo field of the pmix_app_t structure.
10
   PMIx v2.2
               PMIX_APP_INFO_CREATE(m, n)
11
               IN
12
13
                   Pointer to the pmix app t structure (handle)
14
               IN
15
                   Number of directives to be allocated (size t)
    3.2.29.7 Create the pmix_info_t array of application directives
16
               Create an array of pmix info t structures for passing application-level directives, updating the
17
               ninfo field of the pmix_app_t structure.
18
   PMIx v2.2
19
               PMIX APP INFO CREATE (m, n)
               IN
20
21
                   Pointer to the pmix_app_t structure (handle)
22
               IN
23
                   Number of directives to be allocated (size t)
```

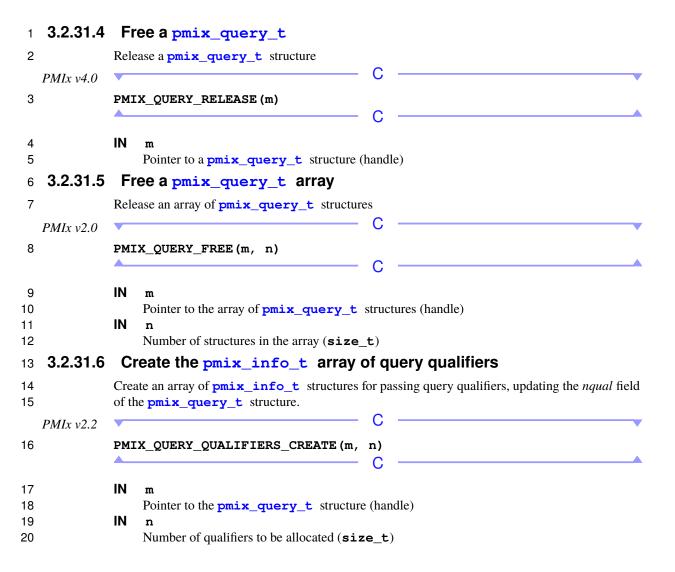
3.2.30 Query Structure 2 The pmix_query_t structure is used by PMIx_Query_info_nb to describe a single query 3 operation. PMIx v2.0 typedef struct pmix_query { 4 5 char **keys; 6 pmix_info_t *qualifiers; 7 size_t nqual; 8 } pmix query t; 3.2.31 Query structure support macros The following macros are provided to support the **pmix_query_t** structure. 10 Initialize the pmix_query_t structure 3.2.31.1 12 Initialize the **pmix query t** fields PMIx v2.0 PMIX QUERY CONSTRUCT (m) 13 IN 14 15 Pointer to the structure to be initialized (pointer to **pmix_query_t**) 3.2.31.2 Destruct the pmix query t structure 16 Destruct the **pmix_query_t** fields 17 PMIx v2.018 PMIX_QUERY_DESTRUCT (m) IN 19 20 Pointer to the structure to be destructed (pointer to **pmix_query_t**) 3.2.31.3 Create a pmix_query_t array 21 22 Allocate and initialize an array of **pmix_query_t** structures PMIx v2.0PMIX QUERY CREATE (m, n) 23 INOUT m 24

Address where the pointer to the array of **pmix_query_t** structures shall be stored (handle)

Number of structures to be allocated (size t)

IN

25



3.2.32 Attribute registration structure

2 The pmix_regattr_t structure is used to register attribute support for a PMIx function.

```
PMIx v4.0
3
            typedef struct pmix regattr {
4
                 char *name;
5
                pmix_key_t *string;
6
                pmix_data_type_t type;
7
                pmix_info_t *info;
8
                 size t ninfo;
9
                 char **description;
            } pmix regattr t;;
10
```

Note that in this structure:

- the name is the actual name of the attribute e.g., "PMIX_MAX_PROCS"; and
- the *string* is the literal string value of the attribute e.g., "pmix.max.size" for the **PMIX_MAX_PROCS** attribute
- type must be a PMIx data type identifying the type of data associated with this attribute.
- the *info* array contains machine-usable information regarding the range of accepted values. This
 may include entries for PMIX_MIN_VALUE, PMIX_MAX_VALUE, PMIX_ENUM_VALUE, or
 a combination of them. For example, an attribute that supports all positive integers might
 delineate it by including a pmix_info_t with a key of PMIX_MIN_VALUE, type of
 PMIX_INT, and value of zero. The lack of an entry for PMIX_MAX_VALUE indicates that
 there is no ceiling to the range of accepted values.
- *ninfo* indicates the number of elements in the *info* array
- The description field consists of a NULL-terminated array of strings describing the attribute, optionally including a human-readable description of the range of accepted values e.g., "ALL POSITIVE INTEGERS", or a comma-delimited list of enum value names. No correlation between the number of entries in the description and the number of elements in the info array is implied or required.

The attribute *name* and *string* fields must be **NULL**-terminated strings composed of standard alphanumeric values supported by common utilities such as *strcmp*.

——— Advice to PMIx library implementers —————

Although not strictly required, PMIx library implementers are strongly encouraged to provide both human-readable and machine-parsable descriptions of supported attributes.

11 12

13

14 15

16

17

18

19

20 21

22

23

24

25 26

27 28

29

30

1 2		Although not strictly required, host environments are strongly encouraged to provide both human-readable and machine-parsable descriptions of supported attributes when registering them.
3	3.2.33	Attribute registration structure support macros
4		The following macros are provided to support the pmix_regattr_t structure.
5	3.2.33.1	Initialize the pmix_regattr_t structure
6	PMIx v4.0	Initialize the pmix_regattr_t fields C
7		PMIX_REGATTR_CONSTRUCT (m)
8		IN m
9		Pointer to the structure to be initialized (pointer to pmix_regattr_t)
10	3.2.33.2	Destruct the pmix_regattr_t structure
11		Destruct the <pre>pmix_regattr_t</pre> fields, releasing all strings.
	PMIx v4.0	C
12		PMIX_REGATTR_DESTRUCT (m)
13 14		IN m Pointer to the structure to be destructed (pointer to pmix_regattr_t)
15	3.2.33.3	Create a pmix_regattr_t array
16		Allocate and initialize an array of pmix_regattr_t structures
	PMIx v4.0	C
17		PMIX_REGATTR_CREATE (m, n)
18 19 20		INOUT m Address where the pointer to the array of pmix_regattr_t structures shall be stored (handle)
21		IN n
22		Number of structures to be allocated (size_t)

——— Advice to PMIx server hosts ————

```
3.2.33.4 Free a pmix_regattr_t array
2
               Release an array of pmix_regattr_t structures
   PMIx v4.0
 3
               PMIX REGATTR FREE (m, n)
               INOUT m
4
5
                    Pointer to the array of pmix_regattr_t structures (handle)
6
               IN
 7
                    Number of structures in the array (size t)
    3.2.33.5
8
               Load a pmix_regattr_t structure
               Load values into a pmix regattr t structure. The macro can be called multiple times to add
9
               as many strings as desired to the same structure by passing the same address and a NULL key to the
10
               macro. Note that the t type value must be given each time.
11
   PMIx v4.0
12
               PMIX REGATTR LOAD (a, n, k, t, ni, v)
               IN
13
                    Pointer to the structure to be loaded (pointer to pmix_proc_t)
14
               IN
15
16
                    String name of the attribute (string)
               IN
17
                    Key value to be loaded ( pmix_key_t )
18
               IN
19
20
                    Type of data associated with the provided key ( pmix_data_type_t )
21
               IN
                   Number of pmix_info_t elements to be allocated in info(size_t)
22
               IN
23
                    One-line description to be loaded (more can be added separately) (string)
24
    3.2.33.6
                Transfer a pmix_regattr_t to another pmix_regattr_t
25
26
27
               Non-destructively transfer the contents of a pmix regattr t structure to another one.
   PMIx v4.0
               PMIX REGATTR XFER(m, n)
28
               INOUT m
29
                    Pointer to the destination pmix_regattr_t structure (handle)
30
               IN
31
                    Pointer to the source pmix_regattr_t structure (handle)
32
```

```
PMIx Group Directives
   3.2.34
              The pmix_group_opt_t type is an enumerated type used with the PMIx_Group_join API
  PMIx v4.0
3
              to indicate accept or decline of the invitation - these are provided for readability of user code:
              PMIX_GROUP_DECLINE
                                      Decline the invitation
 4
 5
              PMIX_GROUP_ACCEPT
                                     Accept the invitation.
            Byte Object Type
   3.2.35
7
              The pmix_byte_object_t structure describes a raw byte sequence.
   PMIx v1.0
              typedef struct pmix_byte_object {
8
9
                  char *bytes;
                  size_t size;
10
11
              } pmix_byte_object_t;
   3.2.36
              Byte object support macros
              The following macros support the pmix_byte_object_t structure.
13
   3.2.36.1
              Initialize the pmix_byte_object_t structure
15
              Initialize the pmix byte object t fields
   PMIx v2.0
16
              PMIX BYTE OBJECT CONSTRUCT (m)
17
              IN
                  Pointer to the structure to be initialized (pointer to pmix_byte_object_t)
18
   3.2.36.2
              Destruct the pmix_byte_object_t structure
19
20
              Clear the pmix byte object t fields
   PMIx v2.0
21
              PMIX BYTE OBJECT DESTRUCT (m)
```

Pointer to the structure to be destructed (pointer to pmix_byte_object_t)

IN

```
3.2.36.3 Create a pmix_byte_object_t structure
2
              Allocate and intitialize an array of pmix_byte_object_t structures
                                                      C
   PMIx v2.0
3
              PMIX_BYTE_OBJECT_CREATE(m, n)
              INOUT m
4
5
                   Address where the pointer to the array of pmix_byte_object_t structures shall be
6
                   stored (handle)
7
              IN
                   n
                   Number of structures to be allocated (size_t)
8
   3.2.36.4 Free a pmix_byte_object_t array
10
              Release an array of pmix_byte_object_t structures
   PMIx v2.0
              PMIX_BYTE_OBJECT_FREE(m, n)
11
              IN
12
                   Pointer to the array of pmix byte object t structures (handle)
13
14
              IN
                   Number of structures in the array (size_t)
15
   3.2.36.5 Load a pmix_byte_object_t structure
16
              Load values into a pmix_byte_object_t
17
   PMIx v2.0
              PMIX BYTE OBJECT LOAD (b, d, s)
18
              IN
19
20
                   Pointer to the structure to be loaded (pointer to pmix_byte_object_t)
              IN
21
                   Pointer to the data to be loaded (char*)
22
              IN
23
                   Number of bytes in the data array (size t)
24
```

3.2.37 Data Array Structure

23

```
2
              The pmix_data_array_t structure defines an array data structure.
   PMIx v2.0
 3
              typedef struct pmix_data_array {
 4
                   pmix_data_type_t type;
 5
                   size_t size;
                   void *array;
 6
 7
              } pmix_data_array_t;
    3.2.38
              Data array support macros
              The following macros support the pmix_data_array_t structure.
 9
    3.2.38.1
               Initialize a pmix_data_array_t structure
              Initialize the pmix data array t fields, allocating memory for the array of the indicated type.
11
                                                      C
   PMIx v2.2
              PMIX DATA ARRAY_CONSTRUCT(m, n, t)
12
              IN
13
                  Pointer to the structure to be initialized (pointer to pmix_data_array_t)
14
              IN
15
16
                  Number of elements in the array (size_t)
17
              IN
                  PMIx data type of the array elements ( pmix_data_type_t )
18
    3.2.38.2
               Destruct a pmix_data_array_t structure
19
20
              Destruct the pmix data array t, releasing the memory in the array.
   PMIx v2.2
21
              PMIX DATA ARRAY CONSTRUCT (m)
22
              IN
```

Pointer to the structure to be destructed (pointer to pmix_data_array_t)

```
3.2.38.3 Create a pmix_data_array_t structure
               Allocate memory for the pmix_data_array_t object itself, and then allocate memory for the
2
 3
               array of the indicated type.
   PMIx v2.2
               PMIX DATA ARRAY CREATE (m, n, t)
 4
5
               INOUT m
6
                   Variable to be set to the address of the structure (pointer to pmix_data_array_t)
7
               IN
8
                   Number of elements in the array (size_t)
               IN
9
                   PMIx data type of the array elements ( pmix_data_type_t )
10
    3.2.38.4 Free a pmix_data_array_t structure
11
               Release the memory in the array, and then release the pmix data array_t object itself.
12
   PMIx v2.2
13
               PMIX_DATA_ARRAY_FREE (m)
14
               IN
                   Pointer to the structure to be released (pointer to pmix data array t)
15
    3.2.39 Argument Array Macros
17
               The following macros support the construction and release of NULL-terminated argy arrays of
               strings.
18
    3.2.39.1 Argument array extension
19
20
               Summary
               Append a string to a NULL-terminated, argy-style array of strings.
21
22
               PMIX ARGV APPEND(r, a, b);
               OUT r
23
24
                   Status code indicating success or failure of the operation ( pmix_status_t )
25
               INOUT a
                   Argument list (pointer to NULL-terminated array of strings)
26
27
               IN
                   Argument to append to the list (string)
28
```

Description 1 2 This function helps the caller build the **argy** portion of **pmix app t** structure, arrays of keys for querying, or other places where argy-style string arrays are required in the way that the PRI 3 4 expects it to be constructed. Advice to users 5 The provided argument is copied into the destination array - thus, the source string can be free'd without affecting the array once the macro has completed. 6 3.2.39.2 Argument array prepend Summary 8 Prepend a string to a NULL-terminated, argy-style array of strings. 9 10 PMIX ARGV PREPEND(r, a, b); OUT r 11 12 Status code indicating success or failure of the operation (pmix_status_t) INOUT a 13 14 Argument list (pointer to NULL-terminated array of strings) IN 15 16 Argument to append to the list (string) **Description** 17 This function helps the caller build the argy portion of pmix app_t structure, arrays of keys 18 for querying, or other places where argy-style string arrays are required in the way that the PRI 19 20 expects it to be constructed. — Advice to users — The provided argument is copied into the destination array - thus, the source string can be free'd 21 without affecting the array once the macro has completed. 22

3.2.39.3 Argument array extension - unique 1 2 Summary Append a string to a NULL-terminated, argy-style array of strings, but only if the provided 3 argument doesn't already exist somewhere in the array. 4 PMIX ARGV_APPEND_UNIQUE(r, a, b); 5 OUT r 6 7 Status code indicating success or failure of the operation (pmix status t) INOUT a 8 9 Argument list (pointer to NULL-terminated array of strings) 10 IN 11 Argument to append to the list (string) **Description** 12 13 This function helps the caller build the **argy** portion of **pmix app t** structure, arrays of keys for querying, or other places where argy-style string arrays are required in the way that the PRI 14 15 expects it to be constructed. Advice to users 16 The provided argument is copied into the destination array - thus, the source string can be free'd without affecting the array once the macro has completed. 17 3.2.39.4 Argument array release Summary 19 Free an argy-style array and all of the strings that it contains 20 21 PMIX ARGV FREE(a); IN 22 Argument list (pointer to NULL-terminated array of strings) 23 Description 24 This function releases the array and all of the strings it contains. 25

3.2.39.5 Argument array split 1 2 Summary Split a string into a NULL-terminated argy array. 3 PMIX_ARGV_SPLIT(a, b, c); 4 OUT a 5 Resulting argv-style array (char**) 6 7 IN String to be split (char*) 8 IN 9 Delimiter character (char) 10 Description 11 12 Split an input string into a NULL-terminated argy array. Do not include empty strings in the 13 resulting array. Advice to users -All strings are inserted into the argy array by value; the newly-allocated array makes no references 14 to the src_string argument (i.e., it can be freed after calling this function without invalidating the 15 16 output argy array) 3.2.39.6 Argument array join 17 Summary 18 Join all the elements of an argy array into a single newly-allocated string. 19 20 PMIX ARGV JOIN(a, b, c); 21 OUT a Resulting string (char*) 22 IN 23 Argy-style array to be joined (**char****) 24 25 IN 26 Delimiter character (char) Description 27 Join all the elements of an argv array into a single newly-allocated string. 28

3.2.39.7 Argument array count 2 Summary Return the length of a NULL-terminated argy array. 3 4 PMIX_ARGV_COUNT(r, a); 5 OUT r 6 Number of strings in the array (integer) 7 IN Argy-style array (char**) **Description** 9 Count the number of elements in an argv array 10 3.2.39.8 Argument array copy 11 Summary 12 Copy an argy array, including copying all off its strings. 13 14 PMIX_ARGV_COPY(a, b); OUT a 15 New argy-style array (char**) 16 IN 17 18 Argv-style array (char**) Description 19 Copy an argy array, including copying all off its strings. 20 3.2.40 Set Environment Variable 22 Summary 23 Set an environment variable in a **NULL**-terminated, env-style array

1 PMIX SETENV(r, name, value, env); OUT r 2 Status code indicating success or failure of the operation (pmix_status_t) 3 IN 4 5 Argument name (string) value 6 IN 7 Argument value (string) INOUT env 8 9 Environment array to update (pointer to array of strings) **Description** 10 11 Similar to **setenv** from the C API, this allows the caller to set an environment variable in the 12 specified env array, which could then be passed to the pmix app t structure or any other destination. 13 Advice to users 14 The provided name and value are copied into the destination environment array - thus, the source strings can be free'd without affecting the array once the macro has completed. 15

3.3 Generalized Data Types Used for Packing/Unpacking

17

18 19

20 21

22

23 24 The **pmix_data_type_t** structure is a **uint16_t** type for identifying the data type for packing/unpacking purposes. New data type values introduced in this version of the Standard are shown in **magenta**.

— Advice to PMIx library implementers -

The following constants can be used to set a variable of the type <code>pmix_data_type_t</code>. Data types in the PMIx Standard are defined in terms of the C-programming language. Implementers wishing to support other languages should provide the equivalent definitions in a language-appropriate manner. Additionally, a PMIx implementation may choose to add additional types.

```
Undefined
 1
              PMIX_UNDEF
 2
                             Boolean (converted to/from native true/false) (bool)
              PMIX_BOOL
 3
                             A byte of data (uint8_t)
              PMIX_BYTE
 4
                               NULL terminated string (char*)
              PMIX STRING
5
              PMIX SIZE
                             Size size t
6
              PMIX PID
                           Operating process identifier (PID) (pid t)
7
                            Integer (int)
              PMIX INT
8
                             8-byte integer (int8_t)
              PMIX_INT8
                              16-byte integer (int16_t)
9
              PMIX_INT16
10
              PMIX INT32
                              32-byte integer (int32_t)
                              64-byte integer (int64_t)
11
              PMIX INT64
12
                             Unsigned integer (unsigned int)
              PMIX UINT
13
              PMIX UINT8
                              Unsigned 8-byte integer (uint8 t)
14
                               Unsigned 16-byte integer (uint16 t)
              PMIX UINT16
15
                               Unsigned 32-byte integer (uint32_t)
              PMIX UINT32
                               Unsigned 64-byte integer (uint64_t)
16
              PMIX_UINT64
17
                              Float (float)
              PMIX FLOAT
18
              PMIX_DOUBLE
                               Double (double)
19
                                Time value (struct timeval)
              PMIX TIMEVAL
20
              PMIX TIME
                             Time (time t)
                               Status code pmix status t
21
              PMIX STATUS
22
                              Value ( pmix_value_t )
              PMIX_VALUE
23
                             Process ( pmix_proc_t )
              PMIX_PROC
24
                            Application context
              PMIX_APP
25
              PMIX INFO
                             Info object
                              Pointer to data
26
              PMIX PDATA
27
              PMIX BUFFER
                               Buffer
              PMIX_BYTE_OBJECT
                                     Byte object ( pmix_byte_object_t )
28
29
                             Key/value pair
              PMIX_KVAL
30
                                Persistance (pmix_persistence_t)
              PMIX_PERSIST
31
              PMIX_POINTER
                                 Pointer to an object (void*)
32
                              Scope ( pmix_scope_t )
              PMIX SCOPE
                                    Range for data ( pmix_data_range_t )
33
              PMIX DATA RANGE
                                PMIx command code (used internally)
34
              PMIX_COMMAND
35
              PMIX INFO DIRECTIVES
                                          Directives flag for pmix info t (
36
                  pmix info directives t)
37
              PMIX DATA TYPE
                                   Data type code ( pmix_data_type_t )
                                    Process state ( pmix_proc_state_t )
38
              PMIX_PROC_STATE
39
                                   Process information ( pmix_proc_info_t )
              PMIX PROC INFO
40
              PMIX_DATA_ARRAY
                                    Data array ( pmix_data_array_t )
41
                                   Process rank ( pmix_rank_t )
              PMIX_PROC_RANK
```

```
1
                                Query structure ( pmix_query_t )
               PMIX_QUERY
2
               PMIX COMPRESSED STRING
                                                String compressed with zlib (char*)
                                             Allocation directive ( pmix_alloc_directive_t )
 3
               PMIX ALLOC DIRECTIVE
 4
               PMIX_IOF_CHANNEL
                                        Input/output forwarding channel ( pmix_iof_channel_t )
 5
                                Environmental variable structure (pmix envar t)
               PMIX ENVAR
6
               PMIX COORD
                                Structure containing fabric coordinates (pmix coord t)
7
                                   Structure supporting attribute registrations (pmix regattr t)
               PMIX REGATTR
8
               PMIX REGEX
                                Regular expressions - can be a valid NULL-terminated string or an arbitrary
9
                    array of bytes
10
               PMIX JOB STATE
                                     Job state ( pmix_job_state_t )
11
               PMIX LINK STATE
                                      Link state ( pmix_link_state_t )
12
               PMIX DATA TYPE MAX
                                           A starting point for implementer-specific data types. Values above
                    this are guaranteed not to conflict with PMIx values. Definitions should always be based on
13
14
                    the PMIX_DATA_TYPE_MAX constant and not a specific value as the value of the constant
15
                    may change.
```

3.4 Reserved attributes

17

18

19 20

21

22

23

24

25

26

27

28 29

30

31

33

34

35

The PMIx standard defines a relatively small set of APIs and the caller may customize the behavior of the API by passing one or more attributes to that API. Additionally, attributes may be keys passed to PMIx_Get calls to access the specified values from the system.

Each attribute is represented by a *key* string, and a type for the associated *value*. This section defines a set of **reserved** keys which are prefixed with **pmix**. to designate them as PMIx standard reserved keys. All definitions were introduced in version 1 of the standard unless otherwise marked.

Applications or associated libraries (e.g., MPI) may choose to define additional attributes. The attributes defined in this section are of the system and job as opposed to the attributes that the application (or associated libraries) might choose to expose. Due to this extensibility the **PMIX Get** API will return **PMIX ERR NOT FOUND** if the provided *key* cannot be found.

Attributes added in this version of the standard are shown in *magenta* to distinguish them from those defined in prior versions, which are shown in *black*. Deprecated attributes are shown in *green* and will be removed in future versions of the standard.

PMIX_ATTR_UNDEF NULL (NULL)

Constant representing an undefined attribute.

32 3.4.1 Initialization attributes

These attributes are defined to assist the caller with initialization by passing them into the appropriate initialization API - thus, they are not typically accessed via the **PMIx Get** API.

```
PMIX_EVENT_BASE "pmix.evbase" (struct event_base *)
```

```
Pointer to libevent event_base to use in place of the internal progress thread.
 1
2
               PMIX_SERVER_TOOL_SUPPORT "pmix.srvr.tool" (bool)
3
                     The host RM wants to declare itself as willing to accept tool connection requests.
 4
               PMIX_SERVER_REMOTE_CONNECTIONS "pmix.srvr.remote" (bool)
5
                     Allow connections from remote tools. Forces the PMIx server to not exclusively use
6
                     loopback device.
7
               PMIX SERVER SYSTEM SUPPORT "pmix.srvr.sys" (bool)
                     The host RM wants to declare itself as being the local system server for PMIx connection
8
9
                     requests.
               PMIX SERVER SESSION SUPPORT "pmix.srvr.sess" (bool)
10
                     The host RM wants to declare itself as being the local session server for PMIx connection
11
12
                     requests.
               PMIX_SERVER_START_TIME "pmix.srvr.strtime" (char*)
13
                     Time when the server started - i.e., when the server created it's rendezvous file (given in
14
                     ctime string format)
15
               PMIX_SERVER_TMPDIR "pmix.srvr.tmpdir" (char*)
16
                     Top-level temporary directory for all client processes connected to this server, and where the
17
18
                     PMIx server will place its tool rendezvous point and contact information.
19
               PMIX SYSTEM TMPDIR "pmix.sys.tmpdir" (char*)
20
                     Temporary directory for this system, and where a PMIx server that declares itself to be a
21
                     system-level server will place a tool rendezvous point and contact information.
22
               PMIX SERVER ENABLE MONITORING "pmix.srv.monitor" (bool)
23
                     Enable PMIx internal monitoring by the PMIx server.
               PMIX_SERVER_NSPACE "pmix.srv.nspace" (char*)
24
                     Name of the namespace to use for this PMIx server.
25
               PMIX_SERVER_RANK "pmix.srv.rank" (pmix_rank_t)
26
27
                     Rank of this PMIx server
               PMIX_SERVER_GATEWAY "pmix.srv.gway" (bool)
28
29
                     Server is acting as a gateway for PMIx requests that cannot be serviced on backend nodes
30
                     (e.g., logging to email)
    3.4.2 Identification attributes
               These attributes are defined to identify a process and it's associated PMIx-enabled library. They are
32
               not typically accessed via the PMIx Get API, and thus are not associated with a particular rank.
33
34
               PMIX USERID "pmix.euid" (uint32 t)
35
                     Effective user id.
               PMIX_GRPID "pmix.egid" (uint32_t)
36
37
                     Effective group id.
               PMIX_DSTPATH "pmix.dstpath" (char*)
38
                     Path to shared memory data storage (dstore) files.
39
40
               PMIX VERSION INFO "pmix.version" (char*)
                <sup>1</sup>http://libevent.org/
```

```
1
                    PMIx version of contractor.
2
              PMIX_REQUESTOR_IS_TOOL "pmix.req.tool" (bool)
 3
                    The requesting process is a PMIx tool.
4
              PMIX_REQUESTOR_IS_CLIENT "pmix.req.client" (bool)
5
                    The requesting process is a PMIx client.
6
              PMIX PSET NAME "pmix.pset.nm" (char*)
7
                    User-assigned name for the process set containing the given process.
8
              PMIX_REINCARNATION "pmix.reinc" (uint32_t)
                    Number of times this process has been re-instantiated - i.e, a value of zero indicates that the
9
10
                    process has never failed and been restarted.
    3.4.3 Programming model attributes
11
12
              These attributes are associated with programming models.
13
              PMIX PROGRAMMING MODEL "pmix.pqm.model" (char*)
                    Programming model being initialized (e.g., "MPI" or "OpenMP")
14
15
              PMIX MODEL LIBRARY NAME "pmix.mdl.name" (char*)
                    Programming model implementation ID (e.g., "OpenMPI" or "MPICH")
16
17
              PMIX_MODEL_LIBRARY_VERSION "pmix.mld.vrs" (char*)
                    Programming model version string (e.g., "2.1.1")
18
              PMIX_THREADING_MODEL "pmix.threads" (char*)
19
                    Threading model used (e.g., "pthreads")
20
21
              PMIX MODEL NUM THREADS "pmix.mdl.nthrds" (uint64 t)
22
                    Number of active threads being used by the model
23
              PMIX_MODEL_NUM_CPUS "pmix.mdl.ncpu" (uint64_t)
                    Number of cpus being used by the model
24
              PMIX_MODEL_CPU_TYPE "pmix.mdl.cputype" (char*)
25
26
                    Granularity - "hwthread", "core", etc.
              PMIX_MODEL_PHASE_NAME "pmix.mdl.phase" (char*)
27
28
                    User-assigned name for a phase in the application execution (e.g., "cfd reduction")
              PMIX MODEL PHASE TYPE "pmix.mdl.ptype" (char*)
29
30
                    Type of phase being executed (e.g., "matrix multiply")
              PMIX_MODEL_AFFINITY_POLICY "pmix.mdl.tap" (char*)
31
                    Thread affinity policy - e.g.: "master" (thread co-located with master thread), "close" (thread
32
33
                    located on cpu close to master thread), "spread" (threads load-balanced across available cpus)
   3.4.4 UNIX socket rendezvous socket attributes
35
              These attributes are used to describe a UNIX socket for rendezvous with the local RM by passing
              them into the relevant initialization API - thus, they are not typically accessed via the PMIx Get
36
37
              API.
38
              PMIX USOCK DISABLE "pmix.usock.disable" (bool)
39
                    Disable legacy UNIX socket (usock) support
```

```
1
               PMIX_SOCKET_MODE "pmix.sockmode" (uint32_t)
                    POSIX mode_t (9 bits valid)
2
3
               PMIX SINGLE LISTENER "pmix.sing.listnr" (bool)
4
                    Use only one rendezvous socket, letting priorities and/or environment parameters select the
5
                    active transport.
    3.4.5 TCP connection attributes
 7
               These attributes are used to describe a TCP socket for rendezvous with the local RM by passing
               them into the relevant initialization API - thus, they are not typically accessed via the PMIx_Get
8
9
               API.
10
               PMIX_TCP_REPORT_URI "pmix.tcp.repuri" (char*)
11
                    If provided, directs that the TCP uniform resource identifier (URI) be reported and indicates
                    the desired method of reporting: '-' for stdout, '+' for stderr, or filename.
12
               PMIX TCP URI "pmix.tcp.uri" (char*)
13
14
                    The URI of the PMIx server to connect to, or a file name containing it in the form of
15
                    file: <name of file containing it>.
               PMIX_TCP_IF_INCLUDE "pmix.tcp.ifinclude" (char*)
16
                    Comma-delimited list of devices and/or Classless Inter-Domain Routing (CIDR) notation to
17
18
                    include when establishing the TCP connection.
19
               PMIX_TCP_IF_EXCLUDE "pmix.tcp.ifexclude" (char*)
                    Comma-delimited list of devices and/or CIDR notation to exclude when establishing the
20
                    TCP connection.
21
22
               PMIX_TCP_IPV4_PORT "pmix.tcp.ipv4" (int)
23
                    The IPv4 port to be used.
               PMIX_TCP_IPV6_PORT "pmix.tcp.ipv6" (int)
24
25
                    The IPv6 port to be used.
26
               PMIX TCP DISABLE IPV4 "pmix.tcp.disipv4" (bool)
                    Set to true to disable IPv4 family of addresses.
27
28
               PMIX_TCP_DISABLE_IPV6 "pmix.tcp.disipv6" (bool)
29
                    Set to true to disable IPv6 family of addresses.
    3.4.6
             Global Data Storage (GDS) attributes
31
               These attributes are used to define the behavior of the GDS used to manage key/value pairs by
32
               passing them into the relevant initialization API - thus, they are not typically accessed via the
33
               PMIx Get API.
```

34

35

1 3.4.7 General process-level attributes

```
2
               These attributes are used to define process attributes and are referenced by their process rank.
               PMIX CPUSET "pmix.cpuset" (char*)
 3
                     HWLOC<sup>2</sup> bitmap applied to the process upon launch.
 4
               PMIX_CREDENTIAL "pmix.cred" (char*)
5
6
                     Security credential assigned to the process.
7
               PMIX SPAWNED "pmix.spawned" (bool)
8
                     true if this process resulted from a call to PMIx Spawn. Lack of inclusion (i.e., a return
9
                     status of PMIX_ERR_NOT_FOUND ) corresponds to a value of false for this attribute.
10
               PMIX ARCH "pmix.arch" (uint32 t)
                     Architecture flag.
11
   3.4.8 Scratch directory attributes
12
13
               These attributes are used to define an application scratch directory and are referenced using the
14
               PMIX RANK WILDCARD rank.
15
               PMIX_TMPDIR "pmix.tmpdir" (char*)
16
                     Full path to the top-level temporary directory assigned to the session.
17
               PMIX_NSDIR "pmix.nsdir" (char*)
                     Full path to the temporary directory assigned to the namespace, under PMIX TMPDIR.
18
19
               PMIX PROCDIR "pmix.pdir" (char*)
20
                     Full path to the subdirectory under PMIX_NSDIR assigned to the process.
21
               PMIX_TDIR_RMCLEAN "pmix.tdir.rmclean" (bool)
22
                     Resource Manager will clean session directories
    3.4.9 Relative Rank Descriptive Attributes
24
               These attributes are used to describe information about relative ranks as assigned by the RM, and
25
               thus are referenced using the process rank except where noted.
26
               PMIX_CLUSTER_ID "pmix.clid" (char*)
                     A string name for the cluster this allocation is on
27
28
               PMIX PROCID "pmix.procid" (pmix proc t)
                     Process identifier
29
30
               PMIX_NSPACE "pmix.nspace" (char*)
31
                     Namespace of the job - may be a numerical value expressed as a string, but is often a
                     non-numerical string carrying information solely of use to the system.
32
               PMIX_JOBID "pmix.jobid" (char*)
33
34
                     Job identifier assigned by the scheduler - may be identical to the namespace, but is often a
```

numerical value expressed as a string (e.g., "12345.3").

PMIX_APPNUM "pmix.appnum" (uint32_t)

35

²https://www.open-mpi.org/projects/hwloc/

1	Application number within the job.
2	PMIX_RANK "pmix.rank" (pmix_rank_t)
3	Process rank within the job.
4	PMIX_GLOBAL_RANK "pmix.grank" (pmix_rank_t)
5	Process rank spanning across all jobs in this session.
6	PMIX_APP_RANK "pmix.apprank" (pmix_rank_t)
7	Process rank within this application.
8	<pre>PMIX_NPROC_OFFSET "pmix.offset" (pmix_rank_t)</pre>
9	Starting global rank of this job - referenced using PMIX_RANK_WILDCARD.
10	<pre>PMIX_LOCAL_RANK "pmix.lrank" (uint16_t)</pre>
11	Local rank on this node within this job.
12	<pre>PMIX_NODE_RANK "pmix.nrank" (uint16_t)</pre>
13	Process rank on this node spanning all jobs.
14	<pre>PMIX_PACKAGE_RANK "pmix.pkgrank" (uint16_t)</pre>
15	Process rank within this job on the package where this process resides
16	<pre>PMIX_LOCALLDR "pmix.lldr" (pmix_rank_t)</pre>
17	Lowest rank on this node within this job - referenced using PMIX_RANK_WILDCARD.
18	<pre>PMIX_APPLDR "pmix.aldr" (pmix_rank_t)</pre>
19	Lowest rank in this application within this job - referenced using PMIX_RANK_WILDCARD.
20	PMIX_PROC_PID "pmix.ppid" (pid_t)
21	PID of specified process.
22	<pre>PMIX_SESSION_ID "pmix.session.id" (uint32_t)</pre>
23	Session identifier assigned by the scheduler - referenced using PMIX_RANK_WILDCARD.
24	<pre>PMIX_NODE_LIST "pmix.nlist" (char*)</pre>
25	Comma-delimited list of nodes running processes for the specified namespace - referenced
26	using PMIX_RANK_WILDCARD.
27	<pre>PMIX_ALLOCATED_NODELIST "pmix.alist" (char*)</pre>
28	Comma-delimited list or regular expression of all nodes in this allocation regardless of
29	whether or not they currently host processes - referenced using PMIX_RANK_WILDCARD.
30	PMIX_HOSTNAME "pmix.hname" (char*)
31	Name of the host (e.g., where a specified process is running, or a given device is located).
32	<pre>PMIX_HOSTNAME_ALIASES "pmix.alias" (char*)</pre>
33	Comma-delimited list of names by which this node is known.
34	PMIX_HOSTNAME_KEEP_FQDN "pmix.fqdn" (bool)
35	FQDN hostnames are being retained
36	<pre>PMIX_NODEID "pmix.nodeid" (uint32_t)</pre>
37	Node identifier expressed as the node's index (beginning at zero) in an array of nodes within
38	the active session. The value must be unique and directly correlate to the PMIX_HOSTNAME
39	of the node - i.e., users can interchangeably reference the same location using either the
40	PMIX_HOSTNAME or corresponding PMIX_NODEID .
41	PMIX_LOCAL_PEERS "pmix.lpeers" (char*)
42	Comma-delimited list of ranks on this node within the specified namespace - referenced
43	using PMIX_RANK_WILDCARD.

```
1
              PMIX_LOCAL_PROCS "pmix.lprocs" (pmix_proc_t array)
2
                    Array of pmix proc t of all processes on the specified node - referenced using
                    PMIX RANK WILDCARD.
3
4
              PMIX_LOCAL_CPUSETS "pmix.lcpus" (char*)
5
                    Colon-delimited cpusets of local peers within the specified namespace - referenced using
6
                    PMIX_RANK_WILDCARD.
7
              PMIX PROC URI "pmix.puri" (char*)
8
                    URI containing contact information for a given process.
9
              PMIX LOCALITY "pmix.loc" (uint16 t)
                    Relative locality of the specified process to the requestor.
10
              PMIX_PARENT_ID "pmix.parent" (pmix_proc_t)
11
12
                    Process identifier of the parent process of the calling process.
13
              PMIX_EXIT_CODE "pmix.exit.code" (int)
14
                    Exit code returned when process terminated
```

3.4.10 Information retrieval attributes

The following attributes are used to specify the level of information (e.g., **session**, **job**, or **application**) being requested where ambiguity may exist - see 5.1.5 for examples of their use.

PMIX_SESSION_INFO "pmix.ssn.info" (bool)

Return information about the specified session. If information about a session other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX_SESSION_ID** attribute identifying the desired target.

PMIX_JOB_INFO "pmix.job.info" (bool)

Return information about the specified job or namespace. If information about a job or namespace other than the one containing the requesting process is desired, then the attribute array must contain a PMIX_JOBID or PMIX_NSPACE attribute identifying the desired target. Similarly, if information is requested about a job or namespace in a session other than the one containing the requesting process, then an attribute identifying the target session must be provided.

PMIX APP INFO "pmix.app.info" (bool)

Return information about the specified application. If information about an application other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX_APPNUM** attribute identifying the desired target. Similarly, if information is requested about an application in a job or session other than the one containing the requesting process, then attributes identifying the target job and/or session must be provided.

PMIX NODE INFO "pmix.node.info" (bool)

Return information about the specified node. If information about a node other than the one containing the requesting process is desired, then the attribute array must contain either the **PMIX_NODEID** or **PMIX_HOSTNAME** attribute identifying the desired target.

3.4.11 Information storage attributes

ambiguous.

The following attributes are used to assemble information by its level (e.g., **session**, **job**, or **application**) for storage where ambiguity may exist - see 11.1.3.1 for examples of their use.

PMIX_SESSION_INFO_ARRAY "pmix.ssn.arr" (pmix_data_array_t)

Provide an array of pmix_info_t containing session-level information. The

PMIX_SESSION_ID attribute is required to be included in the array.

PMIX_JOB_INFO_ARRAY "pmix.job.arr" (pmix_data_array_t)
Provide an array of pmix_info_t containing job-level information. The
PMIX_SESSION_ID attribute of the session containing the job is required to be included in the array whenever the PMIx server library may host multiple sessions (e.g., when executing with a host RM daemon). As information is registered one job (aka namespace) at a time via the PMIx_server_register_nspace API, there is no requirement that the array contain either the PMIX_NSPACE or PMIX_JOBID attributes when used in that context (though either or both of them may be included). At least one of the job identifiers must be provided in all other contexts where the job being referenced is

PMIX_APP_INFO_ARRAY "pmix.app.arr" (pmix_data_array_t)

Provide an array of pmix_info_t containing app-level information. The PMIX_NSPACE or PMIX_JOBID attributes of the job containing the application, plus its PMIX_APPNUM attribute, must to be included in the array when the array is not included as part of a call to PMIx_server_register_nspace - i.e., when the job containing the application is ambiguous. The job identification is otherwise optional.

PMIX_PROC_INFO_ARRAY "pmix.pdata" (pmix_data_array_t)

Provide an array of pmix_info_t containing process-level information. The

PMIX_RANK and PMIX_NSPACE attributes, or the PMIX_PROCID attribute, are
required to be included in the array when the array is not included as part of a call to

PMIx_server_register_nspace - i.e., when the job containing the process is
ambiguous. All three may be included if desired. When the array is included in some
broader structure that identifies the job, then only the PMIX_RANK or the PMIX_PROCID
attribute must be included (the others are optional).

PMIX_NODE_INFO_ARRAY "pmix.node.arr" (pmix_data_array_t)

Provide an array of pmix_info_t containing node-level information. At a minimum, either the PMIX_NODEID or PMIX_HOSTNAME attribute is required to be included in the array, though both may be included.

PMIX_SERVER_INFO_ARRAY "pmix.srv.arr" (pmix_data_array_t)
Array of data on a given server, starting with its PMIX_NSPACE
Note that these assemblages can be used hierarchically:

- a PMIX_JOB_INFO_ARRAY might contain multiple PMIX_APP_INFO_ARRAY elements, each describing values for a specific application within the job
- a PMIX_JOB_INFO_ARRAY could contain a PMIX_NODE_INFO_ARRAY for each node hosting processes from that job, each array describing job-level values for that node

 a PMIX_SESSION_INFO_ARRAY might contain multiple PMIX_JOB_INFO_ARRAY elements, each describing a job executing within the session. Each job array could, in turn, contain both application and node arrays, thus providing a complete picture of the active operations within the allocation

— Advice to PMIx library implementers –

PMIx implementations must be capable of properly parsing and storing any hierarchical depth of information arrays. The resulting stored values are must to be accessible via both PMIx_Get and PMIx_Query_info_nb APIs, assuming appropriate directives are provided by the caller.

3.4.12 Size information attributes

1

2

3

4

5

6

7

9

10 11

12

13

14 15

16

17

18

19 20

21

22

23

24 25

26 27

28

29

30

31

32

33 34

35

36

37

These attributes are used to describe the size of various dimensions of the PMIx universe - all are referenced using **PMIX RANK WILDCARD**.

PMIX UNIV SIZE "pmix.univ.size" (uint32 t)

Number of allocated slots in a session - each slot may or may not be occupied by an executing process. Note that this attribute is equivalent to providing the PMIX_MAX_PROCS entry in the PMIX_SESSION_INFO_ARRAY array - it is included in the Standard for historical reasons.

PMIX_JOB_SIZE "pmix.job.size" (uint32_t)

Total number of processes in this job across all contained applications. Note that this value can be different from PMIX_MAX_PROCS. For example, users may choose to subdivide an allocation (running several jobs in parallel within it), and dynamic programming models may support adding and removing processes from a running job on-the-fly. In the latter case, PMIx events must be used to notify processes within the job that the job size has changed.

PMIX_JOB_NUM_APPS "pmix.job.napps" (uint32_t)

Number of applications in this job.

PMIX_APP_SIZE "pmix.app.size" (uint32_t)

Number of processes in this application.

PMIX_LOCAL_SIZE "pmix.local.size" (uint32_t)

Number of processes in this job or application on this node.

PMIX NODE SIZE "pmix.node.size" (uint32 t)

Number of processes across all jobs on this node.

PMIX_MAX_PROCS "pmix.max.size" (uint32_t)

Maximum number of processes that can be executed in this context (session, namespace, application, or node). Typically, this is a constraint imposed by a scheduler or by user settings in a hostfile or other resource description.

PMIX_NUM_SLOTS "pmix.num.slots" (uint32_t)

Number of slots allocated in this context (session, namespace, application, or node). Note that this attribute is the equivalent to **PMIX_MAX_PROCS** used in the corresponding context - it is included in the Standard for historical reasons.

1 2 3		<pre>PMIX_NUM_NODES "pmix.num.nodes" (uint32_t) Number of nodes in this session, or that are currently executing processes from the associated namespace or application.</pre>
4	3.4.13	Memory information attributes
5 6		These attributes are used to describe memory available and used in the system - all are referenced using ${\tt PMIX_RANK_WILDCARD}$.
7 8 9 0 1 2		PMIX_AVAIL_PHYS_MEMORY "pmix.pmem" (uint64_t) Total available physical memory on this node. PMIX_DAEMON_MEMORY "pmix.dmn.mem" (float) Megabytes of memory currently used by the RM daemon. PMIX_CLIENT_AVG_MEMORY "pmix.cl.mem.avg" (float) Average Megabytes of memory used by client processes.
3	3.4.14	Topology information attributes
4 5		These attributes are used to describe topology information in the PMIx universe - all are referenced using PMIX_RANK_WILDCARD except where noted.
6 7 8		PMIX_LOCAL_TOPO "pmix.ltopo" (char*) eXtensible Markup Language (XML) representation of local node topology. PMIX_TOPOLOGY "pmix.topo" (hwloc_topology_t)
9 20 21		Pointer to the PMIx client's internal hwloc topology object. PMIX_TOPOLOGY_XML "pmix.topo.xml" (char*) XML-based description of topology
22 23 24		PMIX_TOPOLOGY_FILE "pmix.topo.file" (char*) Full path to file containing XML topology description PMIX_TOPOLOGY_SIGNATURE "pmix.toposig" (char*)
25 26 27		Topology signature string. PMIX_LOCALITY_STRING "pmix.locstr" (char*) String describing a process's bound location - referenced using the process's rank. The string
28 29 30		is of the form: NM%s:SK%s:L3%s:L2%s:L1%s:CR%s:HT%s Where the %s is replaced with an integer index or inclusive range for hwloc. NM identifies
11 12 13		the numa node(s). SK identifies the socket(s). L3 identifies the L3 cache(s). L2 identifies the L2 cache(s). L1 identifies the L1 cache(s). CR identifies the cores(s). HT identifies the hardware thread(s). If your architecture does not have the specified hardware designation
14 15 16		then it can be omitted from the signature. For example: NM0:SK0:L30-4:L20-4:L10-4:CR0-4:HT0-39. This means numa node 0, socket 0, L3 caches 0, 1, 2, 3, 4, L2 caches 0-4, L1 caches
17 18 19		0-4, cores 0, 1, 2, 3, 4, and hardware threads 0-39. PMIX_HWLOC_SHMEM_ADDR "pmix.hwlocaddr" (size_t) Address of the HWLOC shared memory segment

1		PMIX_HWLOC_SHMEM_SIZE "pmix.hwlocsize" (size_t)
2		Size of the HWLOC shared memory segment.
3		PMIX_HWLOC_SHMEM_FILE "pmix.hwlocfile" (char*)
4		Path to the HWLOC shared memory file.
5		PMIX_HWLOC_XML_V1 "pmix.hwlocxml1" (char*)
6		XML representation of local topology using HWLOC's v1.x format.
7		PMIX_HWLOC_XML_V2 "pmix.hwlocxml2" (char*)
8		XML representation of local topology using HWLOC's v2.x format.
9		PMIX_HWLOC_SHARE_TOPO "pmix.hwlocsh" (bool)
0		Share the HWLOC topology via shared memory
11		PMIX_HWLOC_HOLE_KIND "pmix.hwlocholek" (char*)
12		Kind of VM "hole" HWLOC should use for shared memory
13	3.4.15	Request-related attributes
14		These attributes are used to influence the behavior of various PMIx operations - they do not
15		represent values accessed using the PMIx_Get API.
16		PMIX_COLLECT_DATA "pmix.collect" (bool)
17		Collect data and return it at the end of the operation.
8		PMIX_TIMEOUT "pmix.timeout" (int)
19		Time in seconds before the specified operation should time out (θ indicating infinite) in
20		error. The timeout parameter can help avoid "hangs" due to programming errors that prevent
21		the target process from ever exposing its data.
22		PMIX_IMMEDIATE "pmix.immediate" (bool)
23		Specified operation should immediately return an error from the PMIx server if the requested
24		data cannot be found - do not request it from the host RM.
25		PMIX_WAIT "pmix.wait" (int)
26		Caller requests that the PMIx server wait until at least the specified number of values are
27		found (0 indicates all and is the default).
28		PMIX_COLLECTIVE_ALGO "pmix.calgo" (char*)
29		Comma-delimited list of algorithms to use for the collective operation. PMIx does not
30		impose any requirements on a host environment's collective algorithms. Thus, the
31		acceptable values for this attribute will be environment-dependent - users are encouraged to
32		check their host environment for supported values.
33		PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)
34		If true , indicates that the requested choice of algorithm is mandatory.
35		PMIX_NOTIFY_LAUNCH "pmix.note.lnch" (bool)
36		Notify the requestor upon launch of the child job and return its namespace in the event
37		PMIX_NOTIFY_COMPLETION "pmix.notecomp" (bool)
38		Notify the parent process upon termination of child job.
39		PMIX_RANGE "pmix.range" (pmix_data_range_t)
10		Value for calls to publish/lookup/unpublish or for monitoring event notifications.
11		PMIX_PERSISTENCE "pmix.persist" (pmix_persistence_t)

1		Value for calls to PMIx_Publish.
2		PMIX_DATA_SCOPE "pmix.scope" (pmix_scope_t)
3		Scope of the data to be found in a PMIx_Get call.
4		PMIX_OPTIONAL "pmix.optional" (bool)
5		Look only in the client's local data store for the requested value - do not request data from
6		the PMIx server if not found.
7		<pre>PMIX_GET_STATIC_VALUES "pmix.get.static" (bool)</pre>
8		Request that any pointers in the returned value point directly to values in the key-value store
9		PMIX_EMBED_BARRIER "pmix.embed.barrier" (bool)
10		Execute a blocking fence operation before executing the specified operation. For example,
11		PMIx_Finalize does not include an internal barrier operation by default. This attribute
12		would direct PMIx_Finalize to execute a barrier as part of the finalize operation.
13		<pre>PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t)</pre>
14		Status returned by job upon its termination. The status will be communicated as part of a
15		PMIx event payload provided by the host environment upon termination of a job. Note that
16		generation of the PMIX_EVENT_JOB_END event is optional and host environments may
17		choose to provide it only upon request.
18		<pre>PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)</pre>
19		State of the specified process as of the last report - may not be the actual current state based
20		on update rate.
21		<pre>PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t)</pre>
22		Status returned by a process upon its termination. The status will be communicated as part
23		of a PMIx event payload provided by the host environment upon termination of a process.
24		Note that generation of the PMIX_PROC_TERMINATED event is optional and host
25		environments may choose to provide it only upon request.
26		<pre>PMIX_NOTIFY_LAUNCH "pmix.note.lnch" (bool)</pre>
27		Notify the requester upon launch of the child job and return its namespace in the event
28		<pre>PMIX_GET_REFRESH_CACHE "pmix.get.refresh" (bool)</pre>
29		When retrieving data for a remote process, refresh the existing local data cache for the
30		process in case new values have been put and committed by it since the last refresh
31	3.4.16	Server-to-PMIx library attributes
		-
32		Attributes used by the host environment to pass data to its PMIx server library. The data will then
33		be parsed and provided to the local PMIx clients. These attributes are all referenced using
34		PMIX_RANK_WILDCARD except where noted.
35		PMIX_REGISTER_NODATA "pmix.reg.nodata" (bool)
36		Registration is for this namespace only, do not copy job data - this attribute is not accessed
37		using the PMIx_Get
38		PMIX_PROC_DATA "pmix.pdata" (pmix_data_array_t)
39		Deprecated in favor of the PMIX_PROC_INFO_ARRAY attribute
40		PMIX_NODE_MAP "pmix.nmap" (char*)
41		Regular expression of nodes - see 11.1.3.1 for an explanation of its generation.

```
1
              PMIX_PROC_MAP "pmix.pmap" (char*)
2
                    Regular expression describing processes on each node - see 11.1.3.1 for an explanation of its
 3
                    generation.
4
              PMIX_ANL_MAP "pmix.anlmap" (char*)
                    Process mapping in Argonne National Laboratory's PMI-1/PMI-2 notation.
5
6
              PMIX APP MAP TYPE "pmix.apmap.type" (char*)
7
                    Type of mapping used to layout the application (e.g., cyclic).
8
              PMIX APP MAP REGEX "pmix.apmap.regex" (char*)
9
                    Regular expression describing the result of the process mapping.
10
              PMIX REQUIRED KEY "pmix.reg.key" (char*)
                    Key the user needs prior to responding from a dmodex request
11
   3.4.17 Server-to-Client attributes
12
              Attributes used internally to communicate data from the PMIx server to the PMIx client - they do
13
              not represent values accessed using the PMIx_Get API.
14
15
              PMIX PROC BLOB "pmix.pblob" (pmix byte object t)
16
                    Packed blob of process data.
              PMIX_MAP_BLOB "pmix.mblob" (pmix_byte_object_t)
17
18
                    Packed blob of process location.
    3.4.18 Event handler registration and notification attributes
20
              Attributes to support event registration and notification - they are values passed to the event
21
              registration and notification APIs and therefore are not accessed using the PMIx Get API.
22
              PMIX_EVENT_HDLR_NAME "pmix.evname" (char*)
23
                    String name identifying this handler.
24
              PMIX EVENT HDLR FIRST "pmix.evfirst" (bool)
25
                    Invoke this event handler before any other handlers.
26
              PMIX_EVENT_HDLR_LAST "pmix.evlast" (bool)
27
                    Invoke this event handler after all other handlers have been called.
28
              PMIX_EVENT_HDLR_FIRST_IN_CATEGORY "pmix.evfirstcat" (bool)
29
                    Invoke this event handler before any other handlers in this category.
              PMIX_EVENT_HDLR_LAST_IN_CATEGORY "pmix.evlastcat" (bool)
30
                    Invoke this event handler after all other handlers in this category have been called.
31
32
              PMIX EVENT HDLR BEFORE "pmix.evbefore" (char*)
33
                    Put this event handler immediately before the one specified in the (char*) value.
34
              PMIX_EVENT_HDLR_AFTER "pmix.evafter" (char*)
                    Put this event handler immediately after the one specified in the (char*) value.
35
              PMIX_EVENT_HDLR_PREPEND "pmix.evprepend" (bool)
36
37
                    Prepend this handler to the precedence list within its category.
              PMIX EVENT HDLR APPEND "pmix.evappend" (bool)
38
39
                    Append this handler to the precedence list within its category.
```

```
PMIX_EVENT_CUSTOM_RANGE "pmix.evrange" (pmix_data_array_t*)
 1
 2
                    Array of pmix proc t defining range of event notification.
 3
              PMIX_EVENT_AFFECTED_PROC "pmix.evproc" (pmix_proc_t)
 4
                    The single process that was affected.
 5
              PMIX EVENT AFFECTED PROCS "pmix.evaffected" (pmix data array t*)
 6
                    Array of pmix proc t defining affected processes.
 7
              PMIX EVENT NON DEFAULT "pmix.evnondef" (bool)
 8
                    Event is not to be delivered to default event handlers.
              PMIX_EVENT_RETURN_OBJECT "pmix.evobject" (void *)
 9
10
                    Object to be returned whenever the registered callback function cbfunc is invoked. The
                    object will only be returned to the process that registered it.
11
              PMIX_EVENT_DO_NOT_CACHE "pmix.evnocache" (bool)
12
                    Instruct the PMIx server not to cache the event.
13
14
              PMIX EVENT SILENT TERMINATION "pmix.evsilentterm" (bool)
15
                    Do not generate an event when this job normally terminates.
              PMIX_EVENT_PROXY "pmix.evproxy" (pmix_proc_t*)
16
17
                    PMIx server that sourced the event
              PMIX EVENT TEXT MESSAGE "pmix.evtext" (char*)
18
                    Text message suitable for output by recipient - e.g., describing the cause of the event
19
   3.4.19 Fault tolerance attributes
20
21
              Attributes to support fault tolerance behaviors - they are values passed to the event notification API
22
              and therefore are not accessed using the PMIx Get API.
23
              PMIX EVENT TERMINATE SESSION "pmix.evterm.sess" (bool)
                    The RM intends to terminate this session.
24
25
              PMIX EVENT TERMINATE JOB "pmix.evterm.job" (bool)
26
                    The RM intends to terminate this job.
27
              PMIX_EVENT_TERMINATE_NODE "pmix.evterm.node" (bool)
28
                    The RM intends to terminate all processes on this node.
              PMIX_EVENT_TERMINATE_PROC "pmix.evterm.proc" (bool)
29
                    The RM intends to terminate just this process.
30
31
              PMIX EVENT ACTION TIMEOUT "pmix.evtimeout" (int)
32
                    The time in seconds before the RM will execute error response.
              PMIX EVENT NO TERMINATION "pmix.evnoterm" (bool)
33
                    Indicates that the handler has satisfactorily handled the event and believes termination of the
34
35
                    application is not required.
36
              PMIX_EVENT_WANT_TERMINATION "pmix.evterm" (bool)
                    Indicates that the handler has determined that the application should be terminated
37
```

3.4.20 Spawn attributes

2

3

5

6

7

8

9 10

11

12

13

14 15

16

17

18

19 20

21 22

23

24 25

26 27

28 29

30

31

32 33

34

35

36

37 38

39

40

41 42 Attributes used to describe PMIx_Spawn behavior - they are values passed to the PMIx_Spawn API and therefore are not accessed using the PMIx Get API when used in that context. However, some of the attributes defined in this section can be provided by the host environment for other purposes - e.g., the environment might provide the PMIX MAPPER attribute in the job-related information so that an application can use PMIx_Get to discover the layout algorithm used for determining process locations. Multi-use attributes and their respective access reference rank are denoted below. PMIX_PERSONALITY "pmix.pers" (char*) Name of personality to use. PMIX HOST "pmix.host" (char*) Comma-delimited list of hosts to use for spawned processes. PMIX_HOSTFILE "pmix.hostfile" (char*) Hostfile to use for spawned processes. PMIX ADD HOST "pmix.addhost" (char*) Comma-delimited list of hosts to add to the allocation. PMIX ADD HOSTFILE "pmix.addhostfile" (char*) Hostfile listing hosts to add to existing allocation. PMIX_PREFIX "pmix.prefix" (char*) Prefix to use for starting spawned processes. PMIX_WDIR "pmix.wdir" (char*) Working directory for spawned processes. PMIX MAPPER "pmix.mapper" (char*) Mapping mechanism to use for placing spawned processes - when accessed using PMIx_Get, use the PMIX_RANK_WILDCARD value for the rank to discover the mapping mechanism used for the provided namespace. PMIX DISPLAY MAP "pmix.dispmap" (bool) Display process mapping upon spawn. PMIX_PPR "pmix.ppr" (char*) Number of processes to spawn on each identified resource. PMIX_MAPBY "pmix.mapby" (char*) Process mapping policy - when accessed using PMIx Get, use the PMIX RANK WILDCARD value for the rank to discover the mapping policy used for the provided namespace PMIX RANKBY "pmix.rankby" (char*) Process ranking policy - when accessed using PMIx Get, use the PMIX_RANK_WILDCARD value for the rank to discover the ranking algorithm used for the provided namespace

PMIX BINDTO "pmix.bindto" (char*)

Process binding policy - when accessed using **PMIx** Get, use the

PMIX_RANK_WILDCARD value for the rank to discover the binding policy used for the provided namespace

1	PMIX_PRELOAD_BIN "pmix.preloadbin" (bool)
2	Preload binaries onto nodes.
3	<pre>PMIX_PRELOAD_FILES "pmix.preloadfiles" (char*)</pre>
4	Comma-delimited list of files to pre-position on nodes.
5	PMIX_NON_PMI "pmix.nonpmi" (bool)
6	Spawned processes will not call PMIx_Init.
7	<pre>PMIX_STDIN_TGT "pmix.stdin" (uint32_t)</pre>
8	Spawned process rank that is to receive any forwarded stdin .
9	PMIX_DEBUGGER_DAEMONS "pmix.debugger" (bool)
0	Spawned application consists of debugger daemons.
1	PMIX_COSPAWN_APP "pmix.cospawn" (bool)
2	Designated application is to be spawned as a disconnected job. Meaning that it is not part of
3	the "comm_world" of the parent process.
4	PMIX_SET_SESSION_CWD "pmix.ssncwd" (bool)
5	Set the application's current working directory to the session working directory assigned by
6	the RM - when accessed using PMIx_Get , use the PMIX_RANK_WILDCARD value for
7	the rank to discover the session working directory assigned to the provided namespace
8	PMIX_TAG_OUTPUT "pmix.tagout" (bool)
9	Tag application output with the identity of the source process.
20	PMIX_TIMESTAMP_OUTPUT "pmix.tsout" (bool)
21	Timestamp output from applications.
22	PMIX_MERGE_STDERR_STDOUT "pmix.mergeerrout" (bool)
23	Merge stdout and stderr streams from application processes.
24	PMIX_OUTPUT_TO_FILE "pmix.outfile" (char*)
25	Direct application output (both stdout and stderr) into files of form " <filename>.rank"</filename>
26	PMIX_OUTPUT_TO_DIRECTORY "pmix.outdir" (char*)
27	Direct application output into files of form " <directory>/<jobid>/rank.<rank>/stdout[err]"</rank></jobid></directory>
28	PMIX_INDEX_ARGV "pmix.indxargv" (bool)
29	Mark the argv with the rank of the process.
30	PMIX_CPUS_PER_PROC "pmix.cpuperproc" (uint32_t)
31	Number of cpus to assign to each rank - when accessed using PMIx_Get , use the
32	PMIX_RANK_WILDCARD value for the rank to discover the cpus/process assigned to the
3	provided namespace
34	PMIX_NO_PROCS_ON_HEAD "pmix.nolocal" (bool)
35	Do not place processes on the head node.
86	PMIX_NO_OVERSUBSCRIBE "pmix.noover" (bool)
37	Do not oversubscribe the cpus.
8	PMIX_REPORT_BINDINGS "pmix.repbind" (bool)
19	Report bindings of the individual processes.
-0	PMIX_CPU_LIST "pmix.cpulist" (char*)
1	List of cpus to use for this job - when accessed using PMIx_Get, use the
-2	PMIX_RANK_WILDCARD value for the rank to discover the cpu list used for the provided
3	namespace

```
PMIX_JOB_RECOVERABLE "pmix.recover" (bool)
1
2
                    Application supports recoverable operations.
              PMIX JOB_CONTINUOUS "pmix.continuous" (bool)
 3
4
                    Application is continuous, all failed processes should be immediately restarted.
              PMIX MAX RESTARTS "pmix.maxrestarts" (uint32 t)
5
6
                    Maximum number of times to restart a job - when accessed using PMIx Get, use the
7
                    PMIX RANK WILDCARD value for the rank to discover the max restarts for the provided
8
                    namespace
9
              PMIX_SPAWN_TOOL "pmix.spwn.tool" (bool)
                    Indicate that the job being spawned is a tool
10
              PMIX_CMD_LINE "pmix.cmd.line" (char*)
11
                    Command line used to execute the specified process (e.g., "mpirun -n 2 -map-by foo
12
                    ./myapp")
13
14
              PMIX FORK EXEC AGENT "pmix.fe.agnt" (char*)
                    Command line of fork/exec agent to be used for starting local processes
15
              PMIX_TIMEOUT_STACKTRACES "pmix.tim.stack" (bool)
16
17
                    Include process stacktraces in timeout report from a job
18
              PMIX TIMEOUT REPORT STATE "pmix.tim.state" (bool)
19
                    Report process states in timeout report from a job
              PMIX_APP_ARGV "pmix.app.argv" (char*)
20
                    Consolidated argy passed to the spawn command for the given process (e.g., "./myapp arg1
21
22
                    arg2 arg3")
   3.4.21 Query attributes
23
24
              Attributes used to describe PMIx_Query_info_nb behavior - these are values passed to the
              PMIx_Query_info_nb API and therefore are not passed to the PMIx_Get API.
25
              PMIX_QUERY_SUPPORTED_KEYS "pmix.qry.keys" (char*)
26
                    Returns comma-delimited list of keys supported by the query function. NO QUALIFIERS
27
28
              PMIX QUERY SUPPORTED QUALIFIERS "pmix.gry.quals" (char*)
29
                    Return comma-delimited list of qualifiers supported by a query on the provided key, instead
30
                    of actually performing the query on the key. NO QUALIFIERS
              PMIX_QUERY_REFRESH_CACHE "pmix.qry.rfsh" (bool)
31
                    Retrieve updated information from server. NO QUALIFIERS
32
              PMIX QUERY NAMESPACES "pmix.grv.ns" (char*)
33
                    Request a comma-delimited list of active namespaces. NO QUALIFIERS
34
              PMIX_QUERY_NAMESPACE_INFO "pmix.qry.nsinfo" (pmix_data_array_t*)
35
36
                    Return an array of active namespace information - each element will itself contain an array
                    including the namespace plus the command line of the application executing within it.
37
                    SUPPORTED QUALIFIERS: PMIX_NSPACE of specific namespace whose info is being
38
39
                    requested
40
              PMIX_QUERY_JOB_STATUS "pmix.qry.jst" (pmix_status_t)
```

1	Status of a specified, currently executing job. REQUIRED QUALIFIER: PMIX_NSPACE
2	indicating the namespace whose status is being queried
3	<pre>PMIX_QUERY_QUEUE_LIST "pmix.qry.qlst" (char*)</pre>
4	Request a comma-delimited list of scheduler queues. NO QUALIFIERS
5	PMIX_QUERY_QUEUE_STATUS "pmix.qry.qst" (TBD)
6	Status of a specified scheduler queue. SUPPORTED QUALIFIERS: PMIX_ALLOC_QUEUE
7	naming specific queue whose status is being requested
8	<pre>PMIX_QUERY_PROC_TABLE "pmix.qry.ptable" (char*)</pre>
9	Input namespace of the job whose information is being requested returns (
0	<pre>pmix_data_array_t) an array of pmix_proc_info_t . REQUIRED QUALIFIER:</pre>
1	PMIX_NSPACE indicating the namespace whose process table is being queried
2	<pre>PMIX_QUERY_LOCAL_PROC_TABLE "pmix.qry.lptable" (char*)</pre>
3	Input namespace of the job whose information is being requested returns (
4	<pre>pmix_data_array_t) an array of pmix_proc_info_t for processes in job on same</pre>
5	node. REQUIRED QUALIFIER: PMIX_NSPACE indicating the namespace whose process
6	table is being queried
7	PMIX_QUERY_AUTHORIZATIONS "pmix.qry.auths" (bool)
8	Return operations the PMIx tool is authorized to perform. NO QUALIFIERS
9	PMIX_QUERY_SPAWN_SUPPORT "pmix.qry.spawn" (bool)
20	Return a comma-delimited list of supported spawn attributes. NO QUALIFIERS
21	PMIX_QUERY_DEBUG_SUPPORT "pmix.qry.debug" (bool)
22	Return a comma-delimited list of supported debug attributes. NO QUALIFIERS
23	PMIX_QUERY_MEMORY_USAGE "pmix.qry.mem" (bool)
24	Return information on memory usage for the processes indicated in the qualifiers.
25	SUPPORTED QUALIFIERS: PMIX_NSPACE and PMIX_RANK, or PMIX_PROCID of
26	specific process(es) whose memory usage is being requested
27	PMIX_QUERY_LOCAL_ONLY "pmix.qry.local" (bool)
28	Constrain the query to local information only. NO QUALIFIERS
29	PMIX_QUERY_REPORT_AVG "pmix.qry.avg" (bool)
30	Report only average values for sampled information. NO QUALIFIERS
81	PMIX_QUERY_REPORT_MINMAX "pmix.qry.minmax" (bool) Report minimum and maximum values. NO QUALIFIERS
12 13	PMIX_QUERY_ALLOC_STATUS "pmix.query.alloc" (char*)
34	String identifier of the allocation whose status is being requested. NO QUALIFIERS
95	PMIX_TIME_REMAINING "pmix.time.remaining" (char*)
36	Query number of seconds (uint32_t) remaining in allocation for the specified namespace.
37	SUPPORTED QUALIFIERS: PMIX_NSPACE of the namespace whose info is being
18	requested (defaults to allocation containing the caller)
19	PMIX_QUERY_ATTRIBUTE_SUPPORT "pmix.qry.attrs" (bool)
0	Query list of supported attributes for specified APIs. SUPPORTED QUALIFIERS:
1	PMIX_CLIENT_FUNCTIONS, PMIX_SERVER_FUNCTIONS,
2	PMIX_TOOL_FUNCTIONS, and/or PMIX_HOST_FUNCTIONS
3	PMIX_QUERY_NUM_PSETS "pmix.qry.psetnum" (size_t)
-	

```
1
                    Return the number of psets defined in the specified range (defaults to session).
2
              PMIX QUERY PSET NAMES "pmix.grv.psets" (char*)
3
                    Return a comma-delimited list of the names of the psets defined in the specified range
4
                    (defaults to session).
              PMIX_QUERY_AVAIL_SERVERS "pmix.qry.asrvrs" (pmix_data_array_t*)
5
                    Return an array of pmix info t, each element itself containing an array of
6
                    pmix info t of available data for servers on this node to which the caller might be able
7
8
                    to connect. Each array must include at least one of the rendezvous-required pieces of
9
                    information.
   3.4.22 Log attributes
10
              Attributes used to describe PMIx Log nb behavior - these are values passed to the
11
              PMIx Log nb API and therefore are not accessed using the PMIx Get API.
12
              PMIX_LOG_SOURCE "pmix.log.source" (pmix_proc_t*)
13
14
                    ID of source of the log request
15
              PMIX LOG STDERR "pmix.log.stderr" (char*)
                    Log string to stderr.
16
              PMIX_LOG_STDOUT "pmix.log.stdout" (char*)
17
18
                    Log string to stdout.
19
              PMIX_LOG_SYSLOG "pmix.log.syslog" (char*)
                    Log data to syslog. Defaults to ERROR priority. Will log to global syslog if available,
20
                    otherwise to local syslog
21
22
              PMIX LOG LOCAL SYSLOG "pmix.log.lsys" (char*)
23
                    Log data to local syslog. Defaults to ERROR priority.
              PMIX_LOG_GLOBAL_SYSLOG "pmix.log.gsys" (char*)
24
                    Forward data to system "gateway" and log msg to that syslog Defaults to ERROR priority.
25
              PMIX LOG SYSLOG PRI "pmix.log.syspri" (int)
26
                    Syslog priority level
27
28
              PMIX_LOG_TIMESTAMP "pmix.log.tstmp" (time_t)
29
                    Timestamp for log report
              PMIX LOG GENERATE TIMESTAMP "pmix.log.gtstmp" (bool)
30
                    Generate timestamp for log
31
              PMIX LOG TAG OUTPUT "pmix.log.tag" (bool)
32
                    Label the output stream with the channel name (e.g., "stdout")
33
              PMIX_LOG_TIMESTAMP_OUTPUT "pmix.log.tsout" (bool)
34
                    Print timestamp in output string
35
              PMIX LOG XML OUTPUT "pmix.log.xml" (bool)
36
37
                    Print the output stream in XML format
              PMIX LOG ONCE "pmix.log.once" (bool)
38
                    Only log this once with whichever channel can first support it, taking the channels in priority
39
40
                    order
41
              PMIX_LOG_MSG "pmix.log.msg" (pmix_byte_object_t)
```

```
1
                    Message blob to be sent somewhere.
2
              PMIX LOG_EMAIL "pmix.log.email" (pmix_data_array_t)
3
                    Log via email based on pmix_info_t containing directives.
4
              PMIX_LOG_EMAIL_ADDR "pmix.log.emaddr" (char*)
5
                    Comma-delimited list of email addresses that are to receive the message.
6
              PMIX LOG EMAIL SENDER ADDR "pmix.log.emfaddr" (char*)
7
                    Return email address of sender
8
              PMIX LOG EMAIL_SUBJECT "pmix.log.emsub" (char*)
9
                    Subject line for email.
10
              PMIX LOG EMAIL MSG "pmix.log.emmsg" (char*)
11
                    Message to be included in email.
12
              PMIX LOG EMAIL SERVER "pmix.log.esrvr" (char*)
                    Hostname (or IP address) of estmp server
13
14
              PMIX LOG EMAIL SRVR PORT "pmix.log.esrvrprt" (int32 t)
15
                    Port the email server is listening to
16
              PMIX_LOG_GLOBAL_DATASTORE "pmix.log.gstore" (bool)
17
                    Store the log data in a global data store (e.g., database)
              PMIX_LOG_JOB_RECORD "pmix.log.jrec" (bool)
18
                    Log the provided information to the host environment's job record
19
   3.4.23 Resource manager attributes
20
21
              Attributes used to describe the RM - these are values assigned by the host environment and accessed
22
              using the PMIx Get API. The value of the provided namespace is unimportant but should be
              given as the namespace of the requesting process and a rank of PMIX_RANK_WILDCARD used to
23
              indicate that the information will be found with the job-level information.
24
25
              PMIX RM NAME "pmix.rm.name" (char*)
26
                    String name of the RM.
27
              PMIX_RM_VERSION "pmix.rm.version" (char*)
28
                    RM version string.
    3.4.24 Environment variable attributes
29
30
              Attributes used to adjust environment variables - these are values passed to the PMIx_Spawn API
              and are not accessed using the PMIx_Get API.
31
32
              PMIX SET ENVAR "pmix.envar.set" (pmix envar t*)
33
                    Set the envar to the given value, overwriting any pre-existing one
              PMIX_UNSET_ENVAR "pmix.envar.unset" (char*)
34
35
                    Unset the environment variable specified in the string.
              PMIX ADD ENVAR "pmix.envar.add" (pmix envar t*)
36
37
                    Add the environment variable, but do not overwrite any pre-existing one
```

38

PMIX_PREPEND_ENVAR "pmix.envar.prepnd" (pmix_envar_t*)

```
1
                    Prepend the given value to the specified environmental value using the given separator
2
                    character, creating the variable if it doesn't already exist
 3
              PMIX_APPEND_ENVAR "pmix.envar.appnd" (pmix_envar_t*)
4
                    Append the given value to the specified environmental value using the given separator
                    character, creating the variable if it doesn't already exist
5
              PMIX_FIRST_ENVAR "pmix.envar.first" (pmix_envar_t*)
6
7
                    Ensure the given value appears first in the specified envar using the separator character,
8
                    creating the envar if it doesn't already exist
   3.4.25 Job Allocation attributes
              Attributes used to describe the job allocation - these are values passed to and/or returned by the
10
              PMIx Allocation request nb and PMIx Allocation request APIs and are not
11
              accessed using the PMIx Get API
12
13
              PMIX_ALLOC_REQ_ID "pmix.alloc.reqid" (char*)
14
                    User-provided string identifier for this allocation request which can later be used to query
15
                    status of the request.
              PMIX ALLOC ID "pmix.alloc.id" (char*)
16
                    A string identifier (provided by the host environment) for the resulting allocation which can
17
                    later be used to reference the allocated resources in, for example, a call to PMIx Spawn.
18
19
              PMIX ALLOC QUEUE "pmix.alloc.queue" (char*)
                    Name of the WLM queue being referenced.
20
              PMIX_ALLOC_NUM_NODES "pmix.alloc.nnodes" (uint64_t)
21
22
                    The number of nodes.
23
              PMIX_ALLOC_NODE_LIST "pmix.alloc.nlist" (char*)
                    Regular expression of the specific nodes.
24
25
              PMIX ALLOC NUM CPUS "pmix.alloc.ncpus" (uint64 t)
26
                    Number of cpus.
              PMIX_ALLOC_NUM_CPU_LIST "pmix.alloc.ncpulist" (char*)
27
28
                    Regular expression of the number of cpus for each node.
              PMIX_ALLOC_CPU_LIST "pmix.alloc.cpulist" (char*)
29
30
                    Regular expression of the specific cpus indicating the cpus involved.
              PMIX ALLOC MEM SIZE "pmix.alloc.msize" (float)
31
32
                    Number of Megabytes.
              PMIX ALLOC NETWORK "pmix.alloc.net" (array)
33
                    Changed to PMIX_ALLOC_FABRIC
34
              PMIX_ALLOC_FABRIC "pmix.alloc.net" (array)
35
                    Array of pmix_info_t describing requested fabric resources. This must include at least:
36
                    PMIX_ALLOC_FABRIC_ID, PMIX_ALLOC_FABRIC_TYPE, and
37
                    PMIX_ALLOC_FABRIC_ENDPTS, plus whatever other descriptors are desired.
38
              PMIX_ALLOC_NETWORK_ID "pmix.alloc.netid" (char*)
39
                    Changed to PMIX_ALLOC_FABRIC_ID
40
41
              PMIX ALLOC FABRIC ID "pmix.alloc.netid" (char*)
```

1	The key to be used when accessing this requested fabric allocation. The allocation will be
2	returned/stored as a pmix_data_array_t of pmix_info_t indexed by this key and
3	containing at least one entry with the same key and the allocated resource description. The
4	type of the included value depends upon the fabric support. For example, a TCP allocation
5	might consist of a comma-delimited string of socket ranges such as
6	"32000-32100,33005,38123-38146". Additional entries will consist of any provided
7	resource request directives, along with their assigned values. Examples include:
8	<pre>PMIX_ALLOC_FABRIC_TYPE - the type of resources provided;</pre>
9	PMIX_ALLOC_FABRIC_PLANE - if applicable, what plane the resources were assigned
10	from; PMIX_ALLOC_FABRIC_QOS - the assigned QoS; PMIX_ALLOC_BANDWIDTH -
11	the allocated bandwidth; PMIX_ALLOC_FABRIC_SEC_KEY - a security key for the
12	requested fabric allocation. NOTE: the assigned values may differ from those requested,
13	especially if PMIX_INFO_REQD was not set in the request.
14	PMIX_ALLOC_BANDWIDTH "pmix.alloc.bw" (float)
15	Mbits/sec.
16	<pre>PMIX_ALLOC_NETWORK_QOS "pmix.alloc.netqos" (char*)</pre>
17	Changed to PMIX_ALLOC_FABRIC_QOS
18	<pre>PMIX_ALLOC_FABRIC_QOS "pmix.alloc.netqos" (char*)</pre>
19	Quality of service level.
20	<pre>PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)</pre>
21	Time in seconds.
22	<pre>PMIX_ALLOC_NETWORK_TYPE "pmix.alloc.nettype" (char*)</pre>
23	Changed to PMIX_ALLOC_FABRIC_TYPE
24	<pre>PMIX_ALLOC_FABRIC_TYPE "pmix.alloc.nettype" (char*)</pre>
25	Type of desired transport (e.g., "tcp", "udp")
26	<pre>PMIX_ALLOC_NETWORK_PLANE "pmix.alloc.netplane" (char*)</pre>
27	Changed to PMIX_ALLOC_FABRIC_PLANE
28	<pre>PMIX_ALLOC_FABRIC_PLANE "pmix.alloc.netplane" (char*)</pre>
29	ID string for the NIC (aka plane) to be used for this allocation (e.g., CIDR for Ethernet)
30	<pre>PMIX_ALLOC_NETWORK_ENDPTS "pmix.alloc.endpts" (size_t)</pre>
31	Changed to PMIX_ALLOC_FABRIC_ENDPTS
32	<pre>PMIX_ALLOC_FABRIC_ENDPTS "pmix.alloc.endpts" (size_t)</pre>
33	Number of endpoints to allocate per process
34	<pre>PMIX_ALLOC_NETWORK_ENDPTS_NODE "pmix.alloc.endpts.nd" (size_t)</pre>
35	Changed to PMIX_ALLOC_FABRIC_ENDPTS_NODE
36	<pre>PMIX_ALLOC_FABRIC_ENDPTS_NODE "pmix.alloc.endpts.nd" (size_t)</pre>
37	Number of endpoints to allocate per node
38	<pre>PMIX_ALLOC_NETWORK_SEC_KEY "pmix.alloc.nsec" (pmix_byte_object_t)</pre>
39	Changed to PMIX_ALLOC_FABRIC_SEC_KEY
40	<pre>PMIX_ALLOC_FABRIC_SEC_KEY "pmix.alloc.nsec" (pmix_byte_object_t)</pre>
41	Fabric security key

1 3.4.26 Job control attributes

```
2
              Attributes used to request control operations on an executing application - these are values passed
              to the PMIx Job control nb API and are not accessed using the PMIx Get API.
3
              PMIX_JOB_CTRL_ID "pmix.jctrl.id" (char*)
 4
                    Provide a string identifier for this request. The user can provide an identifier for the
5
6
                    requested operation, thus allowing them to later request status of the operation or to
7
                    terminate it. The host, therefore, shall track it with the request for future reference.
8
              PMIX_JOB_CTRL_PAUSE "pmix.jctrl.pause" (bool)
9
                    Pause the specified processes.
              PMIX JOB CTRL RESUME "pmix.jctrl.resume" (bool)
10
                    Resume ("un-pause") the specified processes.
11
12
              PMIX_JOB_CTRL_CANCEL "pmix.jctrl.cancel" (char*)
                    Cancel the specified request - the provided request ID must match the
13
                    PMIX JOB CTRL ID provided to a previous call to PMIx Job control. An ID of
14
15
                    NULL implies cancel all requests from this requestor.
              PMIX_JOB_CTRL_KILL "pmix.jctrl.kill" (bool)
16
                    Forcibly terminate the specified processes and cleanup.
17
              PMIX_JOB_CTRL_RESTART "pmix.jctrl.restart" (char*)
18
                    Restart the specified processes using the given checkpoint ID.
19
20
              PMIX JOB CTRL CHECKPOINT "pmix.jctrl.ckpt" (char*)
                    Checkpoint the specified processes and assign the given ID to it.
21
              PMIX JOB CTRL CHECKPOINT EVENT "pmix.jctrl.ckptev" (bool)
22
23
                    Use event notification to trigger a process checkpoint.
              PMIX_JOB_CTRL_CHECKPOINT_SIGNAL "pmix.jctrl.ckptsig" (int)
24
                    Use the given signal to trigger a process checkpoint.
25
              PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT "pmix.jctrl.ckptsig" (int)
26
27
                    Time in seconds to wait for a checkpoint to complete.
28
              PMIX JOB CTRL CHECKPOINT METHOD
29
               "pmix.jctrl.ckmethod" (pmix data array t)
30
                    Array of pmix info t declaring each method and value supported by this application.
              PMIX JOB CTRL SIGNAL "pmix.jctrl.sig" (int)
31
32
                    Send given signal to specified processes.
33
              PMIX_JOB_CTRL_PROVISION "pmix.jctrl.pvn" (char*)
                    Regular expression identifying nodes that are to be provisioned.
34
              PMIX_JOB_CTRL_PROVISION_IMAGE "pmix.jctrl.pvnimg" (char*)
35
                    Name of the image that is to be provisioned.
36
              PMIX JOB CTRL PREEMPTIBLE "pmix.jctrl.preempt" (bool)
37
38
                    Indicate that the job can be pre-empted.
              PMIX_JOB_CTRL_TERMINATE "pmix.jctrl.term" (bool)
39
                    Politely terminate the specified processes.
40
              PMIX_REGISTER_CLEANUP "pmix.reg.cleanup" (char*)
41
42
                    Comma-delimited list of files to be removed upon process termination
```

1		<pre>PMIX_REGISTER_CLEANUP_DIR "pmix.reg.cleanupdir" (char*)</pre>
2		Comma-delimited list of directories to be removed upon process termination
3		PMIX_CLEANUP_RECURSIVE "pmix.clnup.recurse" (bool)
4		Recursively cleanup all subdirectories under the specified one(s)
5		PMIX_CLEANUP_EMPTY "pmix.clnup.empty" (bool)
6		Only remove empty subdirectories
7		<pre>PMIX_CLEANUP_IGNORE "pmix.clnup.ignore" (char*)</pre>
8		Comma-delimited list of filenames that are not to be removed
9		PMIX_CLEANUP_LEAVE_TOPDIR "pmix.clnup.lvtop" (bool)
0		When recursively cleaning subdirectories, do not remove the top-level directory (the one
11		given in the cleanup request)
12	3.4.27	Monitoring attributes
13		Attributes used to control monitoring of an executing application- these are values passed to the
14		PMIx_Process_monitor_nb API and are not accessed using the PMIx_Get API.
15		<pre>PMIX_MONITOR_ID "pmix.monitor.id" (char*)</pre>
16		Provide a string identifier for this request.
17		PMIX_MONITOR_CANCEL "pmix.monitor.cancel" (char*)
18		Identifier to be canceled (NULL means cancel all monitoring for this process).
19		PMIX_MONITOR_APP_CONTROL "pmix.monitor.appctrl" (bool)
20		The application desires to control the response to a monitoring event.
21		PMIX_MONITOR_HEARTBEAT "pmix.monitor.mbeat" (void)
22		Register to have the PMIx server monitor the requestor for heartbeats.
23		PMIX_SEND_HEARTBEAT "pmix.monitor.beat" (void)
24		Send heartbeat to local PMIx server.
25		PMIX_MONITOR_HEARTBEAT_TIME "pmix.monitor.btime" (uint32_t)
26		Time in seconds before declaring heartbeat missed.
27		PMIX_MONITOR_HEARTBEAT_DROPS "pmix.monitor.bdrop" (uint32_t)
28		Number of heartbeats that can be missed before generating the event.
29		<pre>PMIX_MONITOR_FILE "pmix.monitor.fmon" (char*)</pre>
30		Register to monitor file for signs of life.
31		PMIX_MONITOR_FILE_SIZE "pmix.monitor.fsize" (bool)
32		Monitor size of given file is growing to determine if the application is running.
33		PMIX_MONITOR_FILE_ACCESS "pmix.monitor.faccess" (char*)
34		Monitor time since last access of given file to determine if the application is running.
35		PMIX_MONITOR_FILE_MODIFY "pmix.monitor.fmod" (char*)
36		Monitor time since last modified of given file to determine if the application is running.
37		PMIX_MONITOR_FILE_CHECK_TIME "pmix.monitor.ftime" (uint32_t)
38		Time in seconds between checking the file.
39		PMIX_MONITOR_FILE_DROPS "pmix.monitor.fdrop" (uint32_t)
10		Number of file checks that can be missed before generating the event.

1 3.4.28 Security attributes

```
Attributes for managing security credentials
2 PMIx v3.0
3
               PMIX CRED TYPE "pmix.sec.ctype" (char*)
                     When passed in PMIx_Get_credential, a prioritized, comma-delimited list of desired
4
5
                     credential types for use in environments where multiple authentication mechanisms may be
6
                     available. When returned in a callback function, a string identifier of the credential type.
7
               PMIX_CRYPTO_KEY "pmix.sec.key" (pmix_byte_object_t)
8
                     Blob containing crypto key
   3.4.29 IO Forwarding attributes
               Attributes used to control IO forwarding behavior
10 PMIx v3.0
11
               PMIX IOF CACHE SIZE "pmix.iof.csize" (uint32 t)
12
                     The requested size of the server cache in bytes for each specified channel. By default, the
                     server is allowed (but not required) to drop all bytes received beyond the max size.
13
               PMIX IOF DROP OLDEST "pmix.iof.old" (bool)
14
15
                     In an overflow situation, drop the oldest bytes to make room in the cache.
16
               PMIX_IOF_DROP_NEWEST "pmix.iof.new" (bool)
17
                     In an overflow situation, drop any new bytes received until room becomes available in the
18
                     cache (default).
               PMIX_IOF_BUFFERING_SIZE "pmix.iof.bsize" (uint32_t)
19
                     Controls grouping of IO on the specified channel(s) to avoid being called every time a bit of
20
21
                     IO arrives. The library will execute the callback whenever the specified number of bytes
22
                     becomes available. Any remaining buffered data will be "flushed" upon call to deregister the
                     respective channel.
23
24
               PMIX_IOF_BUFFERING_TIME "pmix.iof.btime" (uint32_t)
25
                     Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering
26
                     size, this prevents IO from being held indefinitely while waiting for another payload to arrive.
27
               PMIX IOF COMPLETE "pmix.iof.cmp" (bool)
                     Indicates whether or not the specified IO channel has been closed by the source.
28
               PMIX_IOF_PUSH_STDIN "pmix.iof.stdin" (bool)
29
30
                     Used by a tool to request that the PMIx library collect the tool's stdin and forward it to the
                     processes specified in the PMIx_IOF_push call
31
               PMIX_IOF_STOP "pmix.iof.stop" (bool)
32
                     Stop forwarding the specified channel(s)
33
               PMIX_IOF_TAG_OUTPUT "pmix.iof.tag" (bool)
34
35
                     Tag output with the channel it comes from.
               PMIX IOF TIMESTAMP OUTPUT "pmix.iof.ts" (bool)
36
                     Timestamp output
37
               PMIX_IOF_XML_OUTPUT "pmix.iof.xml" (bool)
38
39
                     Format output in XML
               PMIX_IOF_PUSH_STDIN "pmix.iof.stdin" (()
40
```

1		b
2		ool) Used by a tool to request that the PMIx library collect the tool's stdin and forward it to the
3		processes specified in the PMIx_IOF_push call
4	3.4.30	Application setup attributes
5	PMIx v3.0	Attributes for controlling contents of application setup data
6		PMIX_SETUP_APP_ENVARS "pmix.setup.env" (bool)
7		Harvest and include relevant environmental variables
8		PMIX_SETUP_APP_NONENVARS ""pmix.setup.nenv" (bool)
9		Include all relevant data other than environmental variables
0		PMIX_SETUP_APP_ALL "pmix.setup.all" (bool)
11		Include all relevant data
12	3.4.31	Attribute support level attributes
13		PMIX_CLIENT_FUNCTIONS "pmix.client.fns" (bool)
14		Request a list of functions supported by the PMIx client library
15		PMIX_CLIENT_ATTRIBUTES "pmix.client.attrs" (bool)
16		Request attributes supported by the PMIx client library
17		PMIX_SERVER_FUNCTIONS "pmix.srvr.fns" (bool)
18		Request a list of functions supported by the PMIx server library
19		PMIX_SERVER_ATTRIBUTES "pmix.srvr.attrs" (bool)
20		Request attributes supported by the PMIx server library
21		PMIX_HOST_FUNCTIONS "pmix.srvr.fns" (bool)
22		Request a list of functions supported by the host environment
23		PMIX_HOST_ATTRIBUTES "pmix.host.attrs" (bool)
24		Request attributes supported by the host environment
25		PMIX_TOOL_FUNCTIONS "pmix.tool.fns" (bool)
26		Request a list of functions supported by the PMIx tool library
27		PMIX_TOOL_ATTRIBUTES "pmix.setup.env" (bool)
28		Request attributes supported by the PMIx tool library functions
29	3.4.32	Descriptive attributes
30		PMIX_MAX_VALUE "pmix.descr.maxval" (varies)
31		Used in pmix_regattr_t to describe the maximum valid value for the associated
32		attribute.
33		<pre>PMIX_MIN_VALUE "pmix.descr.minval" (varies)</pre>
34		Used in pmix_regattr_t to describe the minimum valid value for the associated
35		attribute.
36		<pre>PMIX_ENUM_VALUE "pmix.descr.enum" (char*)</pre>
37		Used in pmix_regattr_t to describe accepted values for the associated attribute.
88		Numerical values shall be presented in a form convertible to the attribute's declared data
39		type. Named values (i.e., values defined by constant names via a typical C-language enum
10		declaration) must be provided as their numerical equivalent.

1 3.4.33 Process group attributes

40

```
2 PMIx v4.0
               Attributes for controlling the PMIx Group APIs
 3
               PMIX GROUP ID "pmix.grp.id" (char*)
                     User-provided group identifier
 4
 5
               PMIX_GROUP_LEADER "pmix.grp.ldr" (bool)
 6
                     This process is the leader of the group
 7
               PMIX GROUP OPTIONAL "pmix.grp.opt" (bool)
                     Participation is optional - do not return an error if any of the specified processes terminate
 8
 9
                     without having joined. The default is false
10
               PMIX GROUP NOTIFY TERMINATION "pmix.grp.notterm" (bool)
                     Notify remaining members when another member terminates without first leaving the group.
11
12
                     The default is false
13
               PMIX_GROUP_INVITE_DECLINE "pmix.grp.decline" (bool)
14
                     Notify the inviting process that this process does not wish to participate in the proposed
15
                     group The default is true
               PMIX_GROUP_FT_COLLECTIVE "pmix.grp.ftcoll" (bool)
16
                     Adjust internal tracking for terminated processes. Default is false
17
18
               PMIX GROUP MEMBERSHIP "pmix.grp.mbrs" (pmix data array t*)
19
                     Array of group member ID's
20
               PMIX_GROUP_ASSIGN_CONTEXT_ID "pmix.grp.actxid" (bool)
                     Requests that the RM assign a new context identifier to the newly created group. The
21
22
                     identifier is an unsigned, size t value that the RM guarantees to be unique across the range
23
                     specified in the request. Thus, the value serves as a means of identifying the group within
24
                     that range. If no range is specified, then the request defaults to PMIX RANGE SESSION.
               PMIX_GROUP_CONTEXT_ID "pmix.grp.ctxid" (size_t)
25
26
                     Context identifier assigned to the group by the host RM.
27
               PMIX_GROUP_LOCAL_ONLY "pmix.grp.lcl" (bool)
                     Group operation only involves local processes. PMIx implementations are required to
28
29
                     automatically scan an array of group members for local vs remote processes - if only local
30
                     processes are detected, the implementation need not execute a global collective for the
31
                     operation unless a context ID has been requested from the host environment. This can result
                     in significant time savings. This attribute can be used to optimize the operation by indicating
32
                     whether or not only local processes are represented, thus allowing the implementation to
33
34
                     bypass the scan. The default is false
35
               PMIX_GROUP_ENDPT_DATA "pmix.grp.endpt" (pmix_byte_object_t)
                     Data collected to be shared during group construction
36
    3.4.34
                Storage-related attributes
38 PMIx v4.0
               Attributes related to storage systems
39
               PMIX STORAGE ID "pmix.strg.id" (char*)
```

Identifier of the storage system being referenced

```
1
               PMIX_STORAGE_PATH "pmix.strg.path" (char*)
2
                    Mount point corresponding to a specified storage ID
3
               PMIX_STORAGE_TYPE "pmix.strg.type" (char*)
4
                    Qualifier indicating the type of storage being referenced by a query e.g., lustre, gpfs, online,
5
                    fabric-attached, ...)
6
               PMIX STORAGE BW "pmix.strq.bw" (float)
7
                    Overall bandwidth (in Megabytes[base2]/sec) of specified storage system
               PMIX STORAGE AVAIL BW "pmix.strg.availbw" (float)
8
9
                    Overall bandwidth (in Megabytes[base2]/sec) of specified storage system that is available for
                    use by the calling program
10
               PMIX_STORAGE_OBJECT_LIMIT "pmix.strg.obj" (uint64_t)
11
                    Overall limit on number of objects (e.g., inodes) of specified storage system
12
               PMIX_STORAGE_OBJECTS_FREE "pmix.strg.objf" (uint64_t)
13
14
                    Number of free objects (e.g., inodes) on specified storage system
               PMIX STORAGE OBJECTS AVAIL "pmix.strg.obja" (uint64 t)
15
                    Number of objects (e.g., inodes) on specified storage system that are available for use by the
16
17
                    calling program
18
               PMIX STORAGE CAPACITY LIMIT "pmix.strg.cap" (uint64 t)
19
                    Overall capacity (in Megabytes[base2]) of specified storage system
               PMIX_STORAGE_CAPACITY_FREE "pmix.strg.free" (uint64_t)
20
                    Free capacity (in Megabytes[base2]) of specified storage system
21
22
               PMIX_STORAGE_CAPACITY_AVAIL "pmix.strg.avail" (uint64_t)
23
                    Capacity (in Megabytes[[base2]]) of specified storage system that is available for use by the
24
                    calling program
25
               PMIX QUERY STORAGE LIST "pmix.strg.list" (char*)
26
                    Return comma-delimited list of identifiers for all available storage systems
```

7 3.5 Callback Functions

PMIx provides blocking and nonblocking versions of most APIs. In the nonblocking versions, a callback is activated upon completion of the operation. This section describes many of those callbacks.

3.5.1 Release Callback Function

Summary

The pmix_release_cbfunc_t is used by the pmix_modex_cbfunc_t and pmix_info_cbfunc_t operations to indicate that the callback data may be reclaimed/freed by the caller.

28

29

30

32 33

34

```
Format
1
   PMIx v1.0
2
              typedef void (*pmix_release_cbfunc_t)
 3
                    (void *cbdata)
              INOUT cbdata
4
                   Callback data passed to original API call (memory reference)
 5
              Description
6
              Since the data is "owned" by the host server, provide a callback function to notify the host server
 7
8
              that we are done with the data so it can be released.
   3.5.2
            Modex Callback Function
              Summary
10
11
              The pmix_modex_cbfunc_t is used by the pmix_server_fencenb_fn_t and
12
              pmix_server_dmodex_req_fn_t PMIx server operations to return modex business card
              exchange (BCX) data.
13
   PMIx v1.0
14
              typedef void (*pmix modex cbfunc t)
                    (pmix_status_t status,
15
                    const char *data, size t ndata,
16
                    void *cbdata,
17
                    pmix release cbfunc t release fn,
18
19
                    void *release cbdata)
              IN
20
                  status
                   Status associated with the operation (handle)
21
22
              IN
                   data
23
                   Data to be passed (pointer)
              IN
                  ndata
24
                   size of the data (size_t)
25
                  cbdata
26
27
                   Callback data passed to original API call (memory reference)
28
              IN release fn
                   Callback for releasing data (function pointer)
29
30
              IN
                   release cbdata
31
                   Pointer to be passed to release_fn (memory reference)
```

Description

A callback function that is solely used by PMIx servers, and not clients, to return modex BCX data in response to "fence" and "get" operations. The returned blob contains the data collected from each server participating in the operation.

3.5.3 Spawn Callback Function

Summary

The pmix_spawn_cbfunc_t is used on the PMIx client side by PMIx_Spawn_nb and on the PMIx server side by pmix_server_spawn_fn_t.

PMIx v1.0

1

3

4

5

6

7

8

9

10

11

13

14

15

16 17

18 19

20 21

22 23

24

25

26

27

28

29

30 31

32

33

12 **IN** status

Status associated with the operation (handle)

IN nspace

Namespace string (pmix_nspace_t)

IN cbdata

Callback data passed to original API call (memory reference)

Description

The callback will be executed upon launch of the specified applications in **PMIx_Spawn_nb**, or upon failure to launch any of them.

The *status* of the callback will indicate whether or not the spawn succeeded. The *nspace* of the spawned processes will be returned, along with any provided callback data. Note that the returned *nspace* value will not be protected by the PRI upon return from the callback function, so the receiver must copy it if it needs to be retained.

3.5.4 Op Callback Function

Summary

The pmix op cbfunc t is used by operations that simply return a status.

```
PMIx v1.0
```

IN status

Status associated with the operation (handle)

IN cbdata

Callback data passed to original API call (memory reference)

```
Description
1
2
               Used by a wide range of PMIx API's including PMIx Fence nb,
               pmix server client connected fn t, PMIx server register nspace. This
 3
 4
               callback function is used to return a status to an often nonblocking operation.
    3.5.5
             Lookup Callback Function
               Summary
6
 7
               The pmix_lookup_cbfunc_t is used by PMIx_Lookup_nb to return data.
   PMIx v1.0
8
               typedef void (*pmix_lookup_cbfunc_t)
9
                    (pmix status t status,
10
                     pmix_pdata_t data[], size_t ndata,
                     void *cbdata);
11
12
               IN
                   status
13
                   Status associated with the operation (handle)
14
15
                   Array of data returned ( pmix_pdata_t )
               IN ndata
16
17
                   Number of elements in the data array (size_t)
               IN
18
19
                   Callback data passed to original API call (memory reference)
20
               Description
               A callback function for calls to PMIx_Lookup_nb The function will be called upon completion
21
22
               of the command with the status indicating the success or failure of the request. Any retrieved data
               will be returned in an array of pmix_pdata_t structs. The namespace and rank of the process
23
24
               that provided each data element is also returned.
25
               Note that these structures will be released upon return from the callback function, so the receiver
26
               must copy/protect the data prior to returning if it needs to be retained.
    3.5.6 Value Callback Function
               Summary
28
               The pmix_value_cbfunc_t is used by PMIx_Get_nb to return data.
29
   PMIx v1.0
30
               typedef void (*pmix value cbfunc t)
                    (pmix_status_t status,
31
32
                     pmix value t *kv, void *cbdata);
```

```
IN
 1
                    status
 2
                    Status associated with the operation (handle)
3
               IN
                    Key/value pair representing the data ( pmix_value_t )
 4
                    cbdata
5
               IN
6
                    Callback data passed to original API call (memory reference)
7
               Description
8
               A callback function for calls to PMIx_Get_nb . The status indicates if the requested data was
9
               found or not. A pointer to the pmix value t structure containing the found data is returned.
10
               The pointer will be NULL if the requested data was not found.
    3.5.7
              Info Callback Function
               Summarv
12
13
               The pmix_info_cbfunc_t is a general information callback used by various APIs.
   PMIx v2.0
14
               typedef void (*pmix info cbfunc t)
15
                     (pmix status t status,
                     pmix_info_t info[], size_t ninfo,
16
                     void *cbdata,
17
                     pmix_release_cbfunc_t release_fn,
18
                     void *release cbdata);
19
20
               IN status
                    Status associated with the operation (pmix status t)
21
22
                  info
                    Array of pmix_info_t returned by the operation (pointer)
23
                   ninfo
24
               IN
25
                    Number of elements in the info array (size t)
               IN
26
                   cbdata
27
                    Callback data passed to original API call (memory reference)
               IN release fn
28
                    Function to be called when done with the info data (function pointer)
29
               IN
                    release_cbdata
30
                    Callback data to be passed to release_fn (memory reference)
31
               Description
32
               The status indicates if requested data was found or not. An array of pmix info t will contain
33
               the key/value pairs.
34
```

3.5.8 Event Handler Registration Callback Function

```
2
              The pmix_evhdlr_reg_cbfunc_t callback function.

    Advice to users -

              The PMIx ad hoc v1.0 Standard defined an error handler registration callback function with a
 3
              compatible signature, but with a different type definition function name
 4
 5
              (pmix errhandler reg cbfunc t). It was removed from the v2.0 Standard and is not included in this
              document to avoid confusion.
 6
   PMIx v2.0
7
              typedef void (*pmix_evhdlr_reg_cbfunc_t)
8
                    (pmix_status_t status,
                    size_t evhdlr_ref,
9
10
                    void *cbdata)
              IN status
11
12
                   Status indicates if the request was successful or not (pmix status t)
              IN evhdlr ref
13
                   Reference assigned to the event handler by PMIx — this reference * must be used to
14
15
                   deregister the err handler (size t)
              IN
16
17
                   Callback data passed to original API call (memory reference)
18
              Description
19
              Define a callback function for calls to PMIx_Register_event_handler Deprecated in v4.0
              in favor of pmix hdlr reg cbfunc t.
20
    3.5.9
             Notification Handler Completion Callback Function
              Summary
22
23
              The pmix_event_notification_cbfunc_fn_t is called by event handlers to indicate
              completion of their operations.
24
   PMIx v2.0
              typedef void (*pmix_event_notification_cbfunc_fn_t)
25
                    (pmix_status_t status,
26
                    pmix_info_t *results, size_t nresults,
27
                    pmix op cbfunc t cbfunc, void *thiscbdata,
28
                    void *notification cbdata);
29
```

1	IN	status
2		Status returned by the event handler's operation (pmix_status_t)
3	IN	results
4		Results from this event handler's operation on the event (pmix_info_t)
5	IN	nresults
6		Number of elements in the results array (size_t)
7	IN	cbfunc
8		<pre>pmix_op_cbfunc_t function to be executed when PMIx completes processing the</pre>
9		callback (function reference)
10	IN	thiscbdata
11		Callback data that was passed in to the handler (memory reference)
12	IN	cbdata
13		Callback data to be returned when PMIx executes cbfunc (memory reference)
14	Des	scription
15	Defi	ne a callback by which an event handler can notify the PMIx library that it has completed its
16	resp	onse to the notification. The handler is <i>required</i> to execute this callback so the library can
17	dete	rmine if additional handlers need to be called. The handler shall return
18	PMI	X_EVENT_ACTION_COMPLETE if no further action is required. The return status of each
19	even	t handler and any returned pmix_info_t structures will be added to the <i>results</i> array of
20	pmi	x_info_t passed to any subsequent event handlers to help guide their operation.

3.5.10 Notification Function

Summary

The pmix_notification_fn_t is called by PMIx to deliver notification of an event.

provided info array and execute any other required cleanup operations.

Advice to users -

If non-NULL, the provided callback function will be called to allow the event handler to release the

The PMIx *ad hoc* v1.0 Standard defined an error notification function with an identical name, but different signature than the v2.0 Standard described below. The *ad hoc* v1.0 version was removed from the v2.0 Standard is not included in this document to avoid confusion.

PMIx v2.0

21

22

24

25

26

27

	<pre>(size_t evhdlr_registration_id, pmix_status_t status, const pmix_proc_t *source, pmix_info_t info[], size_t ninfo, pmix_info_t results[], size_t nresults, pmix_event_notification_cbfunc_fn_t cbfunc, void *cbdata);</pre>
	C
IN	<pre>evhdlr_registration_id Registration number of the handler being called (size_t)</pre>
IN	status
	Status associated with the operation (pmix_status_t)
IN	source
	Identifier of the process that generated the event (pmix_proc_t). If the source is the SM
	then the nspace will be empty and the rank will be PMIX_RANK_UNDEF
IN	info
	Information describing the event (pmix_info_t). This argument will be NULL if no
	additional information was provided by the event generator.
IN	ninfo
	Number of elements in the info array (size_t)
IN	results
	Aggregated results from prior event handlers servicing this event (pmix_info_t). This
	argument will be NULL if this is the first handler servicing the event, or if no prior handlers provided results.
IN	nresults
114	Number of elements in the results array (size_t)
IN	cbfunc
	<pre>pmix_event_notification_cbfunc_fn_t callback function to be executed upon</pre>
	completion of the handler's operation and prior to handler return (function reference).
	cbdata
IN	CDOATA

processes. Thus, the info array may be NULL or may contain detailed information of the event. It is

the responsibility of the application to parse any provided info array for defined key-values if it so

desires.

L	\sim	I۱/I	\sim	tΩ	110	ers

1 Possible uses of the *info* array include:

- for the host RM to alert the process as to planned actions, such as aborting the session, in response to the reported event
- provide a timeout for alternative action to occur, such as for the application to request an alternate response to the event

For example, the RM might alert the application to the failure of a node that resulted in termination of several processes, and indicate that the overall session will be aborted unless the application requests an alternative behavior in the next 5 seconds. The application then has time to respond with a checkpoint request, or a request to recover from the failure by obtaining replacement nodes and restarting from some earlier checkpoint.

Support for these options is left to the discretion of the host RM. Info keys are included in the common definitions above but may be augmented by environment vendors.

Advice to PMIx server hosts —

On the server side, the notification function is used to inform the PMIx server library's host of a detected event in the PMIx server library. Events generated by PMIx clients are communicated to the PMIx server library, but will be relayed to the host via the

pmix_server_notify_event_fn_t function pointer, if provided.

17 3.5.11 Server Setup Application Callback Function

- The PMIx_server_setup_application callback function.
- **Summary**
- 20 Provide a function by which the resource manager can receive application-specific environmental variables and other setup data prior to launch of an application.

```
Format
1
   PMIx v2.0
2
              typedef void (*pmix_setup_application_cbfunc_t)(
 3
                                             pmix status t status,
 4
                                             pmix info t info[], size t ninfo,
                                             void *provided_cbdata,
5
6
                                             pmix_op_cbfunc_t cbfunc, void *cbdata)
              IN
7
                   status
                   returned status of the request ( pmix_status_t )
8
9
              IN
                  info
10
                   Array of info structures (array of handles)
11
                   Number of elements in the info array (integer)
12
13
                   provided_cbdata
14
                   Data originally passed to call to PMIx server setup application (memory
                   reference)
15
              IN
                   cbfunc
16
17
                   pmix_op_cbfunc_t function to be called when processing completed (function reference)
18
              IN
                   cbdata
19
                   Data to be passed to the cbfunc callback function (memory reference)
              Description
20
21
              Define a function to be called by the PMIx server library for return of application-specific setup
22
              data in response to a request from the host RM. The returned info array is owned by the PMIx
23
              server library and will be free'd when the provided cbfunc is called.
               Server Direct Modex Response Callback Function
    3.5.12
25
              The PMIx server dmodex request callback function.
26
              Summary
27
              Provide a function by which the local PMIx server library can return connection and other data
              posted by local application processes to the host resource manager.
28
              Format
29
   PMIx v1.0
30
              typedef void (*pmix_dmodex_response_fn_t)(pmix_status_t status,
31
                                             char *data, size t sz,
32
                                             void *cbdata);
```

IN 1 status Returned status of the request (pmix_status_t) 2 3 IN Pointer to a data "blob" containing the requested information (handle) 4 5 IN 6 Number of bytes in the *data* blob (integer) 7 IN cbdata 8 Data passed into the initial call to **PMIx_server_dmodex_request** (memory reference) 9 Description Define a function to be called by the PMIx server library for return of information posted by a local 10 application process (via PMIx_Put with subsequent PMIx_Commit) in response to a request 11 from the host RM. The returned data blob is owned by the PMIx server library and will be free'd 12 13 upon return from the function. 3.5.13 PMIx Client Connection Callback Function Summary 15 16 Callback function for incoming connection request from a local client 17 Format PMIx v1.018 typedef void (*pmix_connection_cbfunc_t)(int incoming sd, void *cbdata) 19 20 IN incoming_sd (integer) 21 IN cbdata 22 23 (memory reference) **Description** 24 25 Callback function for incoming connection requests from local clients - only used by host 26 environments that wish to directly handle socket connection requests. 3.5.14 PMIx Tool Connection Callback Function Summary 28 29 Callback function for incoming tool connections.

```
Format
1
   PMIx v2.0
2
               typedef void (*pmix_tool_connection_cbfunc_t)(
3
                                                     pmix_status_t status,
 4
                                                     pmix proc t *proc, void *cbdata)
5
               IN
                    status
6
                    pmix_status_t value (handle)
7
               IN
8
                    pmix_proc_t structure containing the identifier assigned to the tool (handle)
9
               IN
                  cbdata
                    Data to be passed (memory reference)
10
               Description
11
12
               Callback function for incoming tool connections. The host environment shall provide a
13
               namespace/rank identifier for the connecting tool.
                               Advice to PMIx server hosts ——
14
               It is assumed that rank=0 will be the normal assignment, but allow for the future possibility of a
15
               parallel set of tools connecting, and thus each process requiring a unique rank.
```

6 3.5.15 Credential callback function

17 **Summary**

18

Callback function to return a requested security credential

Format 1 PMIx v3.0 2 typedef void (*pmix credential cbfunc t)(3 pmix_status_t status, 4 pmix byte object t *credential, pmix_info_t info[], size_t ninfo, 5 6 void *cbdata) C IN status 7 8 pmix_status_t value (handle) 9 IN credential pmix_byte_object_t structure containing the security credential (handle) 10 IN 11 Array of provided by the system to pass any additional information about the credential - e.g., 12 the identity of the issuing agent. (handle) 13 14 IN ninfo 15 Number of elements in *info* (size t) IN cbdata 16 Object passed in original request (memory reference) 17 **Description** 18 19 Define a callback function to return a requested security credential. Information provided by the issuing agent can subsequently be used by the application for a variety of purposes. Examples 20 21 include: 22 • checking identified authorizations to determine what requests/operations are feasible as a means 23 to steering workflows • compare the credential type to that of the local SMS for compatibility 24 Advice to users 25 The credential is opaque and therefore understandable only by a service compatible with the issuer. The *info* array is owned by the PMIx library and is not to be released or altered by the receiving 26 27 party.

3.5.16 Credential validation callback function

Summary

Callback function for security credential validation

29

```
Format
 1
   PMIx v3.0
 2
                typedef void (*pmix validation cbfunc t) (
 3
                                                        pmix_status_t status,
                                                        pmix info t info[], size t ninfo,
 4
                                                        void *cbdata);
 5
                IN
 6
                     status
 7
                     pmix_status_t value (handle)
 8
                IN
                     info
 9
                     Array of pmix info t provided by the system to pass any additional information about
                     the authentication - e.g., the effective userid and group id of the certificate holder, and any
10
                     related authorizations (handle)
11
                IN
12
                    ninfo
                     Number of elements in info (size_t)
13
14
                IN
                     cbdata
                     Object passed in original request (memory reference)
15
                Description
16
17
                Define a validation callback function to indicate if a provided credential is valid, and any
18
                corresponding information regarding authorizations and other security matters.
                                                    Advice to users
19
                The precise contents of the array will depend on the host environment and its associated security
20
                system. At the minimum, it is expected (but not required) that the array will contain entries for the
21
                PMIX USERID and PMIX GRPID of the client described in the credential. The info array is
                owned by the PMIx library and is not to be released or altered by the receiving party.
22
```

3.5.17 IOF delivery function

Summary

24 25

Callback function for delivering forwarded IO to a process

```
Format
 1
   PMIx v3.0
 2
               typedef void (*pmix iof cbfunc t) (
 3
                                                      size t iofhdlr, pmix iof channel t channel,
 4
                                                      pmix proc t *source, char *payload,
                                                      pmix_info_t info[], size_t ninfo);
 5
               IN
                    iofhdlr
 6
 7
                    Registration number of the handler being invoked (size t)
 8
               IN
                    channel
 9
                    bitmask identifying the channel the data arrived on (pmix iof channel t)
10
               IN
                    source
                    Pointer to a pmix_proc_t identifying the namespace/rank of the process that generated the
11
12
                    data (char*)
13
               IN
                    payload
14
                    Pointer to character array containing the data.
               IN
                    info
15
                    Array of pmix_info_t provided by the source containing metadata about the payload.
16
                    This could include PMIX_IOF_COMPLETE (handle)
17
               IN
                    ninfo
18
19
                    Number of elements in info (size t)
               Description
20
21
               Define a callback function for delivering forwarded IO to a process. This function will be called
22
               whenever data becomes available, or a specified buffering size and/or time has been met.
                                                  Advice to users
23
               Multiple strings may be included in a given payload, and the payload may not be NULL terminated.
               The user is responsible for releasing the payload memory. The info array is owned by the PMIx
24
25
               library and is not to be released or altered by the receiving party.
```

26 3.5.18 IOF and Event registration function

Summary

Callback function for calls to register handlers, e.g., event notification and IOF requests.

27

```
Format
1
   PMIx v3.0
              typedef void (*pmix_hdlr_reg_cbfunc_t) (pmix_status_t status,
2
 3
                                                                size_t refid,
 4
                                                                void *cbdata);
              IN
5
                  status
6
                   PMIX SUCCESS or an appropriate error constant ( pmix status t )
7
              IN
8
                   reference identifier assigned to the handler by PMIx, used to deregister the handler (size_t)
9
              IN
                   cbdata
                   object provided to the registration call (pointer)
10
              Description
11
12
              Callback function for calls to register handlers, e.g., event notification and IOF requests.
   3.6
            Constant String Functions
14
              Provide a string representation for several types of values. Note that the provided string is statically
              defined and must NOT be free'd.
15
16
              Summary
17
              String representation of a pmix_status_t .
   PMIx v1.0
18
              const char*
              PMIx_Error_string(pmix_status_t status);
19
20
              Summary
              String representation of a pmix_proc_state_t.
21
   PMIx v2.0
22
              const char*
              PMIx Proc state string(pmix proc state t state);
23
```

```
Summary
1
2
             String representation of a pmix scope t.
  PMIx v2.0
3
             const char*
4
             PMIx Scope string(pmix scope t scope);
             Summary
5
             String representation of a pmix_persistence_t.
6
  PMIx v2.0
7
             const char*
             PMIx_Persistence_string(pmix_persistence_t persist);
8
             Summary
9
             String representation of a pmix_data_range_t .
10
  PMIx v2.0
11
             const char*
12
             PMIx Data range string(pmix data range t range);
             Summary
13
14
             String representation of a pmix_info_directives_t.
  PMIx v2.0
15
             const char*
             PMIx_Info_directives_string(pmix_info_directives_t directives);
16
                                                 C
17
             Summary
18
             String representation of a pmix_data_type_t.
  PMIx v2.0
19
             const char*
20
             PMIx Data type string(pmix data type t type);
```

```
Summary
1
2
            String representation of a pmix alloc directive t.
  PMIx v2.0
3
            const char*
            PMIx_Alloc_directive_string(pmix_alloc_directive_t directive);
4
            Summary
5
6
            String representation of a pmix_iof_channel_t.
  PMIx v3.0
7
            const char*
            PMIx_IOF_channel_string(pmix_iof_channel_t channel);
8
            Summary
9
            String representation of a pmix_job_state_t.
10
  PMIx v4.0
11
            const char*
12
            PMIx Job state string(pmix job state t state);
            Summary
13
14
            String representation of a pmix_job_state_t.
  PMIx v4.0
            const char*
15
            PMIx_Job_state_string(pmix_job_state_t state);
16
17
            Summary
18
            String representation of a PMIx attribute
  PMIx v4.0
19
            const char*
20
            PMIx Get attribute string(char *attributename);
```

1	Summary
2	Return the PMIx attribute name corresponding to the given attribute string
PMIx v4.0	C
3	const char*
4	<pre>PMIx_Get_attribute_name(char *attributestring);</pre>
	C
5	Summary
6	String representation of a pmix_link_state_t
PMIx v4.0	C
7	const char*
8	<pre>PMIx_Link_state_string(pmix_link_state_t state);</pre>
	C -

CHAPTER 4

Initialization and Finalization

The PMIx library is required to be initialized and finalized around the usage of most of the APIs. 1 The APIs that may be used outside of the initialized and finalized region are noted. All other APIs 2 must be used inside this region. 3 There are three sets of initialization and finalization functions depending upon the role of the 4 5 process in the PMIx universe. Each of these functional sets are described in this chapter. Note that a process can only call one of the init/finalize functional pairs - e.g., a process that calls the client 6 7 initialization function cannot also call the tool or server initialization functions, and must call the 8 corresponding client finalize. — Advice to users ————— 9 Processes that initialize as a server or tool automatically are given access to all client APIs. Server 10 initialization includes setting up the infrastructure to support local clients - thus, it necessarily includes overhead and an increased memory footprint. Tool initialization automatically searches for 11 a server to which it can connect — if declared as a *launcher*, the PMIx library sets up the required 12 "hooks" for other tools (e.g., debuggers) to attach to it. 13 14 **4.1** Query 15 The API defined in this section can be used by any PMIx process, regardless of their role in the PMIx universe. 16 4.1.1 PMIx Initialized Format 18 PMIx v1.0 int PMIx Initialized(void) 19 A value of 1 (true) will be returned if the PMIx library has been initialized, and 0 (false) otherwise. 20 21 The return value is an integer for historical reasons as that was the signature of prior PMI libraries.

Description 1 2 Check to see if the PMIx library has been initialized using any of the init functions: PMIx Init, 3 PMIx server init, or PMIx tool init. 4.1.2 PMIx Get_version 5 Summary Get the PMIx version information. 6 7 Format *PMIx v1.0* 8 const char* PMIx Get version(void) **Description** 9 Get the PMIx version string. Note that the provided string is statically defined and must *not* be 10 free'd. 11 Client Initialization and Finalization 13 Initialization and finalization routines for PMIx clients. Advice to users The PMIx ad hoc v1.0 Standard defined the PMIx Init function, but modified the function 14 signature in the v1.2 version. The ad hoc v1.0 version is not included in this document to avoid 15 16 confusion. 17 **4.2.1** PMIx_Init

18

19

Summary

Initialize the PMIx client library

1	Format C —
PMIx v1 2 3 4	pmix_status_t PMIx_Init(pmix_proc_t *proc,
5 6 7 8 9	<pre>INOUT proc proc structure (handle) IN info Array of pmix_info_t structures (array of handles) IN ninfo Number of element in the info array (size_t)</pre>
1	Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant. Optional Attributes
2	The following attributes are optional for implementers of PMIx libraries:
3 4 5	PMIX_USOCK_DISABLE "pmix.usock.disable" (bool) Disable legacy UNIX socket (usock) support If the library supports Unix socket connections, this attribute may be supported for disabling it.
6 7 8	PMIX_SOCKET_MODE "pmix.sockmode" (uint32_t) POSIX mode_t (9 bits valid) If the library supports socket connections, this attribute may be supported for setting the socket mode.
9 20 21 22	<pre>PMIX_SINGLE_LISTENER "pmix.sing.listnr" (bool) Use only one rendezvous socket, letting priorities and/or environment parameters select the active transport. If the library supports multiple methods for clients to connect to servers, this attribute may be supported for disabling all but one of them.</pre>
23 24 25 26	<pre>PMIX_TCP_REPORT_URI "pmix.tcp.repuri" (char*) If provided, directs that the TCP URI be reported and indicates the desired method of reporting: '-' for stdout, '+' for stderr, or filename. If the library supports TCP socket connections, this attribute may be supported for reporting the URI.</pre>
27 28 29	<pre>PMIX_TCP_IF_INCLUDE "pmix.tcp.ifinclude" (char*) Comma-delimited list of devices and/or CIDR notation to include when establishing the TCP connection. If the library supports TCP socket connections, this attribute may be supported for specifying the interfaces to be used.</pre>
31 32 33 34	PMIX_TCP_IF_EXCLUDE "pmix.tcp.ifexclude" (char*) Comma-delimited list of devices and/or CIDR notation to exclude when establishing the TCP connection. If the library supports TCP socket connections, this attribute may be supported for specifying the interfaces that are <i>not</i> to be used.

1 PMIX_TCP_IPV4_PORT "pmix.tcp.ipv4" (int) The IPv4 port to be used. If the library supports IPV4 connections, this attribute may be 2 supported for specifying the port to be used. 3 PMIX_TCP_IPV6_PORT "pmix.tcp.ipv6" (int) 4 The IPv6 port to be used. If the library supports IPV6 connections, this attribute may be 5 6 supported for specifying the port to be used. PMIX_TCP_DISABLE_IPV4 "pmix.tcp.disipv4" (bool) 7 8 Set to **true** to disable IPv4 family of addresses. If the library supports IPV4 connections, this attribute may be supported for disabling it. 9 10 PMIX TCP DISABLE IPV6 "pmix.tcp.disipv6" (bool) Set to **true** to disable IPv6 family of addresses. If the library supports IPV6 connections, 11 this attribute may be supported for disabling it. 12 PMIX_EVENT_BASE "pmix.evbase" (struct event_base *) 13 Pointer to libevent **base** to use in place of the internal progress thread. 14 PMIX_GDS_MODULE "pmix.gds.mod" (char*) 15 Comma-delimited string of desired modules. This attribute is specific to the PRI and 16 controls only the selection of GDS module for internal use by the process. Module selection 17 for interacting with the server is performed dynamically during the connection process. 18 **Description** 19 Initialize the PMIx client, returning the process identifier assigned to this client's application in the 20 provided pmix_proc_t struct. Passing a value of **NULL** for this parameter is allowed if the user 21 wishes solely to initialize the PMIx system and does not require return of the identifier at that time. 22 23 When called, the PMIx client shall check for the required connection information of the local PMIx server and establish the connection. If the information is not found, or the server connection fails, 24 then an appropriate error constant shall be returned. 25 26 If successful, the function shall return **PMIX_SUCCESS** and fill the *proc* structure (if provided) 27 with the server-assigned namespace and rank of the process within the application. In addition, all startup information provided by the resource manager shall be made available to the client process 28 29 via subsequent calls to PMIx_Get . The PMIx client library shall be reference counted, and so multiple calls to PMIx_Init are 30 31 allowed by the standard. Thus, one way for an application process to obtain its namespace and rank is to simply call **PMIx Init** with a non-NULL *proc* parameter. Note that each call to 32

PMIx Init must be balanced with a call to PMIx Finalize to maintain the reference count.

Each call to PMIx Init may contain an array of pmix info t structures passing directives to

33 34

35

the PMIx client library as per the above attributes.

¹http://libevent.org/

Multiple calls to PMIx_Init shall not include conflicting directives. The PMIx_Init function 1 2 will return an error when directives that conflict with prior directives are encountered. 4.2.2 PMIx Finalize Summary 4 Finalize the PMIx client library. 5 Format 6 PMIx v1.07 pmix status t 8 PMIx_Finalize(const pmix_info_t info[], size_t ninfo) 9 IN info Array of pmix info t structures (array of handles) 10 11 IN Number of element in the *info* array (size_t) 12 13 Returns **PMIX SUCCESS** or a negative value corresponding to a PMIx error constant. Optional Attributes The following attributes are optional for implementers of PMIx libraries: 14 PMIX_EMBED_BARRIER "pmix.embed.barrier" (bool) 15 Execute a blocking fence operation before executing the specified operation. For example, 16 17 **PMIx_Finalize** does not include an internal barrier operation by default. This attribute would direct **PMIx_Finalize** to execute a barrier as part of the finalize operation. 18 **Description** 19 20 Decrement the PMIx client library reference count. When the reference count reaches zero, the library will finalize the PMIx client, closing the connection with the local PMIx server and 21 22 releasing all internally allocated memory. Server Initialization and Finalization 24 Initialization and finalization routines for PMIx servers. 4.3.1 PMIx_server_init Summary 26

Initialize the PMIx server.

```
Format
1
   PMIx v1.0
2
              pmix status t
              PMIx server init(pmix server module t *module,
 3
                                  pmix info t info[], size t ninfo)
 4
              INOUT module
5
6
                  pmix server module t structure (handle)
7
8
                  Array of pmix info t structures (array of handles)
9
              IN ninfo
10
                  Number of elements in the info array (size t)
11
              Returns PMIX SUCCESS or a negative value corresponding to a PMIx error constant.
                                           Required Attributes
              ______
              The following attributes are required to be supported by all PMIx libraries:
12
              PMIX_SERVER_NSPACE "pmix.srv.nspace" (char*)
13
                   Name of the namespace to use for this PMIx server.
14
              PMIX SERVER RANK "pmix.srv.rank" (pmix rank t)
15
                   Rank of this PMIx server
16
              PMIX SERVER TMPDIR "pmix.srvr.tmpdir" (char*)
17
18
                   Top-level temporary directory for all client processes connected to this server, and where the
19
                   PMIx server will place its tool rendezvous point and contact information.
20
              PMIX_SYSTEM_TMPDIR "pmix.sys.tmpdir" (char*)
21
                   Temporary directory for this system, and where a PMIx server that declares itself to be a
22
                   system-level server will place a tool rendezvous point and contact information.
23
              PMIX SERVER TOOL SUPPORT "pmix.srvr.tool" (bool)
                   The host RM wants to declare itself as willing to accept tool connection requests.
24
25
              PMIX SERVER SYSTEM SUPPORT "pmix.srvr.sys" (bool)
26
                   The host RM wants to declare itself as being the local system server for PMIx connection
27
                         ______
```

	→ Optional Attributes
1	The following attributes are optional for implementers of PMIx libraries:
2 3 4	<pre>PMIX_USOCK_DISABLE "pmix.usock.disable" (bool) Disable legacy UNIX socket (usock) support If the library supports Unix socket connections, this attribute may be supported for disabling it.</pre>
5 6 7	PMIX_SOCKET_MODE "pmix.sockmode" (uint32_t) POSIX <i>mode_t</i> (9 bits valid) If the library supports socket connections, this attribute may be supported for setting the socket mode.
8 9 10 11	PMIX_TCP_REPORT_URI "pmix.tcp.repuri" (char*) If provided, directs that the TCP URI be reported and indicates the desired method of reporting: '-' for stdout, '+' for stderr, or filename. If the library supports TCP socket connections, this attribute may be supported for reporting the URI.
12 13 14 15	<pre>PMIX_TCP_IF_INCLUDE "pmix.tcp.ifinclude" (char*) Comma-delimited list of devices and/or CIDR notation to include when establishing the TCP connection. If the library supports TCP socket connections, this attribute may be supported for specifying the interfaces to be used.</pre>
16 17 18 19	<pre>PMIX_TCP_IF_EXCLUDE "pmix.tcp.ifexclude" (char*) Comma-delimited list of devices and/or CIDR notation to exclude when establishing the TCP connection. If the library supports TCP socket connections, this attribute may be supported for specifying the interfaces that are not to be used.</pre>
20 21 22	PMIX_TCP_IPV4_PORT "pmix.tcp.ipv4" (int) The IPv4 port to be used. If the library supports IPV4 connections, this attribute may be supported for specifying the port to be used.
23 24 25	PMIX_TCP_IPV6_PORT "pmix.tcp.ipv6" (int) The IPv6 port to be used. If the library supports IPV6 connections, this attribute may be supported for specifying the port to be used.
26 27 28	<pre>PMIX_TCP_DISABLE_IPV4 "pmix.tcp.disipv4" (bool) Set to true to disable IPv4 family of addresses. If the library supports IPV4 connections, this attribute may be supported for disabling it.</pre>
29 30 31	PMIX_TCP_DISABLE_IPV6 "pmix.tcp.disipv6" (bool) Set to true to disable IPv6 family of addresses. If the library supports IPV6 connections, this attribute may be supported for disabling it.
32 33	PMIX_SERVER_REMOTE_CONNECTIONS "pmix.srvr.remote" (bool) Allow connections from remote tools. Forces the PMIx server to not exclusively use

be supported for enabling or disabling it.

PMIX_EVENT_BASE "pmix.evbase" (struct event_base *)

34

35

Pointer to libevent² **event_base** to use in place of the internal progress thread. 1 2 Description 3 Initialize the PMIx server support library, and provide a pointer to a pmix_server_module_t structure containing the caller's callback functions. The array of pmix_info_t structs is used to 4 pass additional info that may be required by the server when initializing. For example, it may 5 include the PMIX_SERVER_TOOL_SUPPORT attribute, thereby indicating that the daemon is 6 7 willing to accept connection requests from tools. Advice to PMIx server hosts -8 Providing a value of **NULL** for the *module* argument is permitted, as is passing an empty *module* 9 structure. Doing so indicates that the host environment will not provide support for multi-node 10 operations such as PMIx Fence, but does intend to support local clients access to information. 4.3.2 PMIx server finalize Summary 12 Finalize the PMIx server library. 13 14 Format PMIx v1.0pmix_status_t 15 16 PMIx server finalize(void) 17 Returns **PMIX SUCCESS** or a negative value corresponding to a PMIx error constant. Description 18 19 Finalize the PMIx server support library, terminating all connections to attached tools and any local 20 clients. All memory usage is released.

²http://libevent.org/

CHAPTER 5

1

3

4 5

6 7

8

11

Key/Value Management

Management of key-value pairs in PMIx is a distributed responsibility. While the stated objective of the PMIx community is to eliminate collective operations, it is recognized that the traditional method of posting/exchanging data must be supported until that objective can be met. This method relies on processes to discover and post their local information which is collected by the local PMIx server library. Global exchange of the posted information is then executed via a collective operation performed by the host SMS servers. The PMIx_Put and PMIx_Commit APIs, plus an attribute directing PMIx_Fence to globally collect the data posted by processes, are provided for this purpose.

5.1 Setting and Accessing Key/Value Pairs

10 **5.1.1 PMIx Put**

Summary

```
12
               Push a key/value pair into the client's namespace.
               Format
13
   PMIx v1.0
14
               pmix_status_t
15
               PMIx_Put(pmix_scope_t scope,
16
                           const pmix key t key,
17
                          pmix value t *val)
               IN
                    scope
18
19
                    Distribution scope of the provided value (handle)
               IN
20
21
                    key ( pmix_key_t )
               IN
                    value
22
                    Reference to a pmix_value_t structure (handle)
23
               Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.
24
```

1	Description
2	Push a value int
3	locally until PM

 Push a value into the client's namespace. The client's PMIx library will cache the information locally until **PMIx_Commit** is called.

The provided *scope* is passed to the local PMIx server, which will distribute the data to other processes according to the provided scope. The **pmix_scope_t** values are defined in Section 3.2.10 on page 38. Specific implementations may support different scope values, but all implementations must support at least **PMIX_GLOBAL**.

The <code>pmix_value_t</code> structure supports both string and binary values. PMIx implementations will support heterogeneous environments by properly converting binary values between host architectures, and will copy the provided <code>value</code> into internal memory.

Advice to PMIx library implementers –

The PMIx server library will properly pack/unpack data to accommodate heterogeneous environments. The host SMS is not involved in this action. The *value* argument must be copied - the caller is free to release it following return from the function.

Advice to users

The value is copied by the PMIx client library. Thus, the application is free to release and/or modify the value once the call to **PMIx_Put** has completed.

Note that keys starting with a string of "pmix" are exclusively reserved for the PMIx standard and must not be used in calls to PMIx_Put. Thus, applications should never use a defined "PMIX_" attribute as the key in a call to PMIx_Put.

5.1.2 PMIx_Get

Summary

Retrieve a key/value pair from the client's namespace.

```
Format
1
   PMIx v1.0
2
               pmix status t
 3
               PMIx_Get(const pmix_proc_t *proc, const pmix_key_t key,
                           const pmix info t info[], size t ninfo,
 4
                           pmix_value_t **val)
5
               IN
6
                    proc
7
                    process reference (handle)
8
               IN
9
                    key to retrieve (pmix key t)
10
               IN
                  info
                    Array of info structures (array of handles)
11
               IN ninfo
12
13
                    Number of element in the info array (integer)
               OUT val
14
                    value (handle)
15
16
               Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.
                                               Required Attributes
17
               The following attributes are required to be supported by all PMIx libraries:
18
               PMIX OPTIONAL "pmix.optional" (bool)
19
                     Look only in the client's local data store for the requested value - do not request data from
20
                     the PMIx server if not found.
21
               PMIX_IMMEDIATE "pmix.immediate" (bool)
22
                     Specified operation should immediately return an error from the PMIx server if the requested
23
                     data cannot be found - do not request it from the host RM.
24
               PMIX_DATA_SCOPE "pmix.scope" (pmix_scope_t)
                     Scope of the data to be found in a PMIx Get call.
25
26
               PMIX SESSION INFO "pmix.ssn.info" (bool)
27
                     Return information about the specified session. If information about a session other than the
                     one containing the requesting process is desired, then the attribute array must contain a
28
                     PMIX_SESSION_ID attribute identifying the desired target.
29
30
               PMIX JOB INFO "pmix.job.info" (bool)
```

Return information about the specified job or namespace. If information about a job or namespace other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX_JOBID** or **PMIX_NSPACE** attribute identifying the desired target. Similarly, if information is requested about a job or namespace in a session other than the one containing the requesting process, then an attribute identifying the target session must be provided.

PMIX APP INFO "pmix.app.info" (bool)

Return information about the specified application. If information about an application other than the one containing the requesting process is desired, then the attribute array must contain a PMIX_APPNUM attribute identifying the desired target. Similarly, if information is requested about an application in a job or session other than the one containing the requesting process, then attributes identifying the target job and/or session must be provided.

PMIX_NODE_INFO "pmix.node.info" (bool)

Return information about the specified node. If information about a node other than the one containing the requesting process is desired, then the attribute array must contain either the **PMIX_NODEID** or **PMIX_HOSTNAME** attribute identifying the desired target.

PMIX_GET_STATIC_VALUES "pmix.get.static" (bool)

Request that any pointers in the returned value point directly to values in the key-value store and indicate that the address provided for the return value points to a statically defined memory location. Returned non-pointer values should therefore be copied directly into the provided memory. Pointers in the returned value should point directly to values in the key-value store. User is responsible for *not* releasing memory on any returned pointer value. Note that a return status of PMIX_ERR_GET_MALLOC_REQD indicates that direct pointers could not be supported - thus, the returned data contains allocated memory that the user must release.

------ Optional Attributes ------

The following attributes are optional for host environments:

PMIX_TIMEOUT "pmix.timeout" (int)

Time in seconds before the specified operation should time out (θ indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Advice to PMIx library implementers -

We recommend that implementation of the PMIX_TIMEOUT attribute be left to the host environment due to race condition considerations between delivery of the data by the host environment versus internal timeout in the PMIx server library. Implementers that choose to support PMIX_TIMEOUT directly in the PMIx server library must take care to resolve the race condition and should avoid passing PMIX_TIMEOUT to the host environment so that multiple competing timeouts are not created.

Description

Retrieve information for the specified *key* associated with the process identified in the given <code>pmix_proc_t</code>, returning a pointer to the value in the given address. In general, data posted by a process via <code>PMIx_Put</code> and data that refers directly to a process-related value must be retrieved by specifying the rank of the target process. All other information is retrievable using a rank of <code>PMIX_RANK_WILDCARD</code>, as illustrated in 5.1.5. See 3.4.10 for an explanation regarding use of the <code>level</code> attributes, and <code>PMIx_server_register_nspace</code> for a description of the available information.

This is a blocking operation - the caller will block until either the specified data becomes available from the specified rank in the *proc* structure, the operation times out should the **PMIX_TIMEOUT** attribute have been given, or the **PMIX_OPTIONAL** or the **PMIX_IMMEDIATE** conditions are met. The caller is responsible for freeing all memory associated with the returned *value* when no longer required.

The *info* array is used to pass user requests regarding the get operation.

21 5.1.3 PMIx Get nb

22 Summary

Nonblocking **PMIx_Get** operation.

Format

PMIx v1.0

,

	C
IN proc	
process refer	rence (handle)
IN key	
key to retrie	ve (string)
<pre>IN info</pre>	
•	o structures (array of handles)
IN ninfo	
	elements in the <i>info</i> array (integer)
IN cbfunc	
	action (function reference)
IN cbdata	
Data to be p	assed to the callback function (memory reference)
Returns one of the	e following:
will be returned	ss, indicating that the request is being processed by the host environment - result in the provided <i>cbfunc</i> . Note that the library must not invoke the callback o returning from the API.
	TION_SUCCEEDED , indicating that the request was immediately processed and ss - the $cbfunc$ will not be called
	onstant indicating either an error in the input or that the request was immediately failed - the <i>cbfunc</i> will <i>not</i> be called
If executed, the st constants:	atus returned in the provided callback function will be one of the following
• PMIX_SUCCE	SS The requested data has been returned
• PMIX_ERR_N	OT_FOUND The requested data was not available
• a non-zero PM	x error constant indicating a reason for the request's failure
~	Required Attributes
The following attr	ributes are required to be supported by all PMIx libraries:
	L "pmix.optional" (bool) in the client's local data store for the requested value - do not request data from erver if not found.
PMIX IMMEDIA	TE "pmix.immediate" (bool)

Specified operation should immediately return an error from the PMIx server if the requested

PMIX_DATA_SCOPE "pmix.scope" (pmix_scope_t)

Scope of the data to be found in a PMIx_Get call.

data cannot be found - do not request it from the host RM.

1 PMIX_SESSION_INFO "pmix.ssn.info" (bool) 2 Return information about the specified session. If information about a session other than the 3 one containing the requesting process is desired, then the attribute array must contain a PMIX SESSION_ID attribute identifying the desired target. 4 5 PMIX_JOB_INFO "pmix.job.info" (bool) 6 Return information about the specified job or namespace. If information about a job or 7 namespace other than the one containing the requesting process is desired, then the attribute 8 array must contain a PMIX JOBID or PMIX NSPACE attribute identifying the desired 9 target. Similarly, if information is requested about a job or namespace in a session other than the one containing the requesting process, then an attribute identifying the target session 10 11 must be provided. 12 PMIX APP INFO "pmix.app.info" (bool) Return information about the specified application. If information about an application other 13 than the one containing the requesting process is desired, then the attribute array must 14 contain a PMIX APPNUM attribute identifying the desired target. Similarly, if information is 15 requested about an application in a job or session other than the one containing the requesting 16 17 process, then attributes identifying the target job and/or session must be provided. 18 PMIX NODE INFO "pmix.node.info" (bool) Return information about the specified node. If information about a node other than the one 19 20 containing the requesting process is desired, then the attribute array must contain either the 21 **PMIX NODEID** or **PMIX HOSTNAME** attribute identifying the desired target. 22 PMIX_GET_STATIC_VALUES "pmix.get.static" (bool) Request that any pointers in the returned value point directly to values in the key-value store 23 and indicate that user takes responsibility for properly releasing memory on the returned 24 25 value (i.e., free'ing the value structure but not the pointer fields). Note that a return status of PMIX_ERR_GET_MALLOC_REQD indicates that direct pointers could not be supported -26 thus, the returned data contains allocated memory that the user must release. 27 Optional Attributes 28 The following attributes are optional for host environments that support this operation:

"pmix.timeout" (int)

the target process from ever exposing its data.

Time in seconds before the specified operation should time out (θ indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent

29

30

31 32 PMIX TIMEOUT

Advice to PMIx library implementers —

We recommend that implementation of the PMIX_TIMEOUT attribute be left to the host environment due to race condition considerations between delivery of the data by the host environment versus internal timeout in the PMIx server library. Implementers that choose to support PMIX_TIMEOUT directly in the PMIx server library must take care to resolve the race condition and should avoid passing PMIX_TIMEOUT to the host environment so that multiple competing timeouts are not created.

Description

The callback function will be executed once the specified data becomes available from the identified process and retrieved by the local server. See **PMIx_Get** for a full description.

5.1.4 PMIx_Store_internal

11 Summary

Store some data locally for retrieval by other areas of the proc.

Format

PMIx v1.0

1

2

4

5

6

7 8

9

12

13

14

15

16

17

18

19 20

21 22

23

24

25

26

27

C

IN proc

process reference (handle)

IN key

key to retrieve (string)

IN val

Value to store (handle)

Returns **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant.

Description

Store some data locally for retrieval by other areas of the proc. This is data that has only internal scope - it will never be "pushed" externally.

5.1.5 Accessing information: examples

This section provides examples illustrating methods for accessing information at various levels. The intent of the examples is not to provide comprehensive coding guidance, but rather to illustrate how PMIx_Get can be used to obtain information on a session, job, application, process, and node.

5.1.5.1 Session-level information

2

3

4

5

6 7

8

9

10

11

12

13 14

30

31

32

33

The **PMIx_Get** API does not include an argument for specifying the **session** associated with the information being requested. Information regarding the session containing the requestor can be obtained by the following methods:

- for session-level attributes (e.g., PMIX_UNIV_SIZE), specifying the requestor's namespace and a rank of PMIX_RANK_WILDCARD; or
- for non-specific attributes (e.g., PMIX_NUM_NODES), including the PMIX_SESSION_INFO
 attribute to indicate that the session-level information for that attribute is being requested

Example requests are shown below:

```
15
            pmix info t info;
16
            pmix value t *value;
            pmix status t rc;
17
            pmix proc t myproc, wildcard;
18
19
20
            /* initialize the client library */
21
            PMIx_Init(&myproc, NULL, 0);
22
23
            /* get the #slots in our session */
            PMIX_PROC_LOAD(&wildcard, myproc.nspace, PMIX_RANK_WILDCARD);
24
25
            rc = PMIx_Get(&wildcard, PMIX_UNIV_SIZE, NULL, 0, &value);
26
            /* get the #nodes in our session */
27
            PMIX INFO LOAD (&info, PMIX SESSION INFO, NULL, PMIX BOOL);
28
            rc = PMIx Get(&wildcard, PMIX NUM NODES, &info, 1, &value);
29
```

Information regarding a different session can be requested by either specifying the namespace and a rank of PMIX_RANK_WILDCARD for a process in the target session, or adding the PMIX_SESSION_ID attribute identifying the target session. In the latter case, the *proc* argument to PMIx_Get will be ignored:

```
1
             pmix info t info[2];
2
             pmix value t *value;
3
             pmix_status_t rc;
4
             pmix_proc_t myproc;
5
             uint32_t sid;
6
7
             /* initialize the client library */
8
             PMIx_Init(&myproc, NULL, 0);
9
10
             /* get the #nodes in a different session */
             sid = 12345;
11
             PMIX INFO LOAD (&info[0], PMIX SESSION INFO, NULL, PMIX BOOL);
12
13
             PMIX INFO LOAD(&info[1], PMIX SESSION ID, &sid, PMIX UINT32);
             rc = PMIx_Get(&myproc, PMIX_NUM_NODES, info, 2, &value);
14
   5.1.5.2
            Job-level information
15
16
             Information regarding a job can be obtained by the following methods:
17
             • for job-level attributes (e.g., PMIX_JOB_SIZE or PMIX_JOB_NUM_APPS), specifying the
               namespace of the job and a rank of PMIX_RANK_WILDCARD for the proc argument to
18
               PMIx Get; or
19
20
             • for non-specific attributes (e.g., PMIX NUM NODES), including the PMIX JOB INFO
21
               attribute to indicate that the job-level information for that attribute is being requested
22
             Example requests are shown below:
23
             pmix_info_t info;
24
             pmix value t *value;
25
             pmix status t rc;
26
             pmix_proc_t myproc, wildcard;
27
28
             /* initialize the client library */
             PMIx_Init(&myproc, NULL, 0);
29
30
31
             /* get the #apps in our job */
             PMIX_PROC_LOAD(&wildcard, myproc.nspace, PMIX_RANK_WILDCARD);
32
33
             rc = PMIx_Get(&wildcard, PMIX_JOB_NUM_APPS, NULL, 0, &value);
34
35
             /* get the #nodes in our job */
36
             PMIX_INFO_LOAD(&info, PMIX_JOB_INFO, NULL, PMIX_BOOL);
37
             rc = PMIx Get(&wildcard, PMIX NUM NODES, &info, 1, &value);
```

5.1.5.3 Application-level information

2

4 5

6

7 8

9

10

11 12

13 14

37

Information regarding an application can be obtained by the following methods:

- for application-level attributes (e.g., **PMIX_APP_SIZE**), specifying the namespace and rank of a process within that application;
- for application-level attributes (e.g., PMIX_APP_SIZE), including the PMIX_APPNUM
 attribute specifying the application whose information is being requested. In this case, the
 namespace field of the *proc* argument is used to reference the job containing the application the rank field is ignored;
- or application-level attributes (e.g., PMIX_APP_SIZE), including the PMIX_APPNUM and PMIX_NSPACE or PMIX_JOBID attributes specifying the job/application whose information is being requested. In this case, the *proc* argument is ignored;
- for non-specific attributes (e.g., **PMIX_NUM_NODES**), including the **PMIX_APP_INFO** attribute to indicate that the application-level information for that attribute is being requested

Example requests are shown below:

```
15
            pmix info t info;
16
            pmix_value_t *value;
17
            pmix_status_t rc;
18
           pmix_proc_t myproc, otherproc;
            uint32_t appsize, appnum;
19
20
21
            /* initialize the client library */
22
            PMIx_Init(&myproc, NULL, 0);
23
            /* get the #processes in our application */
24
            rc = PMIx Get(&myproc, PMIX APP SIZE, NULL, 0, &value);
25
            appsize = value->data.uint32;
26
27
            /* get the #nodes in an application containing "otherproc".
28
29
             * Note that the rank of a process in the other application
             * must be obtained first - a simple method is shown here */
30
31
            /* assume for this example that we are in the first application
32
             * and we want the #nodes in the second application - use the
33
34
             * rank of the first process in that application, remembering
             * that ranks start at zero */
35
36
            PMIX_PROC_LOAD(&otherproc, myproc.nspace, appsize);
```

```
1
           PMIX_INFO_LOAD(&info, PMIX_APP_INFO, NULL, PMIX_BOOL);
2
           rc = PMIx Get(&otherproc, PMIX NUM NODES, &info, 1, &value);
3
4
           /* alternatively, we can directly ask for the #nodes in
5
             * the second application in our job, again remembering that
6
             * application numbers start with zero */
7
           appnum = 1;
8
           PMIX_INFO_LOAD(&appinfo[0], PMIX_APP_INFO, NULL, PMIX_BOOL);
9
           PMIX INFO LOAD (&appinfo[1], PMIX APPNUM, &appnum, PMIX UINT32);
10
           rc = PMIx_Get(&myproc, PMIX_NUM_NODES, appinfo, 2, &value);
11
```

C

5.1.5.4 Process-level information

Process-level information is accessed by providing the namespace and rank of the target process. In the absence of any directive as to the level of information being requested, the PMIx library will always return the process-level value.

16 5.1.5.5 Node-level information

12 13

14

15

17

18

19 20

21

22

23

24 25

37

Information regarding a node within the system can be obtained by the following methods:

- for node-level attributes (e.g., **PMIX_NODE_SIZE**), specifying the namespace and rank of a process executing on the target node;
- for node-level attributes (e.g., PMIX_NODE_SIZE), including the PMIX_NODEID or PMIX_HOSTNAME attribute specifying the node whose information is being requested. In this case, the *proc* argument's values are ignored; or
- for non-specific attributes (e.g., PMIX_MAX_PROCS), including the PMIX_NODE_INFO
 attribute to indicate that the node-level information for that attribute is being requested

Example requests are shown below:

```
26
            pmix info t info[2];
27
            pmix_value_t *value;
            pmix_status_t rc;
28
            pmix_proc_t myproc, otherproc;
29
30
            uint32_t nodeid;
31
32
            /* initialize the client library */
33
            PMIx_Init(&myproc, NULL, 0);
34
35
            /* get the #procs on our node */
            rc = PMIx Get(&myproc, PMIX NODE SIZE, NULL, 0, &value);
36
```

```
/* get the #slots on another node */

PMIX_INFO_LOAD(&info[0], PMIX_NODE_INFO, NULL, PMIX_BOOL);

PMIX_INFO_LOAD(&info[1], PMIX_HOSTNAME, "remotehost", PMIX_STRING);

rc = PMIx_Get(&myproc, PMIX_MAX_PROCS, info, 2, &value);

Advice to users

An explanation of the use of PMIx_Get versus PMIx_Query_info_nb is provided in 7.1.4.1.
```

5.2 Exchanging Key/Value Pairs

The APIs defined in this section push key/value pairs from the client to the local PMIx server, and circulate the data between PMIx servers for subsequent retrieval by the local clients.

10 5.2.1 PMIx Commit

11 Summary

Push all previously **PMIx_Put** values to the local PMIx server.

Format

PMIx v1.0

12

15

16

17

18

19

20

21

Returns **PMIX SUCCESS** or a negative value corresponding to a PMIx error constant.

Description

This is an asynchronous operation. The PRI will immediately return to the caller while the data is transmitted to the local server in the background.

Advice to users -

The local PMIx server will cache the information locally - i.e., the committed data will not be circulated during PMIx_Commit . Availability of the data upon completion of PMIx_Commit is therefore implementation-dependent.

5.2.2 PMIx Fence 2 Summary 3 Execute a blocking barrier across the processes identified in the specified array, collecting information posted via PMIx_Put as directed. 4 Format 5 *PMIx v1.0* pmix status t 6 7 PMIx_Fence(const pmix_proc_t procs[], size_t nprocs, const pmix info t info[], size t ninfo) 8 — C -9 IN procs 10 Array of pmix proc t structures (array of handles) IN 11 Number of element in the *procs* array (integer) 12 info 13 14 Array of info structures (array of handles) 15 IN ninfo Number of element in the *info* array (integer) 16 17 Returns **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant. _____ Required Attributes 18 The following attributes are required to be supported by all PMIx libraries: 19 PMIX COLLECT DATA "pmix.collect" (bool) Collect data and return it at the end of the operation. 20 **A**-----**-**Optional Attributes -----_____ 21 The following attributes are optional for host environments: 22 PMIX_TIMEOUT "pmix.timeout" (int) Time in seconds before the specified operation should time out (0 indicating infinite) in 23 error. The timeout parameter can help avoid "hangs" due to programming errors that prevent 24 the target process from ever exposing its data. 25 PMIX COLLECTIVE ALGO "pmix.calgo" (char*) 26 Comma-delimited list of algorithms to use for the collective operation. PMIx does not 27 28 impose any requirements on a host environment's collective algorithms. Thus, the acceptable values for this attribute will be environment-dependent - users are encouraged to 29 30 check their host environment for supported values.

31

PMIX COLLECTIVE ALGO REQD "pmix.calregd" (bool)

If **true**, indicates that the requested choice of algorithm is mandatory. 1 Advice to PMIx library implementers -2 We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host 3 environment due to race condition considerations between completion of the operation versus 4 internal timeout in the PMIx server library. Implementers that choose to support PMIX TIMEOUT 5 directly in the PMIx server library must take care to resolve the race condition and should avoid 6 passing PMIX TIMEOUT to the host environment so that multiple competing timeouts are not 7 created. **Description** 8 9 Passing a **NULL** pointer as the *procs* parameter indicates that the fence is to span all processes in 10 the client's namespace. Each provided pmix proc t struct can pass PMIX RANK WILDCARD to indicate that all processes in the given namespace are participating. 11 12 The *info* array is used to pass user requests regarding the fence operation. 13 Note that for scalability reasons, the default behavior for PMIx Fence is to not collect the data. Advice to PMIx library implementers -14 **PMIx_Fence** and its non-blocking form are both *collective* operations. Accordingly, the PMIx server library is required to aggregate participation by local clients, passing the request to the host 15 environment once all local participants have executed the API. 16 Advice to PMIx server hosts — 17 The host will receive a single call for each collective operation. It is the responsibility of the host to 18 identify the nodes containing participating processes, execute the collective across all participating nodes, and notify the local PMIx server library upon completion of the global collective. 19 5.2.3 PMIx Fence nb Summary 21

Execute a nonblocking **PMIx_Fence** across the processes identified in the specified array of processes, collecting information posted via **PMIx_Put** as directed.

1		Format
P	PMIx v1.0	· · · · · · · · · · · · · · · · · · ·
2		pmix_status_t
3		<pre>PMIx_Fence_nb(const pmix_proc_t procs[], size_t nprocs,</pre>
4		<pre>const pmix_info_t info[], size_t ninfo,</pre>
5		<pre>pmix_op_cbfunc_t cbfunc, void *cbdata)</pre>
		C
6		IN procs
7		Array of pmix_proc_t structures (array of handles)
8		IN nprocs
9		Number of element in the <i>procs</i> array (integer)
10		IN info
11		Array of info structures (array of handles)
12		IN ninfo
13		Number of element in the <i>info</i> array (integer)
14		IN cbfunc
15		Callback function (function reference)
16		IN cbdata
17		Data to be passed to the callback function (memory reference)
18		Returns one of the following:
19		• PMIX_SUCCESS, indicating that the request is being processed by the host environment - result
20		will be returned in the provided <i>cbfunc</i> . Note that the library must not invoke the callback
21		function prior to returning from the API.
22		• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
23		returned <i>success</i> - the <i>cbfunc</i> will <i>not</i> be called. This can occur if the collective involved only
24		processes on the local node.
25		• a PMIx error constant indicating either an error in the input or that the request was immediately
26		processed and failed - the <i>cbfunc</i> will <i>not</i> be called
		·
		Required Attributes
27		The following attributes are required to be supported by all PMIx libraries:
28		PMIX_COLLECT_DATA "pmix.collect" (bool)
29		Collect data and return it at the end of the operation.
		A

	Optional Attributes
--	---------------------

The following attributes are optional for host environments that support this operation:

PMIX_TIMEOUT "pmix.timeout" (int)

Time in seconds before the specified operation should time out (θ indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

PMIX COLLECTIVE ALGO "pmix.calgo" (char*)

Comma-delimited list of algorithms to use for the collective operation. PMIx does not impose any requirements on a host environment's collective algorithms. Thus, the acceptable values for this attribute will be environment-dependent - users are encouraged to check their host environment for supported values.

PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)

If **true**, indicates that the requested choice of algorithm is mandatory.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Note that PMIx libraries may choose to implement an optimization for the case where only the calling process is involved in the fence operation by immediately returning
PMIX_OPERATION_SUCCEEDED from the client's call in lieu of passing the fence operation to a
PMIx server. Fence operations involving more than just the calling process must be communicated to the PMIx server for proper execution of the included barrier behavior.

Similarly, fence operations that involve only processes that are clients of the same PMIx server may be resolved by that server without referral to its host environment as no inter-node coordination is required.

Description

Nonblocking **PMIx_Fence** routine. Note that the function will return an error if a **NULL** callback function is given.

Note that for scalability reasons, the default behavior for **PMIx_Fence_nb** is to not collect the data.

See the **PMIx_Fence** description for further details.

5.3 Publish and Lookup Data

The APIs defined in this section publish data from one client that can be later exchanged and looked 2 3 up by another client. PMIx libraries that support any of the functions in this section are required to support all of them. 4 — Advice to PMIx server hosts — Host environments that support any of the functions in this section are required to support all of 5 6 5.3.1 PMIx Publish Summary 8 Publish data for later access via PMIx_Lookup. 9 **Format** 10 *PMIx v1.0* 11 pmix_status_t PMIx Publish(const pmix_info_t info[], size_t ninfo) 12 13 IN info 14 Array of info structures (array of handles) 15 IN ninfo 16 Number of element in the *info* array (integer) 17 Returns **PMIX SUCCESS** or a negative value corresponding to a PMIx error constant. Required Attributes _____ 18 PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is 19 20 required to add the PMIX_USERID and the PMIX_GRPID attributes of the client process that 21 published the info.

Optional Attributes The following attributes are optional for host environments that support this operation: 1 PMIX_TIMEOUT "pmix.timeout" (int) 3 Time in seconds before the specified operation should time out (θ indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent 4 the target process from ever exposing its data. 5 PMIX_RANGE "pmix.range" (pmix_data_range_t) 6 Value for calls to publish/lookup/unpublish or for monitoring event notifications. 7 8 PMIX_PERSISTENCE "pmix.persist" (pmix_persistence_t) Value for calls to PMIx Publish. 9 Advice to PMIx library implementers — 10 We recommend that implementation of the PMIX_TIMEOUT attribute be left to the host environment due to race condition considerations between completion of the operation versus 11 12 internal timeout in the PMIx server library. Implementers that choose to support PMIX_TIMEOUT 13 directly in the PMIx server library must take care to resolve the race condition and should avoid 14 passing **PMIX TIMEOUT** to the host environment so that multiple competing timeouts are not 15 created. **Description** 16 17 Publish the data in the *info* array for subsequent lookup. By default, the data will be published into 18 the PMIX RANGE SESSION range and with PMIX PERSIST APP persistence. Changes to 19 those values, and any additional directives, can be included in the pmix_info_t array. Attempts to access the data by processes outside of the provided data range will be rejected. The persistence 20 parameter instructs the server as to how long the data is to be retained. 21 22 The blocking form will block until the server confirms that the data has been sent to the PMIx server and that it has obtained confirmation from its host SMS daemon that the data is ready to be 23 looked up. Data is copied into the backing key-value data store, and therefore the *info* array can be 24 25 released upon return from the blocking function call. Advice to users -Publishing duplicate keys is permitted provided they are published to different ranges. 26 —— Advice to PMIx library implementers ———— Implementations should, to the best of their ability, detect duplicate keys being posted on the same 27 data range and protect the user from unexpected behavior by returning the 28 PMIX_ERR_DUPLICATE_KEY error. 29

5.3.2 PMIx Publish nb Summary 3 Nonblocking PMIx Publish routine. Format *PMIx v1.0* 5 pmix status t 6 PMIx Publish nb(const pmix info t info[], size t ninfo, 7 pmix op cbfunc t cbfunc, void *cbdata) IN info 8 9 Array of info structures (array of handles) 10 IN ninfo Number of element in the *info* array (integer) 11 IN cbfunc 12 Callback function **pmix_op_cbfunc_t** (function reference) 13 IN cbdata 14 15 Data to be passed to the callback function (memory reference) Returns one of the following: 16 17 • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result 18 will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API. 19 20 • PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and 21 returned success - the cbfunc will not be called 22 • a PMIx error constant indicating either an error in the input or that the request was immediately 23 processed and failed - the cbfunc will not be called Required Attributes -----24 PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is 25 required to add the PMIX_USERID and the PMIX_GRPID attributes of the client process that 26

27

published the info.

		▼ Optional Attributes
1		The following attributes are optional for host environments that support this operation:
2 3 4 5		PMIX_TIMEOUT "pmix.timeout" (int) Time in seconds before the specified operation should time out (0 indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.
6 7		<pre>PMIX_RANGE "pmix.range" (pmix_data_range_t) Value for calls to publish/lookup/unpublish or for monitoring event notifications.</pre>
8 9		<pre>PMIX_PERSISTENCE "pmix.persist" (pmix_persistence_t) Value for calls to PMIx_Publish .</pre>
		Advice to PMIx library implementers
10 11 12 13 14 15		We recommend that implementation of the PMIX_TIMEOUT attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support PMIX_TIMEOUT directly in the PMIx server library must take care to resolve the race condition and should avoid passing PMIX_TIMEOUT to the host environment so that multiple competing timeouts are not created.
16 17 18		Description Nonblocking PMIx_Publish routine. The non-blocking form will return immediately, executing the callback when the PMIx server receives confirmation from its host SMS daemon.
19 20		Note that the function will return an error if a NULL callback function is given, and that the <i>info</i> array must be maintained until the callback is provided.
21	5.3.3	PMIx_Lookup
22		Summary
23		Lookup information published by this or another process with PMIx_Publish or

PMIx_Publish_nb.

1	Format
PMIx v	
2	pmix_status_t
3	PMIx_Lookup(pmix_pdata_t data[], size_t ndata,
4	<pre>const pmix_info_t info[], size_t ninfo)</pre>
5	INOUT data
6	Array of publishable data structures (array of handles)
7	IN ndata
8	Number of elements in the <i>data</i> array (integer)
9	IN info
0	Array of info structures (array of handles)
1	IN ninfo
2	Number of elements in the <i>info</i> array (integer)
3	Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.
	▼
4	PMIx libraries are not required to directly support any attributes for this function. However, any
5	provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is
6	required to add the PMIX_USERID and the PMIX_GRPID attributes of the client process that is
7	requesting the info.
	A
	▼ Optional Attributes
8	The following attributes are optional for host environments that support this operation:
9	<pre>PMIX_TIMEOUT "pmix.timeout" (int)</pre>
0	Time in seconds before the specified operation should time out (θ indicating infinite) in
1	error. The timeout parameter can help avoid "hangs" due to programming errors that prevent
2	the target process from ever exposing its data.
3	<pre>PMIX_RANGE "pmix.range" (pmix_data_range_t)</pre>
4	Value for calls to publish/lookup/unpublish or for monitoring event notifications.
5	<pre>PMIX_WAIT "pmix.wait" (int)</pre>
6	Caller requests that the PMIx server wait until at least the specified number of values are
7	found (θ indicates all and is the default).
	A

Advice to PMIx library implementers -

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Lookup information published by this or another process. By default, the search will be conducted across the **PMIX_RANGE_SESSION** range. Changes to the range, and any additional directives, can be provided in the **pmix_info_t** array. Data is returned provided the following conditions are met:

- the requesting process resides within the range specified by the publisher. For example, data
 published to PMIX_RANGE_LOCAL can only be discovered by a process executing on the same
 node
- the provided key matches the published key within that data range
- the data was published by a process with corresponding user and/or group IDs as the one looking up the data. There currently is no option to override this behavior such an option may become available later via an appropriate pmix_info_t directive.

The *data* parameter consists of an array of <code>pmix_pdata_t</code> struct with the keys specifying the requested information. Data will be returned for each key in the associated *value* struct. Any key that cannot be found will return with a data type of <code>PMIX_UNDEF</code>. The function will return <code>PMIX_SUCCESS</code> if any values can be found, so the caller must check each data element to ensure it was returned.

The proc field in each **pmix_pdata_t** struct will contain the namespace/rank of the process that published the data.

Advice to users

Although this is a blocking function, it will not wait by default for the requested data to be published. Instead, it will block for the time required by the server to lookup its current data and return any found items. Thus, the caller is responsible for ensuring that data is published prior to executing a lookup, using **PMIX_WAIT** to instruct the server to wait for the data to be published, or for retrying until the requested data is found.

5.3.4 PMIx Lookup nb Summary 2 Nonblocking version of PMIx Lookup. 3 Format PMIx v1.0 5 pmix status t 6 PMIx Lookup nb(char **keys, 7 const pmix_info_t info[], size_t ninfo, pmix lookup cbfunc t cbfunc, void *cbdata) 8 9 IN keys 10 Array to be provided to the callback (array of strings) IN 11 Array of info structures (array of handles) 12 IN ninfo 13 14 Number of element in the *info* array (integer) 15 IN cbfunc Callback function (handle) 16 IN cbdata 17 Callback data to be provided to the callback function (pointer) 18 Returns one of the following: 19 20 • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result will be returned in the provided cbfunc. Note that the library must not invoke the callback 21 22 function prior to returning from the API. 23 • a PMIx error constant indicating an error in the input - the *cbfunc* will *not* be called Required Attributes

PMIx libraries are not required to directly support any attributes for this function. However, any

provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is *required* to add the **PMIX USERID** and the **PMIX GRPID** attributes of the client process that is

▲-----**-**

requesting the info.

24

25

26

Optional Attributes

The following attributes are optional for host environments that support this operation:

PMIX TIMEOUT "pmix.timeout" (int)

Time in seconds before the specified operation should time out (θ indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

PMIX RANGE "pmix.range" (pmix data range t)

Value for calls to publish/lookup/unpublish or for monitoring event notifications.

PMIX WAIT "pmix.wait" (int)

Caller requests that the PMIx server wait until at least the specified number of values are found (0 indicates all and is the default).

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Non-blocking form of the **PMIx_Lookup** function. Data for the provided NULL-terminated *keys* array will be returned in the provided callback function. As with **PMIx_Lookup**, the default behavior is to not wait for data to be published. The *info* array can be used to modify the behavior as previously described by **PMIx_Lookup**. Both the *info* and *keys* arrays must be maintained until the callback is provided.

3 5.3.5 PMIx_Unpublish

Summary

Unpublish data posted by this process using the given keys.

1		Format
F	PMIx v1.0	
2		pmix_status_t
3 4		PMIx_Unpublish(char **keys,
4		const pmix_info_t info[], size_t ninfo)
		O
5		IN info
6		Array of info structures (array of handles)
7 8		Number of element in the <i>info</i> array (integer)
9		Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.
		Required Attributes
10 11 12 13		PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is required to add the PMIX_USERID and the PMIX_GRPID attributes of the client process that is requesting the operation.
		▼Optional Attributes
14		The following attributes are optional for host environments that support this operation:
15 16 17 18		PMIX_TIMEOUT "pmix.timeout" (int) Time in seconds before the specified operation should time out (0 indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.
19 20		PMIX_RANGE "pmix.range" (pmix_data_range_t) Value for calls to publish/lookup/unpublish or for monitoring event notifications.
		Advice to PMIx library implementers —
21 22 23 24 25 26		We recommend that implementation of the PMIX_TIMEOUT attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support PMIX_TIMEOUT directly in the PMIx server library must take care to resolve the race condition and should avoid passing PMIX_TIMEOUT to the host environment so that multiple competing timeouts are not created.

Description 1 2 Unpublish data posted by this process using the given keys. The function will block until the data has been removed by the server (i.e., it is safe to publish that key again). A value of **NULL** for the 3 4 keys parameter instructs the server to remove all data published by this process. 5 By default, the range is assumed to be PMIX RANGE SESSION. Changes to the range, and any 6 additional directives, can be provided in the *info* array. 5.3.6 PMIx Unpublish nb **Summary** 8 Nonblocking version of PMIx_Unpublish. 9 **Format** 10 PMIx v1.011 pmix_status_t 12 PMIx Unpublish nb(char **keys, 13 const pmix_info_t info[], size_t ninfo, pmix_op_cbfunc_t cbfunc, void *cbdata) 14 IN 15 keys 16 (array of strings) 17 IN info 18 Array of info structures (array of handles) 19 IN ninfo Number of element in the *info* array (integer) 20 21 IN cbfunc 22 Callback function **pmix_op_cbfunc_t** (function reference) IN 23 cbdata Data to be passed to the callback function (memory reference) 24 25 Returns one of the following: 26 • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result will be returned in the provided cbfunc. Note that the library must not invoke the callback 27 function prior to returning from the API. 28 29 • PMIX OPERATION SUCCEEDED, indicating that the request was immediately processed and 30 returned success - the cbfunc will not be called 31

• a PMIx error constant indicating either an error in the input or that the request was immediately

processed and failed - the cbfunc will not be called

32

Required Attributes PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is required to add the PMIX USERID and the PMIX GRPID attributes of the client process that is requesting the operation. Optional Attributes The following attributes are optional for host environments that support this operation: PMIX TIMEOUT "pmix.timeout" (int) Time in seconds before the specified operation should time out (θ indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data. PMIX_RANGE "pmix.range" (pmix_data_range_t) Value for calls to publish/lookup/unpublish or for monitoring event notifications. ——— Advice to PMIx library implementers —— We recommend that implementation of the PMIX TIMEOUT attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support PMIX_TIMEOUT directly in the PMIx server library must take care to resolve the race condition and should avoid passing PMIX TIMEOUT to the host environment so that multiple competing timeouts are not created.

Description

Non-blocking form of the **PMIx_Unpublish** function. The callback function will be executed once the server confirms removal of the specified data. The *info* array must be maintained until the callback is provided.

1

2

5 6

7

8

9

10

11

12

13

14 15

16

17

18

19

20

CHAPTER 6

Process Management

1 This chapter defines functionality used by clients to create and destroy/abort processes in the PMIx universe.

3 **6.1 Abort**

PMIx provides a dedicated API by which an application can request that specified processes be aborted by the system.

6 6.1.1 PMIx Abort

```
Summary
 8
               Abort the specified processes
               Format
   PMIx v1.0
10
               pmix_status_t
               PMIx_Abort(int status, const char msg[],
11
                             pmix_proc_t procs[], size_t nprocs)
12
                                                    — С
               IN
13
                    Error code to return to invoking environment (integer)
14
15
               IN
16
                    String message to be returned to user (string)
               IN
                    procs
17
                    Array of pmix proc t structures (array of handles)
18
               IN
19
                    nprocs
                    Number of elements in the procs array (integer)
20
               Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.
21
```

Description

1 2

3

4

5

6

7

8

9

10 11

12

13

14

15

Request that the host resource manager print the provided message and abort the provided array of procs. A Unix or POSIX environment should handle the provided status as a return error code from the main program that launched the application. A **NULL** for the *procs* array indicates that all processes in the caller's namespace are to be aborted, including itself. Passing a **NULL** msg parameter is allowed.

Advice to users

The response to this request is somewhat dependent on the specific resource manager and its configuration (e.g., some resource managers will not abort the application if the provided status is zero unless specifically configured to do so, and some cannot abort subsets of processes in an application), and thus lies outside the control of PMIx itself. However, the PMIx client library shall inform the RM of the request that the specified procs be aborted, regardless of the value of the provided status.

Note that race conditions caused by multiple processes calling **PMIx Abort** are left to the server implementation to resolve with regard to which status is returned and what messages (if any) are printed.

6.2 **Process Creation**

- 17 The **PMIx_Spawn** commands spawn new processes and/or applications in the PMIx universe. 18 This may include requests to extend the existing resource allocation or obtain a new one, depending
- 19 upon provided and supported attributes.

6.2.1 PMIx Spawn

Summary 21 22

Spawn a new job.

I	Format
<i>PMIx v1.0</i>	
2	pmix_status_t
3	<pre>PMIx_Spawn(const pmix_info_t job_info[], size_t ninfo,</pre>
4	<pre>const pmix_app_t apps[], size_t napps,</pre>
5	char nspace[])
	C
6	<pre>IN job_info</pre>
7	Array of info structures (array of handles)
8	IN ninfo
9	Number of elements in the <i>job_info</i> array (integer)
0	IN apps
1	Array of pmix_app_t structures (array of handles)
2	IN napps
3	Number of elements in the <i>apps</i> array (integer)
4	OUT nspace
5	Namespace of the new job (string)
6	Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.
	▼ Required Attributes
7	PMIx libraries are not required to directly support any attributes for this function. However, any
8	provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is
9	required to add the following attributes to those provided before passing the request to the host:
20	PMIX_SPAWNED "pmix.spawned" (bool)
21	true if this process resulted from a call to PMIx_Spawn . Lack of inclusion (i.e., a return
22	status of PMIX_ERR_NOT_FOUND) corresponds to a value of false for this attribute.
23	<pre>PMIX_PARENT_ID "pmix.parent" (pmix_proc_t)</pre>
24	Process identifier of the parent process of the calling process.
25	PMIX_REQUESTOR_IS_CLIENT "pmix.req.client" (bool)
26	The requesting process is a PMIx client.
27	PMIX_REQUESTOR_IS_TOOL "pmix.req.tool" (bool)
28	The requesting process is a PMIx tool.
29	
80	Host environments that implement support for PMIx_Spawn are required to pass the
31	PMIX_SPAWNED and PMIX_PARENT_ID attributes to all PMIx servers launching new child
32	processes so those values can be returned to clients upon connection to the PMIx server. In
33	addition, they are required to support the following attributes when present in either the job_info or
34	the info array of an element of the apps array:

1 2	PMIX_WDIR "pmix.wdir" (char*) Working directory for spawned processes.
3 4 5 6	PMIX_SET_SESSION_CWD "pmix.ssncwd" (bool) Set the application's current working directory to the session working directory assigned by the RM - when accessed using PMIx_Get , use the PMIX_RANK_WILDCARD value for the rank to discover the session working directory assigned to the provided namespace
7 8	PMIX_PREFIX "pmix.prefix" (char*) Prefix to use for starting spawned processes.
9 10	PMIX_HOST "pmix.host" (char*) Comma-delimited list of hosts to use for spawned processes.
11 12	PMIX_HOSTFILE "pmix.hostfile" (char*) Hostfile to use for spawned processes.
	▼ Optional Attributes
13	The following attributes are optional for host environments that support this operation:
14 15	PMIX_ADD_HOSTFILE "pmix.addhostfile" (char*) Hostfile listing hosts to add to existing allocation.
16 17	PMIX_ADD_HOST "pmix.addhost" (char*) Comma-delimited list of hosts to add to the allocation.
18 19	PMIX_PRELOAD_BIN "pmix.preloadbin" (bool) Preload binaries onto nodes.
20 21	<pre>PMIX_PRELOAD_FILES "pmix.preloadfiles" (char*)</pre>
22 23	PMIX_PERSONALITY "pmix.pers" (char*) Name of personality to use.
24 25 26 27	PMIX_MAPPER "pmix.mapper" (char*) Mapping mechanism to use for placing spawned processes - when accessed using PMIx_Get , use the PMIX_RANK_WILDCARD value for the rank to discover the mapping mechanism used for the provided namespace.
28 29	PMIX_DISPLAY_MAP "pmix.dispmap" (bool) Display process mapping upon spawn.
30 31	PMIX_PPR "pmix.ppr" (char*) Number of processes to spawn on each identified resource.
32	PMIX_MAPBY "pmix.mapby" (char*)

1 2 3	Process mapping policy - when accessed using PMIx_Get , use the PMIX_RANK_WILDCARD value for the rank to discover the mapping policy used for the provided namespace
4 5 6 7	PMIX_RANKBY "pmix.rankby" (char*) Process ranking policy - when accessed using PMIx_Get, use the PMIX_RANK_WILDCARD value for the rank to discover the ranking algorithm used for the provided namespace
8 9 0 1	<pre>PMIX_BINDTO "pmix.bindto" (char*) Process binding policy - when accessed using PMIx_Get , use the PMIX_RANK_WILDCARD value for the rank to discover the binding policy used for the provided namespace</pre>
2 3	PMIX_NON_PMI "pmix.nonpmi" (bool) Spawned processes will not call PMIx_Init.
4 5	PMIX_STDIN_TGT "pmix.stdin" (uint32_t) Spawned process rank that is to receive any forwarded stdin.
6 7	<pre>PMIX_FWD_STDIN "pmix.fwd.stdin" (bool) Forward stdin from this process to the designated process.</pre>
8 9	PMIX_FWD_STDOUT "pmix.fwd.stdout" (bool) Forward stdout from spawned processes to this process.
20 21	<pre>PMIX_FWD_STDERR "pmix.fwd.stderr" (bool) Forward stderr from spawned processes to this process.</pre>
22 23	PMIX_DEBUGGER_DAEMONS "pmix.debugger" (bool) Spawned application consists of debugger daemons.
24 25	PMIX_TAG_OUTPUT "pmix.tagout" (bool) Tag application output with the identity of the source process.
26 27	PMIX_TIMESTAMP_OUTPUT "pmix.tsout" (bool) Timestamp output from applications.
28 29	<pre>PMIX_MERGE_STDERR_STDOUT "pmix.mergeerrout" (bool) Merge stdout and stderr streams from application processes.</pre>
30 31	<pre>PMIX_OUTPUT_TO_FILE "pmix.outfile" (char*)</pre>
32 33	PMIX_INDEX_ARGV "pmix.indxargv" (bool) Mark the argv with the rank of the process.
34	PMIX CPUS PER PROC "pmix.cpuperproc" (uint32 t)

1 Number of cpus to assign to each rank - when accessed using **PMIx_Get**, use the 2 PMIX RANK WILDCARD value for the rank to discover the cpus/process assigned to the provided namespace 3 4 PMIX NO PROCS ON HEAD "pmix.nolocal" (bool) 5 Do not place processes on the head node. PMIX NO OVERSUBSCRIBE "pmix.noover" (bool) 6 7 Do not oversubscribe the cpus. 8 PMIX REPORT BINDINGS "pmix.repbind" (bool) 9 Report bindings of the individual processes. PMIX CPU LIST "pmix.cpulist" (char*) 10 List of cpus to use for this job - when accessed using PMIx_Get, use the 11 12 PMIX_RANK_WILDCARD value for the rank to discover the cpu list used for the provided 13 namespace 14 PMIX JOB RECOVERABLE "pmix.recover" (bool) Application supports recoverable operations. 15 16 PMIX_JOB_CONTINUOUS "pmix.continuous" (bool) 17 Application is continuous, all failed processes should be immediately restarted. 18 PMIX_MAX_RESTARTS "pmix.maxrestarts" (uint32_t) Maximum number of times to restart a job - when accessed using PMIx_Get, use the 19 PMIX_RANK_WILDCARD value for the rank to discover the max restarts for the provided 20 21 namespace 22 PMIX NOTIFY COMPLETION "pmix.notecomp" (bool) Notify the parent process upon termination of child job. 23

Description

Spawn a new job. The assigned namespace of the spawned applications is returned in the *nspace* parameter. A **NULL** value in that location indicates that the caller doesn't wish to have the namespace returned. The *nspace* array must be at least of size one more than **PMIX_MAX_NSLEN**.

By default, the spawned processes will be PMIx "connected" to the parent process upon successful launch (see PMIx_Connect description for details). Note that this only means that (a) the parent process will be given a copy of the new job's information so it can query job-level info without incurring any communication penalties, (b) newly spawned child processes will receive a copy of the parent processes job-level info, and (c) both the parent process and members of the child job will receive notification of errors from processes in their combined assemblage.

24 25

26 27

28

29

30

31

Advice to users

Behavior of individual resource managers may differ, but it is expected that failure of any application process to start will result in termination/cleanup of all processes in the newly spawned job and return of an error code to the caller.

4 6.2.2 PMIx_Spawn_nb

Summary

Nonblocking version of the **PMIx_Spawn** routine.

Format

PMIx v1.0

1

2

3

5

8

9 10

11

12

13 14

15

17 18

19

20

21 22

23

24

25 26 С

- IN job_info
 - Array of info structures (array of handles)
- IN ninfo
 - Number of elements in the *job_info* array (integer)
- 16 IN apps
 - Array of pmix_app_t structures (array of handles)
 - IN cbfunc
 - Callback function pmix_spawn_cbfunc_t (function reference)
 - IN cbdata
 - Data to be passed to the callback function (memory reference)
 - Returns one of the following:
 - PMIX_SUCCESS, indicating that the request is being processed by the host environment result
 will be returned in the provided *cbfunc*. Note that the library must not invoke the callback
 function prior to returning from the API.
 - a PMIx error constant indicating an error in the request the *cbfunc* will *not* be called

Required Attributes

PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is required to add the following attributes to those provided before passing the request to the host:

PMIX SPAWNED "pmix.spawned" (bool)

true if this process resulted from a call to **PMIx_Spawn**. Lack of inclusion (i.e., a return status of **PMIX_ERR_NOT_FOUND**) corresponds to a value of **false** for this attribute.

PMIX_PARENT_ID "pmix.parent" (pmix_proc_t)

Process identifier of the parent process of the calling process.

PMIX_REQUESTOR_IS_CLIENT "pmix.req.client" (bool)

The requesting process is a PMIx client.

PMIX REQUESTOR IS TOOL "pmix.req.tool" (bool)

The requesting process is a PMIx tool.

Host environments that implement support for PMIx_Spawn are required to pass the PMIX_SPAWNED and PMIX_PARENT_ID attributes to all PMIx servers launching new child processes so those values can be returned to clients upon connection to the PMIx server. In addition, they are required to support the following attributes when present in either the *job_info* or the *info* array of an element of the *apps* array:

PMIX WDIR "pmix.wdir" (char*)

Working directory for spawned processes.

```
PMIX SET SESSION CWD "pmix.ssncwd" (bool)
```

Set the application's current working directory to the session working directory assigned by the RM - when accessed using <code>PMIx_Get</code>, use the <code>PMIX_RANK_WILDCARD</code> value for the rank to discover the session working directory assigned to the provided namespace

```
PMIX_PREFIX "pmix.prefix" (char*)
```

Prefix to use for starting spawned processes.

```
PMIX HOST "pmix.host" (char*)
```

Comma-delimited list of hosts to use for spawned processes.

```
PMIX HOSTFILE "pmix.hostfile" (char*)
```

Hostfile to use for spawned processes.

1

2

3

5

6 7

8

9 10

11

12 13 14

15

16

17 18

19

20

21

22

23

24 25

26 27

28

29

	▼ Optional Attributes
1	The following attributes are optional for host environments that support this operation:
2	<pre>PMIX_ADD_HOSTFILE "pmix.addhostfile" (char*) Hostfile listing hosts to add to existing allocation.</pre>
4 5	<pre>PMIX_ADD_HOST "pmix.addhost" (char*) Comma-delimited list of hosts to add to the allocation.</pre>
6 7	<pre>PMIX_PRELOAD_BIN "pmix.preloadbin" (bool) Preload binaries onto nodes.</pre>
8 9	<pre>PMIX_PRELOAD_FILES "pmix.preloadfiles" (char*) Comma-delimited list of files to pre-position on nodes.</pre>
10 11	PMIX_PERSONALITY "pmix.pers" (char*) Name of personality to use.
12 13 14 15	<pre>PMIX_MAPPER "pmix.mapper" (char*) Mapping mechanism to use for placing spawned processes - when accessed using PMIx_Get , use the PMIX_RANK_WILDCARD value for the rank to discover the mapping mechanism used for the provided namespace.</pre>
16 17	PMIX_DISPLAY_MAP "pmix.dispmap" (bool) Display process mapping upon spawn.
18 19	PMIX_PPR "pmix.ppr" (char*) Number of processes to spawn on each identified resource.
20 21 22 23	<pre>PMIX_MAPBY "pmix.mapby" (char*) Process mapping policy - when accessed using PMIx_Get , use the PMIX_RANK_WILDCARD value for the rank to discover the mapping policy used for the provided namespace</pre>
24 25 26 27	<pre>PMIX_RANKBY "pmix.rankby" (char*) Process ranking policy - when accessed using PMIx_Get , use the PMIX_RANK_WILDCARD value for the rank to discover the ranking algorithm used for the provided namespace</pre>
28 29 30 31	<pre>PMIX_BINDTO "pmix.bindto" (char*) Process binding policy - when accessed using PMIx_Get , use the PMIX_RANK_WILDCARD value for the rank to discover the binding policy used for the provided namespace</pre>
32 33	PMIX_NON_PMI "pmix.nonpmi" (bool) Spawned processes will not call PMIx_Init.
34 35	PMIX_STDIN_TGT "pmix.stdin" (uint32_t) Spawned process rank that is to receive any forwarded stdin.

1 2	PMIX_FWD_STDIN "pmix.fwd.stdin" (bool) Forward stdin from this process to the designated process.
3 4	PMIX_FWD_STDOUT "pmix.fwd.stdout" (bool) Forward stdout from spawned processes to this process.
5 6	PMIX_FWD_STDERR "pmix.fwd.stderr" (bool) Forward stderr from spawned processes to this process.
7 8	PMIX_DEBUGGER_DAEMONS "pmix.debugger" (bool) Spawned application consists of debugger daemons.
9 10	PMIX_TAG_OUTPUT "pmix.tagout" (bool) Tag application output with the identity of the source process.
1 2	PMIX_TIMESTAMP_OUTPUT "pmix.tsout" (bool) Timestamp output from applications.
13 14	PMIX_MERGE_STDERR_STDOUT "pmix.mergeerrout" (bool) Merge stdout and stderr streams from application processes.
15 16	<pre>PMIX_OUTPUT_TO_FILE "pmix.outfile" (char*)</pre>
17 18	PMIX_INDEX_ARGV "pmix.indxargv" (bool) Mark the argv with the rank of the process.
19 20 21 22	PMIX_CPUS_PER_PROC "pmix.cpuperproc" (uint32_t) Number of cpus to assign to each rank - when accessed using PMIx_Get , use the PMIX_RANK_WILDCARD value for the rank to discover the cpus/process assigned to the provided namespace
23 24	PMIX_NO_PROCS_ON_HEAD "pmix.nolocal" (bool) Do not place processes on the head node.
25 26	<pre>PMIX_NO_OVERSUBSCRIBE "pmix.noover" (bool) Do not oversubscribe the cpus.</pre>
27 28	PMIX_REPORT_BINDINGS "pmix.repbind" (bool) Report bindings of the individual processes.
29 30 31 32	<pre>PMIX_CPU_LIST "pmix.cpulist" (char*) List of cpus to use for this job - when accessed using PMIx_Get , use the PMIX_RANK_WILDCARD value for the rank to discover the cpu list used for the provided namespace</pre>
33 34	PMIX_JOB_RECOVERABLE "pmix.recover" (bool) Application supports recoverable operations.
35 36	PMIX_JOB_CONTINUOUS "pmix.continuous" (bool) Application is continuous, all failed processes should be immediately restarted.

1 PMIX_MAX_RESTARTS "pmix.maxrestarts" (uint32_t) 2 Maximum number of times to restart a job - when accessed using **PMIx Get**, use the 3 PMIX RANK WILDCARD value for the rank to discover the max restarts for the provided

namespace

Description

5 6

7

8

9

10

12

13 14

15

16

17

18

19

20 21

22 23

24

25

26 27

28

29

30

Nonblocking version of the PMIx Spawn routine. The provided callback function will be executed upon successful start of all specified application processes.

Advice to users

Behavior of individual resource managers may differ, but it is expected that failure of any application process to start will result in termination/cleanup of all processes in the newly spawned job and return of an error code to the caller.

6.3 **Connecting and Disconnecting Processes**

This section defines functions to connect and disconnect processes in two or more separate PMIx namespaces. The PMIx definition of *connected* solely implies that the host environment should treat the failure of any process in the assemblage as a reportable event, taking action on the assemblage as if it were a single application. For example, if the environment defaults (in the absence of any application directives) to terminating an application upon failure of any process in that application, then the environment should terminate all processes in the connected assemblage upon failure of any member.

— Advice to PMIx server hosts —

The host environment may choose to assign a new namespace to the connected assemblage and/or assign new ranks for its members for its own internal tracking purposes. However, it is not required to communicate such assignments to the participants (e.g., in response to an appropriate call to PMIx Query info nb). The host environment is required to generate a PMIX ERR INVALID TERMINATION event should any process in the assemblage terminate or call **PMIx_Finalize** without first *disconnecting* from the assemblage.

The *connect* operation does not require the exchange of job-level information nor the inclusion of information posted by participating processes via PMIx_Put. Indeed, the callback function utilized in pmix_server_connect_fn_t cannot pass information back into the PMIx server library. However, host environments are advised that collecting such information at the participating daemons represents an optimization opportunity as participating processes are likely to request such information after the connect operation completes.

Advice to users -

Attempting to *connect* processes solely within the same namespace is essentially a *no-op* operation.

While not explicitly prohibited, users are advised that a PMIx implementation or host environment may return an error in such cases.

Neither the PMIx implementation nor host environment are required to provide any tracking support for the assemblage. Thus, the application is responsible for maintaining the membership list of the assemblage.

6.3.1 PMIx Connect

Summary

Connect namespaces.

Format

PMIx v1.0

4

5 6

8

9

10

11

12

13

14

15

16

17

18

19

20

21 22

23

24

pmix_status_t

_____ C _

IN procs

Array of proc structures (array of handles)

IN nprocs

Number of elements in the *procs* array (integer)

IN info

Array of info structures (array of handles)

IN ninfo

Number of elements in the *info* array (integer)

Returns **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant.

Required Attributes

PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing.

	→ Optional Attributes
1	The following attributes are optional for host environments that support this operation:
2 3 4 5	PMIX_TIMEOUT "pmix.timeout" (int) Time in seconds before the specified operation should time out (0 indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.
6 7 8 9 10	PMIX_COLLECTIVE_ALGO "pmix.calgo" (char*) Comma-delimited list of algorithms to use for the collective operation. PMIx does not impose any requirements on a host environment's collective algorithms. Thus, the acceptable values for this attribute will be environment-dependent - users are encouraged to check their host environment for supported values.
11 12	PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool) If true, indicates that the requested choice of algorithm is mandatory.
	Advice to PMIx library implementers —
13 14 15 16 17	We recommend that implementation of the PMIX_TIMEOUT attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support PMIX_TIMEOUT directly in the PMIx server library must take care to resolve the race condition and should avoid passing PMIX_TIMEOUT to the host environment so that multiple competing timeouts are not created.

Description

 Record the processes specified by the *procs* array as *connected* as per the PMIx definition. The function will return once all processes identified in *procs* have called either **PMIx_Connect** or its non-blocking version, *and* the host environment has completed any supporting operations required to meet the terms of the PMIx definition of *connected* processes.

Advice to users

All processes engaged in a given **PMIx_Connect** operation must provide the identical *procs* array as ordering of entries in the array and the method by which those processes are identified (e.g., use of **PMIX_RANK_WILDCARD** versus listing the individual processes) *may* impact the host environment's algorithm for uniquely identifying an operation.

Advice to PMIx library implementers

PMIx_Connect and its non-blocking form are both *collective* operations. Accordingly, the PMIx server library is required to aggregate participation by local clients, passing the request to the host environment once all local participants have executed the API.

Advice to PMIx server hosts —

The host will receive a single call for each collective operation. It is the responsibility of the host to identify the nodes containing participating processes, execute the collective across all participating nodes, and notify the local PMIx server library upon completion of the global collective.

Processes that combine via **PMIx_Connect** must call **PMIx_Disconnect** prior to finalizing and/or terminating - any process in the assemblage failing to meet this requirement will cause a **PMIX_ERR_INVALID_TERMINATION** event to be generated.

A process can only engage in one connect operation involving the identical *procs* array at a time. However, a process can be simultaneously engaged in multiple connect operations, each involving a different *procs* array.

As in the case of the **PMIx_Fence** operation, the *info* array can be used to pass user-level directives regarding the algorithm to be used for any collective operation involved in the operation, timeout constraints, and other options available from the host RM.

6.3.2 PMIx_Connect_nb

Summary

Nonblocking PMIx Connect nb routine.

1	Format
<i>PMIx</i>	v1.0 V
2	pmix_status_t
3	<pre>PMIx_Connect_nb(const pmix_proc_t procs[], size_t nprocs,</pre>
4	<pre>const pmix_info_t info[], size_t ninfo,</pre>
5	<pre>pmix_op_cbfunc_t cbfunc, void *cbdata)</pre>
	C
6	IN procs
7	Array of proc structures (array of handles)
8	IN nprocs
9	Number of elements in the <i>procs</i> array (integer)
10	<pre>IN info</pre>
11	Array of info structures (array of handles)
12	IN ninfo
13	Number of element in the <i>info</i> array (integer)
14	IN cbfunc
15	Callback function pmix_op_cbfunc_t (function reference)
16	IN cbdata
17	Data to be passed to the callback function (memory reference)
18	Returns one of the following:
19	• PMIX_SUCCESS, indicating that the request is being processed by the host environment - result
20	will be returned in the provided <i>cbfunc</i> . Note that the library must not invoke the callback
21	function prior to returning from the API.
22	• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
23	returned success - the cbfunc will not be called
	·
24	• a PMIx error constant indicating either an error in the input or that the request was immediately
25	processed and failed - the <i>cbfunc</i> will <i>not</i> be called
	Required Attributes
26	PMIx libraries are not required to directly support any attributes for this function. However, any
27	provided attributes must be passed to the host SMS daemon for processing.

Optional Attributes

The following attributes are optional for host environments that support this operation:

PMIX_TIMEOUT "pmix.timeout" (int)

Time in seconds before the specified operation should time out (θ indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

PMIX COLLECTIVE ALGO "pmix.calgo" (char*)

Comma-delimited list of algorithms to use for the collective operation. PMIx does not impose any requirements on a host environment's collective algorithms. Thus, the acceptable values for this attribute will be environment-dependent - users are encouraged to check their host environment for supported values.

PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)

If **true**, indicates that the requested choice of algorithm is mandatory.

Advice to PMIx library implementers —

We recommend that implementation of the PMIX_TIMEOUT attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support PMIX_TIMEOUT directly in the PMIx server library must take care to resolve the race condition and should avoid passing PMIX_TIMEOUT to the host environment so that multiple competing timeouts are not created.

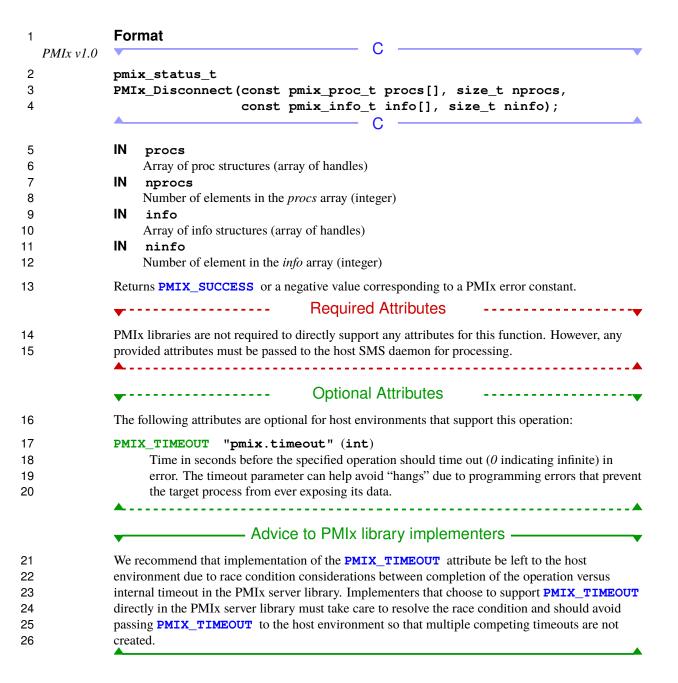
Description

Nonblocking version of PMIx_Connect. The callback function is called once all processes identified in *procs* have called either PMIx_Connect or its non-blocking version, *and* the host environment has completed any supporting operations required to meet the terms of the PMIx definition of *connected* processes. See the advice provided in the description for PMIx_Connect for more information.

6.3.3 PMIx Disconnect

Summary

Disconnect a previously connected set of processes.



Description

Disconnect a previously connected set of processes. A PMIX_ERR_INVALID_OPERATION error will be returned if the specified set of *procs* was not previously *connected* via a call to PMIx_Connect or its non-blocking form. The function will return once all processes identified in *procs* have called either PMIx_Disconnect or its non-blocking version, *and* the host environment has completed any required supporting operations.

Advice to users

All processes engaged in a given **PMIx_Disconnect** operation must provide the identical *procs* array as ordering of entries in the array and the method by which those processes are identified (e.g., use of **PMIX_RANK_WILDCARD** versus listing the individual processes) *may* impact the host environment's algorithm for uniquely identifying an operation.

Advice to PMIx library implementers -

PMIx_Disconnect and its non-blocking form are both *collective* operations. Accordingly, the PMIx server library is required to aggregate participation by local clients, passing the request to the host environment once all local participants have executed the API.

Advice to PMIx server hosts

The host will receive a single call for each collective operation. The host will receive a single call for each collective operation. It is the responsibility of the host to identify the nodes containing participating processes, execute the collective across all participating nodes, and notify the local PMIx server library upon completion of the global collective.

A process can only engage in one disconnect operation involving the identical *procs* array at a time. However, a process can be simultaneously engaged in multiple disconnect operations, each involving a different *procs* array.

As in the case of the **PMIx_Fence** operation, the *info* array can be used to pass user-level directives regarding the algorithm to be used for any collective operation involved in the operation, timeout constraints, and other options available from the host RM.

6.3.4 PMIx_Disconnect_nb

Summary

Nonblocking **PMIx Disconnect** routine.

		Format
	PMIx v1.0	· · · · · · · · · · · · · · · · · · ·
2		pmix_status_t
3		<pre>PMIx_Disconnect_nb(const pmix_proc_t procs[], size_t nprocs,</pre>
4		<pre>const pmix_info_t info[], size_t ninfo,</pre>
5		<pre>pmix_op_cbfunc_t cbfunc, void *cbdata);</pre>
		C —
6		IN procs
7		Array of proc structures (array of handles)
8		IN nprocs
9		Number of elements in the <i>procs</i> array (integer)
10		IN info
11		Array of info structures (array of handles)
12 13		IN ninfo
13 14		Number of element in the <i>info</i> array (integer) IN cbfunc
15		Callback function pmix_op_cbfunc_t (function reference)
16		IN cbdata
17		Data to be passed to the callback function (memory reference)
18		Returns one of the following:
10		
19 20		• PMIX_SUCCESS, indicating that the request is being processed by the host environment - result will be returned in the provided <i>cbfunc</i> . Note that the library must not invoke the callback
21		function prior to returning from the API.
22 23		 PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned success - the cbfunc will not be called
23		·
24		• a PMIx error constant indicating either an error in the input or that the request was immediately
25		processed and failed - the <i>cbfunc</i> will <i>not</i> be called
		Required Attributes
26		PMIx libraries are not required to directly support any attributes for this function. However, any
27		provided attributes must be passed to the host SMS daemon for processing.
		A
		▼ Optional Attributes
28		The following attributes are optional for host environments that support this operation:
29 30		PMIX_TIMEOUT "pmix.timeout" (int) Time in seconds before the specified operation should time out (0 indicating infinite) in
30 31		error. The timeout parameter can help avoid "hangs" due to programming errors that prevent
32		the target process from ever exposing its data.

Advice to PMIx library implementers —

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Nonblocking **PMIx_Disconnect** routine. The callback function is called once all processes identified in *procs* have called either **PMIx_Disconnect_nb** or its blocking version, *and* the host environment has completed any required supporting operations. See the advice provided in the description for **PMIx_Disconnect** for more information.

6.4 IO Forwarding

This section defines functions by which tools (e.g., debuggers) can request forwarding of input/output to/from other processes. The term "tool" widely refers to non-computational programs executed by the user or system administrator to monitor or control a principal computational program. Tools almost always interact with either the host environment, user applications, or both to perform administrative and support functions. For example, a debugger tool might be used to remotely control the processes of a parallel application, monitoring their behavior on a step-by-step basis.

Underlying the operation of many tools is a common need to forward stdin from the tool to targeted processes, and to return stdout/stderr from those processes for display on the user's console. Historically, each tool developer was responsible for creating their own IO forwarding subsystem. However, with the introduction of PMIx as a standard mechanism for interacting between applications and the host environment, it has become possible to relieve tool developers of this burden.

Advice to PMIx server hosts

The responsibility of the host environment in forwarding of IO falls into the following areas:

- Capturing output from specified child processes
- Forwarding that output to the host of the PMIx server library that requested it
- Delivering that payload to the PMIx server library via the **PMIx_server_IOF_deliver**API for final dispatch

It is the responsibility of the PMIx library to buffer, format, and deliver the payload to the requesting client.

Advice to users -

The forwarding of IO via PMIx requires that both the host environment and the tool support PMIx, but does not impose any similar requirements on the application itself.

6.4.1 PMIx_IOF_pull

4 Summary

Register to receive output forwarded from a set of remote processes.

Format

PMIx v3.0

1

5

6

7

8

9 10

11

12

15 16

17 18

19

21 22

23

24

25

26 27

28

29

C

- IN procs
- Array of proc structures identifying desired source processes (array of handles)
- 14 **IN** nprocs
 - Number of elements in the *procs* array (integer)
 - IN directives
 - Array of pmix_info_t structures (array of handles)
 - IN ndirs
 - Number of elements in the *directives* array (integer)
- 20 IN channel
 - Bitmask of IO channels included in the request (pmix_iof_channel_t)
 - IN cbfunc
 - Callback function for delivering relevant output (pmix_iof_cbfunc_t function reference)
 - IN regcbfunc
 - Function to be called when registration is completed (pmix_hdlr_reg_cbfunc_t function reference)
 - IN regcbdata
 - Data to be passed to the *regcbfunc* callback function (memory reference)

1 If regcbfunc is **NULL**, the function call will be treated as a blocking call. In this case, the returned status will be either (a) the IOF handler reference identifier if the value is greater than or equal to 2 zero, or (b) a negative error code indicative of the reason for the failure. 3 4 If the regcbfunc is non-NULL, the function call will be treated as a non-blocking call and will return the following: 5 PMIX SUCCESS indicating that the request has been accepted for processing and the provided 6 callback function will be executed upon completion of the operation. Note that the library 7 must not invoke the callback function prior to returning from the API. The IOF handler 8 9 identifier will be returned in the callback 10 a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this case, the provided callback function will not be executed. 11 Required Attributes ______ 12 The following attributes are required for PMIx libraries that support IO forwarding: 13 PMIX_IOF_CACHE_SIZE "pmix.iof.csize" (uint32_t) The requested size of the server cache in bytes for each specified channel. By default, the 14 server is allowed (but not required) to drop all bytes received beyond the max size. 15 16 PMIX IOF DROP OLDEST "pmix.iof.old" (bool) In an overflow situation, drop the oldest bytes to make room in the cache. 17 PMIX_IOF_DROP_NEWEST "pmix.iof.new" (bool) 18 In an overflow situation, drop any new bytes received until room becomes available in the 19 cache (default). 20 **▲**------------ Optional Attributes ------21 The following attributes are optional for PMIx libraries that support IO forwarding: 22 PMIX_IOF_BUFFERING_SIZE "pmix.iof.bsize" (uint32_t) 23 Controls grouping of IO on the specified channel(s) to avoid being called every time a bit of IO arrives. The library will execute the callback whenever the specified number of bytes 24 becomes available. Any remaining buffered data will be "flushed" upon call to deregister the 25 respective channel. 26 27 PMIX_IOF_BUFFERING_TIME "pmix.iof.btime" (uint32_t) 28 Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering 29 size, this prevents IO from being held indefinitely while waiting for another payload to arrive. 30 31 PMIX_IOF_TAG_OUTPUT "pmix.iof.tag" (bool) 32 Tag output with the channel it comes from. 33 PMIX_IOF_TIMESTAMP_OUTPUT "pmix.iof.ts" (bool)

```
Timestamp output
 1
               PMIX IOF XML OUTPUT "pmix.iof.xml" (bool)
2
 3
                     Format output in XML
               Description
4
               Register to receive output forwarded from a set of remote processes.
5
                                                Advice to users
               Providing a NULL function pointer for the cbfunc parameter will cause output for the indicated
6
 7
               channels to be written to their corresponding stdout/stderr file descriptors. Use of
8
               PMIX RANK WILDCARD to specify all processes in a given namespace is supported but should
               be used carefully due to bandwidth considerations.
9
10 6.4.2
             PMIx_IOF_deregister
               Summary
11
12
               Deregister from output forwarded from a set of remote processes.
               Format
13
   PMIx v3.0
14
               pmix status t
               PMIx IOF deregister(size t iofhdlr,
15
                                         const pmix_info_t directives[], size_t ndirs,
16
                                         pmix_op_cbfunc_t cbfunc, void *cbdata)
17
               IN
                    iofhdlr
18
                   Registration number returned from the pmix hdlr req cbfunc t callback from the
19
20
                   call to PMIx IOF pull (size t)
                    directives
               IN
21
22
                   Array of pmix info t structures (array of handles)
23
               IN
                   Number of elements in the directives array (integer)
24
25
               IN
                    cbfunc
                   Callback function to be called when deregistration has been completed. (function reference)
26
27
               IN
                    cbdata
28
                   Data to be passed to the cbfunc callback function (memory reference)
```

If *cbfunc* is **NULL**, the function will be treated as a *blocking* call and the result of the operation 1 returned in the status code. 2 3 If cbfunc is non-NULL, the function will be treated as a non-blocking call and return one of the 4 following: 5 • PMIX_SUCCESS, indicating that the request is being processed - result will be returned in the provided cbfunc. Note that the library must not invoke the callback function prior to returning 6 from the API. 7 • PMIX OPERATION SUCCEEDED, indicating that the request was immediately processed and 8 9 returned success - the cbfunc will not be called • a PMIx error constant indicating either an error in the input or that the request was immediately 10 processed and failed - the cbfunc will not be called 11 The returned status code will be one of the following: 12 13 **PMIX_SUCCESS** The IOF handler was successfully deregistered. PMIX_ERR_BAD_PARAM The provided *iofhdlr* was unrecognized. 14 **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation does not support this function. 15 **Description** 16 Deregister from output forwarded from a set of remote processes. 17 Advice to PMIx library implementers 18 Any currently buffered IO should be flushed upon receipt of a deregistration request. All received

20 6.4.3 PMIx_IOF_push

21 Summary

19

22

Push data collected locally (typically from stdin or a file) to stdin of the target recipients.

IO after receipt of the request shall be discarded.

1		Format
1	PMIx v3.0	
2		pmix_status_t
3		<pre>PMIx_IOF_push(const pmix_proc_t targets[], size_t ntargets,</pre>
4		<pre>pmix_byte_object_t *bo,</pre>
5		<pre>const pmix_info_t directives[], size_t ndirs,</pre>
6		<pre>pmix_op_cbfunc_t cbfunc, void *cbdata)</pre>
		C -
7		IN targets
8		Array of proc structures identifying desired target processes (array of handles)
9		IN ntargets
0		Number of elements in the <i>targets</i> array (integer)
1		IN bo
2		Pointer to pmix_byte_object_t containing the payload to be delivered (handle)
3		IN directives
4		Array of pmix_info_t structures (array of handles)
5		IN ndirs Number of elements in the directives arroy (integer)
6 7		Number of elements in the <i>directives</i> array (integer) IN directives
8		Array of pmix_info_t structures (array of handles)
9		IN cbfunc
20		Callback function to be called when operation has been completed. (pmix_op_cbfunc_t
21		function reference)
2		IN cbdata
23		Data to be passed to the cbfunc callback function (memory reference)
24		If <i>cbfunc</i> is NULL , the function will be treated as a <i>blocking</i> call and the result of the operation
25		returned in the status code.
26		If <i>cbfunc</i> is non- NULL , the function will be treated as a <i>non-blocking</i> call and return one of the
27		following:
28		• PMIX_SUCCESS, indicating that the request is being processed - result will be returned in the
9		provided <i>cbfunc</i> . Note that the library must not invoke the callback function prior to returning
0		from the API.
31		• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
2		returned success - the cbfunc will not be called
3		• a PMIx error constant indicating either an error in the input or that the request was immediately
34		processed and failed - the <i>cbfunc</i> will <i>not</i> be called
15		The returned status code will be one of the following:
86 87		PMIX_SUCCESS The provided data has been accepted for transmission - it is not indicative of the payload being delivered to any member of the provided <i>targets</i>

PMIX_ERR_NOT_SUPPORTED The PMIx implementation does not support this function. 1 a PMIx error constant indicating the nature of the error 2 Required Attributes The following attributes are required for PMIx libraries that support IO forwarding: 3 PMIX IOF CACHE SIZE "pmix.iof.csize" (uint32 t) 4 5 The requested size of the server cache in bytes for each specified channel. By default, the server is allowed (but not required) to drop all bytes received beyond the max size. 6 PMIX IOF DROP OLDEST "pmix.iof.old" (bool) 7 In an overflow situation, drop the oldest bytes to make room in the cache. 8 9 PMIX IOF DROP NEWEST "pmix.iof.new" (bool) In an overflow situation, drop any new bytes received until room becomes available in the 10 cache (default). 11 ------ Optional Attributes -------The following attributes are optional for PMIx libraries that support IO forwarding: 12 13 PMIX_IOF_BUFFERING_SIZE "pmix.iof.bsize" (uint32_t) Controls grouping of IO on the specified channel(s) to avoid being called every time a bit of 14 IO arrives. The library will execute the callback whenever the specified number of bytes 15 becomes available. Any remaining buffered data will be "flushed" upon call to deregister the 16 respective channel. 17 18 PMIX IOF BUFFERING TIME "pmix.iof.btime" (uint32 t) 19 Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering size, this prevents IO from being held indefinitely while waiting for another payload to 20 21 22 **Description** Push data collected locally (typically from stdin or a file) to stdin of the target recipients. 23 Advice to users -24 Execution of the *cbfunc* callback function serves as notice that the PMIx library no longer requires the caller to maintain the bo data object - it does not indicate delivery of the payload to the targets. 25 Use of PMIX RANK WILDCARD to specify all processes in a given namespace is supported but 26 27 should be used carefully due to bandwidth considerations.

CHAPTER 7

Job Management and Reporting

The job management APIs provide an application with the ability to orchestrate its operation in partnership with the SMS. Members of this category include the

PMIx_Allocation_request_nb, PMIx_Job_control_nb, and

PMIx Process monitor nb APIs.

7.1 Query

 As the level of interaction between applications and the host SMS grows, so too does the need for the application to query the SMS regarding its capabilities and state information. PMIx provides a generalized query interface for this purpose, along with a set of standardized attribute keys to support a range of requests. This includes requests to determine the status of scheduling queues and active allocations, the scope of API and attribute support offered by the SMS, namespaces of active jobs, location and information about a job's processes, and information regarding available resources.

An example use-case for the PMIx_Query_info_nb API is to ensure clean job completion. Time-shared systems frequently impose maximum run times when assigning jobs to resource allocations. To shut down gracefully, e.g., to write a checkpoint before termination, it is necessary for an application to periodically query the resource manager for the time remaining in its allocation. This is especially true on systems for which allocation times may be shortened or lengthened from the original time limit. Many resource managers provide APIs to dynamically obtain this information, but each API is specific to the resource manager.

PMIx supports this use-case by defining an attribute key (PMIX_TIME_REMAINING) that can be used with the PMIx_Query_info_nb interface to obtain the number of seconds remaining in the current job allocation. Note that one could alternatively use the PMIx_Register_event_handler API to register for an event indicating incipient job termination, and then use the PMIx_Job_control_nb API to request that the host SMS generate an event a specified amount of time prior to reaching the maximum run time. PMIx provides such alternate methods as a means of maximizing the probability of a host system supporting at least one method by which the application can obtain the desired service.

The following APIs support query of various session and environment values.

7.1.1 PMIx_Resolve_peers

Summary

Obtain the array of processes within the specified namespace that are executing on a given node.

```
Format
1
   PMIx v1.0
2
               pmix status t
 3
               PMIx Resolve peers (const char *nodename,
                                        const pmix nspace t nspace,
 4
                                        pmix_proc_t **procs, size_t *nprocs)
5
               IN
                    nodename
6
 7
                    Name of the node to query (string)
8
               IN
                   nspace
9
                    namespace (string)
10
               OUT procs
                    Array of process structures (array of handles)
11
               OUT nprocs
12
13
                    Number of elements in the procs array (integer)
               Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.
14
               Description
15
               Given a nodename, return the array of processes within the specified nspace that are executing on
16
17
               that node. If the nspace is NULL, then all processes on the node will be returned. If the specified
               node does not currently host any processes, then the returned array will be NULL, and nprocs will
18
19
               be 0. The caller is responsible for releasing the procs array when done with it. The
               PMIX PROC FREE macro is provided for this purpose.
20
    7.1.2
              PMIx Resolve nodes
               Summary
22
               Return a list of nodes hosting processes within the given namespace.
23
               Format
24
   PMIx v1.0
25
               pmix status t
               PMIx_Resolve_nodes(const char *nspace, char **nodelist)
26
27
               IN
                    nspace
                    Namespace (string)
28
29
               OUT nodelist
30
                    Comma-delimited list of nodenames (string)
31
               Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.
```

```
Description
1
2
               Given a nspace, return the list of nodes hosting processes within that namespace. The returned
               string will contain a comma-delimited list of nodenames. The caller is responsible for releasing the
 3
 4
               string when done with it.
    7.1.3
             PMIx Query info
               Summary
6
 7
               Query information about the system in general.
               Format
8
   PMIx v4.0
9
               pmix status t
10
               PMIx_Query_info(pmix_query_t queries[], size_t nqueries,
                                    pmix_info_t *info[], size_t *ninfo)
11
                                                —— C
12
               IN
                    queries
13
                    Array of query structures (array of handles)
               IN
                    nqueries
14
                    Number of elements in the queries array (integer)
15
               INOUT info
16
17
                    Address where a pointer to an array of pmix_info_t containing the results of the query
                    can be returned (memory reference)
18
               INOUT ninfo
19
                    Address where the number of elements in info can be returned (handle)
20
21
               Returns one of the following:
22
               • PMIX SUCCESS All data has been returned
               • PMIX_ERR_NOT_FOUND None of the requested data was available
23
24
               • PMIX ERR PARTIAL SUCCESS Some of the data has been returned
               • PMIX_ERR_NOT_SUPPORTED The host RM does not support this function
25
26
               • a non-zero PMIx error constant indicating a reason for the request's failure
                                               Required Attributes
               PMIx libraries that support this API are required to support the following attributes:
27
28
               PMIX_QUERY_REFRESH_CACHE "pmix.qry.rfsh" (bool)
                     Retrieve updated information from server. NO QUALIFIERS
29
30
               PMIX_SESSION_INFO "pmix.ssn.info" (bool)
```

 Return information about the specified session. If information about a session other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX_SESSION_ID** attribute identifying the desired target.

PMIX JOB INFO "pmix.job.info" (bool)

Return information about the specified job or namespace. If information about a job or namespace other than the one containing the requesting process is desired, then the attribute array must contain a PMIX_JOBID or PMIX_NSPACE attribute identifying the desired target. Similarly, if information is requested about a job or namespace in a session other than the one containing the requesting process, then an attribute identifying the target session must be provided.

PMIX_APP_INFO "pmix.app.info" (bool)

Return information about the specified application. If information about an application other than the one containing the requesting process is desired, then the attribute array must contain a PMIX_APPNUM attribute identifying the desired target. Similarly, if information is requested about an application in a job or session other than the one containing the requesting process, then attributes identifying the target job and/or session must be provided.

PMIX NODE INFO "pmix.node.info" (bool)

Return information about the specified node. If information about a node other than the one containing the requesting process is desired, then the attribute array must contain either the **PMIX_NODEID** or **PMIX_HOSTNAME** attribute identifying the desired target.

PMIX_PROCID "pmix.procid" (pmix_proc_t)

Process identifier Specifies the process ID whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Only required when the request is for information on a specific process.

PMIX_NSPACE "pmix.nspace" (char*)

Namespace of the job - may be a numerical value expressed as a string, but is often a non-numerical string carrying information solely of use to the system. Specifies the namespace of the process whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must be accompanied by the **PMIX_RANK** attribute. Only required when the request is for information on a specific process.

PMIX_RANK "pmix.rank" (pmix_rank_t)

Process rank within the job. Specifies the rank of the process whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must be accompanied by the **PMIX_NSPACE** attribute. Only required when the request is for information on a specific process.

PMIX_QUERY_ATTRIBUTE_SUPPORT "pmix.qry.attrs" (bool)

Query list of supported attributes for specified APIs. SUPPORTED QUALIFIERS: PMIX CLIENT FUNCTIONS, PMIX SERVER FUNCTIONS,

PMIX_TOOL_FUNCTIONS, and/or PMIX_HOST_FUNCTIONS

1 2	PMIX_CLIENT_ATTRIBUTES "pmix.client.attrs" (bool) Request attributes supported by the PMIx client library
3 4	PMIX_SERVER_ATTRIBUTES "pmix.srvr.attrs" (bool) Request attributes supported by the PMIx server library
5 6	PMIX_HOST_ATTRIBUTES "pmix.host.attrs" (bool) Request attributes supported by the host environment
7 8	PMIX_TOOL_ATTRIBUTES "pmix.setup.env" (bool) Request attributes supported by the PMIx tool library functions
9 0 1 2 3	Note that inclusion of the PMIX_PROCID directive and either the PMIX_NSPACE or the PMIX_RANK attribute will return a PMIX_ERR_BAD_PARAM result, and that the inclusion of a process identifier must apply to all keys in that pmix_query_t . Queries for information on multiple specific processes therefore requires submitting multiple pmix_query_t structures, each referencing one process.
4 5 6 7	PMIx libraries are not required to directly support any other attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is <i>required</i> to add the PMIX_USERID and the PMIX_GRPID attributes of the client process making the request.
8	
9 20	Host environments that support this operation are required to support the following attributes as qualifiers to the request:
21 22 23 24	PMIX_PROCID "pmix.procid" (pmix_proc_t) Process identifier Specifies the process ID whose information is being requested - e.g., a query asking for the PMIX_LOCAL_RANK of a specified process. Only required when the request is for information on a specific process.
25 26 27 28 29	PMIX_NSPACE "pmix.nspace" (char*) Namespace of the job - may be a numerical value expressed as a string, but is often a non-numerical string carrying information solely of use to the system. Specifies the namespace of the process whose information is being requested - e.g., a query asking for the PMIX_LOCAL_RANK of a specified process. Must be accompanied by the PMIX_RANK attribute. Only required when the request is for information on a specific process.
81 82 83 84	<pre>PMIX_RANK "pmix.rank" (pmix_rank_t) Process rank within the job. Specifies the rank of the process whose information is being requested - e.g., a query asking for the PMIX_LOCAL_RANK of a specified process. Must be accompanied by the PMIX_NSPACE attribute. Only required when the request is for information on a specific process.</pre>

1 2 3 4 5	Note that inclusion of the PMIX_PROCID directive and either the PMIX_NSPACE or the PMIX_RANK attribute will return a PMIX_ERR_BAD_PARAM result, and that the inclusion of a process identifier must apply to all keys in that pmix_query_t. Queries for information on multiple specific processes therefore requires submitting multiple pmix_query_t structures, each referencing one process.
	▼Optional Attributes
6	The following attributes are optional for host environments that support this operation:
7 8	<pre>PMIX_QUERY_NAMESPACES "pmix.qry.ns" (char*) Request a comma-delimited list of active namespaces. NO QUALIFIERS</pre>
9 10 11	PMIX_QUERY_JOB_STATUS "pmix.qry.jst" (pmix_status_t) Status of a specified, currently executing job. REQUIRED QUALIFIER: PMIX_NSPACE indicating the namespace whose status is being queried
12 13	<pre>PMIX_QUERY_QUEUE_LIST "pmix.qry.qlst" (char*) Request a comma-delimited list of scheduler queues. NO QUALIFIERS</pre>
14 15 16	PMIX_QUERY_QUEUE_STATUS "pmix.qry.qst" (TBD) Status of a specified scheduler queue. SUPPORTED QUALIFIERS: PMIX_ALLOC_QUEUE naming specific queue whose status is being requested
17 18 19 20	<pre>PMIX_QUERY_PROC_TABLE "pmix.qry.ptable" (char*) Input namespace of the job whose information is being requested returns (pmix_data_array_t) an array of pmix_proc_info_t . REQUIRED QUALIFIER: PMIX_NSPACE indicating the namespace whose process table is being queried</pre>
21 22 23 24 25	PMIX_QUERY_LOCAL_PROC_TABLE "pmix.qry.lptable" (char*) Input namespace of the job whose information is being requested returns (pmix_data_array_t) an array of pmix_proc_info_t for processes in job on same node. REQUIRED QUALIFIER: PMIX_NSPACE indicating the namespace whose process table is being queried
26 27	PMIX_QUERY_SPAWN_SUPPORT "pmix.qry.spawn" (bool) Return a comma-delimited list of supported spawn attributes. NO QUALIFIERS
28 29	PMIX_QUERY_DEBUG_SUPPORT "pmix.qry.debug" (bool) Return a comma-delimited list of supported debug attributes. NO QUALIFIERS
30 31 32 33	PMIX_QUERY_MEMORY_USAGE "pmix.qry.mem" (bool) Return information on memory usage for the processes indicated in the qualifiers. SUPPORTED QUALIFIERS: PMIX_NSPACE and PMIX_RANK, or PMIX_PROCID of specific process(es) whose memory usage is being requested
34	PMIX_QUERY_REPORT_AVG "pmix.qry.avg" (bool)

1	Report only average values for sampled information. NO QUALIFIERS
2 3	PMIX_QUERY_REPORT_MINMAX "pmix.qry.minmax" (bool) Report minimum and maximum values. NO QUALIFIERS
4 5	<pre>PMIX_QUERY_ALLOC_STATUS "pmix.query.alloc" (char*) String identifier of the allocation whose status is being requested. NO QUALIFIERS</pre>
6 7 8 9	<pre>PMIX_TIME_REMAINING "pmix.time.remaining" (char*) Query number of seconds (uint32_t) remaining in allocation for the specified namespace. SUPPORTED QUALIFIERS: PMIX_NSPACE of the namespace whose info is being requested (defaults to allocation containing the caller)</pre>
10 11 12	PMIX_SERVER_URI "pmix.srvr.uri" (char*) URI of the PMIx server to be contacted. Requests the URI of the specified PMIx server's PMIx connection. Defaults to requesting the information for the local PMIx server.
13 14 15 16	PMIX_PROC_URI "pmix.puri" (char*) URI containing contact information for a given process. Requests the URI of the specified PMIx server's out-of-band connection. Defaults to requesting the information for the local PMIx server.
17 18 19 20 21	Description Query information about the system in general. This can include a list of active namespaces, fabric topology, etc. Also can be used to query node-specific info such as the list of peers executing on a given node. We assume that the host RM will exercise appropriate access control on the information.
22 23 24 25	The returned <i>status</i> indicates if requested data was found or not. The returned array of <code>pmix_info_t</code> will contain each key that was provided and the corresponding value that was found. Requests for keys that are not found will return the key paired with a value of type <code>PMIX_UNDEF</code> . The caller is responsible for releasing the returned array.
	Advice to PMIx library implementers —
26 27 28 29 30 31	Information returned from PMIx_Query_info shall be locally cached so that retrieval by subsequent calls to PMIx_Get , PMIx_Query_info , or PMIx_Query_info_nb can succeed with minimal overhead. The local cache shall be checked prior to querying the PMIx server and/or the host environment. Queries that include the PMIX_QUERY_REFRESH_CACHE attribute shall bypass the local cache and retrieve a new value for the query, refreshing the values in the cache upon return.

7.1.4 PMIx_Query_info_nb Summary Query information about the system in general. 3 Format 4 PMIx v2.0 5 pmix status t 6 PMIx Query info nb(pmix query t queries[], size t nqueries, 7 pmix info cbfunc t cbfunc, void *cbdata) IN 8 queries 9 Array of query structures (array of handles) 10 IN nqueries Number of elements in the *queries* array (integer) 11 IN cbfunc 12 Callback function **pmix_info_cbfunc_t** (function reference) 13 IN cbdata 14 15 Data to be passed to the callback function (memory reference) Returns one of the following: 16 17 • PMIX_SUCCESS indicating that the request has been accepted for processing and the provided callback function will be executed upon completion of the operation. Note that the library must 18 not invoke the callback function prior to returning from the API. 19 20 • a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this case, the provided callback function will not be executed 21 22 If executed, the status returned in the provided callback function will be one of the following 23 constants: • PMIX SUCCESS All data has been returned 24 • PMIX ERR NOT FOUND None of the requested data was available 25 26 • PMIX ERR PARTIAL SUCCESS Some of the data has been returned 27 • PMIX_ERR_NOT_SUPPORTED The host RM does not support this function 28 • a non-zero PMIx error constant indicating a reason for the request's failure Required Attributes PMIx libraries that support this API are required to support the following attributes: 29 PMIX QUERY REFRESH CACHE "pmix.gry.rfsh" (bool) 30 Retrieve updated information from server. NO QUALIFIERS 31 32 PMIX SESSION INFO "pmix.ssn.info" (bool)

 Return information about the specified session. If information about a session other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX_SESSION_ID** attribute identifying the desired target.

PMIX JOB INFO "pmix.job.info" (bool)

Return information about the specified job or namespace. If information about a job or namespace other than the one containing the requesting process is desired, then the attribute array must contain a PMIX_JOBID or PMIX_NSPACE attribute identifying the desired target. Similarly, if information is requested about a job or namespace in a session other than the one containing the requesting process, then an attribute identifying the target session must be provided.

PMIX APP INFO "pmix.app.info" (bool)

Return information about the specified application. If information about an application other than the one containing the requesting process is desired, then the attribute array must contain a PMIX_APPNUM attribute identifying the desired target. Similarly, if information is requested about an application in a job or session other than the one containing the requesting process, then attributes identifying the target job and/or session must be provided.

PMIX_NODE_INFO "pmix.node.info" (bool)

Return information about the specified node. If information about a node other than the one containing the requesting process is desired, then the attribute array must contain either the **PMIX_NODEID** or **PMIX_HOSTNAME** attribute identifying the desired target.

PMIX_PROCID "pmix.procid" (pmix_proc_t)

Process identifier Specifies the process ID whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Only required when the request is for information on a specific process.

PMIX_NSPACE "pmix.nspace" (char*)

Namespace of the job - may be a numerical value expressed as a string, but is often a non-numerical string carrying information solely of use to the system. Specifies the namespace of the process whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must be accompanied by the **PMIX_RANK** attribute. Only required when the request is for information on a specific process.

PMIX_RANK "pmix.rank" (pmix_rank_t)

Process rank within the job. Specifies the rank of the process whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must be accompanied by the **PMIX_NSPACE** attribute. Only required when the request is for information on a specific process.

PMIX_QUERY_ATTRIBUTE_SUPPORT "pmix.qry.attrs" (bool)

Query list of supported attributes for specified APIs. SUPPORTED QUALIFIERS: PMIX CLIENT FUNCTIONS. PMIX SERVER FUNCTIONS.

PMIX_TOOL_FUNCTIONS, and/or PMIX_HOST_FUNCTIONS

1	<pre>PMIX_CLIENT_ATTRIBUTES "pmix.client.attrs" (bool)</pre>
2	Request attributes supported by the PMIx client library
3	<pre>PMIX_SERVER_ATTRIBUTES "pmix.srvr.attrs" (bool)</pre>
4	Request attributes supported by the PMIx server library
5	<pre>PMIX_HOST_ATTRIBUTES "pmix.host.attrs" (bool)</pre>
6	Request attributes supported by the host environment

PMIX_TOOL_ATTRIBUTES "pmix.setup.env" (bool)

Request attributes supported by the PMIx tool library functions

Note that inclusion of the PMIX_PROCID directive and either the PMIX_NSPACE or the PMIX_RANK attribute will return a PMIX_ERR_BAD_PARAM result, and that the inclusion of a process identifier must apply to all keys in that pmix_query_t . Queries for information on multiple specific processes therefore requires submitting multiple pmix_query_t structures, each referencing one process.

PMIx libraries are not required to directly support any other attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is *required* to add the **PMIX_USERID** and the **PMIX_GRPID** attributes of the client process making the request.

Host environments that support this operation are required to support the following attributes as qualifiers to the request:

PMIX_PROCID "pmix.procid" (pmix_proc_t)

Process identifier Specifies the process ID whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Only required when the request is for information on a specific process.

PMIX_NSPACE "pmix.nspace" (char*)

Namespace of the job - may be a numerical value expressed as a string, but is often a non-numerical string carrying information solely of use to the system. Specifies the namespace of the process whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must be accompanied by the **PMIX_RANK** attribute. Only required when the request is for information on a specific process.

PMIX_RANK "pmix.rank" (pmix_rank_t)

Process rank within the job. Specifies the rank of the process whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must be accompanied by the **PMIX_NSPACE** attribute. Only required when the request is for information on a specific process.

Note that inclusion of the PMIX_PROCID directive and either the PMIX_NSPACE or the PMIX_RANK attribute will return a PMIX_ERR_BAD_PARAM result, and that the inclusion of a process identifier must apply to all keys in that pmix_query_t. Queries for information on

1 2	multiple specific processes therefore requires submitting multiple pmix_query_t structures, each referencing one process.
	▼ Optional Attributes
3	The following attributes are optional for host environments that support this operation:
4 5	PMIX_QUERY_NAMESPACES "pmix.qry.ns" (char*) Request a comma-delimited list of active namespaces. NO QUALIFIERS
6 7 8	PMIX_QUERY_JOB_STATUS "pmix.qry.jst" (pmix_status_t) Status of a specified, currently executing job. REQUIRED QUALIFIER: PMIX_NSPACE indicating the namespace whose status is being queried
9 0	PMIX_QUERY_QUEUE_LIST "pmix.qry.qlst" (char*) Request a comma-delimited list of scheduler queues. NO QUALIFIERS
1 2 3	PMIX_QUERY_QUEUE_STATUS "pmix.qry.qst" (TBD) Status of a specified scheduler queue. SUPPORTED QUALIFIERS: PMIX_ALLOC_QUEUE naming specific queue whose status is being requested
4 5 6 7	PMIX_QUERY_PROC_TABLE "pmix.qry.ptable" (char*) Input namespace of the job whose information is being requested returns (pmix_data_array_t) an array of pmix_proc_info_t. REQUIRED QUALIFIER: PMIX_NSPACE indicating the namespace whose process table is being queried
8 9 20 21 22	PMIX_QUERY_LOCAL_PROC_TABLE "pmix.qry.lptable" (char*) Input namespace of the job whose information is being requested returns (pmix_data_array_t) an array of pmix_proc_info_t for processes in job on same node. REQUIRED QUALIFIER: PMIX_NSPACE indicating the namespace whose process table is being queried
23 24	PMIX_QUERY_SPAWN_SUPPORT "pmix.qry.spawn" (bool) Return a comma-delimited list of supported spawn attributes. NO QUALIFIERS
25 26	PMIX_QUERY_DEBUG_SUPPORT "pmix.qry.debug" (bool) Return a comma-delimited list of supported debug attributes. NO QUALIFIERS
27 28 29 30	PMIX_QUERY_MEMORY_USAGE "pmix.qry.mem" (bool) Return information on memory usage for the processes indicated in the qualifiers. SUPPORTED QUALIFIERS: PMIX_NSPACE and PMIX_RANK, or PMIX_PROCID of specific process(es) whose memory usage is being requested
31 32	PMIX_QUERY_REPORT_AVG "pmix.qry.avg" (bool) Report only average values for sampled information. NO QUALIFIERS
3 34	PMIX_QUERY_REPORT_MINMAX "pmix.qry.minmax" (bool) Report minimum and maximum values. NO QUALIFIERS
35	<pre>PMIX_QUERY_ALLOC_STATUS "pmix.query.alloc" (char*)</pre>

1		String identifier of the allocation whose status is being requested. NO QUALIFIERS
2 3 4 5		PMIX_TIME_REMAINING "pmix.time.remaining" (char*) Query number of seconds (uint32_t) remaining in allocation for the specified namespace. SUPPORTED QUALIFIERS: PMIX_NSPACE of the namespace whose info is being requested (defaults to allocation containing the caller)
6 7 8		PMIX_SERVER_URI "pmix.srvr.uri" (char*) URI of the PMIx server to be contacted. Requests the URI of the specified PMIx server's PMIx connection. Defaults to requesting the information for the local PMIx server.
9 10 11 12		PMIX_PROC_URI "pmix.puri" (char*) URI containing contact information for a given process. Requests the URI of the specified PMIx server's out-of-band connection. Defaults to requesting the information for the local PMIx server.
13 14		Description Non-blocking form of the PMIx_Query_info API
15 7. 1	1.4.1	Using PMIx_Get VS PMIx_Query_info
16 17		Both PMIx_Get and PMIx_Query_info can be used to retrieve information about the system. In general, the <i>get</i> operation should be used to retrieve:
18 19		• information provided by the host environment at time of job start. This includes information on the number of processes in the job, their location, and possibly their communication endpoints
20		• information posted by processes via the PMIx_Put function
21 22 23 24 25		This information is largely considered to be <i>static</i> , although this will not necessarily be true for environments supporting dynamic programming models or fault tolerance. Note that the <code>PMIx_Get</code> function only accesses information about execution environments - i.e., its scope is limited to values pertaining to a specific <code>session</code> , <code>job</code> , <code>application</code> , process, or node. It cannot be used to obtain information about areas such as the status of queues in the WLM.
26		In contrast, the <i>query</i> option should be used to access:
27 28		• system-level information (such as the available WLM queues) that would generally not be included in job-level information provided at job start
29 30 31		 dynamic information such as application and queue status, and resource utilization statistics. Note that the PMIX_QUERY_REFRESH_CACHE attribute must be provided on each query to ensure current data is returned
32		• information created post job start, such as process tables
33 34		• information requiring more complex search criteria than supported by the simpler PMIx_Get API

• queries focused on retrieving multi-attribute blocks of data with a single request, thus bypassing the single-key limitation of the **PMIx Get** API

In theory, all information can be accessed via <code>PMIx_Query_info</code> as the local cache is typically the same datastore searched by <code>PMIx_Get</code>. However, in practice, the overhead associated with the <code>query</code> operation may (depending upon implementation) be higher than the simpler <code>get</code> operation due to the need to construct and process the more complex <code>pmix_query_t</code> structure. Thus, requests for a single key value are likely to be accomplished faster with <code>PMIx_Get</code> versus the <code>query</code> operation.

7.1.4.2 Accessing attribute support information

Information as to attributes supported by either the PMIx implementation or its host environment can be obtained via the PMIx_Query_info_nb API. The

PMIX_QUERY_ATTRIBUTE_SUPPORT attribute must be listed as the first entry in the *keys* field of the **pmix_query_t** structure, followed by the name of the function whose attribute support is being requested - support for multiple functions can be requested simultaneously by simply adding the function names to the array of *keys*. Function names *must* be given as user-level API names - e.g., "PMIx_Get", "PMIx_server_setup_application", or "PMIx_tool_connect_to_server".

The desired levels (see 3.4.31) of attribute support are provided as qualifiers. Multiple levels can be requested simultaneously by simply adding elements to the *qualifiers* array. Each qualifier should contain the desired level attribute with the boolean value set to indicate whether or not that level is to be included in the returned information. Failure to provide any levels is equivalent to a request for all levels.

Unlike other queries, queries for attribute support can result in the number of returned <code>pmix_info_t</code> structures being different from the number of queries. Each element in the returned array will correspond to a pair of specified attribute level and function in the query, where the <code>key</code> is the function and the <code>value</code> contains a <code>pmix_data_array_t</code> of <code>pmix_info_t</code>. Each element of the array is marked by a <code>key</code> indicating the requested attribute <code>level</code> with a <code>value</code> composed of a <code>pmix_data_array_t</code> of <code>pmix_regattr_t</code>, each describing a supported attribute for that function, as illustrated in Fig. 7.1 below where the requestor asked for supported attributes of <code>PMIx_Get</code> at the <code>client</code> and <code>server</code> levels, plus attributes of <code>PMIx_Allocation_request</code> at all levels:

The array of returned structures, and their child arrays, are subject to the return rules for the **PMIx_Query_info_nb** API. For example, a request for supported attributes of the **PMIx_Get** function that includes the *host* level will return values for the *client* and *server* levels, plus an array element with a *key* of **PMIX_HOST_ATTRIBUTES** and a value type of **PMIX_UNDEF** indicating that no attributes are supported at that level.

7.2 Allocation Requests

This section defines functionality to request new allocations from the RM, and request modifications to existing allocations. These are primarily used in the following scenarios:

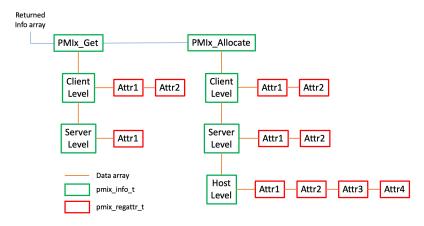


Figure 7.1.: Returned information hierarchy for attribute support request

- Evolving applications that dynamically request and return resources as they execute
 - *Malleable* environments where the scheduler redirects resources away from executing applications for higher priority jobs or load balancing
 - Resilient applications that need to request replacement resources in the face of failures
 - *Rigid* jobs where the user has requested a static allocation of resources for a fixed period of time, but realizes that they underestimated their required time while executing
 - PMIx attempts to address this range of use-cases with a flexible API.

3 7.2.1 PMIx Allocation request

Summarv

10 Request an allocation operation from the host resource manager. **Format** 11 PMIx v3.0 12 pmix status t PMIx Allocation request (pmix alloc directive t directive, 13 14 pmix_info_t info[], size_t ninfo, 15 pmix_info_t *results[], size_t *nresults); 16 IN directive 17 Allocation directive (handle) 18 IN info 19 Array of pmix info t structures (array of handles)

1

2

4 5

6

7

1 2	IN ninfo Number of elements in the <i>info</i> array (integer)
3	INOUT results
4	Address where a pointer to an array of pmix_info_t containing the results of the request
5	can be returned (memory reference)
6	INOUT nresults
7	Address where the number of elements in <i>results</i> can be returned (handle)
8	Returns one of the following:
9	 PMIX_SUCCESS, indicating that the request was processed and returned success
10	• a PMIx error constant indicating either an error in the input or that the request was refused
	Required Attributes
11	PMIx libraries are not required to directly support any attributes for this function. However, any
12	provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is
13	required to add the PMIX_USERID and the PMIX_GRPID attributes of the client process making
14	the request.
15	
16 17	Host environments that implement support for this operation are required to support the following attributes:
18	<pre>PMIX_ALLOC_REQ_ID "pmix.alloc.reqid" (char*)</pre>
19	User-provided string identifier for this allocation request which can later be used to query
20	status of the request.
21	<pre>PMIX_ALLOC_NUM_NODES "pmix.alloc.nnodes" (uint64_t)</pre>
22	The number of nodes.
23	<pre>PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t)</pre>
24	Number of cpus.
25	<pre>PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)</pre>
26	Time in seconds.
	▼ Optional Attributes
27	The following attributes are optional for host environments that support this operation:
28 29	<pre>PMIX_ALLOC_NODE_LIST "pmix.alloc.nlist" (char*) Regular expression of the specific nodes.</pre>
29	
30	PMIX_ALLOC_NUM_CPU_LIST "pmix.alloc.ncpulist" (char*)
31	Regular expression of the number of cpus for each node.
32	PMIX ALLOC CPU LIST "pmix.alloc.cpulist" (char*)

1	Regular expression of the specific cpus indicating the cpus involved.
2 3	PMIX_ALLOC_MEM_SIZE "pmix.alloc.msize" (float) Number of Megabytes.
4 5 6 7	<pre>PMIX_ALLOC_FABRIC "pmix.alloc.net" (array) Array of pmix_info_t describing requested fabric resources. This must include at least: PMIX_ALLOC_FABRIC_ID, PMIX_ALLOC_FABRIC_TYPE, and PMIX_ALLOC_FABRIC_ENDPTS, plus whatever other descriptors are desired.</pre>
8 9 10 11 12 13 14 15 16 17 18 19 20 21	The key to be used when accessing this requested fabric allocation. The allocation will be returned/stored as a pmix_data_array_t of pmix_info_t indexed by this key and containing at least one entry with the same key and the allocated resource description. The type of the included value depends upon the fabric support. For example, a TCP allocation might consist of a comma-delimited string of socket ranges such as "32000-32100,33005,38123-38146". Additional entries will consist of any provided resource request directives, along with their assigned values. Examples include: PMIX_ALLOC_FABRIC_TYPE - the type of resources provided; PMIX_ALLOC_FABRIC_PLANE - if applicable, what plane the resources were assigned from; PMIX_ALLOC_FABRIC_QOS - the assigned QoS; PMIX_ALLOC_BANDWIDTH - the allocated bandwidth; PMIX_ALLOC_FABRIC_SEC_KEY - a security key for the requested fabric allocation. NOTE: the assigned values may differ from those requested, especially if PMIX_INFO_REQD was not set in the request.
22 23	PMIX_ALLOC_BANDWIDTH "pmix.alloc.bw" (float) Mbits/sec.
24 25	<pre>PMIX_ALLOC_FABRIC_QOS</pre>
26 27	<pre>PMIX_ALLOC_FABRIC_TYPE "pmix.alloc.nettype" (char*) Type of desired transport (e.g., "tcp", "udp")</pre>
28 29	<pre>PMIX_ALLOC_FABRIC_PLANE "pmix.alloc.netplane" (char*) ID string for the NIC (aka plane) to be used for this allocation (e.g., CIDR for Ethernet)</pre>
30 31	<pre>PMIX_ALLOC_FABRIC_ENDPTS "pmix.alloc.endpts" (size_t) Number of endpoints to allocate per process</pre>
32 33	<pre>PMIX_ALLOC_FABRIC_ENDPTS_NODE "pmix.alloc.endpts.nd" (size_t) Number of endpoints to allocate per node</pre>
34 35	PMIX_ALLOC_FABRIC_SEC_KEY "pmix.alloc.nsec" (pmix_byte_object_t) Fabric security key

Description

1

3

5

6

7

8

9

10 11

12

13

14 15

16 17

19 20

36

Request an allocation operation from the host resource manager. Several broad categories are envisioned, including the ability to:

- Request allocation of additional resources, including memory, bandwidth, and compute. This
 should be accomplished in a non-blocking manner so that the application can continue to
 progress while waiting for resources to become available. Note that the new allocation will be
 disjoint from (i.e., not affiliated with) the allocation of the requestor thus the termination of one
 allocation will not impact the other.
- Extend the reservation on currently allocated resources, subject to scheduling availability and priorities. This includes extending the time limit on current resources, and/or requesting additional resources be allocated to the requesting job. Any additional allocated resources will be considered as part of the current allocation, and thus will be released at the same time.
- Return no-longer-required resources to the scheduler. This includes the "loan" of resources back to the scheduler with a promise to return them upon subsequent request.

If successful, the returned results for a request for additional resources must include the host resource manager's identifier (PMIX_ALLOC_ID) that the requester can use to specify the resources in, for example, a call to PMIx_Spawn .

3 7.2.2 PMIx Allocation request nb

Returns one of the following:

Summary

Request an allocation operation from the host resource manager.

```
21
              Format
   PMIx v2.0
22
              pmix_status_t
23
              PMIx_Allocation_request_nb(pmix_alloc_directive_t directive,
                                               pmix_info_t info[], size_t ninfo,
24
                                               pmix_info_cbfunc_t cbfunc, void *cbdata);
25
              IN
26
                   directive
27
                  Allocation directive (handle)
              IN
                 info
28
29
                  Array of pmix info t structures (array of handles)
              IN
30
31
                  Number of elements in the info array (integer)
32
              IN
                   cbfunc
33
                  Callback function pmix info cbfunc t (function reference)
34
              IN
                   cbdata
35
                  Data to be passed to the callback function (memory reference)
```

1 2 3	• PMIX_SUCCESS , indicating that the request is being processed by the host environment - result will be returned in the provided <i>cbfunc</i> . Note that the library must not invoke the callback function prior to returning from the API.
4 5	• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will <i>not</i> be called
6 7	 a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will <i>not</i> be called Required Attributes
8 9 10 11	PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is <i>required</i> to add the PMIX_USERID and the PMIX_GRPID attributes of the client process making the request.
13 14	Host environments that implement support for this operation are required to support the following attributes:
15 16 17	<pre>PMIX_ALLOC_REQ_ID "pmix.alloc.reqid" (char*) User-provided string identifier for this allocation request which can later be used to query status of the request.</pre>
18 19	<pre>PMIX_ALLOC_NUM_NODES</pre>
20 21	<pre>PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t) Number of cpus.</pre>
22 23	PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t) Time in seconds.
	▼ Optional Attributes
24	The following attributes are optional for host environments that support this operation:
25 26	<pre>PMIX_ALLOC_NODE_LIST "pmix.alloc.nlist" (char*) Regular expression of the specific nodes.</pre>
27 28	<pre>PMIX_ALLOC_NUM_CPU_LIST "pmix.alloc.ncpulist" (char*) Regular expression of the number of cpus for each node.</pre>
29 30	<pre>PMIX_ALLOC_CPU_LIST "pmix.alloc.cpulist" (char*) Regular expression of the specific cpus indicating the cpus involved.</pre>
31 32	PMIX_ALLOC_MEM_SIZE "pmix.alloc.msize" (float) Number of Megabytes.

```
1
               PMIX_ALLOC_FABRIC "pmix.alloc.net" (array)
                    Array of pmix info t describing requested fabric resources. This must include at least:
2
                    PMIX ALLOC FABRIC ID, PMIX ALLOC FABRIC TYPE, and
 3
 4
                    PMIX_ALLOC_FABRIC_ENDPTS, plus whatever other descriptors are desired.
5
               PMIX_ALLOC_FABRIC_ID "pmix.alloc.netid" (char*)
6
                    The key to be used when accessing this requested fabric allocation. The allocation will be
                    returned/stored as a pmix_data_array_t of pmix_info_t indexed by this key and
7
8
                    containing at least one entry with the same key and the allocated resource description. The
9
                    type of the included value depends upon the fabric support. For example, a TCP allocation
                    might consist of a comma-delimited string of socket ranges such as
10
11
                    "32000-32100,33005,38123-38146". Additional entries will consist of any provided
12
                    resource request directives, along with their assigned values. Examples include:
13
                    PMIX ALLOC FABRIC TYPE - the type of resources provided;
                    PMIX ALLOC FABRIC PLANE - if applicable, what plane the resources were assigned
14
                    from; PMIX ALLOC FABRIC QOS - the assigned QoS; PMIX ALLOC BANDWIDTH -
15
                    the allocated bandwidth; PMIX ALLOC FABRIC SEC KEY - a security key for the
16
17
                    requested fabric allocation. NOTE: the assigned values may differ from those requested,
                    especially if PMIX_INFO_REQD was not set in the request.
18
19
               PMIX_ALLOC_BANDWIDTH "pmix.alloc.bw" (float)
                    Mbits/sec.
20
21
               PMIX ALLOC FABRIC QOS "pmix.alloc.netgos" (char*)
                    Ouality of service level.
22
23
               PMIX_ALLOC_FABRIC_TYPE "pmix.alloc.nettype" (char*)
                    Type of desired transport (e.g., "tcp", "udp")
24
25
               PMIX_ALLOC_FABRIC_PLANE "pmix.alloc.netplane" (char*)
                    ID string for the NIC (aka plane) to be used for this allocation (e.g., CIDR for Ethernet)
26
27
               PMIX_ALLOC_FABRIC_ENDPTS "pmix.alloc.endpts" (size_t)
28
                    Number of endpoints to allocate per process
               PMIX ALLOC FABRIC ENDPTS NODE "pmix.alloc.endpts.nd" (size t)
29
                    Number of endpoints to allocate per node
30
               PMIX ALLOC FABRIC SEC KEY "pmix.alloc.nsec" (pmix byte object t)
31
32
                    Fabric security key
33
```

Description

34

Non-blocking form of the **PMIx_Allocation_request** API.

7.3 Job Control

 This section defines APIs that enable the application and host environment to coordinate the response to failures and other events. This can include requesting termination of the entire job or a subset of processes within a job, but can also be used in combination with other PMIx capabilities (e.g., allocation support and event notification) for more nuanced responses. For example, an application notified of an incipient over-temperature condition on a node could use the PMIx_Allocation_request_nb interface to request replacement nodes while simultaneously using the PMIx_Job_control_nb interface to direct that a checkpoint event be delivered to all processes in the application. If replacement resources are not available, the application might use the PMIx_Job_control_nb interface to request that the job continue at a lower power setting, perhaps sufficient to avoid the over-temperature failure.

The job control APIs can also be used by an application to register itself as available for preemption when operating in an environment such as a cloud or where incentives, financial or otherwise, are provided to jobs willing to be preempted. Registration can include attributes indicating how many resources are being offered for preemption (e.g., all or only some portion), whether the application will require time to prepare for preemption, etc. Jobs that request a warning will receive an event notifying them of an impending preemption (possibly including information as to the resources that will be taken away, how much time the application will be given prior to being preempted, whether the preemption will be a suspension or full termination, etc.) so they have an opportunity to save their work. Once the application is ready, it calls the provided event completion callback function to indicate that the SMS is free to suspend or terminate it, and can include directives regarding any desired restart.

7.3.1 PMIx_Job_control

```
Summary
24
             Request a job control action.
25
             Format
26
   PMIx v3.0
27
             pmix status t
             PMIx_Job_control(const pmix_proc_t targets[], size_t ntargets,
28
                                 const pmix_info_t directives[], size_t ndirs,
29
                                 pmix_info_t *results[], size_t *nresults)
30
                                                – C —
31
             IN
                  targets
                  Array of proc structures (array of handles)
32
33
             IN
                  ntargets
                  Number of element in the targets array (integer)
34
35
             IN
                  directives
```

Array of info structures (array of handles)

1	IN ndirs
2	Number of element in the <i>directives</i> array (integer) INOUT results
4	Address where a pointer to an array of pmix_info_t containing the results of the request
5	can be returned (memory reference)
6	INOUT nresults
7	Address where the number of elements in <i>results</i> can be returned (handle)
8	Returns one of the following:
9 10	• PMIX_SUCCESS, indicating that the request was processed by the host environment and returned <i>success</i> . Details of the result will be returned in the <i>results</i> array
11	• a PMIx error constant indicating either an error in the input or that the request was refused
	▼ Required Attributes
12	PMIx libraries are not required to directly support any attributes for this function. However, any
13	provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is
14 15	required to add the PMIX_USERID and the PMIX_GRPID attributes of the client process making the request.
	the request.
16	
17 18	Host environments that implement support for this operation are required to support the following attributes:
19	PMIX_JOB_CTRL_ID "pmix.jctrl.id" (char*)
20	Provide a string identifier for this request. The user can provide an identifier for the
21	requested operation, thus allowing them to later request status of the operation or to
22	terminate it. The host, therefore, shall track it with the request for future reference.
23	PMIX_JOB_CTRL_PAUSE "pmix.jctrl.pause" (bool)
24	Pause the specified processes.
25	PMIX_JOB_CTRL_RESUME "pmix.jctrl.resume" (bool)
26	Resume ("un-pause") the specified processes.
27	PMIX_JOB_CTRL_KILL "pmix.jctrl.kill" (bool)
28	Forcibly terminate the specified processes and cleanup.
29	PMIX_JOB_CTRL_SIGNAL "pmix.jctrl.sig" (int)
30	Send given signal to specified processes.
31	PMIX_JOB_CTRL_TERMINATE "pmix.jctrl.term" (bool)
32	Politely terminate the specified processes.
33	PMIX_REGISTER_CLEANUP "pmix.reg.cleanup" (char*)
34	Comma-delimited list of files to be removed upon process termination
35	PMIX_REGISTER_CLEANUP_DIR "pmix.reg.cleanupdir" (char*)

```
Comma-delimited list of directories to be removed upon process termination
 1
              PMIX CLEANUP RECURSIVE "pmix.clnup.recurse" (bool)
 2
                   Recursively cleanup all subdirectories under the specified one(s)
 3
              PMIX CLEANUP EMPTY "pmix.clnup.empty" (bool)
 4
 5
                   Only remove empty subdirectories
6
              PMIX CLEANUP IGNORE "pmix.clnup.ignore" (char*)
                   Comma-delimited list of filenames that are not to be removed
 7
8
              PMIX_CLEANUP_LEAVE_TOPDIR "pmix.clnup.lvtop" (bool)
9
                   When recursively cleaning subdirectories, do not remove the top-level directory (the one
10
                   given in the cleanup request)
                                             Optional Attributes
11
              The following attributes are optional for host environments that support this operation:
              PMIX_JOB_CTRL_CANCEL "pmix.jctrl.cancel" (char*)
12
                   Cancel the specified request - the provided request ID must match the
13
                   PMIX_JOB_CTRL_ID provided to a previous call to PMIx_Job_control . An ID of
14
15
                   NULL implies cancel all requests from this requestor.
16
              PMIX JOB CTRL RESTART "pmix.jctrl.restart" (char*)
                   Restart the specified processes using the given checkpoint ID.
17
18
              PMIX_JOB_CTRL_CHECKPOINT "pmix.jctrl.ckpt" (char*)
19
                   Checkpoint the specified processes and assign the given ID to it.
              PMIX JOB CTRL CHECKPOINT_EVENT "pmix.jctrl.ckptev" (bool)
20
21
                   Use event notification to trigger a process checkpoint.
22
              PMIX_JOB_CTRL_CHECKPOINT_SIGNAL "pmix.jctrl.ckptsig" (int)
23
                   Use the given signal to trigger a process checkpoint.
              PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT "pmix.jctrl.ckptsig" (int)
24
25
                   Time in seconds to wait for a checkpoint to complete.
26
              PMIX JOB CTRL CHECKPOINT METHOD
              "pmix.jctrl.ckmethod" (pmix_data_array_t)
27
                   Array of pmix_info_t declaring each method and value supported by this application.
28
29
              PMIX_JOB_CTRL_PROVISION "pmix.jctrl.pvn" (char*)
                   Regular expression identifying nodes that are to be provisioned.
30
              PMIX_JOB_CTRL_PROVISION_IMAGE "pmix.jctrl.pvnimg" (char*)
31
                   Name of the image that is to be provisioned.
32
33
              PMIX JOB CTRL PREEMPTIBLE "pmix.jctrl.preempt" (bool)
```

204

1		A .	Indicate that the job can be pre-empted.
2		Des	scription
3			uest a job control action. The <i>targets</i> array identifies the processes to which the requested job
4		cont	rol action is to be applied. A NULL value can be used to indicate all processes in the caller's
5 6			espace. The use of PMIX_RANK_WILDCARD can also be used to indicate that all processes in given namespace are to be included.
			•
7 8			directives are provided as pmix_info_t structures in the <i>directives</i> array. The callback stion provides a <i>status</i> to indicate whether or not the request was granted, and to provide some
9			rmation as to the reason for any denial in the pmix_info_cbfunc_t array of
10			x_info_t structures.
11	7.3.2	PM1	[x_Job_control_nb
12			mmary
13		Req	uest a job control action.
14		For	rmat
	PMIx v2.0		C
15		nmi	.x_status_t
16		-	<pre>x_Job_control_nb(const pmix_proc_t targets[], size_t ntargets,</pre>
17			const pmix_info_t directives[], size_t ndirs,
8			<pre>pmix_info_cbfunc_t cbfunc, void *cbdata)</pre>
			C
19		IN	targets
20		18.1	Array of proc structures (array of handles)
21		IN	ntargets Number of allowest in the towards array (interest)
22		INI	Number of element in the <i>targets</i> array (integer)
23 24		IN	Array of info atrustures (array of handles)
25		IN	Array of info structures (array of handles) ndirs
26		114	Number of element in the <i>directives</i> array (integer)
27		IN	cbfunc
28		114	Callback function pmix_info_cbfunc_t (function reference)
29		IN	cbdata
30			Data to be passed to the callback function (memory reference)
31		Retu	arns one of the following:
			-
32			MIX_SUCCESS, indicating that the request is being processed by the host environment - result ill be returned in the provided obtains. Note that the library must not invoke the callback
33 34			ill be returned in the provided <i>cbfunc</i> . Note that the library must not invoke the callback unction prior to returning from the API.
۱ ۲ر		10	menon prior to returning from the Arri.

1 2	 PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned success - the cbfunc will not be called
3 4	• a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will <i>not</i> be called
	▼ Required Attributes
5 6 7 8	PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is required to add the PMIX_USERID and the PMIX_GRPID attributes of the client process making the request.
10 11	Host environments that implement support for this operation are required to support the following attributes:
12 13 14 15	<pre>PMIX_JOB_CTRL_ID "pmix.jctrl.id" (char*) Provide a string identifier for this request. The user can provide an identifier for the requested operation, thus allowing them to later request status of the operation or to terminate it. The host, therefore, shall track it with the request for future reference.</pre>
16 17	PMIX_JOB_CTRL_PAUSE "pmix.jctrl.pause" (bool) Pause the specified processes.
18 19	<pre>PMIX_JOB_CTRL_RESUME "pmix.jctrl.resume" (bool) Resume ("un-pause") the specified processes.</pre>
20 21	PMIX_JOB_CTRL_KILL "pmix.jctrl.kill" (bool) Forcibly terminate the specified processes and cleanup.
22 23	PMIX_JOB_CTRL_SIGNAL "pmix.jctrl.sig" (int) Send given signal to specified processes.
24 25	<pre>PMIX_JOB_CTRL_TERMINATE "pmix.jctrl.term" (bool) Politely terminate the specified processes.</pre>
26 27	<pre>PMIX_REGISTER_CLEANUP "pmix.reg.cleanup" (char*) Comma-delimited list of files to be removed upon process termination</pre>
28 29	<pre>PMIX_REGISTER_CLEANUP_DIR "pmix.reg.cleanupdir" (char*) Comma-delimited list of directories to be removed upon process termination</pre>
30 31	PMIX_CLEANUP_RECURSIVE "pmix.clnup.recurse" (bool) Recursively cleanup all subdirectories under the specified one(s)
32 33	PMIX_CLEANUP_EMPTY "pmix.clnup.empty" (bool) Only remove empty subdirectories
34	PMIX_CLEANUP_IGNORE "pmix.clnup.ignore" (char*)

1	Comma-delimited list of filenames that are not to be removed
2 3 4	<pre>PMIX_CLEANUP_LEAVE_TOPDIR "pmix.clnup.lvtop" (bool) When recursively cleaning subdirectories, do not remove the top-level directory (the one given in the cleanup request)</pre>
	▼ Optional Attributes
5	The following attributes are optional for host environments that support this operation:
6 7 8 9	<pre>PMIX_JOB_CTRL_CANCEL "pmix.jctrl.cancel" (char*) Cancel the specified request - the provided request ID must match the PMIX_JOB_CTRL_ID provided to a previous call to PMIx_Job_control . An ID of NULL implies cancel all requests from this requestor.</pre>
0 1	<pre>PMIX_JOB_CTRL_RESTART "pmix.jctrl.restart" (char*) Restart the specified processes using the given checkpoint ID.</pre>
2 3	<pre>PMIX_JOB_CTRL_CHECKPOINT "pmix.jctrl.ckpt" (char*)</pre>
4 5	<pre>PMIX_JOB_CTRL_CHECKPOINT_EVENT "pmix.jctrl.ckptev" (bool) Use event notification to trigger a process checkpoint.</pre>
6 7	<pre>PMIX_JOB_CTRL_CHECKPOINT_SIGNAL "pmix.jctrl.ckptsig" (int) Use the given signal to trigger a process checkpoint.</pre>
8 9	<pre>PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT "pmix.jctrl.ckptsig" (int) Time in seconds to wait for a checkpoint to complete.</pre>
0 1 2	PMIX_JOB_CTRL_CHECKPOINT_METHOD "pmix.jctrl.ckmethod" (pmix_data_array_t) Array of pmix_info_t declaring each method and value supported by this application.
3 4	<pre>PMIX_JOB_CTRL_PROVISION "pmix.jctrl.pvn" (char*) Regular expression identifying nodes that are to be provisioned.</pre>
5 6	<pre>PMIX_JOB_CTRL_PROVISION_IMAGE "pmix.jctrl.pvnimg" (char*) Name of the image that is to be provisioned.</pre>
7 8	PMIX_JOB_CTRL_PREEMPTIBLE "pmix.jctrl.preempt" (bool) Indicate that the job can be pre-empted.

Description

1 2

3

4

5

6

7

8 9

11

12

13

14

15

16 17

18

19

20

21

22

24 25

26

31

32

33

34

PMIx v3.0

Non-blocking form of the PMIx Job control API. The targets array identifies the processes to which the requested job control action is to be applied. A **NULL** value can be used to indicate all processes in the caller's namespace. The use of PMIX RANK WILDCARD can also be used to indicate that all processes in the given namespace are to be included.

The directives are provided as **pmix** info t structures in the *directives* array. The callback function provides a *status* to indicate whether or not the request was granted, and to provide some information as to the reason for any denial in the pmix info cbfunc t array of pmix info t structures.

7.4 **Process and Job Monitoring**

In addition to external faults, a common problem encountered in HPC applications is a failure to make progress due to some internal conflict in the computation. These situations can result in a significant waste of resources as the SMS is unaware of the problem, and thus cannot terminate the job. Various watchdog methods have been developed for detecting this situation, including requiring a periodic "heartbeat" from the application and monitoring a specified file for changes in size and/or modification time.

At the request of SMS vendors and members, a monitoring support interface has been included in the PMIx v2 standard. The defined API allows applications to request monitoring, directing what is to be monitored, the frequency of the associated check, whether or not the application is to be notified (via the event notification subsystem) of stall detection, and other characteristics of the operation. In addition, heartbeat and file monitoring methods have been included in the PRI but are active only when requested.

7.4.1 PMIx Process monitor

Summary

Request that application processes be monitored.

Format

```
27
            pmix status t
28
            PMIx Process monitor(const pmix info t *monitor, pmix status t error,
29
                                  const pmix info t directives[], size t ndirs,
30
                                 pmix_info_t *results[], size_t *nresults)
```

monitor info (handle) IN error status (integer)

1 2 3	IN directives
4 5 6 7 8 9	Number of elements in the <i>directives</i> array (integer) INOUT results Address where a pointer to an array of pmix_info_t containing the results of the request can be returned (memory reference) INOUT nresults Address where the number of elements in results can be returned (handle)
10	Returns one of the following:
1 2	 PMIX_SUCCESS, indicating that the request was processed and returned success. Details of the result will be returned in the results array
13	• a PMIx error constant indicating either an error in the input or that the request was refused
	→ Optional Attributes
4 5 6 7	The following attributes may be implemented by a PMIx library or by the host environment. If supported by the PMIx server library, then the library must not pass the supported attributes to the host environment. All attributes not directly supported by the server library must be passed to the host environment if it supports this operation, and the library is <i>required</i> to add the PMIX_USERID and the PMIX_GRPID attributes of the requesting process:
19 20	<pre>PMIX_MONITOR_ID "pmix.monitor.id" (char*) Provide a string identifier for this request.</pre>
21 22	<pre>PMIX_MONITOR_CANCEL "pmix.monitor.cancel" (char*) Identifier to be canceled (NULL means cancel all monitoring for this process).</pre>
23 24	<pre>PMIX_MONITOR_APP_CONTROL "pmix.monitor.appctrl" (bool) The application desires to control the response to a monitoring event.</pre>
25 26	PMIX_MONITOR_HEARTBEAT "pmix.monitor.mbeat" (void) Register to have the PMIx server monitor the requestor for heartbeats.
27 28	<pre>PMIX_MONITOR_HEARTBEAT_TIME "pmix.monitor.btime" (uint32_t) Time in seconds before declaring heartbeat missed.</pre>
29 30	PMIX_MONITOR_HEARTBEAT_DROPS "pmix.monitor.bdrop" (uint32_t) Number of heartbeats that can be missed before generating the event.
31 32	<pre>PMIX_MONITOR_FILE "pmix.monitor.fmon" (char*) Register to monitor file for signs of life.</pre>
33 34	<pre>PMIX_MONITOR_FILE_SIZE "pmix.monitor.fsize" (bool) Monitor size of given file is growing to determine if the application is running.</pre>
35	PMIX MONITOR FILE ACCESS "pmix.monitor.faccess" (char*)

Monitor time since last access of given file to determine if the application is running. 1 PMIX MONITOR FILE MODIFY "pmix.monitor.fmod" (char*) 2 Monitor time since last modified of given file to determine if the application is running. 3 PMIX MONITOR FILE CHECK TIME "pmix.monitor.ftime" (uint32 t) 4 5 Time in seconds between checking the file. 6 PMIX_MONITOR_FILE_DROPS "pmix.monitor.fdrop" (uint32_t) 7 Number of file checks that can be missed before generating the event. **Description** 8 9 Request that application processes be monitored via several possible methods. For example, that the server monitor this process for periodic heartbeats as an indication that the process has not 10 become "wedged". When a monitor detects the specified alarm condition, it will generate an event 11 notification using the provided error code and passing along any available relevant information. It 12 13 is up to the caller to register a corresponding event handler. 14 The *monitor* argument is an attribute indicating the type of monitor being requested. For example, PMIX_MONITOR_FILE to indicate that the requestor is asking that a file be monitored. 15 The error argument is the status code to be used when generating an event notification alerting that 16 the monitor has been triggered. The range of the notification defaults to 17 PMIX RANGE_NAMESPACE. This can be changed by providing a PMIX_RANGE directive. 18 19 The directives argument characterizes the monitoring request (e.g., monitor file size) and frequency 20 of checking to be done 7.4.2 PMIx Process monitor nb Summary 22 23 Request that application processes be monitored. **Format** 24 PMIx v2.025 pmix_status_t PMIx_Process_monitor_nb(const pmix_info_t *monitor, pmix_status_t error, 26 const pmix info t directives[], size t ndirs, 27 pmix info cbfunc t cbfunc, void *cbdata) 28

C	
IN monitor	
info (handle)	
IN error	
status (integer)	
IN directives	
Array of info structures (array of handles)	
IN ndirs	
Number of elements in the <i>directives</i> array (integer)	
IN cbfunc	
Callback function pmix_info_cbfunc_t (function reference)	
IN cbdata	
Data to be passed to the callback function (memory reference)	
Returns one of the following:	
• PMIX_SUCCESS, indicating that the request is being processed by the host environment - rewill be returned in the provided <i>cbfunc</i> . Note that the library must not invoke the callback function prior to returning from the API.	esult
• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed a returned <i>success</i> - the <i>cbfunc</i> will <i>not</i> be called	and
• a PMIx error constant indicating either an error in the input or that the request was immediate processed and failed - the <i>cbfunc</i> will <i>not</i> be called	ely
▼ Optional Attributes	
The following attributes may be implemented by a PMIx library or by the host environment. If supported by the PMIx server library, then the library must not pass the supported attributes to host environment. All attributes not directly supported by the server library must be passed to the host environment if it supports this operation, and the library is <i>required</i> to add the PMIX_USERID and the PMIX_GRPID attributes of the requesting process:	
PMIX_MONITOR_ID "pmix.monitor.id" (char*) Provide a string identifier for this request.	
<pre>PMIX_MONITOR_CANCEL "pmix.monitor.cancel" (char*) Identifier to be canceled (NULL means cancel all monitoring for this process).</pre>	
PMIX_MONITOR_APP_CONTROL "pmix.monitor.appctrl" (bool) The application desires to control the response to a monitoring event.	
PMIX_MONITOR_HEARTBEAT "pmix.monitor.mbeat" (void)	

Register to have the PMIx server monitor the requestor for heartbeats.

Time in seconds before declaring heartbeat missed.

1	PMIX_MONITOR_HEARTBEAT_DROPS "pmix.monitor.bdrop" (uint32_t) Number of heartbeats that can be missed before generating the event.
3 4	<pre>PMIX_MONITOR_FILE "pmix.monitor.fmon" (char*) Register to monitor file for signs of life.</pre>
5 6	<pre>PMIX_MONITOR_FILE_SIZE "pmix.monitor.fsize" (bool) Monitor size of given file is growing to determine if the application is running.</pre>
7 8	<pre>PMIX_MONITOR_FILE_ACCESS "pmix.monitor.faccess" (char*) Monitor time since last access of given file to determine if the application is running.</pre>
9 10	<pre>PMIX_MONITOR_FILE_MODIFY "pmix.monitor.fmod" (char*) Monitor time since last modified of given file to determine if the application is running.</pre>
11 12	<pre>PMIX_MONITOR_FILE_CHECK_TIME "pmix.monitor.ftime" (uint32_t) Time in seconds between checking the file.</pre>
13 14	PMIX_MONITOR_FILE_DROPS "pmix.monitor.fdrop" (uint32_t) Number of file checks that can be missed before generating the event.
15 16 17 18	Description Non-blocking form of the PMIx_Process_monitor API. The <i>cbfunc</i> function provides a <i>status</i> to indicate whether or not the request was granted, and to provide some information as to the reason for any denial in the pmix_info_cbfunc_t array of pmix_info_t structures.
19 7.4.3	PMIx_Heartbeat
20 21	Summary Send a heartbeat to the PMIx server library
22 <i>PMIx v2.0</i>	Format C
23	PMIx_Heartbeat (void)
24 25 26	Description A simplified macro wrapping PMIx_Process_monitor_nb that sends a heartbeat to the PMIx server library.

₁ 7.5 Logging

The logging interface supports posting information by applications and SMS elements to persistent storage. This function is *not* intended for output of computational results, but rather for reporting status and saving state information such as inserting computation progress reports into the application's SMS job log or error reports to the local syslog.

7.5.1 PMIx_Log

```
Summary
 7
 8
               Log data to a data service.
               Format
 9
   PMIx v3.0
10
               pmix_status_t
               PMIx_Log(const pmix_info_t data[], size_t ndata,
11
                           const pmix_info_t directives[], size_t ndirs)
12
                                                          C
               IN
13
                    data
                    Array of info structures (array of handles)
14
               IN
                   ndata
15
16
                    Number of elements in the data array (size t)
17
                    directives
                    Array of info structures (array of handles)
18
               IN
                   ndirs
19
20
                    Number of elements in the directives array (size_t)
               Return codes are one of the following:
21
22
                PMIX SUCCESS The logging request was successful.
                PMIX_ERR_BAD_PARAM The logging request contains at least one incorrect entry.
23
24
                PMIX_ERR_NOT_SUPPORTED The PMIx implementation or host environment does not
25
                    support this function.
                                                Required Attributes
               If the PMIx library does not itself perform this operation, then it is required to pass any attributes
26
27
               provided by the client to the host environment for processing. In addition, it must include the
               following attributes in the passed info array:
28
29
               PMIX_USERID "pmix.euid" (uint32_t)
                     Effective user id.
30
31
               PMIX_GRPID "pmix.egid" (uint32_t)
32
                     Effective group id.
```

```
1
             Host environments or PMIx libraries that implement support for this operation are required to
 2
             support the following attributes:
             PMIX LOG STDERR "pmix.log.stderr" (char*)
 4
5
                   Log string to stderr.
             PMIX LOG STDOUT "pmix.log.stdout" (char*)
6
7
                   Log string to stdout.
             PMIX LOG SYSLOG "pmix.log.syslog" (char*)
8
                   Log data to syslog. Defaults to ERROR priority. Will log to global syslog if available,
9
                   otherwise to local syslog
10
11
             PMIX LOG LOCAL SYSLOG "pmix.log.lsys" (char*)
                   Log data to local syslog. Defaults to ERROR priority.
12
             PMIX LOG GLOBAL SYSLOG "pmix.log.gsys" (char*)
13
                   Forward data to system "gateway" and log msg to that syslog Defaults to ERROR priority.
14
             PMIX LOG SYSLOG PRI "pmix.log.syspri" (int)
15
16
                   Syslog priority level
17
             PMIX_LOG_ONCE "pmix.log.once" (bool)
                   Only log this once with whichever channel can first support it, taking the channels in priority
18
19

    ▼------ Optional Attributes ------

             The following attributes are optional for host environments or PMIx libraries that support this
20
21
             operation:
22
             PMIX LOG SOURCE "pmix.log.source" (pmix proc t*)
                   ID of source of the log request
23
24
             PMIX LOG TIMESTAMP "pmix.log.tstmp" (time t)
25
                   Timestamp for log report
             PMIX LOG GENERATE TIMESTAMP "pmix.log.gtstmp" (bool)
26
27
                   Generate timestamp for log
             PMIX LOG TAG OUTPUT "pmix.log.tag" (bool)
28
                   Label the output stream with the channel name (e.g., "stdout")
29
             PMIX LOG TIMESTAMP OUTPUT "pmix.log.tsout" (bool)
30
31
                   Print timestamp in output string
32
             PMIX LOG XML OUTPUT "pmix.log.xml" (bool)
33
                   Print the output stream in XML format
```

```
Log via email based on pmix info t containing directives.
 2
 3
               PMIX_LOG_EMAIL_ADDR "pmix.log.emaddr" (char*)
                     Comma-delimited list of email addresses that are to receive the message.
 4
 5
               PMIX_LOG_EMAIL_SUBJECT "pmix.log.emsub" (char*)
 6
                     Subject line for email.
               PMIX_LOG_EMAIL_MSG "pmix.log.emmsg" (char*)
 8
                     Message to be included in email.
 9
               PMIX_LOG_JOB_RECORD "pmix.log.jrec" (bool)
10
                     Log the provided information to the host environment's job record
               PMIX_LOG_GLOBAL_DATASTORE "pmix.log.gstore" (bool)
11
                     Store the log data in a global data store (e.g., database)
12
13
               Description
               Log data subject to the services offered by the host environment. The data to be logged is provided
14
               in the data array. The (optional) directives can be used to direct the choice of logging channel.
15
                                           Advice to users
16
               It is strongly recommended that the PMIx_Log API not be used by applications for streaming data
               as it is not a "performant" transport and can perturb the application since it involves the local PMIx
```

server and host SMS daemon. Note that a return of PMIX_SUCCESS only denotes that the data

was successfully handed to the appropriate system call (for local channels) or the host environment

PMIX_LOG_EMAIL "pmix.log.email" (pmix_data_array_t)

21 **7.5.2 PMIx_Log_nb**

22 Summary

1

17 18

19

20

23

Log data to a data service.

and does not indicate receipt at the final destination.

1		Format
	PMIx v2.0	
2		pmix_status_t
3		<pre>PMIx_Log_nb(const pmix_info_t data[], size_t ndata,</pre>
4		<pre>const pmix_info_t directives[], size_t ndirs,</pre>
5		pmix_op_cbfunc_t cbfunc, void *cbdata)
		C
6		IN data
7		Array of info structures (array of handles)
8		IN ndata
9		Number of elements in the <i>data</i> array (size_t)
10		IN directives
11		Array of info structures (array of handles)
12		IN ndirs
13		Number of elements in the <i>directives</i> array (size_t)
14		IN cbfunc
15		Callback function pmix_op_cbfunc_t (function reference)
16		IN cbdata
17		Data to be passed to the callback function (memory reference)
18		Return codes are one of the following:
19		PMIX_SUCCESS The logging request is valid and is being processed. The resulting status from
20		the operation will be provided in the callback function. Note that the library must not invoke
21		the callback function prior to returning from the API.
22		PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
23		returned <i>success</i> - the <i>cbfunc</i> will <i>not</i> be called
24		PMIX_ERR_BAD_PARAM The logging request contains at least one incorrect entry that prevents
25		it from being processed. The callback function will not be called.
26		PMIX_ERR_NOT_SUPPORTED The PMIx implementation does not support this function. The
27		callback function will not be called.
		Required Attributes
28		If the PMIx library does not itself perform this operation, then it is required to pass any attributes
29		provided by the client to the host environment for processing. In addition, it must include the
30		following attributes in the passed <i>info</i> array:
31		<pre>PMIX_USERID "pmix.euid" (uint32_t)</pre>
32		Effective user id.
33		PMIX_GRPID "pmix.egid" (uint32_t)
34		Effective group id.

1	
2	Host environments or PMIx libraries that implement support for this operation are required to support the following attributes:
4 5	<pre>PMIX_LOG_STDERR "pmix.log.stderr" (char*) Log string to stderr.</pre>
6 7	<pre>PMIX_LOG_STDOUT "pmix.log.stdout" (char*) Log string to stdout.</pre>
8 9 10	<pre>PMIX_LOG_SYSLOG "pmix.log.syslog" (char*) Log data to syslog. Defaults to ERROR priority. Will log to global syslog if available, otherwise to local syslog</pre>
11 12	PMIX_LOG_LOCAL_SYSLOG "pmix.log.lsys" (char*) Log data to local syslog. Defaults to ERROR priority.
13 14	<pre>PMIX_LOG_GLOBAL_SYSLOG "pmix.log.gsys" (char*) Forward data to system "gateway" and log msg to that syslog Defaults to ERROR priority.</pre>
15 16	<pre>PMIX_LOG_SYSLOG_PRI "pmix.log.syspri" (int) Syslog priority level</pre>
17 18 19	PMIX_LOG_ONCE "pmix.log.once" (bool) Only log this once with whichever channel can first support it, taking the channels in priority order
	▼ Optional Attributes
20 21	The following attributes are optional for host environments or PMIx libraries that support this operation:
22 23	<pre>PMIX_LOG_SOURCE "pmix.log.source" (pmix_proc_t*) ID of source of the log request</pre>
24 25	<pre>PMIX_LOG_TIMESTAMP "pmix.log.tstmp" (time_t)</pre>
26 27	<pre>PMIX_LOG_GENERATE_TIMESTAMP "pmix.log.gtstmp" (bool) Generate timestamp for log</pre>
28 29	PMIX_LOG_TAG_OUTPUT "pmix.log.tag" (bool) Label the output stream with the channel name (e.g., "stdout")
30 31	<pre>PMIX_LOG_TIMESTAMP_OUTPUT</pre>
32	Print the output stream in XMI format

PMIX_LOG_EMAIL "pmix.log.email" (pmix_data_array_t) Log via email based on **pmix info** t containing directives. PMIX_LOG_EMAIL_ADDR "pmix.log.emaddr" (char*) Comma-delimited list of email addresses that are to receive the message. PMIX_LOG_EMAIL_SUBJECT "pmix.log.emsub" (char*) Subject line for email. PMIX_LOG_EMAIL_MSG "pmix.log.emmsg" (char*) Message to be included in email. PMIX_LOG_JOB_RECORD "pmix.log.jrec" (bool) Log the provided information to the host environment's job record PMIX_LOG_GLOBAL_DATASTORE "pmix.log.gstore" (bool) Store the log data in a global data store (e.g., database) 12

Description

1

2 3

4 5

6

7

8

9

10

11

13

14 15

16 17

18 19

20

21

22

Log data subject to the services offered by the host environment. The data to be logged is provided in the data array. The (optional) directives can be used to direct the choice of logging channel. The callback function will be executed when the log operation has been completed. The data and directives arrays must be maintained until the callback is provided.

Advice to users -

It is strongly recommended that the PMIx_Log_nb API not be used by applications for streaming data as it is not a "performant" transport and can perturb the application since it involves the local PMIx server and host SMS daemon. Note that a return of PMIX_SUCCESS only denotes that the data was successfully handed to the appropriate system call (for local channels) or the host environment and does not indicate receipt at the final destination.

CHAPTER 8

Event Notification

This chapter defines the PMIx event notification system. These interfaces are designed to support the reporting of events to/from clients and servers, and between library layers within a single process.

4 8.1 Notification and Management

PMIx event notification provides an asynchronous out-of-band mechanism for communicating events between application processes and/or elements of the SMS. Its uses span a wide range that includes fault notification, coordination between multiple programming libraries within a single process, and workflow orchestration for non-synchronous programming models. Events can be divided into two distinct classes:

- *Job-specific events* directly relate to a job executing within the session, such as a debugger attachment, process failure within a related job, or events generated by an application process. Events in this category are to be immediately delivered to the PMIx server library for relay to the related local processes.
- Environment events indirectly relate to a job but do not specifically target the job itself. This category includes SMS-generated events such as Error Check and Correction (ECC) errors, temperature excursions, and other non-job conditions that might directly affect a session's resources, but would never include an event generated by an application process. Note that although these do potentially impact the session's jobs, they are not directly tied to those jobs. Thus, events in this category are to be delivered to the PMIx server library only upon request.

Both SMS elements and applications can register for events of either type.

Advice to PMIx library implementers –

Race conditions can cause the registration to come after events of possible interest (e.g., a memory ECC event that occurs after start of execution but prior to registration, or an application process generating an event prior to another process registering to receive it). SMS vendors are *requested* to cache environment events for some time to mitigate this situation, but are not *required* to do so. However, PMIx implementers are *required* to cache all events received by the PMIx server library and to deliver them to registering clients in the same order in which they were received

9

10

11

3

19

26

22

36

32

Applications must be aware that they may not receive environment events that occur prior to registration, depending upon the capabilities of the host SMS.

The generator of an event can specify the target range for delivery of that event. Thus, the generator can choose to limit notification to processes on the local node, processes within the same job as the generator, processes within the same allocation, other threads within the same process, only the SMS (i.e., not to any application processes), all application processes, or to a custom range based on specific process identifiers. Only processes within the given range that register for the provided event code will be notified. In addition, the generator can use attributes to direct that the event not be delivered to any default event handlers, or to any multi-code handler (as defined below).

Event notifications provide the process identifier of the source of the event plus the event code and any additional information provided by the generator. When an event notification is received by a process, the registered handlers are scanned for their event code(s), with matching handlers assembled into an event chain for servicing. Note that users can also specify a source range when registering an event (using the same range designators described above) to further limit when they are to be invoked. When assembled, PMIx event chains are ordered based on both the specificity of the event handler and user directives at time of handler registration. By default, handlers are grouped into three categories based on the number of event codes that can trigger the callback:

- single-code handlers are serviced first as they are the most specific. These are handlers that are registered against one specific event code.
- multi-code handlers are serviced once all single-code handlers have completed. The handler will be included in the chain upon receipt of an event matching any of the provided codes.
- default handlers are serviced once all multi-code handlers have completed. These handlers are always included in the chain unless the generator specifically excludes them.

Users can specify the callback order of a handler within its category at the time of registration. Ordering can be specified either by providing the relevant returned event handler registration ID or using event handler names, if the user specified an event handler name when registering the corresponding event. Thus, users can specify that a given handler be executed before or after another handler should both handlers appear in an event chain (the ordering is ignored if the other handler isn't included). Note that ordering does not imply immediate relationships. For example, multiple handlers registered to be serviced after event handler A will all be executed after A, but are not guaranteed to be executed in any particular order amongst themselves.

In addition, one event handler can be declared as the *first* handler to be executed in the chain. This handler will always be called prior to any other handler, regardless of category, provided the incoming event matches both the specified range and event code. Only one handler can be so designated — attempts to designate additional handlers as first will return an error. Deregistration of the declared *first* handler will re-open the position for subsequent assignment.

Similarly, one event handler can be declared as the *last* handler to be executed in the chain. This handler will *always* be called after all other handlers have executed, regardless of category, provided the incoming event matches both the specified range and event code. Note that this handler will not be called if the chain is terminated by an earlier handler. Only one handler can be designated as *last* — attempts to designate additional handlers as *last* will return an error. Deregistration of the declared *last* handler will re-open the position for subsequent assignment.

Advice to users

Note that the *last* handler is called *after* all registered default handlers that match the specified range of the incoming event unless a handler prior to it terminates the chain. Thus, if the application intends to define a *last* handler, it should ensure that no default handler aborts the process before it.

Upon completing its work and prior to returning, each handler *must* call the event handler completion function provided when it was invoked (including a status code plus any information to be passed to later handlers) so that the chain can continue being progressed. PMIx automatically aggregates the status and any results of each handler (as provided in the completion callback) with status from all prior handlers so that each step in the chain has full knowledge of what preceded it. An event handler can terminate all further progress along the chain by passing the PMIX EVENT ACTION COMPLETE status to the completion callback function.

7 8.1.1 PMIx_Register_event_handler

Summary

Register an event handler

Format

PMIx v2.0

 IN codes

Array of status codes (array of pmix_status_t)

IN ncodes

Number of elements in the *codes* array (size_t)

IN info

Array of info structures (array of handles)

1 2 3 4 5 6 7 8	<pre>IN ninfo Number of elements in the info array (size_t) IN evhdlr Event handler to be called pmix_notification_fn_t (function reference) IN cbfunc Callback function pmix_evhdlr_reg_cbfunc_t (function reference) IN cbdata Data to be passed to the cbfunc callback function (memory reference)</pre>
9 0 1	If <i>cbfunc</i> is NULL , the function call will be treated as a <i>blocking</i> call. In this case, the returned status will be either (a) the event handler reference identifier if the value is greater than or equal to zero, or (b) a negative error code indicative of the reason for the failure.
12 13	If the <i>cbfunc</i> is non- NULL , the function call will be treated as a <i>non-blocking</i> call and will return the following:
14 15 16 17 18	PMIX_SUCCESS indicating that the request has been accepted for processing and the provided callback function will be executed upon completion of the operation. Note that the library must not invoke the callback function prior to returning from the API. The event handler identifier will be returned in the callback a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this case, the provided callback function will not be executed.
20 21 22	The callback function must not be executed prior to returning from the API, and no events corresponding to this registration may be delivered prior to the completion of the registration callback function (<i>cbfunc</i>).
	▼
23	The following attributes are required to be supported by all PMIx libraries:
24 25	PMIX_EVENT_HDLR_NAME "pmix.evname" (char*) String name identifying this handler.
26 27	PMIX_EVENT_HDLR_FIRST "pmix.evfirst" (bool) Invoke this event handler before any other handlers.
28 29	PMIX_EVENT_HDLR_LAST "pmix.evlast" (bool) Invoke this event handler after all other handlers have been called.
30 31	PMIX_EVENT_HDLR_FIRST_IN_CATEGORY "pmix.evfirstcat" (bool) Invoke this event handler before any other handlers in this category.
32 33	PMIX_EVENT_HDLR_LAST_IN_CATEGORY "pmix.evlastcat" (bool) Invoke this event handler after all other handlers in this category have been called.
34 35	<pre>PMIX_EVENT_HDLR_BEFORE "pmix.evbefore" (char*) Put this event handler immediately before the one specified in the (char*) value.</pre>
36	<pre>PMIX_EVENT_HDLR_AFTER "pmix.evafter" (char*)</pre>

222

1	Put this event handler immediately after the one specified in the (char*) value.
2	<pre>PMIX_EVENT_HDLR_PREPEND "pmix.evprepend" (bool) Prepend this handler to the precedence list within its category.</pre>
4 5	PMIX_EVENT_HDLR_APPEND "pmix.evappend" (bool) Append this handler to the precedence list within its category.
6 7	<pre>PMIX_EVENT_CUSTOM_RANGE "pmix.evrange" (pmix_data_array_t*) Array of pmix_proc_t defining range of event notification.</pre>
8 9	<pre>PMIX_RANGE "pmix.range" (pmix_data_range_t) Value for calls to publish/lookup/unpublish or for monitoring event notifications.</pre>
10 11 12	<pre>PMIX_EVENT_RETURN_OBJECT "pmix.evobject" (void *) Object to be returned whenever the registered callback function cbfunc is invoked. The object will only be returned to the process that registered it.</pre>
13	
14 15	Host environments that implement support for PMIx event notification are required to support the following attributes:
16 17	<pre>PMIX_EVENT_AFFECTED_PROC "pmix.evproc" (pmix_proc_t) The single process that was affected.</pre>
18 19	PMIX_EVENT_AFFECTED_PROCS "pmix.evaffected" (pmix_data_array_t*) Array of pmix_proc_t defining affected processes.
	▼ Optional Attributes
20 21 22	Host environments that support PMIx event notification <i>may</i> offer notifications for environmental events impacting the job and for SMS events relating to the job. The following attributes are optional for host environments that support this operation:
23 24	PMIX_EVENT_TERMINATE_SESSION "pmix.evterm.sess" (bool) The RM intends to terminate this session.
25 26	<pre>PMIX_EVENT_TERMINATE_JOB "pmix.evterm.job" (bool) The RM intends to terminate this job.</pre>
27 28	PMIX_EVENT_TERMINATE_NODE "pmix.evterm.node" (bool) The RM intends to terminate all processes on this node.
29 30	<pre>PMIX_EVENT_TERMINATE_PROC "pmix.evterm.proc" (bool) The RM intends to terminate just this process.</pre>
31 32	PMIX_EVENT_ACTION_TIMEOUT "pmix.evtimeout" (int) The time in seconds before the RM will execute error response.
33	PMIX_EVENT_SILENT_TERMINATION "pmix.evsilentterm" (bool)

Do not generate an event when this job normally terminates.

Description

Register an event handler to report events. Note that the codes being registered do *not* need to be PMIx error constants — any integer value can be registered. This allows for registration of non-PMIx events such as those defined by a particular SMS vendor or by an application itself.

Advice to users

In order to avoid potential conflicts, users are advised to only define codes that lie outside the range of the PMIx standard's error codes. Thus, SMS vendors and application developers should constrain their definitions to positive values or negative values beyond the PMIX EXTERNAL ERR BASE boundary.

Advice to users

As previously stated, upon completing its work, and prior to returning, each handler *must* call the event handler completion function provided when it was invoked (including a status code plus any information to be passed to later handlers) so that the chain can continue being progressed. An event handler can terminate all further progress along the chain by passing the **PMIX_EVENT_ACTION_COMPLETE** status to the completion callback function. Note that the parameters passed to the event handler (e.g., the *info* and *results* arrays) will cease to be valid once the completion function has been called - thus, any information in the incoming parameters that will be referenced following the call to the completion function must be copied.

18 8.1.2 PMIx_Deregister_event_handler

Summary

Deregister an event handler.

1		Format
	<i>PMIx v2.0</i>	· · · · · · · · · · · · · · · · · · ·
2		pmix_status_t
3		<pre>PMIx_Deregister_event_handler(size_t evhdlr_ref,</pre>
4		<pre>pmix_op_cbfunc_t cbfunc,</pre>
5		<pre>void *cbdata);</pre>
		C -
6		<pre>IN evhdlr_ref</pre>
7		Event handler ID returned by registration (size_t)
8		IN cbfunc
9		Callback function to be executed upon completion of operation <pre>pmix_op_cbfunc_t</pre> (function reference)
10 11		IN cbdata
12		Data to be passed to the cbfunc callback function (memory reference)
13		If <i>cbfunc</i> is NULL , the function will be treated as a <i>blocking</i> call and the result of the operation
14		returned in the status code.
15		If <i>cbfunc</i> is non- NULL , the function will be treated as a <i>non-blocking</i> call and return one of the
16		following:
17		• PMIX_SUCCESS, indicating that the request is being processed - result will be returned in the
18		provided cbfunc. Note that the library must not invoke the callback function prior to returning
19		from the API.
20		• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
21		returned <i>success</i> - the <i>cbfunc</i> will <i>not</i> be called
22		• a PMIx error constant indicating either an error in the input or that the request was immediately
23		processed and failed - the <i>cbfunc</i> will <i>not</i> be called
24		The returned status code will be one of the following:
25		PMIX_SUCCESS The event handler was successfully deregistered.
26		PMIX_ERR_BAD_PARAM The provided <i>evhdlr_ref</i> was unrecognized.
27		PMIX_ERR_NOT_SUPPORTED The PMIx implementation does not support event notification.
28		Description
29		Deregister an event handler. Note that no events corresponding to the referenced registration may
30		be delivered following completion of the deregistration operation (either return from the API with
31		PMIX_OPERATION_SUCCEEDED or execution of the <i>cbfunc</i>).
32	8.1.3	PMIx_Notify_event

Report an event for notification via any registered event handler.

Summary

33

34

ı		LOI	miat
	PMIx v2.0		
2		_	ix_status_t
3		PM1	<pre>Ix_Notify_event(pmix_status_t status,</pre>
4			<pre>const pmix_proc_t *source,</pre>
5			<pre>pmix_data_range_t range,</pre>
6			<pre>pmix_info_t info[], size_t ninfo,</pre>
7			<pre>pmix_op_cbfunc_t cbfunc, void *cbdata);</pre>
			C
8		IN	status
9			Status code of the event (pmix_status_t)
0		IN	source
11			Pointer to a pmix_proc_t identifying the original reporter of the event (handle)
12		IN	range
13			Range across which this notification shall be delivered (pmix_data_range_t)
14		IN	info
15			Array of pmix_info_t structures containing any further info provided by the originator of
16			the event (array of handles)
17		IN	ninfo
18			Number of elements in the <i>info</i> array (size_t)
19		IN	cbfunc
20			Callback function to be executed upon completion of operation pmix_op_cbfunc_t
21			(function reference)
22		IN	cbdata
23		114	Data to be passed to the cbfunc callback function (memory reference)
_3			Data to be passed to the colunc canback function (memory reference)
24		If c	bfunc is NULL , the function will be treated as a blocking call and the result of the operation
25		retu	rned in the status code.
06		īf o	before is non MIII I the function will be treated as a new blocking call and naturn one of the
26			bfunc is non- NULL , the function will be treated as a non-blocking call and return one of the
27		10110	owing:
28		PM	IX_SUCCESS The notification request is valid and is being processed. The callback function
29			will be called when the process-local operation is complete and will provide the resulting
30			status of that operation. Note that this does <i>not</i> reflect the success or failure of delivering the
31			event to any recipients. The callback function must not be executed prior to returning from the
32			API.
33		PM	MIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
34			returned success - the cbfunc will not be called
35		PΝ	IX_ERR_BAD_PARAM The request contains at least one incorrect entry that prevents it from
36			being processed. The callback function will <i>not</i> be called.
,0			being processed. The earloack function will not be earled.

1 **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation does not support event notification, 2 or in the case of a PMIx server calling the API, the range extended beyond the local node and the host SMS environment does not support event notification. The callback function will not 3 4 be called. Required Attributes 5 The following attributes are required to be supported by all PMIx libraries: 6 PMIX_EVENT_NON_DEFAULT "pmix.evnondef" (bool) Event is not to be delivered to default event handlers. PMIX_EVENT_CUSTOM_RANGE "pmix.evrange" (pmix_data_array_t*) 8 Array of pmix_proc_t defining range of event notification. 9 10 Host environments that implement support for PMIx event notification are required to provide the 11 following attributes for all events generated by the environment: 12 PMIX_EVENT_AFFECTED_PROC "pmix.evproc" (pmix_proc_t) 13 14 The single process that was affected. PMIX_EVENT_AFFECTED_PROCS "pmix.evaffected" (pmix_data_array_t*) 15 Array of **pmix_proc_t** defining affected processes. 16

Description

17

18

19

20 21

22

23

24 25

26 27

28

29

30

Report an event for notification via any registered event handler. This function can be called by any PMIx process, including application processes, PMIx servers, and SMS elements. The PMIx server calls this API to report events it detected itself so that the host SMS daemon distribute and handle them, and to pass events given to it by its host down to any attached client processes for processing. Examples might include notification of the failure of another process, detection of an impending node failure due to rising temperatures, or an intent to preempt the application. Events may be locally generated or come from anywhere in the system.

Host SMS daemons call the API to pass events down to its embedded PMIx server both for transmittal to local client processes and for the server's own internal processing.

Client application processes can call this function to notify the SMS and/or other application processes of an event it encountered. Note that processes are not constrained to report status values defined in the official PMIx standard — any integer value can be used. Thus, applications are free to define their own internal events and use the notification system for their own internal purposes.

The callback function will be called upon completion of the **notify_event** function's actions. At that time, any messages required for executing the operation (e.g., to send the notification to the local PMIx server) will have been queued, but may not yet have been transmitted. The caller is required to maintain the input data until the callback function has been executed — the sole purpose of the callback function is to indicate when the input data is no longer required.

CHAPTER 9

1

3

4

5 6

7

9

Data Packing and Unpacking

PMIx intentionally does not include support for internode communications in the standard, instead relying on its host SMS environment to transfer any needed data and/or requests between nodes. These operations frequently involve PMIx-defined public data structures that include binary data. Many HPC clusters are homogeneous, and so transferring the structures can be done rather simply. However, greater effort is required in heterogeneous environments to ensure binary data is correctly transferred. PMIx buffer manipulation functions are provided for this purpose via standardized interfaces to ease adoption.

9.1 Data Buffer Type

The pmix_data_buffer_t structure describes a data buffer used for packing and unpacking.

```
PMIx v2.0
10
            typedef struct pmix_data_buffer {
                /** Start of my memory */
11
                char *base_ptr;
12
                /** Where the next data will be packed to
13
14
                     (within the allocated memory starting
15
                    at base_ptr) */
                char *pack ptr;
16
                /** Where the next data will be unpacked
17
18
                    from (within the allocated memory
19
                    starting as base ptr) */
20
                char *unpack ptr;
                /** Number of bytes allocated (starting
21
22
                    at base_ptr) */
                size_t bytes_allocated;
23
24
                /** Number of bytes used by the buffer
25
                     (i.e., amount of data -- including
                    overhead -- packed in the buffer) */
26
27
                size_t bytes_used;
            } pmix_data_buffer_t;
28
```

9.2 **Support Macros**

PMIx provides a set of convenience macros for creating, initiating, and releasing data buffers.

9.2.1 PMIX DATA BUFFER CREATE 4 Summary Allocate memory for a pmix_data_buffer_t object and initialize it 5 6 Format *PMIx v2.0* 7 PMIX DATA BUFFER CREATE (buffer); OUT buffer 8 Variable to be assigned the pointer to the allocated **pmix_data_buffer_t** (handle) 9 10 **Description** This macro uses *calloc* to allocate memory for the buffer and initialize all fields in it 11 9.2.2 PMIX DATA BUFFER RELEASE Summary 13 14 Free a pmix_data_buffer_t object and the data it contains Format 15 PMIx v2.0PMIX DATA BUFFER_RELEASE(buffer); 16 17 IN buffer Pointer to the pmix_data_buffer_t to be released (handle) 18 **Description** 19 20 Free's the data contained in the buffer, and then free's the buffer itself 9.2.3 PMIX DATA BUFFER CONSTRUCT

Initialize a statically declared **pmix** data buffer t object

22 23 Summary

```
Format
1
   PMIx v2.0
2
             PMIX DATA BUFFER CONSTRUCT (buffer);
              IN
3
                 buffer
                  Pointer to the allocated pmix_data_buffer_t that is to be initialized (handle)
 4
              Description
 5
              Initialize a pre-allocated buffer object
 6
   9.2.4
            PMIX DATA BUFFER DESTRUCT
              Summarv
8
              Release the data contained in a pmix_data_buffer_t object
9
10
              Format
   PMIx v2.0
11
             PMIX DATA BUFFER DESTRUCT (buffer);
12
              IN
                  buffer
                  Pointer to the pmix_data_buffer_t whose data is to be released (handle)
13
              Description
14
              Free's the data contained in a pmix_data_buffer_t object
15
   9.2.5
            PMIX DATA BUFFER LOAD
17
              Summary
              Load a blob into a pmix_data_buffer_t object
18
              Format
19
   PMIx v2.0
20
              PMIX DATA BUFFER LOAD (buffer, data, size);
             IN
                buffer
21
                  Pointer to a pre-allocated pmix_data_buffer_t (handle)
22
             IN
23
                  data
24
                  Pointer to a blob (char*)
25
              IN
                  size
26
                  Number of bytes in the blob size_t
```

Description

1

7

9

11

12 13

14

15

16 17

18

19

20

22

25

- 2 Load the given data into the provided **pmix_data_buffer_t** object, usually done in
- 3 preparation for unpacking the provided data. Note that the data is *not* copied into the buffer thus,
- 4 the blob must not be released until after operations on the buffer have completed.

5 9.2.6 PMIX DATA BUFFER UNLOAD

6 Summary

Unload the data from a pmix_data_buffer_t object

Format

PMIx v2.0

PMIX_DATA_BUFFER_UNLOAD (buffer, data, size);

10 **IN** buffer

Pointer to the **pmix_data_buffer_t** whose data is to be extracted (handle)

OUT data

Variable to be assigned the pointer to the extracted blob (**void***)

OUT size

Variable to be assigned the number of bytes in the blob size_t

Description

Extract the data in a buffer, assigning the pointer to the data (and the number of bytes in the blob) to the provided variables, usually done to transmit the blob to a remote process for unpacking. The

buffer's internal pointer will be set to NULL to protect the data upon buffer destruct or release -

thus, the user is responsible for releasing the blob when done with it.

9.3 General Routines

The following routines are provided to support internode transfers in heterogeneous environments.

3 9.3.1 PMIx_Data_pack

24 Summary

Pack one or more values of a specified type into a buffer, usually for transmission to another process

ı	Format
<i>PMIx v2.0</i>	· · · · · · · · · · · · · · · · · · ·
2	pmix_status_t
3	<pre>PMIx_Data_pack(const pmix_proc_t *target,</pre>
4	<pre>pmix_data_buffer_t *buffer,</pre>
5	<pre>void *src, int32_t num_vals,</pre>
6	<pre>pmix_data_type_t type);</pre>
	C
7	IN target
8	Pointer to a pmix_proc_t containing the nspace/rank of the process that will be unpacking
9	the final buffer. A NULL value may be used to indicate that the target is based on the same
10	PMIx version as the caller. Note that only the target's nspace is relevant. (handle)
11	IN buffer
12	Pointer to a pmix_data_buffer_t where the packed data is to be stored (handle)
13	IN src
14	Pointer to a location where the data resides. Strings are to be passed as (char **) — i.e., the
15	caller must pass the address of the pointer to the string as the (void*). This allows the caller to
16	pass multiple strings in a single call. (memory reference)
17	IN num_vals
18	Number of elements pointed to by the <i>src</i> pointer. A string value is counted as a single value
19	regardless of length. The values must be contiguous in memory. Arrays of pointers (e.g.,
20	string arrays) should be contiguous, although the data pointed to need not be contiguous
21	across array entries.(int32_t)
22	IN type The type of the data to be realised (remine alone towns to)
23	The type of the data to be packed (pmix_data_type_t)
24	Returns one of the following:
25	PMIX_SUCCESS The data has been packed as requested
26	PMIX_ERR_NOT_SUPPORTED The PMIx implementation does not support this function.
27	PMIX_ERR_BAD_PARAM The provided buffer or src is NULL
28	PMIX_ERR_UNKNOWN_DATA_TYPE The specified data type is not known to this
29	implementation
30	PMIX_ERR_OUT_OF_RESOURCE Not enough memory to support the operation
31	PMIX_ERROR General error
20	Description
32	Description The real formation made are a great substantial formation that the real field buffer The buffer
33	The pack function packs one or more values of a specified type into the specified buffer. The buffer
34	must have already been initialized via the PMIX_DATA_BUFFER_CREATE or
35	PMIX_DATA_BUFFER_CONSTRUCT macros — otherwise, PMIx_Data_pack will return an
36	error. Providing an unsupported type flag will likewise be reported as an error.
37 38	Note that any data to be packed that is not hard type cast (i.e., not type cast to a specific size) may lose precision when unpacked by a non-homogeneous recipient. The PMIx_Data_pack function
	The first of the superior of t

will do its best to deal with heterogeneity issues between the packer and unpacker in such cases. Sending a number larger than can be handled by the recipient will return an error code (generated upon unpacking) — the error cannot be detected during packing.

The namespace of the intended recipient of the packed buffer (i.e., the process that will be unpacking it) is used solely to resolve any data type differences between PMIx versions. The recipient must, therefore, be known to the user prior to calling the pack function so that the PMIx library is aware of the version the recipient is using. Note that all processes in a given namespace are *required* to use the same PMIx version — thus, the caller must only know at least one process from the target's namespace.

9.3.2 PMIx_Data_unpack

Summary

Unpack values from a pmix_data_buffer_t

Format

PMIx v2.0

C

IN source

Pointer to a **pmix_proc_t** structure containing the nspace/rank of the process that packed the provided buffer. A NULL value may be used to indicate that the source is based on the same PMIx version as the caller. Note that only the source's nspace is relevant. (handle)

IN buffer

A pointer to the buffer from which the value will be extracted. (handle)

INOUT dest

A pointer to the memory location into which the data is to be stored. Note that these values will be stored contiguously in memory. For strings, this pointer must be to (char**) to provide a means of supporting multiple string operations. The unpack function will allocate memory for each string in the array - the caller must only provide adequate memory for the array of pointers. (void*)

INOUT max num values

The number of values to be unpacked — upon completion, the parameter will be set to the actual number of values unpacked. In most cases, this should match the maximum number provided in the parameters — but in no case will it exceed the value of this parameter. Note that unpacking fewer values than are actually available will leave the buffer in an unpackable state — the function will return an error code to warn of this condition.(int32_t)

IN type The type of the data to be unpacked — must be one of the PMIx defined data types (pmix_data_type_t) Returns one of the following: PMIX_SUCCESS The data has been unpacked as requested PMIX ERR NOT SUPPORTED The PMIx implementation does not support this function. PMIX ERR BAD PARAM The provided buffer or dest is NULL PMIX_ERR_UNKNOWN_DATA_TYPE The specified data type is not known to this implementation PMIX ERR OUT OF RESOURCE Not enough memory to support the operation PMIX ERROR General error

Description

The unpack function unpacks the next value (or values) of a specified type from the given buffer. The buffer must have already been initialized via an PMIX_DATA_BUFFER_CREATE or PMIX_DATA_BUFFER_CONSTRUCT call (and assumedly filled with some data) — otherwise, the unpack_value function will return an error. Providing an unsupported type flag will likewise be reported as an error, as will specifying a data type that *does not* match the type of the next item in the buffer. An attempt to read beyond the end of the stored data held in the buffer will also return an error.

NOTE: it is possible for the buffer to be corrupted and that PMIx will *think* there is a proper variable type at the beginning of an unpack region — but that the value is bogus (e.g., just a byte field in a string array that so happens to have a value that matches the specified data type flag). Therefore, the data type error check is *not* completely safe.

Unpacking values is a "nondestructive" process — i.e., the values are not removed from the buffer. It is therefore possible for the caller to re-unpack a value from the same buffer by resetting the unpack_ptr.

Warning: The caller is responsible for providing adequate memory storage for the requested data. The user must provide a parameter indicating the maximum number of values that can be unpacked into the allocated memory. If more values exist in the buffer than can fit into the memory storage, then the function will unpack what it can fit into that location and return an error code indicating that the buffer was only partially unpacked.

Note that any data that was not hard type cast (i.e., not type cast to a specific size) when packed may lose precision when unpacked by a non-homogeneous recipient. PMIx will do its best to deal with heterogeneity issues between the packer and unpacker in such cases. Sending a number larger than can be handled by the recipient will return an error code generated upon unpacking — these errors cannot be detected during packing.

The namespace of the process that packed the buffer is used solely to resolve any data type differences between PMIx versions. The packer must, therefore, be known to the user prior to calling the pack function so that the PMIx library is aware of the version the packer is using. Note

that all processes in a given namespace are *required* to use the same PMIx version — thus, the caller must only know at least one process from the packer's namespace.

9.3.3 PMIx_Data_copy

Summary

Copy a data value from one location to another.

Format

1

4 5

6

10

11 12

13

14

15 16

17

18

19 20

21

22 23

24

25 26

27

28

29

31

32

```
PMIx v2.0

pmix_status_t

PMIx_Data_copy(void **dest, void *src,

pmix_data_type_t type);
```

IN dest

The address of a pointer into which the address of the resulting data is to be stored. (void**)

IN src

A pointer to the memory location from which the data is to be copied (handle)

IN type

The type of the data to be copied — must be one of the PMIx defined data types. (

pmix_data_type_t)

Returns one of the following:

PMIX_SUCCESS The data has been copied as requested

PMIX_ERR_NOT_SUPPORTED The PMIx implementation does not support this function.

PMIX ERR BAD PARAM The provided src or dest is NULL

PMIX_ERR_UNKNOWN_DATA_TYPE The specified data type is not known to this implementation

PMIX_ERR_OUT_OF_RESOURCE Not enough memory to support the operation PMIX_ERROR General error

Description

Since registered data types can be complex structures, the system needs some way to know how to copy the data from one location to another (e.g., for storage in the registry). This function, which can call other copy functions to build up complex data types, defines the method for making a copy of the specified data type.

9.3.4 PMIx Data print

Summary

Pretty-print a data value.

```
Format
 1
   PMIx v2.0
 2
               pmix status t
 3
               PMIx_Data_print(char **output, char *prefix,
 4
                                    void *src, pmix data type t type);
               IN
 5
                    output
 6
                    The address of a pointer into which the address of the resulting output is to be stored.
 7
                    (char**)
 8
               IN
                   prefix
 9
                    String to be prepended to the resulting output (char*)
10
               IN
                    A pointer to the memory location of the data value to be printed (handle)
11
               IN
                    type
12
13
                    The type of the data value to be printed — must be one of the PMIx defined data types. (
                    pmix_data_type_t)
14
               Returns one of the following:
15
                PMIX_SUCCESS The data has been printed as requested
16
17
                PMIX ERR BAD PARAM The provided data type is not recognized.
18
                PMIX_ERR_NOT_SUPPORTED The PMIx implementation does not support this function.
               Description
19
20
               Since registered data types can be complex structures, the system needs some way to know how to
21
               print them (i.e., convert them to a string representation). Primarily for debug purposes.
    9.3.5
              PMIx Data copy payload
23
               Summary
               Copy a payload from one buffer to another
24
               Format
25
   PMIx v2.0
```

Description

 This function will append a copy of the payload in one buffer into another buffer. Note that this is *not* a destructive procedure — the source buffer's payload will remain intact, as will any pre-existing payload in the destination's buffer. Only the unpacked portion of the source payload will be copied.

CHAPTER 10

Security

PMIx utilizes a multi-layered approach toward security that differs for client versus tool processes. *Client* processes (i.e., processes started by the host environment) must be preregistered with the PMIx server library via the PMIx_server_register_client API before they are spawned. This API requires that you pass the expected uid/gid of the client process.

When the client attempts to connect to the PMIx server, the server uses available standard Operating System (OS) methods to determine the effective uid/gid of the process requesting the connection. PMIx implementations shall not rely on any values reported by the client process itself as that would be unsafe. The effective uid/gid reported by the OS is compared to the values provided by the host during registration - if they don't match, the PMIx server is required to drop the connection request. This ensures that the PMIx server does not allow connection from a client that doesn't at least meet some minimal security requirement.

Once the requesting client passes the initial test, the PMIx server can, at the choice of the implementor, perform additional security checks. This may involve a variety of methods such as exchange of a system-provided key or credential. At the conclusion of that process, the PMIx server reports the client connection request to the host via the

pmix_server_client_connected_fn_t interface. The host may then perform any
additional checks and operations before responding with either PMIX_SUCCESS to indicate that
the connection is approved, or a PMIx error constant indicating that the connection request is
refused. In this latter case, the PMIx server is required to drop the connection.

Tools started by the host environment are classed as a subgroup of client processes and follow the client process procedure. However, tools that are not started by the host environment must be handled differently as registration information is not available prior to the connection request. In these cases, the PMIx server library is required to use available standard OS methods to get the effective uid/gid and report them upwards as part of invoking the

pmix_server_tool_connection_fn_t interface, deferring initial security screening to the host. It is recognized that this may represent a security risk - for this reason, PMIx server libraries must not enable tool connections by default. Instead, the host has to explicitly enable them via the **PMIX_SERVER_TOOL_SUPPORT** attribute, thus recognizing the associated risk. Once the host has completed its authentication procedure, it again informs the PMIx server of the result.

Applications and tools often interact with the host environment in ways that require security beyond just verifying the user's identity - e.g., access to that user's relevant authorizations. This is particularly important when tools connect directly to a system-level PMIx server that may be operating at a privileged level. A variety of system management software packages provide authorization services, but the lack of standardized interfaces makes portability problematic.

This section defines two PMIx client-side APIs for this purpose. These are most likely to be used by user-space applications/tools, but are not restricted to that realm.

10.1 Obtaining Credentials

The API for obtaining a credential is a non-blocking operation since the host environment may have to contact a remote credential service. The definition takes into account the potential that the returned credential could be sent via some mechanism to another application that resides in an environment using a different security mechanism. Thus, provision is made for the system to return additional information (e.g., the identity of the issuing agent) outside of the credential itself and visible to the application.

10.1.1 PMIx_Get_credential

Summary

Request a credential from the PMIx server library or the host environment

Format

PMIx v3.0

1

4

5

6

7

8

9

11

12

13

14

15

16

17

18 19

20

21 22

23

24 25

26

C

IN info

Array of pmix_info_t structures (array of handles)

IN ninfo

Number of elements in the *info* array (size_t)

IN credential

Address of a pmix_byte_object_t within which to return credential (handle)

Returns one of the following:

- PMIX_SUCCESS, indicating that the credential has been returned in the provided pmix_byte_object_t
- a PMIx error constant indicating either an error in the input or that the request is unsupported

		▼
1 2		PMIx libraries that choose not to support this operation <i>must</i> return PMIX_ERR_NOT_SUPPORTED when the function is called.
3 4		There are no required attributes for this API. Note that implementations may choose to internally execute integration for some security environments (e.g., directly contacting a <i>munge</i> server).
5 6 7 8		Implementations that support the operation but cannot directly process the client's request must pass any attributes that are provided by the client to the host environment for processing. In addition, the following attributes are required to be included in the <i>info</i> array passed from the PMIx library to the host environment:
9 10		<pre>PMIX_USERID "pmix.euid" (uint32_t)</pre>
11 12		PMIX_GRPID "pmix.egid" (uint32_t) Effective group id.
		▼ Optional Attributes
13		The following attributes are optional for host environments that support this operation:
14 15 16 17		PMIX_TIMEOUT "pmix.timeout" (int) Time in seconds before the specified operation should time out (0 indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.
		Advice to PMIx library implementers —
18 19 20 21 22 23		We recommend that implementation of the PMIX_TIMEOUT attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support PMIX_TIMEOUT directly in the PMIx server library must take care to resolve the race condition and should avoid passing PMIX_TIMEOUT to the host environment so that multiple competing timeouts are not created.
24 25		Description Request a credential from the PMIx server library or the host environment
26	10.1.2	PMIx_Get_credential_nb
27 28		Summary Request a credential from the PMIx server library or the host environment

1	Format
<i>PMIx v3.0</i>	· C
2	pmix_status_t
3	<pre>PMIx_Get_credential_nb(const pmix_info_t info[], size_t ninfo,</pre>
4	<pre>pmix_credential_cbfunc_t cbfunc, void *cbdata</pre>
	C
5	IN info
6	Array of pmix_info_t structures (array of handles)
7	IN ninfo
8	Number of elements in the <i>info</i> array (size_t)
9	IN cbfunc
10 11	Callback function to return credential (pmix_credential_cbfunc_t function reference)
12	IN cbdata
13	Data to be passed to the callback function (memory reference)
14	Returns one of the following:
15 16	• PMIX_SUCCESS , indicating that the request has been communicated to the local PMIx server result will be returned in the provided <i>cbfunc</i>
17 18	• a PMIx error constant indicating either an error in the input or that the request is unsupported - the <i>cbfunc</i> will <i>not</i> be called
	▼
19 20	PMIx libraries that choose not to support this operation <i>must</i> return PMIX_ERR_NOT_SUPPORTED when the function is called.
21 22	There are no required attributes for this API. Note that implementations may choose to internally execute integration for some security environments (e.g., directly contacting a <i>munge</i> server).
23	Implementations that support the operation but cannot directly process the client's request must
24	pass any attributes that are provided by the client to the host environment for processing. In
25	addition, the following attributes are required to be included in the <i>info</i> array passed from the PMIx
26	library to the host environment:
27	<pre>PMIX_USERID "pmix.euid" (uint32_t)</pre>
28	Effective user id.
29	<pre>PMIX_GRPID "pmix.egid" (uint32_t)</pre>
30	Effective group id.
	AA

Optional Attributes The following attributes are optional for host environments that support this operation: 1 2 PMIX_TIMEOUT "pmix.timeout" (int) 3 Time in seconds before the specified operation should time out (θ indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent 4 5 the target process from ever exposing its data. Advice to PMIx library implementers We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host 6 7 environment due to race condition considerations between completion of the operation versus 8 internal timeout in the PMIx server library. Implementers that choose to support PMIX_TIMEOUT 9 directly in the PMIx server library must take care to resolve the race condition and should avoid 10 passing PMIX TIMEOUT to the host environment so that multiple competing timeouts are not created. 11 **Description** 12 13 Request a credential from the PMIx server library or the host environment 10.2 Validating Credentials 15 The API for validating a credential is a non-blocking operation since the host environment may 16 have to contact a remote credential service. Provision is made for the system to return additional 17 information regarding possible authorization limitations beyond simple authentication.

18 10.2.1 PMIx_Validate_credential

19 **Summary**

20

Request validation of a credential by the PMIx server library or the host environment

1		Format
Pl	MIx v3.0	<u> </u>
2		pmix_status_t
3		<pre>PMIx_Validate_credential(const pmix_byte_object_t *cred,</pre>
4		const pmix_info_t info[], size_t ninfo,
5		<pre>pmix_info_t **results, size_t *nresults)</pre>
6		IN cred
7		Pointer to pmix_byte_object_t containing the credential (handle)
8		IN info
9		Array of pmix_info_t structures (array of handles) IN ninfo
10 11		Number of elements in the <i>info</i> array (size_t)
12		INOUT results
13		Address where a pointer to an array of pmix_info_t containing the results of the request
14		can be returned (memory reference)
15		INOUT nresults
16		Address where the number of elements in <i>results</i> can be returned (handle)
17		Returns one of the following:
18 19		• PMIX_SUCCESS, indicating that the request was processed and returned <i>success</i> . Details of the result will be returned in the <i>results</i> array
20		• a PMIx error constant indicating either an error in the input or that the request was refused
		▼ Required Attributes
21		PMIx libraries that choose not to support this operation <i>must</i> return
22		PMIX_ERR_NOT_SUPPORTED when the function is called.
23 24		There are no required attributes for this API. Note that implementations may choose to internally execute integration for some security environments (e.g., directly contacting a <i>munge</i> server).
25		Implementations that support the operation but cannot directly process the client's request must
26		pass any attributes that are provided by the client to the host environment for processing. In
27		addition, the following attributes are required to be included in the <i>info</i> array passed from the PMIx
28		library to the host environment:
29		<pre>PMIX_USERID "pmix.euid" (uint32_t)</pre>
30		Effective user id.
31		<pre>PMIX_GRPID "pmix.egid" (uint32_t)</pre>
32		Effective group id.
		A

Optional Attributes The following attributes are optional for host environments that support this operation: 1 2 PMIX_TIMEOUT "pmix.timeout" (int) 3 Time in seconds before the specified operation should time out (θ indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent 4 the target process from ever exposing its data. 5 Advice to PMIx library implementers We recommend that implementation of the PMIX_TIMEOUT attribute be left to the host 6 7 environment due to race condition considerations between completion of the operation versus 8 internal timeout in the PMIx server library. Implementers that choose to support PMIX_TIMEOUT 9 directly in the PMIx server library must take care to resolve the race condition and should avoid 10 passing PMIX TIMEOUT to the host environment so that multiple competing timeouts are not 11 created. **Description** 12 13 Request validation of a credential by the PMIx server library or the host environment. 10.2.2 PMIx_Validate_credential_nb Summary 15

Request validation of a credential by the PMIx server library or the host environment

1	Format
<i>PMIx v3.0</i>	
2	pmix_status_t
3	PMIx_Validate_credential_nb(const pmix_byte_object_t *cred,
4	<pre>const pmix_info_t info[], size_t ninfo,</pre>
5 6	<pre>pmix_validation_cbfunc_t cbfunc, void *cbdata)</pre>
O	Void *ebdata)
	O
7	IN cred
8	Pointer to pmix_byte_object_t containing the credential (handle)
9	IN info
10 11	Array of pmix_info_t structures (array of handles) IN ninfo
12	Number of elements in the <i>info</i> array (size_t)
13	IN cbfunc
14	Callback function to return result (pmix_validation_cbfunc_t function reference)
15	IN cbdata
16	Data to be passed to the callback function (memory reference)
17	Returns one of the following:
18 19	• PMIX_SUCCESS, indicating that the request has been communicated to the local PMIx server - result will be returned in the provided <i>cbfunc</i>
20 21	• a PMIx error constant indicating either an error in the input or that the request is unsupported - the <i>cbfunc</i> will <i>not</i> be called
	▼ Required Attributes
22 23	PMIx libraries that choose not to support this operation <i>must</i> return PMIX_ERR_NOT_SUPPORTED when the function is called.
24 25	There are no required attributes for this API. Note that implementations may choose to internally execute integration for some security environments (e.g., directly contacting a <i>munge</i> server).
26 27 28 29	Implementations that support the operation but cannot directly process the client's request must pass any attributes that are provided by the client to the host environment for processing. In addition, the following attributes are required to be included in the <i>info</i> array passed from the PMIx library to the host environment:
30 31	PMIX_USERID "pmix.euid" (uint32_t) Effective user id.
32 33	PMIX_GRPID "pmix.egid" (uint32_t) Effective group id.

The following attributes are optional for host environments that support this operation: PMIX_TIMEOUT "pmix.timeout" (int) Time in seconds before the specified operation should time out (0 indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data. Advice to PMIx library implementers We recommend that implementation of the PMIX_TIMEOUT attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support PMIX_TIMEOUT directly in the PMIx server library must take care to resolve the race condition and should avoid passing PMIX_TIMEOUT to the host environment so that multiple competing timeouts are not created.

Description

Request validation of a credential by the PMIx server library or the host environment.

CHAPTER 11

1 2

3

4 5

6 7

8

9

10

12 13

16

24

Server-Specific Interfaces

The RM daemon that hosts the PMIx server library interacts with that library in two distinct manners. First, PMIx provides a set of APIs by which the host can request specific services from its library. This includes generating regular expressions, registering information to be passed to client processes, and requesting information on behalf of a remote process. Note that the host always has access to all PMIx client APIs - the functions listed below are in addition to those available to a PMIx client.

Second, the host can provide a set of callback functions by which the PMIx server library can pass requests upward for servicing by the host. These include notifications of client connection and finalize, as well as requests by clients for information and/or services that the PMIx server library does not itself provide.

11.1 Server Support Functions

The following APIs allow the RM daemon that hosts the PMIx server library to request specific services from the PMIx library.

14 11.1.1 PMIx_generate_regex

15 **Summary**

Generate a compressed representation of the input string.

- 21 String to process (string)
 22 **OUT** output
- Compressed representation of *input* (array of bytes)
 - Returns **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant.

Description

Given a comma-separated list of *input* values, generate a reduced size representation of the input that can be passed down to the PMIx server library's **PMIx_server_register_nspace** API for parsing. The order of the individual values in the *input* string is preserved across the operation. The caller is responsible for releasing the returned data.

The precise compressed representations will be implementation specific. However, all PMIx implementations are required to include a **NULL**-terminated string in the output representation that can be printed for diagnostic purposes.

Advice to PMIx server hosts —

The returned representation may be an arbitrary array of bytes as opposed to a valid NULL-terminated string. However, the method used to generate the representation shall be identified with a colon-delimited string at the beginning of the output. For example, an output starting with "pmix:\0" might indicate that the representation is a PMIx-defined regular expression represented as a NULL-terminated string following the "pmix:\0" prefix. In contrast, an output starting with "blob:\0" might indicate a compressed binary array follows the prefix.

Communicating the resulting output should be done by first packing the returned expression using the <code>PMIx_Data_pack</code>, declaring the input to be of type <code>PMIX_REGEX</code>, and then obtaining the resulting blob to be communicated using the <code>PMIX_DATA_BUFFER_UNLOAD</code> macro. The reciprocal method can be used on the remote end prior to passing the regex into <code>PMIx_server_register_nspace</code>. The pack/unpack routines will ensure proper handling of the data based on the regex prefix.

11.1.2 PMIx generate ppn

Summary

Generate a compressed representation of the input identifying the processes on each node.

Format

PMIx v1.0

pmix_status_t PMIx_generate_ppn(const char *input, char **ppn)

IN input

String to process (string)

OUT ppn

Compressed representation of *input* (array of bytes)

Returns **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant.

Description

The input shall consist of a semicolon-separated list of ranges representing the ranks of processes on each node of the job - e.g., "1-4;2-5;8,10,11,12;6,7,9". Each field of the input must correspond to the node name provided at that position in the input to **PMIx_generate_regex**. Thus, in the example, ranks 1-4 would be located on the first node of the comma-separated list of names provided to **PMIx_generate_regex**, and ranks 2-5 would be on the second name in the list.

Advice to PMIx server hosts -

The returned representation may be an arbitrary array of bytes as opposed to a valid NULL-terminated string. However, the method used to generate the representation shall be identified with a colon-delimited string at the beginning of the output. For example, an output starting with "pmix:" indicates that the representation is a PMIx-defined regular expression represented as a NULL-terminated string. In contrast, an output starting with "blob:\0size=1234:" is a compressed binary array.

Communicating the resulting output should be done by first packing the returned expression using the <code>PMIx_Data_pack</code>, declaring the input to be of type <code>PMIX_REGEX</code>, and then obtaining the blob to be communicated using the <code>PMIX_DATA_BUFFER_UNLOAD</code> macro. The pack/unpack routines will ensure proper handling of the data based on the regex prefix.

11.1.3 PMIx_server_register_nspace

Summary

Setup the data about a particular namespace.

Format

PMIx v1.0

pmix_status_t

C

IN nspace

Character array of maximum size **PMIX_MAX_NSLEN** containing the namespace identifier (string)

IN nlocalprocs

number of local processes (integer)

IN info

Array of info structures (array of handles)

1 2 3 4 5	 IN ninfo Number of elements in the info array (integer) IN cbfunc Callback function pmix_op_cbfunc_t (function reference) IN cbdata Data to be passed to the callback function (memory reference)
7	Returns one of the following:
8 9 10	• PMIX_SUCCESS, indicating that the request is being processed by the host environment - result will be returned in the provided <i>cbfunc</i> . Note that the library must not invoke the callback function prior to returning from the API.
11 12	• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will not be called
13 14	• a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will not be called
	▼
15	The following attributes are required to be supported by all PMIx libraries:
16 17 18	<pre>PMIX_REGISTER_NODATA "pmix.reg.nodata" (bool) Registration is for this namespace only, do not copy job data - this attribute is not accessed using the PMIx_Get</pre>
19	
20 21 22 23	Host environments are required to provide a wide range of session-, job-, application-, node-, and process-level information, and may choose to provide a similarly wide range of optional information. The information is broadly separated into categories based on the <i>level</i> definitions explained in 3.4.11.
24 25 26 27 28	Session-level information may be passed as individual <code>pmix_info_t</code> entries, or as part of a <code>pmix_data_array_t</code> using the <code>PMIX_SESSION_INFO_ARRAY</code> attribute. Session-level information is always accessed using the namespace and the <code>PMIX_RANK_WILDCARD</code> rank, accompanied by the <code>PMIX_SESSION_INFO</code> attribute where ambiguity may exist. The list of data referenced in this way shall include:
29 30 31 32 33	 PMIX_UNIV_SIZE "pmix.univ.size" (uint32_t) Number of allocated slots in a session - each slot may or may not be occupied by an executing process. Note that this attribute is equivalent to providing the PMIX_MAX_PROCS entry in the PMIX_SESSION_INFO_ARRAY array - it is included in the Standard for historical reasons.
34	• PMIX_MAX_PROCS "pmix.max.size" (uint32_t)

1 2 3 4 5	Maximum number of processes that can be executed in this context (session, namespace, application, or node). Typically, this is a constraint imposed by a scheduler or by user settings in a hostfile or other resource description. Must be provided if <pre>PMIX_UNIV_SIZE</pre> is not given. Requires use of the <pre>PMIX_SESSION_INFO</pre> attribute to avoid ambiguity when retrieving it.
6 7 8	• PMIX_SESSION_ID "pmix.session.id" (uint32_t) Session identifier assigned by the scheduler - referenced using PMIX_RANK_WILDCARD .
9	plus the following optional information:
10 11	• PMIX_CLUSTER_ID "pmix.clid" (char*) A string name for the cluster this allocation is on
12 13 14 15	 PMIX_ALLOCATED_NODELIST "pmix.alist" (char*) Comma-delimited list or regular expression of all nodes in this allocation regardless of whether or not they currently host processes - referenced using PMIX_RANK_WILDCARD .
16 17 18	Job-level information may be passed as individual pmix_info_t entries, or as part of a pmix_data_array_t using the PMIX_JOB_INFO_ARRAY attribute. The list of data referenced in this way shall include:
19 20	• PMIX_SERVER_NSPACE "pmix.srv.nspace" (char*) Name of the namespace to use for this PMIx server.
21 22	• PMIX_SERVER_RANK "pmix.srv.rank" (pmix_rank_t) Rank of this PMIx server
23 24 25	 PMIX_NSPACE "pmix.nspace" (char*) Namespace of the job - may be a numerical value expressed as a string, but is often a non-numerical string carrying information solely of use to the system.
26 27 28	• PMIX_JOBID "pmix.jobid" (char*) Job identifier assigned by the scheduler - may be identical to the namespace, but is often a numerical value expressed as a string (e.g., "12345.3").
29 30 31 32 33 34	• PMIX_JOB_SIZE "pmix.job.size" (uint32_t) Total number of processes in this job across all contained applications. Note that this value can be different from PMIX_MAX_PROCS. For example, users may choose to subdivide an allocation (running several jobs in parallel within it), and dynamic programming models may support adding and removing processes from a running job on-the-fly. In the latter case, PMIx events must be used to notify processes within the job that the job size has changed.
36 37 38	 PMIX_MAX_PROCS "pmix.max.size" (uint32_t) Maximum number of processes that can be executed in this context (session, namespace, application, or node). Typically, this is a constraint imposed by a scheduler or by user

1 2	settings in a hostfile or other resource description. Retrieval of this attribute defaults to the job level unless an appropriate specification is given (e.g., PMIX_SESSION_INFO).
3 4	• PMIX_NODE_MAP "pmix.nmap" (char*) Regular expression of nodes - see 11.1.3.1 for an explanation of its generation.
5 6 7	 PMIX_PROC_MAP "pmix.pmap" (char*) Regular expression describing processes on each node - see 11.1.3.1 for an explanation of its generation.
8	plus the following optional information:
9 10	• PMIX_NPROC_OFFSET "pmix.offset" (pmix_rank_t) Starting global rank of this job - referenced using PMIX_RANK_WILDCARD.
11 12 13	 PMIX_JOB_NUM_APPS "pmix.job.napps" (uint32_t) Number of applications in this job. This is a required attribute if more than one application is included in the job
14 15 16 17	 PMIX_MAPBY "pmix.mapby" (char*) Process mapping policy - when accessed using PMIx_Get , use the PMIX_RANK_WILDCARD value for the rank to discover the mapping policy used for the provided namespace
18 19 20 21	 PMIX_RANKBY "pmix.rankby" (char*) Process ranking policy - when accessed using PMIx_Get , use the PMIX_RANK_WILDCARD value for the rank to discover the ranking algorithm used for the provided namespace
22 23 24 25	 PMIX_BINDTO "pmix.bindto" (char*) Process binding policy - when accessed using PMIx_Get , use the PMIX_RANK_WILDCARD value for the rank to discover the binding policy used for the provided namespace
26 27	• PMIX_HOSTNAME_KEEP_FQDN "pmix.fqdn" (bool) FQDN hostnames are being retained
28 29	• PMIX_ANL_MAP "pmix.anlmap" (char*) Process mapping in Argonne National Laboratory's PMI-1/PMI-2 notation.
30 31 32 33 34 35	Application-level information is accessed by providing both the namespace of the job (with the rank set to PMIX_RANK_WILDCARD). If application-level information is requested for an application other than the one the caller belongs to, then the PMIX_APPNUM attribute must be provided. If more than one application is included in the namespace, then the the host environment is also required to supply data consisting of the following items for each application in the job, passed as a PMIX_data_array_t using the PMIX_APP_INFO_ARRAY attribute:
36 37	• PMIX_APPNUM "pmix.appnum" (uint32_t) Application number within the job.

• PMIX_APP_SIZE "pmix.app.size" (uint32_t) 1 Number of processes in this application. 2 3 • PMIX_MAX_PROCS "pmix.max.size" (uint32_t) Maximum number of processes that can be executed in this context (session, namespace, 4 5 application, or node). Typically, this is a constraint imposed by a scheduler or by user 6 settings in a hostfile or other resource description. Requires use of the 7 **PMIX_APP_INFO** attribute to avoid ambiguity when retrieving it. 8 • PMIX APPLDR "pmix.aldr" (pmix rank t) Lowest rank in this application within this job - referenced using 9 PMIX RANK WILDCARD. 10 11 • PMIX WDIR "pmix.wdir" (char*) Working directory for spawned processes. This attribute is required for all registrations, 12 but may be provided as an individual **pmix_info_t** entry if only one application is 13 included in the namespace. 14 15 • PMIX APP ARGV "pmix.app.argv" (char*) Consolidated argy passed to the spawn command for the given process (e.g., "./myapp 16 17 arg1 arg2 arg3") This attribute is required for all registrations, but may be provided as an individual **pmix** info t entry if only one application is included in the namespace. 18 19 plus the following optional information: 20 • PMIX PSET NAME "pmix.pset.nm" (char*) 21 User-assigned name for the process set containing the given process. 22 The data may also include attributes provided by the host environment that identify the 23 programming model (as specified by the user) being executed within the namespace. The PMIx 24 server library may utilize this information to customize the environment to fit that model (e.g., 25 adding environmental variables specified by the corresponding standard for that model): 26 • PMIX_PROGRAMMING_MODEL "pmix.pgm.model" (char*) 27 Programming model being initialized (e.g., "MPI" or "OpenMP") • PMIX MODEL LIBRARY NAME "pmix.mdl.name" (char*) 28 Programming model implementation ID (e.g., "OpenMPI" or "MPICH") 29 30 • PMIX MODEL LIBRARY_VERSION "pmix.mld.vrs" (char*) 31 Programming model version string (e.g., "2.1.1") Node-level information is accessed by providing the namespace of the job with the rank set to 32 PMIX RANK WILDCARD. If node-level information is requested for a node other than the one the 33 34 caller is executing on, then the PMIX NODEID or the PMIX HOSTNAME attribute of the target 35 node must be provided. Registration shall include the following data for each node in the job, 36 passed as a pmix_data_array_t using the PMIX_NODE_INFO_ARRAY attribute:

37

• PMIX_NODEID "pmix.nodeid" (uint32_t)

1 2 3 4	Node identifier expressed as the node's index (beginning at zero) in an array of nodes within the active session. The value must be unique and directly correlate to the PMIX_HOSTNAME of the node - i.e., users can interchangeably reference the same location using either the PMIX_HOSTNAME or corresponding PMIX_NODEID .
5 6	• PMIX_HOSTNAME "pmix.hname" (char*) Name of the host (e.g., where a specified process is running, or a given device is located).
7 8	• PMIX_HOSTNAME_ALIASES "pmix.alias" (char*) Comma-delimited list of names by which this node is known.
9 10	• PMIX_LOCAL_SIZE "pmix.local.size" (uint32_t) Number of processes in this job or application on this node.
1 2	• PMIX_NODE_SIZE "pmix.node.size" (uint32_t) Number of processes across all jobs on this node.
13 14	• PMIX_LOCALLDR "pmix.lldr" (pmix_rank_t) Lowest rank on this node within this job - referenced using PMIX_RANK_WILDCARD.
15 16 17	• PMIX_LOCAL_PEERS "pmix.lpeers" (char*) Comma-delimited list of ranks on this node within the specified namespace - referenced using PMIX_RANK_WILDCARD.
18	plus the following information for the server's own node:
19 20 21	• PMIX_LOCAL_CPUSETS "pmix.lcpus" (char*) Colon-delimited cpusets of local peers within the specified namespace - referenced using PMIX_RANK_WILDCARD.
22 23	• PMIX_TMPDIR "pmix.tmpdir" (char*) Full path to the top-level temporary directory assigned to the session.
24 25	• PMIX_NSDIR "pmix.nsdir" (char*) Full path to the temporary directory assigned to the namespace, under PMIX_TMPDIR.
26 27 28	• PMIX_LOCAL_PROCS "pmix.lprocs" (pmix_proc_t array) Array of pmix_proc_t of all processes on the specified node - referenced using PMIX_RANK_WILDCARD.
29 30	The data may also include one or more of the following methods for passing the HWLOC topology information for the server's own node:
31 32	• PMIX_HWLOC_SHMEM_FILE "pmix.hwlocfile" (char*) Path to the HWLOC shared memory file.
33 34	• PMIX_HWLOC_SHMEM_ADDR "pmix.hwlocaddr" (size_t) Address of the HWLOC shared memory segment.
35 36	• PMIX_HWLOC_SHMEM_SIZE "pmix.hwlocsize" (size_t) Size of the HWLOC shared memory segment.

1 2	• PMIX_HWLOC_XML_V1 "pmix.hwlocxml1" (char*) XML representation of local topology using HWLOC's v1.x format.
3 4	• PMIX_HWLOC_XML_V2 "pmix.hwlocxml2" (char*) XML representation of local topology using HWLOC's v2.x format.
5	plus the following optional information for the server's own node:
6 7	• PMIX_AVAIL_PHYS_MEMORY "pmix.pmem" (uint64_t) Total available physical memory on this node.
8	and the following optional information for arbitrary nodes:
9 10 11 12 13	 PMIX_MAX_PROCS "pmix.max.size" (uint32_t) Maximum number of processes that can be executed in this context (session, namespace, application, or node). Typically, this is a constraint imposed by a scheduler or by user settings in a hostfile or other resource description. Requires use of the PMIX_NODE_INFO attribute to avoid ambiguity when retrieving it.
14 15 16	Process-level information is accessed by providing the namespace of the job and the rank of the target process. Registration shall include the following data for each process in the job, passed as a <pre>pmix_data_array_t</pre> using the <pre>PMIX_PROC_INFO_ARRAY</pre> attribute:
17 18	• PMIX_RANK "pmix.rank" (pmix_rank_t) Process rank within the job.
19 20 21	• PMIX_APPNUM "pmix.appnum" (uint32_t) Application number within the job. This attribute may be omitted if only one application is present in the namespace.
22 23 24	 PMIX_APP_RANK "pmix.apprank" (pmix_rank_t) Process rank within this application. This attribute may be omitted if only one application is present in the namespace.
25 26	• PMIX_GLOBAL_RANK "pmix.grank" (pmix_rank_t) Process rank spanning across all jobs in this session.
27 28	• PMIX_LOCAL_RANK "pmix.lrank" (uint16_t) Local rank on this node within this job.
29 30	• PMIX_NODE_RANK "pmix.nrank" (uint16_t) Process rank on this node spanning all jobs.
31 32 33 34 35	• PMIX_NODEID "pmix.nodeid" (uint32_t) Node identifier expressed as the node's index (beginning at zero) in an array of nodes within the active session. The value must be unique and directly correlate to the PMIX_HOSTNAME of the node - i.e., users can interchangeably reference the same location using either the PMIX_HOSTNAME or corresponding PMIX_NODEID.
36	• PMIX_REINCARNATION "pmix.reinc" (uint32_t)

1 2	Number of times this process has been re-instantiated - i.e, a value of zero indicates that the process has never failed and been restarted.
3 4 5 6	 PMIX_SPAWNED "pmix.spawned" (bool) true if this process resulted from a call to PMIx_Spawn. Lack of inclusion (i.e., a return status of PMIX_ERR_NOT_FOUND) corresponds to a value of false for this attribute.
7	plus the following information for processes that are local to the server:
8 9 0 1 2 3 4 5 6 7 8 9	• PMIX_LOCALITY_STRING "pmix.locstr" (char*) String describing a process's bound location - referenced using the process's rank. The string is of the form: NM%s:SK%s:L3%s:L2%s:L1%s:CR%s:HT%s Where the %s is replaced with an integer index or inclusive range for hwloc. NM identifies the numa node(s). SK identifies the socket(s). L3 identifies the L3 cache(s). L2 identifies the L2 cache(s). L1 identifies the L1 cache(s). CR identifies the cores(s). HT identifies the hardware thread(s). If your architecture does not have the specified hardware designation then it can be omitted from the signature. For example: NM0:SK0:L30-4:L20-4:L10-4:CR0-4:HT0-39. This means numa node 0, socket 0, L3 caches 0, 1, 2, 3, 4, L2 caches 0-4, L1 caches 0-4, cores 0, 1, 2, 3, 4, and hardware threads 0-39.
20 21	 PMIX_PROCDIR "pmix.pdir" (char*) Full path to the subdirectory under PMIX_NSDIR assigned to the process.
22 23	and the following optional information - note that this information can be derived from information already provided by other attributes, but it may be included here for ease of retrieval by users:
24 25 26	• PMIX_HOSTNAME "pmix.hname" (char*) Name of the host (e.g., where a specified process is running, or a given device is located).
27 28 29	Attributes not directly provided by the host environment may be derived by the PMIx server library from other required information and included in the data made available to the server library's clients.
30 31	Description Pass job-related information to the PMIx server library for distribution to local client processes. Advice to PMIx server hosts
32 33	Host environments are required to execute this operation prior to starting any local application process within the given namespace.
34 35 36	The PMIx server must register all namespaces that will participate in collective operations with local processes. This means that the server must register a namespace even if it will not host any local processes from within that namespace if any local process of another namespace might at

some point perform an operation involving one or more processes from the new namespace. This is necessary so that the collective operation can identify the participants and know when it is locally complete.

The caller must also provide the number of local processes that will be launched within this namespace. This is required for the PMIx server library to correctly handle collectives as a collective operation call can occur before all the local processes have been started.

Advice to users -

The number of local processes for any given namespace is generally fixed at the time of application launch. Calls to <code>PMIx_Spawn</code> result in processes launched in their own namespace, not that of their parent. However, it is possible for processes to <code>migrate</code> to another node via a call to <code>PMIx_Job_control_nb</code>, thus resulting in a change to the number of local processes on both the initial node and the node to which the process moved. It is therefore critical that applications not migrate processes without first ensuring that <code>PMIx-based</code> collective operations are not in progress, and that no such operations be initiated until process migration has completed.

11.1.3.1 Assembling the registration information

The following description is not intended to represent the actual layout of information in a given PMIx library. Instead, it is describes how information provided in the *info* parameter of the **PMIx_server_register_nspace** shall be organized for proper processing by a PMIx server library. The ordering of the various information elements is arbitrary - they are presented in a top-down hierarchical form solely for clarity in reading.

Advice to PMIx server hosts -

Creating the *info* array of data requires knowing in advance the number of elements required for the array. This can be difficult to compute and somewhat fragile in practice. One method for resolving the problem is to create a linked list of objects, each containing a single <code>pmix_info_t</code> structure. Allocation and manipulation of the list can then be accomplished using existing standard methods. Upon completion, the final *info* array can be allocated based on the number of elements on the list, and then the values in the list object <code>pmix_info_t</code> structures transferred to the corresponding array element utilizing the <code>PMIX_INFO_XFER</code> macro.

A common building block used in several areas is the construction of a regular expression identifying the nodes involved in that area - e.g., the nodes in a **session** or **job**. PMIx provides several tools to facilitate this operation, beginning by constructing an argv-like array of node names. This array is then passed to the **PMIx_generate_regex** function to create a regular expression parseable by the PMIx server library, as shown below:

```
char **nodes = NULL;
char *nodelist;
char *regex;
size_t n;
pmix_status_t rc;
pmix_info_t info;
/* loop over an array of nodes, adding each
 * name to the array */
for (n=0; n < num_nodes; n++)</pre>
    /* filter the nodes to ignore those not included
     * in the target range (session, job, etc.). In
     * this example, all nodes are accepted */
    PMIX_ARGV_APPEND(&nodes, node[n]->name);
/* join into a comma-delimited string */
nodelist = PMIX ARGV JOIN(nodes, ',');
/* release the array */
PMIX ARGV FREE (nodes);
/* generate regex */
rc = PMIx_generate_regex(nodelist, &regex);
/* release list */
free (nodelist);
/* pass the regex as the value to the PMIX_NODE_MAP key */
PMIX_INFO_LOAD(&info, PMIX_NODE_MAP, regex, PMIX_STRING);
/* release the regex */
free (regex);
```

2

3

4

5

6

7 8

9

10

11

12 13

18 19 20

21

22 23

242526

27

28 29

30 31

32

33

34 35

36

37

38

Changing the filter criteria allows the construction of node maps for any level of information.

A similar method is used to construct the map of processes on each node from the namespace being registered. This may be done for each information level of interest (e.g., to identify the process map for the entire **job** or for each **application** in the job) by changing the search criteria. An example is shown below for the case of creating the process map for a **job**:

(

```
1
            char **ndppn;
2
            char rank[30];
3
            char **ppnarray = NULL;
4
            char *ppn;
5
            char *localranks;
6
            char *regex;
7
            size_t n, m;
8
            pmix_status_t rc;
9
            pmix_info_t info;
10
            /* loop over an array of nodes */
11
            for (n=0; n < num nodes; n++)
12
                /* for each node, construct an array of ranks on that node */
13
14
                ndppn = NULL;
15
                for (m=0; m < node[n]->num procs; m++)
16
                    /* ignore processes that are not part of the target job */
                    if (!PMIX CHECK NSPACE(targetjob, node[n]->proc[m].nspace))
17
18
                        continue;
19
20
                    snprintf(rank, 30, "%d", node[n]->proc[m].rank);
21
                    PMIX_ARGV_APPEND(&ndppn, rank);
22
23
                /* convert the array into a comma-delimited string of ranks */
                localranks = PMIX_ARGV_JOIN(ndppn, ',');
24
                /* release the local array */
25
26
                PMIX ARGV FREE (ndppn);
                /* add this node's contribution to the overall array */
27
28
                PMIX_ARGV_APPEND(&ppnarray, localranks);
29
                /* release the local list */
30
                free(localranks);
31
32
33
            /* join into a semicolon-delimited string */
34
            ppn = PMIX_ARGV_JOIN(ppnarray, ';');
35
            /* release the array */
36
37
            PMIX_ARGV_FREE (ppnarray);
38
39
            /* generate ppn regex */
40
            rc = PMIx_generate_ppn(ppn, &regex);
41
42
            /* release list */
```

```
2 3
```

```
free(ppn);

/* pass the regex as the value to the PMIX_PROC_MAP key */
PMIX_INFO_LOAD(&info, PMIX_PROC_MAP, regex, PMIX_STRING);
/* release the regex */
free(regex);
```

Note that the PMIX_NODE_MAP and PMIX_PROC_MAP attributes are linked in that the order of entries in the process map must match the ordering of nodes in the node map - i.e., there is no provision in the PMIx process map regular expression generator/parser pair supporting an out-of-order node or a node that has no corresponding process map entry (e.g., a node with no processes on it). Armed with these tools, the registration *info* array can be constructed as follows:

The *info* array at this point might look like (where the labels identify the corresponding attribute - e.g., "Session ID" corresponds to the **PMIX_SESSION_ID** attribute):

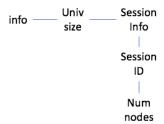


Figure 11.1.: Session-level information elements

Job-level information includes all job-specific values such as PMIX_JOB_SIZE,
 PMIX_JOB_NUM_APPS, and PMIX_JOBID. Since each invocation of
 PMIX_server_register_nspace describes a single job, job-specific values can be specified independently - i.e., in their own pmix_info_t elements of the info array.
 Alternatively, they can be provided as a pmix_data_array_t array of pmix_info_t identified by the PMIX_JOB_INFO_ARRAY attribute - this is required in cases where

non-specific attributes (e.g., **PMIX_NODE_MAP**) are passed to describe aspects of the job. Note that since the invocation only involves a single namespace, there is no need to include the **PMIX_NSPACE** attribute in the array.

Upon conclusion of this step, the *info* array might look like:

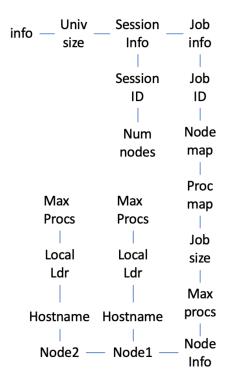


Figure 11.2.: Job-level information elements

Note that in this example, **PMIX_NUM_NODES** is not required as that information is contained in the **PMIX_NODE_MAP** attribute. Similarly, **PMIX_JOB_SIZE** is not technically required as that information is contained in the **PMIX_PROC_MAP** when combined with the corresponding node map - however, there is no issue with including the job size as a separate entry.

The example also illustrates the hierarchical use of the PMIX_NODE_INFO_ARRAY attribute. In this case, we have chosen to pass several job-related values for each node - since those values are non-unique across the job, they must be passed in a node-info container. Note that the choice of what information to pass into the PMIx server library versus what information to derive from other values at time of request is left to the host environment. PMIx implementors in turn may, if they choose, pre-parse registration data to create expanded views (thus enabling faster response to requests at the expense of memory footprint) or to compress views into tighter representations (thus trading minimized footprint for longer response times).

Application-level information includes all application-specific values such as PMIX_APP_SIZE and PMIX_APPLDR. If the job contains only a single application, then the application-specific values can be specified independently - i.e., in their own pmix_info_t elements of the *info* array - or as a pmix_data_array_t array of pmix_info_t using the PMIX_APP_INFO_ARRAY attribute and identifed by including the PMIX_APPNUM attribute in the array. Use of the array format is must in cases where non-specific attributes (e.g., PMIX_NODE_MAP) are passed to describe aspects of the application.

However, in the case of a job consisting of multiple applications, all application-specific values for each application must be provided using the **PMIX_APP_INFO_ARRAY** format, each identified by its **PMIX_APPNUM** value.

Upon conclusion of this step, the *info* array might look like that shown in 11.3, assuming there are two applications in the job being registered:

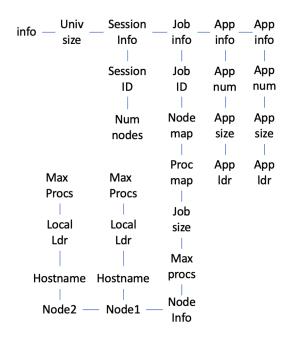


Figure 11.3.: Application-level information elements

- Process-level information includes an entry for each process in the job being registered, each
 entry marked with the PMIX_PROC_INFO_ARRAY attribute. The rank of the process must
 be the first entry in the array this provides efficiency when storing the data. Upon conclusion of
 this step, the *info* array might look like the diagram in 11.4:
- For purposes of this example, node-level information only includes values describing the local node i.e., it does not include information about other nodes in the job or session. In many cases, the values included in this level are unique to it and can be specified independently i.e., in their

Figure 11.4.: Process-level information elements

own pmix_info_t elements of the *info* array. Alternatively, they can be provided as a pmix_data_array_t array of pmix_info_t using the PMIX_NODE_INFO_ARRAY attribute - this is required in cases where non-specific attributes are passed to describe aspects of the node, or where values for multiple nodes are being provided.

The node-level information requires two elements that must be constructed in a manner similar to that used for the node map. The **PMIX_LOCAL_PEERS** value is computed based on the processes on the local node, filtered to select those from the job being registered, as shown below using the tools provided by PMIx:

```
char **ndppn = NULL;
char rank[30];
char *localranks;
size_t m;
pmix_info_t info;

for (m=0; m < mynode->num_procs; m++)
    /* ignore processes that are not part of the target job */
    if (!PMIX_CHECK_NSPACE(targetjob,mynode->proc[m].nspace))
        continue;

    snprintf(rank, 30, "%d", mynode->proc[m].rank);
    PMIX_ARGV_APPEND(&ndppn, rank);

/* convert the array into a comma-delimited string of ranks */
localranks = PMIX_ARGV_JOIN(ndppn, ',');
/* release the local array */
```

```
2
3
               /* pass the string as the value to the PMIX LOCAL PEERS key */
4
               PMIX INFO LOAD (&info, PMIX LOCAL PEERS, localranks, PMIX STRING);
5
               /* release the list */
6
               free(localranks);
7
8
               The PMIX LOCAL CPUSETS value is constructed in a similar manner. In the provided
9
               example, it is assumed that the Hardware Locality (HWLOC) cpuset representation (a
               comma-delimited string of processor IDs) of the processors assigned to each process has
10
               previously been generated and stored on the process description. Thus, the value can be
11
               constructed as shown below:
12
               char **ndcpus = NULL;
13
               char *localcpus;
14
15
               size t m;
               pmix_info_t info;
16
17
18
               for (m=0; m < mynode->num_procs; m++)
                   /* ignore processes that are not part of the target job */
19
20
                   if (!PMIX_CHECK_NSPACE(targetjob, mynode->proc[m].nspace))
21
                        continue;
22
                   PMIX_ARGV_APPEND(&ndcpus, mynode->proc[m].cpuset);
23
24
               /* convert the array into a colon-delimited string */
25
26
               localcpus = PMIX ARGV JOIN(ndcpus, ':');
               /* release the local array */
27
               PMIX ARGV FREE (ndcpus);
28
29
30
               /* pass the string as the value to the PMIX_LOCAL_CPUSETS key */
31
               PMIX INFO LOAD (&info, PMIX LOCAL CPUSETS, localcpus, PMIX STRING);
               /* release the list */
32
33
               free(localcpus);
34
```

35 36 PMIX_ARGV_FREE (ndppn);

Note that for efficiency, these two values can be computed at the same time.

The final *info* array might therefore look like the diagram in 11.5:

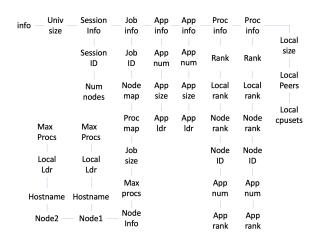


Figure 11.5.: Final information array

11.1.4 PMIx_server_deregister_nspace

```
Summary
2
              Deregister a namespace.
4
              Format
   PMIx v1.0
5
              void PMIx_server_deregister_nspace(const pmix_nspace_t nspace,
6
                                           pmix_op_cbfunc_t cbfunc, void *cbdata)
7
              IN
                  nspace
8
                  Namespace (string)
9
             IN
                  cbfunc
10
                  Callback function pmix op cbfunc t (function reference)
             IN
11
                  cbdata
12
                  Data to be passed to the callback function (memory reference)
```

Description 1 2 Deregister the specified *nspace* and purge all objects relating to it, including any client information from that namespace. This is intended to support persistent PMIx servers by providing an 3 4 opportunity for the host RM to tell the PMIx server library to release all memory for a completed 5 job. Note that the library must not invoke the callback function prior to returning from the API. 11.1.5 PMIx server register client **Summary** 7 8 Register a client process with the PMIx server library. **Format** 9 PMIx v1.010 pmix status t PMIx_server_register_client(const pmix_proc_t *proc, 11 uid t uid, gid t gid, 12 void *server object, 13 pmix op cbfunc t cbfunc, void *cbdata) 14 IN 15 proc 16 pmix_proc_t structure (handle) 17 IN uid user id (integer) 18 IN 19 gid group id (integer) 20 IN server_object 21 22 (memory reference) IN 23 cbfunc Callback function pmix_op_cbfunc_t (function reference) 24 IN 25 26 Data to be passed to the callback function (memory reference) Returns one of the following: 27 • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result 28 29 will be returned in the provided *cbfunc*. Note that the library must not invoke the callback 30 function prior to returning from the API. • PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and 31 32 returned success - the cbfunc will not be called 33 • a PMIx error constant indicating either an error in the input or that the request was immediately

processed and failed - the *cbfunc* will not be called

Description 1 2 Register a client process with the PMIx server library. 3 The host server can also, if it desires, provide an object it wishes to be returned when a server 4 function is called that relates to a specific process. For example, the host server may have an object 5 that tracks the specific client. Passing the object to the library allows the library to provide that 6 object to the host server during subsequent calls related to that client, such as a 7 pmix server client connected fn t function. This allows the host server to access 8 the object without performing a lookup based on the client's namespace and rank. Advice to PMIx server hosts -9 Host environments are required to execute this operation prior to starting the client process. The expected user ID and group ID of the child process allows the server library to properly authenticate 10 clients as they connect by requiring the two values to match. Accordingly, the detected user and 11 group ID's of the connecting process are not included in the 12 pmix server client connected fn t server module function. 13 Advice to PMIx library implementers 14 For security purposes, the PMIx server library should check the user and group ID's of a 15 connecting process against those provided for the declared client process identifier via the PMIx server register client prior to completing the connection. 16 11.1.6 PMIx server deregister client 17 Summary 18 Deregister a client and purge all data relating to it. 19 **Format** 20 PMIx v1.0 21 22 PMIx_server_deregister_client(const pmix_proc_t *proc, pmix_op_cbfunc_t cbfunc, void *cbdata) 23 IN 24 proc 25 pmix proc t structure (handle) IN cbfunc 26 27 Callback function **pmix** op **cbfunc t** (function reference) IN 28 29 Data to be passed to the callback function (memory reference)

Description

The PMIx_server_deregister_nspace API will delete all client information for that namespace. The PMIx server library will automatically perform that operation upon disconnect of all local clients. This API is therefore intended primarily for use in exception cases, but can be called in non-exception cases if desired. Note that the library must not invoke the callback function prior to returning from the API.

11.1.7 PMIx_server_setup_fork

Summary

Setup the environment of a child process to be forked by the host.

Format

PMIx v1.0

1

3

4

5

6

8

10

11

12

13

15

16 17

18

19 20

21

22

23

24 25

26

28

29

pmix_status_t

PMIx_server_setup_fork(const pmix_proc_t *proc,

char ***env)

14 **IN** proc

pmix proc t structure (handle)

IN env

Environment array (array of strings)

Returns **PMIX SUCCESS** or a negative value corresponding to a PMIx error constant.

Description

Setup the environment of a child process to be forked by the host so it can correctly interact with the PMIx server.

Advice to PMIx server hosts

Host environments are required to execute this operation prior to starting the client process.

The PMIx client needs some setup information so it can properly connect back to the server. This function will set appropriate environmental variables for this purpose, and will also provide any environmental variables that were specified in the launch command (e.g., via PMIx_Spawn) plus other values (e.g., variables required to properly initialize the client's fabric library).

11.1.8 PMIx server dmodex request

Summary

Define a function by which the host server can request modex data from the local PMIx server.

Format

PMIx v1.0

C

IN proc

pmix_proc_t structure (handle)

IN cbfunc

Callback function pmix_dmodex_response_fn_t (function reference)

IN cbdata

Data to be passed to the callback function (memory reference)

Returns one of the following:

- PMIX_SUCCESS, indicating that the request is being processed by the host environment result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API.
- a PMIx error constant indicating an error in the input the *cbfunc* will not be called

Description

Define a function by which the host server can request modex data from the local PMIx server. Traditional wireup procedures revolve around the per-process posting of data (e.g., location and endpoint information) via the PMIx_Put and PMIx_Commit functions followed by a PMIx_Fence barrier that globally exchanges the posted information. However, the barrier operation represents a signficant time impact at large scale.

PMIx supports an alternative wireup method known as *Direct Modex* that replaces the barrier-based exchange of all process-posted information with on-demand fetch of a peer's data. In place of the barrier operation, data posted by each process is cached on the local PMIx server. When a process requests the information posted by a particular peer, it first checks the local cache to see if the data is already available. If not, then the request is passed to the local PMIx server, which subsequently requests that its RM host request the data from the RM daemon on the node where the specified peer process is located. Upon receiving the request, the RM daemon passes the request into its PMIx server library using the PMIx_server_dmodex_request function, receiving the response in the provided *cbfunc* once the indicated process has posted its information. The RM daemon then returns the data to the requesting daemon, who subsequently passes the data to its PMIx server library for transfer to the requesting client.

Advice to users -

While direct modex allows for faster launch times by eliminating the barrier operation, per-peer retrieval of posted information is less efficient. Optimizations can be implemented - e.g., by returning posted information from all processes on a node upon first request - but in general direct modex remains best suited for sparsely connected applications.

1 11.1.9	PMIx_server_setup_application
2	Summary
3 4	Provide a function by which the resource manager can request application-specific setup data prior to launch of a job.
	-
5 <i>PMIx v2.</i> 0	Format C
6	pmix_status_t
7	PMIx_server_setup_application(const pmix_nspace_t nspace,
8	pmix_info_t info[], size_t ninfo,
9	<pre>pmix_setup_application_cbfunc_t cbfunc,</pre>
10	void *cbdata)
	C -
11	IN nspace
12	namespace (string)
13	IN info
14	Array of info structures (array of handles)
15 16	IN ninfo Number of elements in the <i>info</i> array (integer)
17	IN cbfunc
18	Callback function pmix_setup_application_cbfunc_t (function reference)
19	IN cbdata
20	Data to be passed to the <i>cbfunc</i> callback function (memory reference)
21	Returns one of the following:
22	• PMIX_SUCCESS, indicating that the request is being processed by the host environment - result
23	will be returned in the provided cbfunc. Note that the library must not invoke the callback
24	function prior to returning from the API.
25	• a PMIx error constant indicating either an error in the input - the <i>cbfunc</i> will not be called
	▼
26	PMIx libraries that support this operation are required to support the following:
27	PMIX_SETUP_APP_ENVARS "pmix.setup.env" (bool)
28	Harvest and include relevant environmental variables
29	<pre>PMIX_SETUP_APP_NONENVARS ""pmix.setup.nenv" (bool)</pre>
30	Include all relevant data other than environmental variables
31	PMIX_SETUP_APP_ALL "pmix.setup.all" (bool)

1	Include all relevant data
2 3 4 5	<pre>PMIX_ALLOC_FABRIC "pmix.alloc.net" (array) Array of pmix_info_t describing requested fabric resources. This must include at least: PMIX_ALLOC_FABRIC_ID, PMIX_ALLOC_FABRIC_TYPE, and PMIX_ALLOC_FABRIC_ENDPTS, plus whatever other descriptors are desired.</pre>
6 7 8 9 10 11 12 13 14 15 16 17 18	The key to be used when accessing this requested fabric allocation. The allocation will be returned/stored as a pmix_data_array_t of pmix_info_t indexed by this key and containing at least one entry with the same key and the allocated resource description. The type of the included value depends upon the fabric support. For example, a TCP allocation might consist of a comma-delimited string of socket ranges such as "32000-32100,33005,38123-38146". Additional entries will consist of any provided resource request directives, along with their assigned values. Examples include: PMIX_ALLOC_FABRIC_TYPE - the type of resources provided; PMIX_ALLOC_FABRIC_PLANE - if applicable, what plane the resources were assigned from; PMIX_ALLOC_FABRIC_QOS - the assigned QoS; PMIX_ALLOC_BANDWIDTH - the allocated bandwidth; PMIX_ALLOC_FABRIC_SEC_KEY - a security key for the requested fabric allocation. NOTE: the assigned values may differ from those requested, especially if PMIX_INFO_REQD was not set in the request.
20 21	<pre>PMIX_ALLOC_FABRIC_SEC_KEY "pmix.alloc.nsec" (pmix_byte_object_t) Fabric security key</pre>
22 23	<pre>PMIX_ALLOC_FABRIC_TYPE "pmix.alloc.nettype" (char*) Type of desired transport (e.g., "tcp", "udp")</pre>
24 25	<pre>PMIX_ALLOC_FABRIC_PLANE "pmix.alloc.netplane" (char*) ID string for the NIC (aka plane) to be used for this allocation (e.g., CIDR for Ethernet)</pre>
26 27	<pre>PMIX_ALLOC_FABRIC_ENDPTS "pmix.alloc.endpts" (size_t) Number of endpoints to allocate per process</pre>
28 29	PMIX_ALLOC_FABRIC_ENDPTS_NODE "pmix.alloc.endpts.nd" (size_t) Number of endpoints to allocate per node
00	Optional Attributes
30	PMIx libraries that support this operation may support the following:
31 32	PMIX_ALLOC_BANDWIDTH "pmix.alloc.bw" (float) Mbits/sec.
33 34	<pre>PMIX_ALLOC_FABRIC_QOS</pre>
35	<pre>PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)</pre>

The following optional attributes may be provided by the host environment to identify the
programming model (as specified by the user) being executed within the application. The PMIx server library may utilize this information to harvest/forward model-specific environmental variables, record the programming model associated with the application, etc.
• PMIX_PROGRAMMING_MODEL "pmix.pgm.model" (char*) Programming model being initialized (e.g., "MPI" or "OpenMP")
• PMIX_MODEL_LIBRARY_NAME "pmix.mdl.name" (char*) Programming model implementation ID (e.g., "OpenMPI" or "MPICH")
• PMIX_MODEL_LIBRARY_VERSION "pmix.mld.vrs" (char*) Programming model version string (e.g., "2.1.1")
Description Provide a function by which the RM can request application-specific setup data (e.g., environmental
variables, fabric configuration and security credentials) from supporting PMIx server library subsystems prior to initiating launch of a job.
Advice to PMIx server hosts
Host environments are required to execute this operation prior to launching a job. In addition to supported directives, the <i>info</i> array must include a description of the job using the PMIX_NODE_MAP and PMIX_PROC_MAP attributes.
This is defined as a non-blocking operation in case contributing subsystems need to perform some potentially time consuming action (e.g., query a remote service) before responding. The returned data must be distributed by the RM and subsequently delivered to the local PMIx server on each node where application processes will execute, prior to initiating execution of those processes.
Advice to PMIx library implementers ————————————————————————————————————
Support for harvesting of environmental variables and providing of local configuration information by the PMIx implementation is optional.

11.1.10 PMIx_Register_attributes

Summary

26

27

Register host environment attribute support for a function.

Format 1 PMIx v4.0 2 pmix status t 3 PMIx Register attributes (char *function, 4 pmix regattr t attrs[], size_t nattrs) 5 6 IN function 7 String name of function (string) 8 IN attrs 9 Array of pmix_regattr_t describing the supported attributes (handle) nattrs 10 IN Number of elements in *attrs* (size_t) 11 12 Returns **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant. **Description** 13 14 The PMIx_Register_attributes function is used by the host environment to register with its PMIx server library the attributes it supports for each pmix server module t function. 15 16 The function is the string name of the server module function (e.g., "register events", 17 "validate credential", or "allocate") whose attributes are being registered. See the **pmix regattr** t entry for a description of the *attrs* array elements. 18 19 Note that the host environment can also query the library (using the PMIx Query_info_nb 20 API) for its attribute support both at the server, client, and tool levels once the host has executed **PMIx_server_init** since the server will internally register those values. 21 Advice to PMIx server hosts — Host environments are strongly encouraged to register all supported attributes immediately after 22 23 initializing the library to ensure that user requests are correctly serviced.

Advice to PMIx library implementers —

PMIx implementations are *required* to register all internally supported attributes for each API during initialization of the library (i.e., when the process calls their respective PMIx init function). Specifically, the implementation *must not* register supported attributes upon first call to a given API as this would prevent users from discovering supported attributes prior to first use of an API.

It is the implementation's responsibility to associate registered attributes for a given **pmix_server_module_t** function with their corresponding user-facing API. Supported attributes *must* be reported to users in terms of their support for user-facing APIs, broken down by the level (see 3.4.31) at which the attribute is supported.

Note that attributes can/will be registered on an API for each level. It is *required* that the implementation support user queries for supported attributes on a per-level basis. Duplicate registrations at the *same* level for a function *shall* return an error - however, duplicate registrations at *different* levels *shall* be independently tracked.

11.1.11 PMIx_server_setup_local_support

Summary

Provide a function by which the local PMIx server can perform any application-specific operations prior to spawning local clients of a given application.

Format

```
PMIx v2.0
```

```
23 IN nspace
24 Namespace (string)
```

IN info

Array of info structures (array of handles)

IN ninfo

Number of elements in the *info* array (size_t)

IN cbfunc

Callback function pmix_op_cbfunc_t (function reference)

IN cbdata

Data to be passed to the callback function (memory reference)

Returns one of the fol	lowing
------------------------	--------

- PMIX_SUCCESS, indicating that the request is being processed by the host environment result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API.
- PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned success - the cbfunc will not be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed the *cbfunc* will not be called

Provide a function by which the local PMIx server can perform any application-specific operations prior to spawning local clients of a given application. For example, a fabric library might need to setup the local driver for "instant on" addressing. The data provided in the *info* array is the data returned to the host RM by the callback function executed as a result of a call to **PMIx server setup application**.

Advice to PMIx server hosts —

Host environments are required to execute this operation prior to starting any local application processes from the specified namespace.

11.1.12 PMIx_server_IOF_deliver

Summary

Provide a function by which the host environment can pass forwarded IO to the PMIx server library for distribution to its clients.

Format

PMIx v3.0

ı	111	source
2		Pointer to pmix_proc_t identifying source of the IO (handle)
3	IN	channel
4		IO channel of the data (pmix_iof_channel_t)
5	IN	bo
6		Pointer to <pre>pmix_byte_object_t</pre> containing the payload to be delivered (handle)
7	IN	info
8		Array of pmix_info_t metadata describing the data (array of handles)
9	IN	ninfo
10		Number of elements in the <i>info</i> array (size_t)
11	IN	cbfunc
12		Callback function <pre>pmix_op_cbfunc_t</pre> (function reference)
13	IN	cbdata
14		Data to be passed to the callback function (memory reference)
15	Retu	arns one of the following:
16	• P	MIX_SUCCESS, indicating that the request is being processed by the host environment - result
17		rill be returned in the provided <i>cbfunc</i> . Note that the library must not invoke the callback
18	fu	unction prior to returning from the API.
19	• P	MIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
20		sturned success - the chfunc will not be called

Provide a function by which the host environment can pass forwarded IO to the PMIx server library for distribution to its clients. The PMIx server library is responsible for determining which of its clients have actually registered for the provided data and delivering it. The *cbfunc* callback function will be called once the PMIx server library no longer requires access to the provided data.

• a PMIx error constant indicating either an error in the input or that the request was immediately

8 11.1.13 PMIx_server_collect_inventory

processed and failed - the cbfunc will not be called

Summary

21

22

23

24

25

26

27

29

30

Collect inventory of resources on a node

Format 1 PMIx v3.0 2 pmix status t 3 PMIx_server_collect_inventory(const pmix_info_t directives[], 4 size t ndirs, 5 pmix_info_cbfunc_t cbfunc, 6 void *cbdata); 7 IN directives Array of **pmix_info_t** directing the request (array of handles) 8 9 IN ndirs Number of elements in the *directives* array (size_t) 10 IN 11 12 Callback function to return collected data (pmix info cbfunc t function reference) IN cbdata 13 14 Data to be passed to the callback function (memory reference) 15 Returns **PMIX** SUCCESS or a negative value corresponding to a PMIx error constant. In the event 16 the function returns an error, the *cbfunc* will not be called. **Description** 17 Provide a function by which the host environment can request its PMIx server library collect an 18 inventory of local resources. Supported resources depends upon the PMIx implementation, but may 19 20 include the local node topology and fabric interfaces. Advice to PMIx server hosts — 21 This is a non-blocking API as it may involve somewhat lengthy operations to obtain the requested 22 information. Inventory collection is expected to be a rare event – at system startup and upon 23 command from a system administrator. Inventory updates are expected to initiate a smaller 24 operation involving only the changed information. For example, replacement of a node would generate an event to notify the scheduler with an inventory update without invoking a global 25 26 inventory operation.

11.1.14 PMIx_server_deliver_inventory

Summary

Pass collected inventory to the PMIx server library for storage

28

PMIx v3.0	C
2	pmix_status_t
3	PMIx_server_deliver_inventory(const_pmix_info_t info[],
4	size_t ninfo,
5	<pre>const pmix_info_t directives[],</pre>
6	size_t ndirs,
7	<pre>pmix_op_cbfunc_t cbfunc,</pre>
8	void *cbdata);
9 10	IN info Array of pmix_info_t containing the inventory (array of handles)
11	IN ninfo
12	Number of elements in the <i>info</i> array (size_t)
13	IN directives
14	Array of pmix_info_t directing the request (array of handles)
15	IN ndirs
16	Number of elements in the <i>directives</i> array (size_t)
17	IN cbfunc
18	Callback function pmix_op_cbfunc_t (function reference)
19	IN cbdata
20	Data to be passed to the callback function (memory reference)
21	Returns one of the following:
22	• PMIX_SUCCESS, indicating that the request is being processed by the host environment - result
23	will be returned in the provided <i>cbfunc</i> . Note that the library must not invoke the callback
24	function prior to returning from the API.
25	• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
26	returned <i>success</i> - the <i>cbfunc</i> will not be called
27	• a PMIx error constant indicating either an error in the input or that the request was immediately
28	processed and failed - the <i>cbfunc</i> will not be called
29	Description
30	Provide a function by which the host environment can pass inventory information obtained from a
31	node to the PMIx server library for storage. Inventory data is subsequently used by the PMIx server
32	library for allocations in response to PMIx_server_setup_application, and may be
33	available to the library's host via the PMIx_Get API (depending upon PMIx implementation).
34	The cbfunc callback function will be called once the PMIx server library no longer requires access
35	to the provided data.

Format

1 11.2 Server Function Pointers

PMIx utilizes a "function-shipping" approach to support for implementing the server-side of the protocol. This method allows RMs to implement the server without being burdened with PMIx internal details. When a request is received from the client, the corresponding server function will be called with the information.

Any functions not supported by the RM can be indicated by a **NULL** for the function pointer. PMIx implementations are required to return a **PMIX_ERR_NOT_SUPPORTED** status to all calls to functions that require host environment support and are not backed by a corresponding server module entry.

The host RM will provide the function pointers in a <code>pmix_server_module_t</code> structure passed to <code>PMIx_server_init</code>. That module structure and associated function references are defined in this section.

Advice to PMIx server hosts -

For performance purposes, the host server is required to return as quickly as possible from all functions. Execution of the function is thus to be done asynchronously so as to allow the PMIx server support library to handle multiple client requests as quickly and scalably as possible.

All data passed to the host server functions is "owned" by the PMIX server support library and must not be free'd. Data returned by the host server via callback function is owned by the host server, which is free to release it upon return from the callback

11.2.1 pmix_server_module_t Module

20 Summary

2

4

5 6

7

8

9

10

11 12

13

14

15

16 17

18

21

List of function pointers that a PMIx server passes to **PMIx_server_init** during startup.

22 Format

```
1
            typedef struct pmix server module 3 0 0 t
2
                /* v1x interfaces */
3
                pmix_server_client_connected_fn_t
                                                      client_connected;
4
                pmix_server_client_finalized_fn_t
                                                      client finalized;
5
                pmix_server_abort_fn_t
                                                      abort;
6
                pmix_server_fencenb_fn_t
                                                      fence nb;
7
                pmix_server_dmodex_req_fn_t
                                                      direct modex;
8
                pmix_server_publish_fn_t
                                                      publish;
9
                pmix_server_lookup_fn_t
                                                      lookup;
                pmix_server_unpublish_fn t
10
                                                      unpublish;
                pmix_server_spawn_fn_t
11
                                                      spawn;
12
                pmix_server_connect_fn_t
                                                      connect;
13
                pmix_server_disconnect_fn_t
                                                      disconnect;
14
                pmix_server_register_events_fn_t
                                                      register_events;
15
                pmix server deregister events fn t
                                                      deregister_events;
16
                pmix server listener fn t
                                                      listener;
17
                /* v2x interfaces */
18
                pmix server notify event fn t
                                                      notify_event;
19
                pmix_server_query_fn_t
                                                      query;
20
                pmix_server_tool_connection_fn_t
                                                      tool_connected;
21
                pmix_server_log_fn_t
                                                      log;
22
                pmix_server_alloc_fn_t
                                                      allocate;
23
                pmix_server_job_control_fn_t
                                                       job_control;
24
                pmix_server_monitor_fn_t
                                                      monitor;
                /* v3x interfaces */
25
                pmix_server_get_cred_fn_t
26
                                                      get_credential;
27
                pmix_server_validate_cred_fn_t
                                                      validate_credential;
28
                pmix_server_iof_fn_t
                                                      iof pull;
29
                pmix_server_stdin_fn_t
                                                      push_stdin;
30
                /* v4x interfaces */
31
                pmix server grp fn t
                                                      group;
```

11.2.2 pmix_server_client_connected_fn_t

pmix_server_fabric_fn_t

pmix_server_module_t;

Summary

32

33

35

36

Notify the host server that a client connected to this server.

fabric;

IN proc
 pmix_proc_t structure (handle)
IN server_object
 object reference (memory reference)
IN cbfunc
 Callback function pmix_op_cbfunc_t (function reference)
IN cbdata
 Data to be passed to the callback function (memory reference)

Returns one of the following:

- PMIX_SUCCESS, indicating that the request is being processed by the host environment result will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function prior to returning from the API.
- PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned *success* the *cbfunc* will not be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed the *cbfunc* will not be called

Description

Notify the host environment that a client has called **PMIx_Init**. Note that the client will be in a blocked state until the host server executes the callback function, thus allowing the PMIx server support library to release the client. The server_object parameter will be the value of the server_object parameter passed to **PMIx_server_register_client** by the host server when registering the connecting client. If provided, an implementation of **pmix_server_client_connected_fn_t** is only required to call the callback function designated. A host server can choose to not be notified when clients connect by setting **pmix_server_client_connected_fn_t** to **NULL**.

It is possible that only a subset of the clients in a namespace call **PMIx_Init**. The server's **pmix_server_client_connected_fn_t** implementation should not depend on being called once per rank in a namespace or delay calling the callback function until all ranks have connected. However, if a rank makes any PMIx calls, it must first call **PMIx_Init** and therefore the server's **pmix_server_client_connected_fn_t** will be called before any other server functions specific to the rank.

Adv	ice to	PM	Ix ser	ver	host	9
/ \U v	וטט נט	1 1 1 1 1		v C i	11031	. •

This operation is an opportunity for a host environment to update the status of the ranks it manages. It is also a convenient and well defined time to perform initialization necessary to support further calls into the server related to that rank.

11.2.3 pmix_server_client_finalized_fn_t

Summary

Notify the host environment that a client called **PMIx_Finalize**.

Format

PMIx v1.0

C

```
IN proc
    pmix_proc_t structure (handle)
IN server_object
    object reference (memory reference)
IN cbfunc
```

Callback function **pmix_op_cbfunc_t** (function reference)

IN cbdata

Data to be passed to the callback function (memory reference)

Returns one of the following:

- PMIX_SUCCESS, indicating that the request is being processed by the host environment result will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function prior to returning from the API.
- PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned *success* the *cbfunc* will not be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed the *cbfunc* will not be called

Notify the host environment that a client called **PMIx_Finalize**. Note that the client will be in a blocked state until the host server executes the callback function, thus allowing the PMIx server support library to release the client. The server_object parameter will be the value of the server_object parameter passed to **PMIx_server_register_client** by the host server when registering the connecting client. If provided, an implementation of **pmix_server_client_finalized_fn_t** is only required to call the callback function designated. A host server can choose to not be notified when clients finalize by setting **pmix_server_client_finalized_fn_t** to **NULL**.

Note that the host server is only being informed that the client has called **PMIx_Finalize**. The client might not have exited. If a client exits without calling **PMIx_Finalize**, the server support library will not call the **pmix_server_client_finalized_fn_t** implementation.

Advice to PMIx server hosts —

This operation is an opportunity for a host server to update the status of the tasks it manages. It is also a convenient and well defined time to release resources used to support that client.

11.2.4 pmix_server_abort_fn_t

Summary

Notify the host environment that a local client called **PMIx Abort**.

Format

```
PMIx v1.0
```

pmix_proc_t procs[],
size_t nprocs,
pmix_op_cbfunc_t cbfunc,
void *cbdata)

1	IN proc
2	<pre>pmix_proc_t structure identifying the process requesting the abort (handle)</pre>
3	<pre>IN server_object</pre>
4	object reference (memory reference)
5	IN status
6	exit status (integer)
7	IN msg
8	exit status message (string)
9	IN procs
10	Array of pmix_proc_t structures identifying the processes to be terminated (array of
11	handles)
12	IN nprocs
13	Number of elements in the <i>procs</i> array (integer)
14	IN cbfunc
15	Callback function <pre>pmix_op_cbfunc_t</pre> (function reference)
16	IN cbdata
17	Data to be passed to the callback function (memory reference)
18	Returns one of the following:
19	• PMIX_SUCCESS, indicating that the request is being processed by the host environment - resul
20	will be returned in the provided <i>cbfunc</i> . Note that the host must not invoke the callback function
21	prior to returning from the API.
20	
22	• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
23	returned <i>success</i> - the <i>cbfunc</i> will not be called
24	• PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the
25	request, even though the function entry was provided in the server module - the cbfunc will not
26	be called
27	• a PMIx error constant indicating either an error in the input or that the request was immediately
28	processed and failed - the <i>cbfunc</i> will not be called
29	Description
30	A local client called PMIx_Abort . Note that the client will be in a blocked state until the host
31	server executes the callback function, thus allowing the PMIx server library to release the client.
32	The array of <i>procs</i> indicates which processes are to be terminated. A NULL indicates that all

11.2.5 pmix_server_fencenb_fn_t

Summary

33

35

36

At least one client called either **PMIx_Fence** or **PMIx_Fence_nb**.

processes in the client's namespace are to be terminated.

PMIx v1.	o	C			
1 MIX VI.	typedef pmix_status_t (*pmix_server_fencenb_fn_t)(
3	const pmix_proc_t procs[],				
4 size_t nprocs,					
5		const pmix_info_t info[],			
6		size_t ninfo,			
7		char *data, size_t ndata,			
8		pmix_modex_cbfunc_t cbfunc,			
9		void *cbdata)			
		C			
0	IN procs	s			
1	_	of pmix_proc_t structures identifying operation participants(array of handles)			
2	IN nproc				
3	_	r of elements in the <i>procs</i> array (integer)			
4	<pre>IN info</pre>				
5	Array of info structures (array of handles)				
6	IN ninfo	· · · · · · · · · · · · · · · · · · ·			
7	Number	r of elements in the <i>info</i> array (integer)			
8	<pre>IN data</pre>				
9	(string)				
20	<pre>IN ndata</pre>				
21	(integer				
22	<pre>IN cbfun</pre>	ıc			
23	Callbac	k function <pre>pmix_modex_cbfunc_t</pre> (function reference)			
24	<pre>IN cbdat</pre>	a			
25	Data to	be passed to the callback function (memory reference)			
26	Returns one of	of the following:			
27	• PMIX SU	CCESS, indicating that the request is being processed by the host environment - result			
28	will be returned in the provided <i>cbfunc</i> . Note that the host must not invoke the callback function				
29		curning from the API.			
	•	-			
30		ERATION_SUCCEEDED , indicating that the request was immediately processed and			
31		uccess - the cbfunc will not be called			
32		R_NOT_SUPPORTED , indicating that the host environment does not support the			
3	•	en though the function entry was provided in the server module - the <i>cbfunc</i> will not			
34	be called				
35	• a PMIx err	or constant indicating either an error in the input or that the request was immediately			
16		and failed - the <i>cbfunc</i> will not be called			

The	following attributes are required to be supported by all host environments:
PMI	X_COLLECT_DATA "pmix.collect" (bool) Collect data and return it at the end of the operation.
~ -	Optional Attributes
The	following attributes are optional for host environments:
PMI	X_TIMEOUT "pmix.timeout" (int) Time in seconds before the specified operation should time out (0 indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that preven the target process from ever exposing its data.
PMI	X_COLLECTIVE_ALGO "pmix.calgo" (char*) Comma-delimited list of algorithms to use for the collective operation. PMIx does not impose any requirements on a host environment's collective algorithms. Thus, the acceptable values for this attribute will be environment-dependent - users are encouraged to check their host environment for supported values.
PMI	X_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool) If true, indicates that the requested choice of algorithm is mandatory.
	Advice to PMIx server hosts
Host	environment are required to return PMIX_ERR_NOT_SUPPORTED if passed an attributed

All local clients in the provided array of *procs* called either **PMIx_Fence** or **PMIx_Fence_nb**. In either case, the host server will be called via a non-blocking function to execute the specified operation once all participating local processes have contributed. All processes in the specified *procs* array are required to participate in the **PMIx_Fence/PMIx_Fence_nb** operation. The callback is to be executed once every daemon hosting at least one participant has called the host server's **pmix server fencenb fn t** function.

Advice to PMIx library implementers ————

The PMIx server library is required to aggregate participation by local clients, passing the request to the host environment once all local participants have executed the API.

— Advice to PMIx server hosts ————

The host will receive a single call for each collective operation. It is the responsibility of the host to identify the nodes containing participating processes, execute the collective across all participating nodes, and notify the local PMIx server library upon completion of the global collective. Data received from each node must be simply concatenated to form an aggregated unit, as shown in the following example:

```
uint8_t *blob1, *blob2, *total;
size_t sz_blob1, sz_blob2, sz_total;

sz_total = sz_blob1 + sz_blob2;
total = (uint8_t*)malloc(sz_total);
memcpy(total, blob1, sz_blob1);
memcpy(&total[sz_blob1], blob2, sz_blob2);
```

Note that the ordering of the data blobs does not matter. The host is responsible for free'ing the *data* object passed to it by the PMIx server library.

The provided data is to be collectively shared with all PMIx servers involved in the fence operation, and returned in the modex *cbfunc*. A **NULL** data value indicates that the local processes had no data to contribute.

The array of *info* structs is used to pass user-requested options to the server. This can include directives as to the algorithm to be used to execute the fence operation. The directives are optional unless the **PMIX_INFO_REQD** flag has been set - in such cases, the host RM is required to return an error if the directive cannot be met.

11.2.6 pmix_server_dmodex_req_fn_t 2 Summary Used by the PMIx server to request its local host contact the PMIx server on the remote node that 3 hosts the specified proc to obtain and return a direct modex blob for that proc. 4 Format 5 PMIx v1.0 6 typedef pmix status t (*pmix server dmodex req fn t) (7 const pmix_proc_t *proc, const pmix info t info[], 8 9 size t ninfo, 10 pmix modex cbfunc t cbfunc, void *cbdata) 11 IN 12 proc pmix_proc_t structure identifying the process whose data is being requested (handle) 13 IN 14 Array of info structures (array of handles) 15 16 IN ninfo Number of elements in the *info* array (integer) 17 18 IN cbfunc Callback function **pmix** modex cbfunc t (function reference) 19 IN 20 21 Data to be passed to the callback function (memory reference) 22 Returns one of the following: 23 • PMIX SUCCESS, indicating that the request is being processed by the host environment - result 24 will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function 25 prior to returning from the API. 26 • PMIX ERR NOT SUPPORTED, indicating that the host environment does not support the 27 request, even though the function entry was provided in the server module - the cbfunc will not 28 be called 29 • a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the cbfunc will not be called 30

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing.

A-----

Optional Attributes The following attributes are optional for host environments that support this operation: 1 2 PMIX TIMEOUT "pmix.timeout" (int) 3 Time in seconds before the specified operation should time out (θ indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent 4 the target process from ever exposing its data. 5 **Description** 6 7 Used by the PMIx server to request its local host contact the PMIx server on the remote node that hosts the specified proc to obtain and return any information that process posted via calls to 8 PMIx Put and PMIx Commit. 9 10 The array of *info* structs is used to pass user-requested options to the server. This can include a 11 timeout to preclude an indefinite wait for data that may never become available. The directives are optional unless the mandatory flag has been set - in such cases, the host RM is required to return an 12 error if the directive cannot be met. 13 11.2.7 pmix server publish fn t Summary 15 Publish data per the PMIx API specification. 16 **Format** 17 PMIx v1.0 typedef pmix_status_t (*pmix_server_publish_fn_t)(18 19 const pmix_proc_t *proc, 20 const pmix_info_t info[], size_t ninfo, 21 22 pmix_op_cbfunc_t cbfunc, void *cbdata) 23 IN 24 proc 25 pmix_proc_t structure of the process publishing the data (handle) 26 IN 27 Array of info structures (array of handles) 28 IN ninfo

IN

cbfunc

29 30

31

Callback function **pmix_op_cbfunc_t** (function reference)

Number of elements in the *info* array (integer)

1 2	IN cbdata Data to be passed to the callback function (memory reference)		
3	Returns one of the following:		
4 5 6	• PMIX_SUCCESS, indicating that the request is being processed by the host environment - result will be returned in the provided <i>cbfunc</i> . Note that the host must not invoke the callback function prior to returning from the API.		
7 8	• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will not be called		
9 10 11	• PMIX_ERR_NOT_SUPPORTED , indicating that the host environment does not support the request, even though the function entry was provided in the server module - the <i>cbfunc</i> will not be called		
12 13	• a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will not be called		
	▼		
14 15	PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed <i>info</i> array:		
16 17	<pre>PMIX_USERID "pmix.euid" (uint32_t)</pre>		
18 19	PMIX_GRPID "pmix.egid" (uint32_t) Effective group id.		
20			
21	Host environments that implement this entry point are required to support the following attributes:		
22 23	<pre>PMIX_RANGE "pmix.range" (pmix_data_range_t) Value for calls to publish/lookup/unpublish or for monitoring event notifications.</pre>		
24 25	<pre>PMIX_PERSISTENCE "pmix.persist" (pmix_persistence_t) Value for calls to PMIx_Publish .</pre>		
	▼ Optional Attributes		
26	The following attributes are optional for host environments that support this operation:		
27 28 29 30	PMIX_TIMEOUT "pmix.timeout" (int) Time in seconds before the specified operation should time out (0 indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.		

Publish data per the PMIx_Publish specification. The callback is to be executed upon completion of the operation. The default data range is left to the host environment, but expected to be PMIX_RANGE_SESSION, and the default persistence PMIX_PERSIST_SESSION or their equivalent. These values can be specified by including the respective attributed in the *info* array.

The persistence indicates how long the server should retain the data.

Advice to PMIx server hosts -

The host environment is not required to guarantee support for any specific range - i.e., the environment does not need to return an error if the data store doesn't support a specified range so long as it is covered by some internally defined range. However, the server must return an error (a) if the key is duplicative within the storage range, and (b) if the server does not allow overwriting of published info by the original publisher - it is left to the discretion of the host environment to allow info-key-based flags to modify this behavior.

The **PMIX_USERID** and **PMIX_GRPID** of the publishing process will be provided to support authorization-based access to published information and must be returned on any subsequent lookup request.

5 11.2.8 pmix_server_lookup_fn_t

Summary

Lookup published data.

Format

```
PMIx v1.0
```

1

3

5

6

7

8 9

10

11

12

13 14

15

17

18

1	IN proc			
2	<pre>pmix_proc_t structure of the process seeking the data (handle)</pre>			
3	IN keys			
4	(array of strings)			
5	IN info			
6	Array of info structures (array of handles)			
7	IN ninfo			
8	Number of elements in the <i>info</i> array (integer)			
9	IN obfunc			
0	Callback function pmix_lookup_cbfunc_t (function reference) IN cbdata			
1 2	Data to be passed to the callback function (memory reference)			
3	Returns one of the following:			
4	• PMIX_SUCCESS, indicating that the request is being processed by the host environment - result			
5	will be returned in the provided <i>cbfunc</i> . Note that the host must not invoke the callback function			
6	prior to returning from the API.			
7	• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and			
8	returned <i>success</i> - the <i>cbfunc</i> will not be called			
9	·			
20	 PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the cbfunc will not 			
.o !1	be called			
.2 .3	• a PMIx error constant indicating either an error in the input or that the request was immediately			
.3	processed and failed - the <i>cbfunc</i> will not be called			
	Required Attributes			
24	PMIx libraries are required to pass any provided attributes to the host environment for processing.			
25	In addition, the following attributes are required to be included in the passed <i>info</i> array:			
26	<pre>PMIX_USERID "pmix.euid" (uint32_t)</pre>			
.0 ?7	Effective user id.			
28	PMIX_GRPID "pmix.egid" (uint32_t)			
.9	Effective group id.			
80				
11	Host environments that implement this entry point are required to support the following attributes:			
32	<pre>PMIX_RANGE "pmix.range" (pmix_data_range_t)</pre>			
3	Value for calls to publish/lookup/unpublish or for monitoring event notifications.			
34	PMIX_WAIT "pmix.wait" (int)			

Caller requests that the PMIx server wait until at least the specified number of values are 1 2 found (0 indicates all and is the default). Optional Attributes 3 The following attributes are optional for host environments that support this operation: 4 PMIX_TIMEOUT "pmix.timeout" (int) 5 Time in seconds before the specified operation should time out (0 indicating infinite) in 6 error. The timeout parameter can help avoid "hangs" due to programming errors that prevent 7 the target process from ever exposing its data. **Description** 8 9 Lookup published data. The host server will be passed a **NULL**-terminated array of string keys 10 identifying the data being requested. 11 The array of *info* structs is used to pass user-requested options to the server. The default data range 12 is left to the host environment, but expected to be PMIX RANGE SESSION. This can include a 13 wait flag to indicate that the server should wait for all data to become available before executing the 14 callback function, or should immediately callback with whatever data is available. In addition, a 15 timeout can be specified on the wait to preclude an indefinite wait for data that may never be published. 16 Advice to PMIx server hosts — 17 The PMIX USERID and PMIX GRPID of the requesting process will be provided to support authorization-based access to published information. The host environment is not required to 18 guarantee support for any specific range - i.e., the environment does not need to return an error if 19 20 the data store doesn't support a specified range so long as it is covered by some internally defined 21 range.

11.2.9 pmix_server_unpublish_fn_t

Summary

Delete data from the data store.

23

1		Format			
PMIx v1.0					
2		<pre>typedef pmix_status_t (*pmix_server_unpublish_fn_t)(</pre>			
3		<pre>const pmix_proc_t *proc,</pre>			
4		char **keys,			
5		<pre>const pmix_info_t info[],</pre>			
6		size_t ninfo,			
7		<pre>pmix_op_cbfunc_t cbfunc,</pre>			
8		void *cbdata)			
		C			
9		IN proc			
10		<pre>pmix_proc_t structure identifying the process making the request (handle)</pre>			
11		IN keys			
12		(array of strings)			
13		IN info			
14		Array of info structures (array of handles)			
15		IN ninfo			
16		Number of elements in the <i>info</i> array (integer)			
17		IN cbfunc			
18		Callback function pmix_op_cbfunc_t (function reference)			
19		IN cbdata			
20		Data to be passed to the callback function (memory reference)			
21		Returns one of the following:			
22 23 24		• PMIX_SUCCESS, indicating that the request is being processed by the host environment - result will be returned in the provided <i>cbfunc</i> . Note that the host must not invoke the callback function prior to returning from the API.			
25 26		• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will not be called			
27 28 29		• PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the <i>cbfunc</i> will not be called			
30 31		• a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will not be called			
		▼			
32 33		PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed <i>info</i> array:			
34 35		PMIX_USERID "pmix.euid" (uint32_t) Effective user id.			

1 2	PMIX_GRPID "pmix.egid" (uint32_t) Effective group id.
3	
4	Host environments that implement this entry point are required to support the following attributes:
5 6	PMIX_RANGE "pmix.range" (pmix_data_range_t) Value for calls to publish/lookup/unpublish or for monitoring event notifications.
	▼ Optional Attributes
7	The following attributes are optional for host environments that support this operation:
8 9 10 11	PMIX_TIMEOUT "pmix.timeout" (int) Time in seconds before the specified operation should time out (0 indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.
12 13 14 15	Description Delete data from the data store. The host server will be passed a NULL -terminated array of string keys, plus potential directives such as the data range within which the keys should be deleted. The default data range is left to the host environment, but expected to be PMIX_RANGE_SESSION . The callback is to be executed upon completion of the delete procedure.
	Advice to PMIx server hosts
17 18 19 20 21	The PMIX_USERID and PMIX_GRPID of the requesting process will be provided to support authorization-based access to published information. The host environment is not required to guarantee support for any specific range - i.e., the environment does not need to return an error if the data store doesn't support a specified range so long as it is covered by some internally defined range.

22 11.2.10 pmix_server_spawn_fn_t

Summary

Spawn a set of applications/processes as per the PMIx_Spawn API.

23

DIA	romat
PMIx	
2	<pre>typedef pmix_status_t (*pmix_server_spawn_fn_t)(</pre>
3	<pre>const pmix_proc_t *proc,</pre>
4	<pre>const pmix_info_t job_info[],</pre>
5	size_t ninfo,
6	<pre>const pmix_app_t apps[],</pre>
7	size_t napps,
8	<pre>pmix_spawn_cbfunc_t cbfunc,</pre>
9	<pre>void *cbdata)</pre>
	^ C
0	IN proc
1	<pre>pmix_proc_t structure of the process making the request (handle)</pre>
2	IN job_info
3	Array of info structures (array of handles)
4	IN ninfo
5	Number of elements in the <i>jobinfo</i> array (integer)
6	IN apps
7	Array of pmix_app_t structures (array of handles)
8	IN napps
9	Number of elements in the <i>apps</i> array (integer)
20	IN cbfunc
.0 ?1	
	Callback function <pre>pmix_spawn_cbfunc_t</pre> (function reference) IN cbdata
22	
23	Data to be passed to the callback function (memory reference)
24	Returns one of the following:
25	• PMIX_SUCCESS, indicating that the request is being processed by the host environment - result
26	will be returned in the provided <i>cbfunc</i> . Note that the host must not invoke the callback function
27	prior to returning from the API.
28	• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
29	returned <i>success</i> - the <i>cbfunc</i> will not be called
80	• PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the
31	request, even though the function entry was provided in the server module - the <i>cbfunc</i> will not
32	be called
33	• a PMIx error constant indicating either an error in the input or that the request was immediately
34	processed and failed - the <i>cbfunc</i> will not be called

Required Attributes PMIx libraries are required to pass any provided attributes to the host environment for processing. 1 2 In addition, the following attributes are required to be included in the passed *info* array: 3 PMIX USERID "pmix.euid" (uint32 t) Effective user id. 4 PMIX GRPID "pmix.egid" (uint32_t) 5 6 Effective group id. 7 8 Host environments that provide this module entry point are required to pass the PMIX SPAWNED 9 and PMIX_PARENT_ID attributes to all PMIx servers launching new child processes so those 10 values can be returned to clients upon connection to the PMIx server. In addition, they are required 11 to support the following attributes when present in either the job info or the info array of an element of the apps array: 12 PMIX WDIR "pmix.wdir" (char*) 13 Working directory for spawned processes. 14 15 PMIX SET SESSION CWD "pmix.ssncwd" (bool) Set the application's current working directory to the session working directory assigned by 16 the RM - when accessed using PMIx_Get , use the PMIX_RANK_WILDCARD value for 17 18 the rank to discover the session working directory assigned to the provided namespace 19 PMIX PREFIX "pmix.prefix" (char*) 20 Prefix to use for starting spawned processes. 21 PMIX HOST "pmix.host" (char*) 22 Comma-delimited list of hosts to use for spawned processes. 23 PMIX_HOSTFILE "pmix.hostfile" (char*) 24 Hostfile to use for spawned processes. _____ Optional Attributes 25 The following attributes are optional for host environments that support this operation: 26 PMIX_ADD_HOSTFILE "pmix.addhostfile" (char*) Hostfile listing hosts to add to existing allocation. 27 28 PMIX ADD HOST "pmix.addhost" (char*) Comma-delimited list of hosts to add to the allocation. 29 30 PMIX PRELOAD BIN "pmix.preloadbin" (bool) 31 Preload binaries onto nodes. 32 PMIX PRELOAD FILES "pmix.preloadfiles" (char*)

```
Comma-delimited list of files to pre-position on nodes.
 1
              PMIX PERSONALITY "pmix.pers" (char*)
 2
                    Name of personality to use.
 3
              PMIX MAPPER "pmix.mapper" (char*)
 4
5
                    Mapping mechanism to use for placing spawned processes - when accessed using
                    PMIx Get, use the PMIX RANK WILDCARD value for the rank to discover the mapping
6
                    mechanism used for the provided namespace.
7
              PMIX_DISPLAY_MAP "pmix.dispmap" (bool)
8
9
                    Display process mapping upon spawn.
10
              PMIX PPR "pmix.ppr" (char*)
11
                    Number of processes to spawn on each identified resource.
              PMIX_MAPBY "pmix.mapby" (char*)
12
13
                    Process mapping policy - when accessed using PMIx_Get, use the
                    PMIX_RANK_WILDCARD value for the rank to discover the mapping policy used for the
14
                    provided namespace
15
16
              PMIX_RANKBY "pmix.rankby" (char*)
17
                    Process ranking policy - when accessed using PMIx Get, use the
18
                    PMIX_RANK_WILDCARD value for the rank to discover the ranking algorithm used for the
                    provided namespace
19
20
              PMIX_BINDTO "pmix.bindto" (char*)
21
                    Process binding policy - when accessed using PMIx Get, use the
                    PMIX RANK WILDCARD value for the rank to discover the binding policy used for the
22
23
                    provided namespace
              PMIX_NON_PMI "pmix.nonpmi" (bool)
24
25
                    Spawned processes will not call PMIx Init.
              PMIX STDIN TGT "pmix.stdin" (uint32 t)
26
27
                    Spawned process rank that is to receive any forwarded stdin.
28
              PMIX_FWD_STDIN "pmix.fwd.stdin" (bool)
29
                    Forward stdin from this process to the designated process.
30
              PMIX FWD STDOUT "pmix.fwd.stdout" (bool)
                    Forward stdout from spawned processes to this process.
31
32
              PMIX FWD STDERR "pmix.fwd.stderr" (bool)
33
                    Forward stderr from spawned processes to this process.
              PMIX DEBUGGER_DAEMONS "pmix.debugger" (bool)
34
                    Spawned application consists of debugger daemons.
35
36
              PMIX TAG OUTPUT "pmix.tagout" (bool)
```

1	Tag application output with the identity of the source process.
2 3	PMIX_TIMESTAMP_OUTPUT "pmix.tsout" (bool) Timestamp output from applications.
4 5	PMIX_MERGE_STDERR_STDOUT "pmix.mergeerrout" (bool) Merge stdout and stderr streams from application processes.
6 7	<pre>PMIX_OUTPUT_TO_FILE "pmix.outfile" (char*) Direct application output (both stdout and stderr) into files of form "<filename>.rank"</filename></pre>
8 9	PMIX_INDEX_ARGV "pmix.indxargv" (bool) Mark the argv with the rank of the process.
10 11 12 13	<pre>PMIX_CPUS_PER_PROC "pmix.cpuperproc" (uint32_t) Number of cpus to assign to each rank - when accessed using PMIx_Get , use the PMIX_RANK_WILDCARD value for the rank to discover the cpus/process assigned to the provided namespace</pre>
14 15	<pre>PMIX_NO_PROCS_ON_HEAD "pmix.nolocal" (bool) Do not place processes on the head node.</pre>
16 17	<pre>PMIX_NO_OVERSUBSCRIBE "pmix.noover" (bool) Do not oversubscribe the cpus.</pre>
18 19	PMIX_REPORT_BINDINGS "pmix.repbind" (bool) Report bindings of the individual processes.
20 21 22 23	<pre>PMIX_CPU_LIST "pmix.cpulist" (char*) List of cpus to use for this job - when accessed using PMIx_Get , use the PMIX_RANK_WILDCARD value for the rank to discover the cpu list used for the provided namespace</pre>
24 25	PMIX_JOB_RECOVERABLE "pmix.recover" (bool) Application supports recoverable operations.
26 27	PMIX_JOB_CONTINUOUS "pmix.continuous" (bool) Application is continuous, all failed processes should be immediately restarted.
28 29 30 31	<pre>PMIX_MAX_RESTARTS "pmix.maxrestarts" (uint32_t) Maximum number of times to restart a job - when accessed using PMIx_Get , use the PMIX_RANK_WILDCARD value for the rank to discover the max restarts for the provided namespace</pre>
32 33 34 35	PMIX_TIMEOUT "pmix.timeout" (int) Time in seconds before the specified operation should time out (0 indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Spawn a set of applications/processes as per the PMIx_Spawn API. Note that applications are not required to be MPI or any other programming model. Thus, the host server cannot make any assumptions as to their required support. The callback function is to be executed once all processes have been started. An error in starting any application or process in this request shall cause all applications and processes in the request to be terminated, and an error returned to the originating caller.

Note that a timeout can be specified in the job_info array to indicate that failure to start the requested job within the given time should result in termination to avoid hangs.

11.2.11 pmix_server_connect_fn_t

Summary

Record the specified processes as *connected*.

```
Format
```

```
14
15
```

PMIx v1.0

1

2

3

5

6

7

8

9

11 12

13

16

17

18 19

20

21

22

23

24

25

26

27

28 29

30 31

32

33

34

35

36

erver connect for the

IN procs

Array of pmix proc t structures identifying participants (array of handles)

IN nprocs

Number of elements in the *procs* array (integer)

IN info

Array of info structures (array of handles)

IN ninfo

Number of elements in the *info* array (integer)

IN cbfunc

Callback function **pmix_op_cbfunc_t** (function reference)

IN cbdata

Data to be passed to the callback function (memory reference)

Returns one of the following:

• PMIX_SUCCESS, indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function prior to returning from the API.

1 2	• PMIX_OPERATION_SUCCEEDED , indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will not be called
3 4 5	• PMIX_ERR_NOT_SUPPORTED , indicating that the host environment does not support the request, even though the function entry was provided in the server module - the <i>cbfunc</i> will not be called
6 7	• a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will not be called
	▼ Required Attributes
8	PMIx libraries are required to pass any provided attributes to the host environment for processing.
	▼ Optional Attributes
9	The following attributes are optional for host environments that support this operation:
0 1 2 3	PMIX_TIMEOUT "pmix.timeout" (int) Time in seconds before the specified operation should time out (0 indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.
4 5 6 7 8	PMIX_COLLECTIVE_ALGO "pmix.calgo" (char*) Comma-delimited list of algorithms to use for the collective operation. PMIx does not impose any requirements on a host environment's collective algorithms. Thus, the acceptable values for this attribute will be environment-dependent - users are encouraged to check their host environment for supported values.
9 0	<pre>PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool) If true, indicates that the requested choice of algorithm is mandatory.</pre>

2

3

4

5

6 7

8

9

10

13

Record the processes specified by the *procs* array as *connected* as per the PMIx definition. The callback is to be executed once every daemon hosting at least one participant has called the host server's **pmix_server_connect_fn_t** function, and the host environment has completed any supporting operations required to meet the terms of the PMIx definition of *connected* processes.

- Advice to PMIx library implementers -

The PMIx server library is required to aggregate participation by local clients, passing the request to the host environment once all local participants have executed the API.

Advice to PMIx server hosts –

The host will receive a single call for each collective operation. It is the responsibility of the host to identify the nodes containing participating processes, execute the collective across all participating nodes, and notify the local PMIx server library upon completion of the global collective.

11 11.2.12 pmix_server_disconnect_fn_t

12 Summary

Disconnect a previously connected set of processes.

1		Format
2 3 4 5 6 7	Ix v1.0	<pre>typedef pmix_status_t (*pmix_server_disconnect_fn_t)(</pre>
8		void *cbdata)
9 10 11 12 13 14 15 16 17 18 19		IN procs Array of pmix_proc_t structures identifying participants (array of handles) IN nprocs Number of elements in the procs array (integer) IN info Array of info structures (array of handles) IN ninfo Number of elements in the info array (integer) IN cbfunc Callback function pmix_op_cbfunc_t (function reference) IN cbdata Data to be passed to the callback function (memory reference)
21		Returns one of the following:
22 23 24		• PMIX_SUCCESS, indicating that the request is being processed by the host environment - result will be returned in the provided <i>cbfunc</i> . Note that the host must not invoke the callback function prior to returning from the API.
25 26		• PMIX_OPERATION_SUCCEEDED , indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will not be called
27 28 29		• PMIX_ERR_NOT_SUPPORTED , indicating that the host environment does not support the request, even though the function entry was provided in the server module - the <i>cbfunc</i> will not be called
30 31		• a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will not be called
		▼ Required Attributes
32		PMIx libraries are required to pass any provided attributes to the host environment for processing.

	→ Optional Attributes
1	The following attributes are optional for host environments that support this operation:
2 3 4 5	PMIX_TIMEOUT "pmix.timeout" (int) Time in seconds before the specified operation should time out (0 indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.
6	Description
7	Disconnect a previously connected set of processes. The callback is to be executed once every
8	daemon hosting at least one participant has called the host server's has called the
9	<pre>pmix_server_disconnect_fn_t function, and the host environment has completed any</pre>
10	required supporting operations.
	Advice to PMIx library implementers ————————————————————————————————————
11	The PMIx server library is required to aggregate participation by local clients, passing the request
12	to the host environment once all local participants have executed the API.
	Advice to PMIx server hosts
13	The host will receive a single call for each collective operation. It is the responsibility of the host to
14	identify the nodes containing participating processes, execute the collective across all participating
15	nodes, and notify the local PMIx server library upon completion of the global collective.
16	A PMIX_ERR_INVALID_OPERATION error must be returned if the specified set of <i>procs</i> was
17	not previously <i>connected</i> via a call to the pmix_server_connect_fn_t function.

18 11.2.13 pmix_server_register_events_fn_t

Summary

19 20

Register to receive notifications for the specified events.

1		Format
	<i>PMIx v1.0</i>	
2		<pre>typedef pmix_status_t (*pmix_server_register_events_fn_t)(</pre>
3		<pre>pmix_status_t *codes,</pre>
4		size_t ncodes,
5		<pre>const pmix_info_t info[],</pre>
6		size_t ninfo,
7		<pre>pmix_op_cbfunc_t cbfunc,</pre>
8		<pre>void *cbdata)</pre>
		C
9		IN codes
10		Array of pmix_status_t values (array of handles)
11		IN ncodes
12		Number of elements in the <i>codes</i> array (integer)
13		IN info
14		Array of info structures (array of handles)
15		IN ninfo
16		Number of elements in the <i>info</i> array (integer)
17		IN cbfunc
18		Callback function <pre>pmix_op_cbfunc_t</pre> (function reference)
19		IN cbdata
20		Data to be passed to the callback function (memory reference)
21		Returns one of the following:
22 23 24		• PMIX_SUCCESS, indicating that the request is being processed by the host environment - result will be returned in the provided <i>cbfunc</i> . Note that the host must not invoke the callback function prior to returning from the API.
25 26		• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will not be called
27 28 29		• PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the <i>cbfunc</i> will not be called
30 31		• a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will not be called
		▼
32 33		PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed <i>info</i> array:
34 35		PMIX_USERID "pmix.euid" (uint32_t) Effective user id.

1 2	PMIX_GRPID "pmix.egid" (uint32_t) Effective group id.
3 4 5	Description Register to receive notifications for the specified status codes. The <i>info</i> array included in this API is reserved for possible future directives to further steer notification. Advice to PMIx library implementers
6 7	The PMIx server library must track all client registrations for subsequent notification. This module function shall only be called when:
8 9 10	 the client has requested notification of an environmental code (i.e., a PMIx code in the range beyond PMIX_ERR_SYS_OTHER) or a code that lies outside the defined PMIx range of constants; and
l1 l2	• the PMIx server library has not previously requested notification of that code - i.e., the host environment is to be contacted only once a given unique code value
	Advice to PMIx server hosts
13 14 15 16	The host environment is required to pass to its PMIx server library all non-environmental events that directly relate to a registered namespace without the PMIx server library explicitly requesting them. Environmental events are to be translated to their nearest PMIx equivalent code as defined in the range between PMIX_ERR_SYS_BASE and PMIX_ERR_SYS_OTHER (inclusive).

17 11.2.14 pmix_server_deregister_events_fn_t

Summary

18

19

Deregister to receive notifications for the specified events.

1		Format
2 3 4 5 6	PMIx v1.0	<pre>typedef pmix_status_t (*pmix_server_deregister_events_fn_t)(</pre>
		C —
7		IN codes
8		Array of pmix_status_t values (array of handles)
9		IN ncodes
10		Number of elements in the <i>codes</i> array (integer)
1 2		IN cbfunc Callback function pmix_op_cbfunc_t (function reference)
13		IN cbdata
14		Data to be passed to the callback function (memory reference)
15		Returns one of the following:
6 7 8		• PMIX_SUCCESS , indicating that the request is being processed by the host environment - result will be returned in the provided <i>cbfunc</i> . Note that the host must not invoke the callback function prior to returning from the API.
19		• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will not be called
21 22 23		• PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the <i>cbfunc</i> will not be called
24 25		• a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will not be called
26 27 28		Description Deregister to receive notifications for the specified events to which the PMIx server has previously registered.
		Advice to PMIx library implementers —
<u>29</u> 30		The PMIx server library must track all client registrations. This module function shall only be called when:
31 32 33		 the library is deregistering environmental codes (i.e., a PMIx codes in the range between PMIX_ERR_SYS_BASE and PMIX_ERR_SYS_OTHER, inclusive) or codes that lies outside the defined PMIx range of constants; and

1 • no client (including the server library itself) remains registered for notifications on any included code - i.e., a code should be included in this call only when no registered notifications against it 2 remain. 3 11.2.15 pmix_server_notify_event_fn_t Summary 5 Notify the specified processes of an event. 6 **Format** PMIx v2.08 typedef pmix_status_t (*pmix_server_notify_event_fn_t)(pmix_status_t code, const pmix_proc_t *source, 9 10 pmix_data_range_t range, 11 pmix_info_t info[], 12 size_t ninfo, pmix_op_cbfunc_t cbfunc, 13 void *cbdata); 14 IN 15 code The **pmix_status_t** event code being referenced structure (handle) 16 IN 17 18 pmix proc t of process that generated the event (handle) 19 IN range 20 pmix data range t range over which the event is to be distributed (handle) info 21 IN Optional array of pmix_info_t structures containing additional information on the event 22 (array of handles) 23 IN ninfo 24 Number of elements in the *info* array (integer) 25 26 IN Callback function pmix_op_cbfunc_t (function reference) 27 cbdata 28 IN Data to be passed to the callback function (memory reference) 29 30 Returns one of the following: • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result 31 32 will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function 33 prior to returning from the API.

34

35

• PMIX OPERATION SUCCEEDED, indicating that the request was immediately processed and

returned success - the cbfunc will not be called

• PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the 1 2 request, even though the function entry was provided in the server module - the cbfunc will not be called 3 4 • a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the cbfunc will not be called 5 -----Required Attributes PMIx libraries are required to pass any provided attributes to the host environment for processing. 6 Host environments that provide this module entry point are required to support the following 8 attributes: 9 10 PMIX_RANGE "pmix.range" (pmix_data_range_t) Value for calls to publish/lookup/unpublish or for monitoring event notifications. 11 12 Description 13 Notify the specified processes (described through a combination of range and attributes provided in the *info* array) of an event generated either by the PMIx server itself or by one of its local clients. 14 The process generating the event is provided in the *source* parameter, and any further descriptive 15 16 information is included in the info array. Advice to PMIx server hosts ——— 17 The callback function is to be executed once the host environment no longer requires that the PMIx 18 server library maintain the provided data structures. It does not necessarily indicate that the event 19 has been delivered to any process, nor that the event has been distributed for delivery

11.2.16 pmix_server_listener_fn_t

Summary

Register a socket the host server can monitor for connection requests.

20

21

```
Format
1
   PMIx v1.0
2
               typedef pmix_status_t (*pmix_server_listener_fn_t)(
 3
                                                     int listening sd,
                                                     pmix connection cbfunc t cbfunc,
 4
                                                     void *cbdata)
5
               IN
6
                    incoming sd
 7
                    (integer)
8
               IN
                    cbfunc
9
                    Callback function pmix connection cbfunc t (function reference)
               IN
                    cbdata
10
                    (memory reference)
11
12
               Returns PMIX_SUCCESS indicating that the request is accepted, or a negative value
               corresponding to a PMIx error constant indicating that the request has been rejected.
13
               Description
14
15
               Register a socket the host environment can monitor for connection requests, harvest them, and then
16
               call the PMIx server library's internal callback function for further processing. A listener thread is
17
               essential to efficiently harvesting connection requests from large numbers of local clients such as
               occur when running on large SMPs. The host server listener is required to call accept on the
18
19
               incoming connection request, and then pass the resulting socket to the provided cbfunc. A NULL
20
               for this function will cause the internal PMIx server to spawn its own listener thread.
    11.2.17
                 pmix_server_query_fn_t
22
               Summary
               Query information from the resource manager.
23
               Format
24
   PMIx v2.0
25
               typedef pmix_status_t (*pmix_server_query_fn_t)(
26
                                                     pmix_proc_t *proct,
27
                                                     pmix query t *queries, size t nqueries,
28
                                                     pmix info cbfunc t cbfunc,
                                                     void *cbdata)
29
               IN
30
                    proct
31
                    pmix_proc_t structure of the requesting process (handle)
               IN
32
                    queries
```

Array of pmix query t structures (array of handles)

1 2 3 4 5 6	 IN nqueries Number of elements in the queries array (integer) IN cbfunc Callback function pmix_info_cbfunc_t (function reference) IN cbdata Data to be passed to the callback function (memory reference)
7	Returns one of the following:
8 9 10	• PMIX_SUCCESS, indicating that the request is being processed by the host environment - result will be returned in the provided <i>cbfunc</i> . Note that the host must not invoke the callback function prior to returning from the API.
11 12	• PMIX_OPERATION_SUCCEEDED , indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will not be called
13 14 15	• PMIX_ERR_NOT_SUPPORTED , indicating that the host environment does not support the request, even though the function entry was provided in the server module - the <i>cbfunc</i> will not be called
16 17	• a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will not be called
	▼
18 19	PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed <i>info</i> array:
20 21	<pre>PMIX_USERID "pmix.euid" (uint32_t)</pre>
22 23	PMIX_GRPID "pmix.egid" (uint32_t) Effective group id.
	▼Optional Attributes
24	The following attributes are optional for host environments that support this operation:
25 26	PMIX_QUERY_NAMESPACES "pmix.qry.ns" (char*) Request a comma-delimited list of active namespaces. NO QUALIFIERS
27 28 29	PMIX_QUERY_JOB_STATUS "pmix.qry.jst" (pmix_status_t) Status of a specified, currently executing job. REQUIRED QUALIFIER: PMIX_NSPACE indicating the namespace whose status is being queried
30 31	<pre>PMIX_QUERY_QUEUE_LIST "pmix.qry.qlst" (char*) Request a comma-delimited list of scheduler queues. NO QUALIFIERS</pre>
32	<pre>PMIX_QUERY_QUEUE_STATUS "pmix.qry.qst" (TBD)</pre>

1 2	Status of a specified scheduler queue. SUPPORTED QUALIFIERS: PMIX_ALLOC_QUEUE naming specific queue whose status is being requested
3 4 5 6	<pre>PMIX_QUERY_PROC_TABLE "pmix.qry.ptable" (char*) Input namespace of the job whose information is being requested returns (pmix_data_array_t) an array of pmix_proc_info_t. REQUIRED QUALIFIER: pMIX_NSPACE indicating the namespace whose process table is being queried</pre>
7 8 9 10	PMIX_QUERY_LOCAL_PROC_TABLE "pmix.qry.lptable" (char*) Input namespace of the job whose information is being requested returns (pmix_data_array_t) an array of pmix_proc_info_t for processes in job on same node. REQUIRED QUALIFIER: PMIX_NSPACE indicating the namespace whose process table is being queried
12 13	PMIX_QUERY_SPAWN_SUPPORT "pmix.qry.spawn" (bool) Return a comma-delimited list of supported spawn attributes. NO QUALIFIERS
14 15	PMIX_QUERY_DEBUG_SUPPORT "pmix.qry.debug" (bool) Return a comma-delimited list of supported debug attributes. NO QUALIFIERS
16 17 18 19	PMIX_QUERY_MEMORY_USAGE "pmix.qry.mem" (bool) Return information on memory usage for the processes indicated in the qualifiers. SUPPORTED QUALIFIERS: PMIX_NSPACE and PMIX_RANK, or PMIX_PROCID of specific process(es) whose memory usage is being requested
20 21	PMIX_QUERY_LOCAL_ONLY "pmix.qry.local" (bool) Constrain the query to local information only. NO QUALIFIERS
22 23	PMIX_QUERY_REPORT_AVG "pmix.qry.avg" (bool) Report only average values for sampled information. NO QUALIFIERS
24 25	PMIX_QUERY_REPORT_MINMAX "pmix.qry.minmax" (bool) Report minimum and maximum values. NO QUALIFIERS
26 27	<pre>PMIX_QUERY_ALLOC_STATUS "pmix.query.alloc" (char*) String identifier of the allocation whose status is being requested. NO QUALIFIERS</pre>
28 29 30 31	PMIX_TIME_REMAINING "pmix.time.remaining" (char*) Query number of seconds (uint32_t) remaining in allocation for the specified namespace. SUPPORTED QUALIFIERS: PMIX_NSPACE of the namespace whose info is being requested (defaults to allocation containing the caller)

```
Description
1
2
               Query information from the host environment. The query will include the namespace/rank of the
               process that is requesting the info, an array of pmix query t describing the request, and a
 3
 4
               callback function/data for the return.
                        ———— Advice to PMIx library implementers ——
5
               The PMIx server library should not block in this function as the host environment may, depending
6
               upon the information being requested, require significant time to respond.
                pmix server tool connection fn t
               Summary
8
9
               Register that a tool has connected to the server.
               Format
10
   PMIx v2.0
               typedef void (*pmix_server_tool_connection_fn_t)(
11
12
                                                    pmix_info_t info[], size_t ninfo,
13
                                                    pmix_tool_connection_cbfunc_t cbfunc,
                                                    void *cbdata)
14
               IN
                    info
15
                   Array of pmix_info_t structures (array of handles)
16
17
               IN
                  ninfo
                   Number of elements in the info array (integer)
18
               IN
                  cbfunc
19
                   Callback function pmix_tool_connection_cbfunc_t (function reference)
20
21
               IN
                  cbdata
22
                   Data to be passed to the callback function (memory reference)
                                              Required Attributes
23
               PMIx libraries are required to pass the following attributes in the info array:
24
               PMIX USERID "pmix.euid" (uint32 t)
                    Effective user id.
25
26
               PMIX_GRPID "pmix.egid" (uint32_t)
27
                    Effective group id.
```

Optional Attributes The following attributes are optional for host environments that support this operation: 1 2 PMIX_FWD_STDOUT "pmix.fwd.stdout" (bool) 3 Forward **stdout** from spawned processes to this process. 4 PMIX_FWD_STDERR "pmix.fwd.stderr" (bool) Forward **stderr** from spawned processes to this process. 5 6 PMIX_FWD_STDIN "pmix.fwd.stdin" (bool) 7 Forward **stdin** from this process to the designated process. Description 8 9 Register that a tool has connected to the server, and request that the tool be assigned a 10 namespace/rank identifier for further interactions. The pmix_info_t array is used to pass qualifiers for the connection request, including the effective uid and gid of the calling tool for 11 12 authentication purposes. Advice to PMIx server hosts -The host environment is solely responsible for authenticating and authorizing the connection, and 13 14 for authorizing all subsequent tool requests. The host must not execute the callback function prior to returning from the API. 15

16 11.2.19 pmix_server_log_fn_t

17 Summary

18

Log data on behalf of a client.

```
Format
1
   PMIx v2.0
              typedef void (*pmix_server_log_fn_t)(
2
3
                                                   const pmix_proc_t *client,
4
                                                   const pmix info t data[], size t ndata,
                                                   const pmix_info_t directives[], size_t ndirs,
5
6
                                                   pmix op cbfunc t cbfunc, void *cbdata)
              IN
                   client
7
8
                   pmix_proc_t structure (handle)
9
              IN
                   data
                   Array of info structures (array of handles)
10
11
12
                   Number of elements in the data array (integer)
13
                   directives
14
                   Array of info structures (array of handles)
                  ndirs
15
              IN
                   Number of elements in the directives array (integer)
16
17
              IN
                  cbfunc
18
                   Callback function pmix_op_cbfunc_t (function reference)
19
              IN cbdata
20
                   Data to be passed to the callback function (memory reference)
                                              Required Attributes
21
              PMIx libraries are required to pass any provided attributes to the host environment for processing.
22
              In addition, the following attributes are required to be included in the passed info array:
23
              PMIX USERID "pmix.euid" (uint32 t)
24
                    Effective user id.
25
              PMIX_GRPID "pmix.egid" (uint32_t)
                    Effective group id.
26
27
28
              Host environments that provide this module entry point are required to support the following
29
              attributes:
              PMIX LOG STDERR "pmix.log.stderr" (char*)
30
31
                    Log string to stderr.
32
              PMIX LOG STDOUT "pmix.log.stdout" (char*)
33
                    Log string to stdout.
34
              PMIX LOG SYSLOG "pmix.log.syslog" (char*)
```

Log data to syslog. Defaults to **ERROR** priority. Will log to global syslog if available, 1 2 otherwise to local syslog Optional Attributes 3 The following attributes are optional for host environments that support this operation: 4 PMIX LOG MSG "pmix.log.msg" (pmix byte object t) Message blob to be sent somewhere. 5 PMIX_LOG_EMAIL "pmix.log.email" (pmix_data_array_t) 6 7 Log via email based on **pmix_info_t** containing directives. 8 PMIX_LOG_EMAIL_ADDR "pmix.log.emaddr" (char*) 9 Comma-delimited list of email addresses that are to receive the message. PMIX_LOG_EMAIL_SUBJECT "pmix.log.emsub" (char*) 10 11 Subject line for email. 12 PMIX_LOG_EMAIL_MSG "pmix.log.emmsg" (char*) Message to be included in email. 13 14 Description Log data on behalf of a client. This function is not intended for output of computational results, but 15 16 rather for reporting status and error messages. The host must not execute the callback function prior to returning from the API. 17 11.2.20 pmix_server_alloc_fn_t 18 19 Summary 20 Request allocation operations on behalf of a client.

```
Format
 1
   PMIx v2.0
 2
               typedef pmix_status_t (*pmix_server_alloc_fn_t)(
 3
                                                      const pmix proc t *client,
 4
                                                      pmix alloc directive t directive,
                                                      const pmix_info_t data[], size_t ndata,
 5
                                                      pmix info cbfunc t cbfunc, void *cbdata)
 6
 7
               IN
                    client
                    pmix proc t structure of process making request (handle)
 8
 9
               IN
                    directive
                    Specific action being requested (pmix alloc directive t)
10
               IN
11
12
                    Array of info structures (array of handles)
13
               IN
                    ndata
14
                    Number of elements in the data array (integer)
                   cbfunc
15
               IN
                    Callback function pmix_info_cbfunc_t (function reference)
16
17
               IN
                    cbdata
18
                    Data to be passed to the callback function (memory reference)
               Returns one of the following:
19
20
               • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result
                  will be returned in the provided cbfunc. Note that the host must not invoke the callback function
21
22
                  prior to returning from the API.
23
               • PMIX OPERATION SUCCEEDED, indicating that the request was immediately processed and
24
                  returned success - the cbfunc will not be called
25
               • PMIX ERR NOT SUPPORTED, indicating that the host environment does not support the
                  request, even though the function entry was provided in the server module - the cbfunc will not
26
27
                  be called
28
               • a PMIx error constant indicating either an error in the input or that the request was immediately
                  processed and failed - the cbfunc will not be called
29
                                                Required Attributes
               PMIx libraries are required to pass any provided attributes to the host environment for processing.
30
               In addition, the following attributes are required to be included in the passed info array:
31
32
               PMIX USERID "pmix.euid" (uint32 t)
                     Effective user id.
33
               PMIX_GRPID "pmix.egid" (uint32_t)
34
                     Effective group id.
35
```

```
1
 2
              Host environments that provide this module entry point are required to support the following
              attributes:
 4
              PMIX ALLOC ID "pmix.alloc.id" (char*)
5
                    A string identifier (provided by the host environment) for the resulting allocation which can
                    later be used to reference the allocated resources in, for example, a call to PMIx Spawn.
6
7
              PMIX ALLOC NUM NODES "pmix.alloc.nnodes" (uint64 t)
                    The number of nodes.
8
9
              PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t)
                    Number of cpus.
10
11
              PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)
                    Time in seconds.
12

    ▼------ Optional Attributes ------

              The following attributes are optional for host environments that support this operation:
13
14
              PMIX ALLOC NODE LIST "pmix.alloc.nlist" (char*)
15
                    Regular expression of the specific nodes.
16
              PMIX_ALLOC_NUM_CPU_LIST "pmix.alloc.ncpulist" (char*)
                    Regular expression of the number of cpus for each node.
17
18
              PMIX ALLOC CPU LIST "pmix.alloc.cpulist" (char*)
                    Regular expression of the specific cpus indicating the cpus involved.
19
20
              PMIX ALLOC MEM SIZE "pmix.alloc.msize" (float)
                    Number of Megabytes.
21
22
              PMIX ALLOC FABRIC "pmix.alloc.net" (array)
                    Array of pmix_info_t describing requested fabric resources. This must include at least:
23
                    PMIX ALLOC FABRIC ID, PMIX ALLOC FABRIC TYPE, and
24
25
                    PMIX ALLOC FABRIC ENDPTS, plus whatever other descriptors are desired.
26
              PMIX ALLOC FABRIC ID "pmix.alloc.netid" (char*)
27
                    The key to be used when accessing this requested fabric allocation. The allocation will be
                    returned/stored as a pmix data array t of pmix info t indexed by this key and
28
29
                    containing at least one entry with the same key and the allocated resource description. The
                    type of the included value depends upon the fabric support. For example, a TCP allocation
30
                    might consist of a comma-delimited string of socket ranges such as
31
                    "32000-32100,33005,38123-38146". Additional entries will consist of any provided
32
                    resource request directives, along with their assigned values. Examples include:
33
34
                    PMIX ALLOC FABRIC TYPE - the type of resources provided;
35
                    PMIX_ALLOC_FABRIC_PLANE - if applicable, what plane the resources were assigned
```

1 from; PMIX_ALLOC_FABRIC_QOS - the assigned QoS; PMIX_ALLOC_BANDWIDTH the allocated bandwidth; PMIX_ALLOC_FABRIC_SEC_KEY - a security key for the 2 requested fabric allocation. NOTE: the assigned values may differ from those requested, 3 4 especially if **PMIX INFO REQD** was not set in the request. 5 PMIX ALLOC BANDWIDTH "pmix.alloc.bw" (float) 6 Mbits/sec. 7 PMIX ALLOC FABRIC QOS "pmix.alloc.netgos" (char*) 8 Quality of service level.

Description

9 10

11

12

13 14

15

16

17 18

19

20

21 22

23

24

26

27

Request new allocation or modifications to an existing allocation on behalf of a client. Several broad categories are envisioned, including the ability to:

- Request allocation of additional resources, including memory, bandwidth, and compute for an
 existing allocation. Any additional allocated resources will be considered as part of the current
 allocation, and thus will be released at the same time.
- Request a new allocation of resources. Note that the new allocation will be disjoint from (i.e., not
 affiliated with) the allocation of the requestor thus the termination of one allocation will not
 impact the other.
- Extend the reservation on currently allocated resources, subject to scheduling availability and priorities.
- Return no-longer-required resources to the scheduler. This includes the *loan* of resources back to the scheduler with a promise to return them upon subsequent request.

The callback function provides a *status* to indicate whether or not the request was granted, and to provide some information as to the reason for any denial in the **pmix_info_cbfunc_t** array of **pmix_info_t** structures.

25 11.2.21 pmix_server_job_control_fn_t

Summary

Execute a job control action on behalf of a client.

1	Format C. ———————————————————————————————————
PMIx v2.0	
2	typedef pmix_status_t (*pmix_server_job_control_fn_t)(
4	<pre>const pmix_proc_t *requestor, const pmix_proc_t targets[], size_t ntargets,</pre>
5	const pmix_proc_t targets[], size_t ntargets, const pmix_info_t directives[], size_t ndirs,
6	pmix_info_cbfunc_t cbfunc, void *cbdata)
	C
7	IN requestor
8	<pre>pmix_proc_t structure of requesting process (handle)</pre>
9	IN targets
10	Array of proc structures (array of handles)
11	IN ntargets
12	Number of elements in the targets array (integer)
13	<pre>IN directives</pre>
14	Array of info structures (array of handles)
15	IN ndirs
16	Number of elements in the <i>info</i> array (integer)
17	IN cbfunc
18	Callback function pmix_op_cbfunc_t (function reference)
19	IN cbdata
20	Data to be passed to the callback function (memory reference)
21	Returns one of the following:
22	• PMIX_SUCCESS, indicating that the request is being processed by the host environment - result
23	will be returned in the provided cbfunc. Note that the host must not invoke the callback function
24	prior to returning from the API.
25	• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
26	returned success - the cbfunc will not be called
27	• PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the
28	request, even though the function entry was provided in the server module - the <i>cbfunc</i> will not
29	be called
30	• a PMIx error constant indicating either an error in the input or that the request was immediately
31	processed and failed - the <i>cbfunc</i> will not be called
	▼
32	PMIx libraries are required to pass any attributes provided by the client to the host environment for
33	processing. In addition, the following attributes are required to be included in the passed <i>info</i> array:
34	PMIX USERID "pmix.euid" (uint32 t)
35	Effective user id.

```
PMIX_GRPID "pmix.egid" (uint32_t)
 1
 2
                    Effective group id.
 3
               Host environments that provide this module entry point are required to support the following
 4
5
               attributes:
6
               PMIX JOB CTRL ID "pmix.jctrl.id" (char*)
 7
                    Provide a string identifier for this request. The user can provide an identifier for the
8
                    requested operation, thus allowing them to later request status of the operation or to
9
                    terminate it. The host, therefore, shall track it with the request for future reference.
               PMIX JOB CTRL PAUSE "pmix.jctrl.pause" (bool)
10
                    Pause the specified processes.
11
12
               PMIX_JOB_CTRL_RESUME "pmix.jctrl.resume" (bool)
                    Resume ("un-pause") the specified processes.
13
14
               PMIX JOB CTRL KILL "pmix.jctrl.kill" (bool)
                    Forcibly terminate the specified processes and cleanup.
15
16
               PMIX_JOB_CTRL_SIGNAL "pmix.jctrl.sig" (int)
17
                    Send given signal to specified processes.
18
               PMIX JOB CTRL TERMINATE "pmix.jctrl.term" (bool)
19
                    Politely terminate the specified processes.
                                               Optional Attributes
               The following attributes are optional for host environments that support this operation:
20
               PMIX_JOB_CTRL_CANCEL "pmix.jctrl.cancel" (char*)
21
22
                    Cancel the specified request - the provided request ID must match the
                    PMIX JOB_CTRL_ID provided to a previous call to PMIx_Job_control . An ID of
23
24
                    NULL implies cancel all requests from this requestor.
               PMIX_JOB_CTRL_RESTART "pmix.jctrl.restart" (char*)
25
26
                    Restart the specified processes using the given checkpoint ID.
27
               PMIX JOB CTRL CHECKPOINT "pmix.jctrl.ckpt" (char*)
                    Checkpoint the specified processes and assign the given ID to it.
28
29
               PMIX_JOB_CTRL_CHECKPOINT_EVENT "pmix.jctrl.ckptev" (bool)
30
                    Use event notification to trigger a process checkpoint.
               PMIX JOB CTRL CHECKPOINT SIGNAL
31
                                                         "pmix.jctrl.ckptsig" (int)
                    Use the given signal to trigger a process checkpoint.
32
33
               PMIX JOB CTRL CHECKPOINT TIMEOUT "pmix.jctrl.ckptsig" (int)
```

```
Time in seconds to wait for a checkpoint to complete.
 1
 2
              PMIX JOB CTRL CHECKPOINT METHOD
               "pmix.jctrl.ckmethod" (pmix_data_array_t)
 3
                    Array of pmix_info_t declaring each method and value supported by this application.
 4
              PMIX_JOB_CTRL_PROVISION "pmix.jctrl.pvn" (char*)
 5
6
                    Regular expression identifying nodes that are to be provisioned.
 7
              PMIX JOB CTRL PROVISION IMAGE "pmix.jctrl.pvnimg" (char*)
8
                    Name of the image that is to be provisioned.
9
              PMIX_JOB_CTRL_PREEMPTIBLE "pmix.jctrl.preempt" (bool)
                    Indicate that the job can be pre-empted.
10
11
              Description
12
              Execute a job control action on behalf of a client. The targets array identifies the processes to
13
              which the requested job control action is to be applied. A NULL value can be used to indicate all
              processes in the caller's namespace. The use of PMIX RANK WILDCARD can also be used to
14
15
              indicate that all processes in the given namespace are to be included.
              The directives are provided as pmix_info_t structures in the directives array. The callback
16
17
              function provides a status to indicate whether or not the request was granted, and to provide some
18
              information as to the reason for any denial in the pmix_info_cbfunc_t array of
19
              pmix_info_t structures.
    11.2.22 pmix server monitor fn t
20
              Summary
21
22
              Request that a client be monitored for activity.
              Format
23
   PMIx v2.0
24
              typedef pmix_status_t (*pmix_server_monitor_fn_t)(
25
                                                   const pmix_proc_t *requestor,
                                                   const pmix_info_t *monitor, pmix_status_t error
26
27
                                                   const pmix_info_t directives[], size_t ndirs,
28
                                                   pmix_info_cbfunc_t cbfunc, void *cbdata);
```

N requestor
<pre>pmix_proc_t structure of requesting process (handle)</pre>
N monitor
<pre>pmix_info_t identifying the type of monitor being requested (handle)</pre>
N error
Status code to use in generating event if alarm triggers (integer)
N directives
Array of info structures (array of handles)
N ndirs
Number of elements in the <i>info</i> array (integer) N cbfunc
Callback function pmix_op_cbfunc_t (function reference)
N cbdata
Data to be passed to the callback function (memory reference)
Returns one of the following:
PMIX_SUCCESS, indicating that the request is being processed by the host environment - result
will be returned in the provided <i>cbfunc</i> . Note that the host must not invoke the callback function
prior to returning from the API.
PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
returned <i>success</i> - the <i>cbfunc</i> will not be called
PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the
request, even though the function entry was provided in the server module - the <i>cbfunc</i> will not
be called
a PMIx error constant indicating either an error in the input or that the request was immediately
processed and failed - the <i>cbfunc</i> will not be called
This entry point is only called for monitoring requests that are not directly supported by the PMIx
erver library itself.
Required Attributes
•
f supported by the PMIx server library, then the library must not pass any supported attributes to
he host environment. Any attributes provided by the client that are not directly supported by the erver library must be passed to the host environment if it provides this module entry. In addition,
the following attributes are required to be included in the passed <i>info</i> array:
PMIX_USERID "pmix.euid" (uint32_t)
Effective user id.
PMIX_GRPID "pmix.egid" (uint32_t)

35

Effective group id.

ı	
2	Host environments are not required to support any specific monitoring attributes.
	▼ Optional Attributes
3	The following attributes may be implemented by a host environment.
4 5	<pre>PMIX_MONITOR_ID "pmix.monitor.id" (char*) Provide a string identifier for this request.</pre>
6 7	<pre>PMIX_MONITOR_CANCEL "pmix.monitor.cancel" (char*) Identifier to be canceled (NULL means cancel all monitoring for this process).</pre>
8 9	<pre>PMIX_MONITOR_APP_CONTROL "pmix.monitor.appctrl" (bool) The application desires to control the response to a monitoring event.</pre>
10 11	<pre>PMIX_MONITOR_HEARTBEAT "pmix.monitor.mbeat" (void) Register to have the PMIx server monitor the requestor for heartbeats.</pre>
12 13	<pre>PMIX_MONITOR_HEARTBEAT_TIME "pmix.monitor.btime" (uint32_t) Time in seconds before declaring heartbeat missed.</pre>
14 15	<pre>PMIX_MONITOR_HEARTBEAT_DROPS "pmix.monitor.bdrop" (uint32_t) Number of heartbeats that can be missed before generating the event.</pre>
16 17	<pre>PMIX_MONITOR_FILE "pmix.monitor.fmon" (char*) Register to monitor file for signs of life.</pre>
18 19	<pre>PMIX_MONITOR_FILE_SIZE "pmix.monitor.fsize" (bool) Monitor size of given file is growing to determine if the application is running.</pre>
20 21	<pre>PMIX_MONITOR_FILE_ACCESS "pmix.monitor.faccess" (char*) Monitor time since last access of given file to determine if the application is running.</pre>
22 23	<pre>PMIX_MONITOR_FILE_MODIFY "pmix.monitor.fmod" (char*) Monitor time since last modified of given file to determine if the application is running.</pre>
24 25	<pre>PMIX_MONITOR_FILE_CHECK_TIME "pmix.monitor.ftime" (uint32_t) Time in seconds between checking the file.</pre>
26 27	<pre>PMIX_MONITOR_FILE_DROPS "pmix.monitor.fdrop" (uint32_t) Number of file checks that can be missed before generating the event.</pre>
	▲-
28	Description

29

Request that a client be monitored for activity.

11.2.23 pmix_server_get_cred_fn_t Summarv Request a credential from the host environment 3 Format PMIx v3.0 5 typedef pmix status t (*pmix server get cred fn t)(6 const pmix proc t *proc, 7 const pmix info t directives[], 8 size t ndirs, 9 pmix credential cbfunc t cbfunc, 10 void *cbdata); C 11 IN proc 12 pmix proc t structure of requesting process (handle) IN 13 directives 14 Array of info structures (array of handles) 15 IN ndirs 16 Number of elements in the *info* array (integer) 17 IN cbfunc 18 Callback function to return the credential (pmix_credential_cbfunc_t function reference) 19 IN cbdata 20 21 Data to be passed to the callback function (memory reference) 22 Returns PMIX SUCCESS, PMIX ERR NOT SUPPORTED indicating that the host environment 23 does not support the request (even though the function entry was provided in the server module), or a negative value corresponding to a PMIx error constant. In the event the function returns an error, 24 25 the cbfunc will not be called. ______ Required Attributes 26 If the PMIx library does not itself provide the requested credential, then it is required to pass any attributes provided by the client to the host environment for processing. In addition, it must include 27 the following attributes in the passed *info* array: 28

29 PMIX_USERID "pmix.euid" (uint32_t)
30 Effective user id.
31 PMIX_GRPID "pmix.egid" (uint32_t)
32 Effective group id.

	→ Optional Attributes
1	The following attributes are optional for host environments that support this operation:
2 3 4 5	<pre>PMIX_CRED_TYPE "pmix.sec.ctype" (char*) When passed in PMIx_Get_credential, a prioritized, comma-delimited list of desired credential types for use in environments where multiple authentication mechanisms may be available. When returned in a callback function, a string identifier of the credential type.</pre>
6 7 8 9	PMIX_TIMEOUT "pmix.timeout" (int) Time in seconds before the specified operation should time out (0 indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.
	Advice to PMIx library implementers
10 11 12 13 14	We recommend that implementation of the PMIX_TIMEOUT attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support PMIX_TIMEOUT directly in the PMIx server library must take care to resolve the race condition and should avoid passing PMIX_TIMEOUT to the host environment so that multiple competing timeouts are not created.
16 17	Description Request a credential from the host environment
18	11.2.24 pmix_server_validate_cred_fn_t
19 20	Summary Request validation of a credential

ı	Format
PMIx v3.0	• C —
2 3 4 5 6 7 8	<pre>typedef pmix_status_t (*pmix_server_validate_cred_fn_t)(</pre>
	C
9 10 11	<pre>IN proc pmix_proc_t structure of requesting process (handle) IN cred</pre>
12 13 14	Pointer to pmix_byte_object_t containing the credential (handle) IN directives Array of info structures (array of handles)
15 16 17	 IN ndirs Number of elements in the <i>info</i> array (integer) IN cbfunc
18 19 20 21	Callback function to return the result (pmix_validation_cbfunc_t function reference) IN cbdata Data to be passed to the callback function (memory reference)
22	Returns one of the following:
23 24	• PMIX_SUCCESS, indicating that the request is being processed by the host environment - result will be returned in the provided <i>cbfunc</i>
25 26	• PMIX_OPERATION_SUCCEEDED , indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will not be called
27 28 29	• PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the <i>cbfunc</i> will not be called
30 31	• a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will not be called
	▼ Required Attributes
32 33 34	If the PMIx library does not itself validate the credential, then it is required to pass any attributes provided by the client to the host environment for processing. In addition, it must include the following attributes in the passed <i>info</i> array:
35	PMIX_USERID "pmix.euid" (uint32_t)

Effective user id. 1 PMIX GRPID "pmix.eqid" (uint32 t) 2 Effective group id. 3 5 Host environments are not required to support any specific attributes. Optional Attributes The following attributes are optional for host environments that support this operation: 6 7 PMIX TIMEOUT "pmix.timeout" (int) 8 Time in seconds before the specified operation should time out (θ indicating infinite) in 9 error. The timeout parameter can help avoid "hangs" due to programming errors that prevent 10 the target process from ever exposing its data. Advice to PMIx library implementers 11 We recommend that implementation of the PMIX_TIMEOUT attribute be left to the host environment due to race condition considerations between completion of the operation versus 12 internal timeout in the PMIx server library. Implementers that choose to support PMIX TIMEOUT 13 14 directly in the PMIx server library must take care to resolve the race condition and should avoid 15 passing PMIX TIMEOUT to the host environment so that multiple competing timeouts are not 16 created. 17 Description 18 Request validation of a credential obtained from the host environment via a prior call to the 19 pmix_server_get_cred_fn_t module entry. 11.2.25 pmix_server_iof_fn_t 20 Summary 21 22 Request the specified IO channels be forwarded from the given array of processes.

```
Format
 1
   PMIx v3.0
 2
               typedef pmix status t (*pmix server iof fn t)(
 3
                                               const pmix_proc_t procs[], size_t nprocs,
 4
                                               const pmix info t directives[], size t ndirs,
 5
                                               pmix_iof_channel_t channels,
 6
                                               pmix op cbfunc t cbfunc, void *cbdata);
 7
               IN
                    procs
 8
                    Array pmix proc t identifiers whose IO is being requested (handle)
 9
               IN
                    nprocs
                    Number of elements in procs (size t)
10
               IN
                    directives
11
12
                    Array of pmix info t structures further defining the request (array of handles)
13
               IN
                    ndirs
14
                    Number of elements in the info array (integer)
                    channels
15
               IN
                    Bitmask identifying the channels to be forwarded (pmix iof channel t)
16
               IN
                    cbfunc
17
18
                    Callback function pmix_op_cbfunc_t (function reference)
19
               IN
                    cbdata
                    Data to be passed to the callback function (memory reference)
20
21
               Returns one of the following:
22
               • PMIX SUCCESS, indicating that the request is being processed by the host environment - result
23
                 will be returned in the provided cbfunc. Note that the library must not invoke the callback
                 function prior to returning from the API.
24
25
               • PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
                 returned success - the cbfunc will not be called
26
27
               • PMIX ERR NOT SUPPORTED, indicating that the host environment does not support the
28
                 request, even though the function entry was provided in the server module - the cbfunc will not
                 be called
29
               • a PMIx error constant indicating either an error in the input or that the request was immediately
30
                 processed and failed - the cbfunc will not be called
31
                                                Required Attributes
               The following attributes are required to be included in the passed info array:
32
               PMIX USERID "pmix.euid" (uint32 t)
33
                     Effective user id.
34
35
               PMIX GRPID "pmix.eqid" (uint32 t)
```

Effective group id.
lost environments that provide this module entry point are required to support the following ttributes:
MIX_IOF_CACHE_SIZE "pmix.iof.csize" (uint32_t) The requested size of the server cache in bytes for each specified channel. By default, the server is allowed (but not required) to drop all bytes received beyond the max size.
MIX_IOF_DROP_OLDEST "pmix.iof.old" (bool) In an overflow situation, drop the oldest bytes to make room in the cache.
MIX_IOF_DROP_NEWEST "pmix.iof.new" (bool) In an overflow situation, drop any new bytes received until room becomes available in the cache (default).
Optional Attributes
The following attributes may be supported by a host environment.
Controls grouping of IO on the specified channel(s) to avoid being called every time a bit IO arrives. The library will execute the callback whenever the specified number of bytes becomes available. Any remaining buffered data will be "flushed" upon call to deregister respective channel.
MIX_IOF_BUFFERING_TIME "pmix.iof.btime" (uint32_t) Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering size, this prevents IO from being held indefinitely while waiting for another payload to arrive.
Description Request the specified IO channels be forwarded from the given array of processes. An error shall eturned in the callback function if the requested service from any of the requested processes can e provided. Advice to PMIx library implementers
The forwarding of stdin is a <i>push</i> process - processes cannot request that it be <i>pulled</i> from some ther source. Requests including the PMIX_FWD_STDIN_CHANNEL channel will return a MIX_ERR_NOT_SUPPORTED error.

11.2.26 pmix_server_stdin_fn_t

```
Summarv
 3
               Pass standard input data to the host environment for transmission to specified recipients.
               Format
   PMIx v3.0
 5
               typedef pmix status t (*pmix server stdin fn t) (
 6
                                                   const pmix proc t *source,
                                                   const pmix_proc_t targets[],
 7
 8
                                                   size t ntargets,
 9
                                                   const pmix info t directives[],
10
                                                   size t ndirs,
                                                   const pmix byte object t *bo,
11
                                                   pmix op cbfunc t cbfunc, void *cbdata);
12
               IN
13
                    source
14
                    pmix proc t structure of source process (handle)
                   targets
15
               IN
                    Array of pmix_proc_t target identifiers (handle)
16
17
               IN
                   ntargets
18
                    Number of elements in the targets array (integer)
19
               IN
                    directives
                    Array of info structures (array of handles)
20
               IN
                    ndirs
21
                    Number of elements in the info array (integer)
22
23
               IN
                    Pointer to pmix_byte_object_t containing the payload (handle)
24
25
               IN
                    cbfunc
26
                    Callback function pmix_op_cbfunc_t (function reference)
27
               IN
                    Data to be passed to the callback function (memory reference)
28
29
               Returns one of the following:
               • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result
30
                 will be returned in the provided cbfunc. Note that the library must not invoke the callback
31
                 function prior to returning from the API.
32
33
               • PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
                 returned success - the cbfunc will not be called
34
35
               • PMIX ERR NOT SUPPORTED, indicating that the host environment does not support the
                 request, even though the function entry was provided in the server module - the cbfunc will not
36
                 be called
37
```

```
• a PMIx error constant indicating either an error in the input or that the request was immediately
 1
2
                processed and failed - the cbfunc will not be called
                  _____
                                            Required Attributes
3
              The following attributes are required to be included in the passed info array:
              PMIX USERID "pmix.euid" (uint32 t)
 4
 5
                   Effective user id.
              PMIX_GRPID "pmix.egid" (uint32_t)
6
 7
                    Effective group id.
              ▲-----
8
              Description
9
              Passes stdin to the host environment for transmission to specified recipients. The host environment
              is responsible for forwarding the data to all locations that host the specified targets and delivering
10
              the payload to the PMIx server library connected to those clients.
11
    11.2.27
                pmix server grp fn t
              Summary
13
14
              Request group operations (construct, destruct, etc.) on behalf of a set of processes.
              Format
15
   PMIx v4.0
16
              typedef pmix_status_t (*pmix_server_grp_fn_t)(
                                               pmix_group_operation_t op, char grp[],
17
18
                                               const pmix_proc_t procs[], size_t nprocs,
                                                const pmix_info_t directives[],
19
20
                                                size t ndirs,
                                               pmix_info_cbfunc_t cbfunc, void *cbdata);
21
22
              IN
                   oр
23
                  pmix_group_operation_t value indicating operation the host is requested to perform
24
                  (integer)
              IN
25
                   grp
                  Character string identifying the group (string)
26
27
              IN
                  Array of pmix proc t identifiers of participants (handle)
28
29
              IN nprocs
                  Number of elements in the procs array (integer)
30
              IN
31
                   directives
32
                  Array of info structures (array of handles)
```

1	IN ndirs
2	Number of elements in the <i>info</i> array (integer)
3	IN cbfunc
4	Callback function pmix_info_cbfunc_t (function reference)
5	IN cbdata
6	Data to be passed to the callback function (memory reference)
7	Returns one of the following:
8	• PMIX_SUCCESS, indicating that the request is being processed by the host environment - result
9	will be returned in the provided cbfunc. Note that the library must not invoke the callback
10	function prior to returning from the API.
11	• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
12	returned <i>success</i> - the <i>cbfunc</i> will not be called
13	• PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the
14	request, even though the function entry was provided in the server module - the <i>cbfunc</i> will not
15	be called
16	• a PMIx error constant indicating either an error in the input or that the request was immediately
17	processed and failed - the <i>cbfunc</i> will not be called
	▼ Optional Attributes
18	The following attributes may be supported by a host environment.
19	PMIX_GROUP_ASSIGN_CONTEXT_ID "pmix.grp.actxid" (bool)
20	Requests that the RM assign a new context identifier to the newly created group. The
21	identifier is an unsigned, size_t value that the RM guarantees to be unique across the range
22	specified in the request. Thus, the value serves as a means of identifying the group within
23	that range. If no range is specified, then the request defaults to PMIX_RANGE_SESSION.
24	PMIX_GROUP_LOCAL_ONLY "pmix.grp.lcl" (bool)
25	Group operation only involves local processes. PMIx implementations are <i>required</i> to
26	automatically scan an array of group members for local vs remote processes - if only local
27	processes are detected, the implementation need not execute a global collective for the
28	operation unless a context ID has been requested from the host environment. This can result
29	in significant time savings. This attribute can be used to optimize the operation by indicating
30	whether or not only local processes are represented, thus allowing the implementation to
31	bypass the scan. The default is false
32	<pre>PMIX_GROUP_ENDPT_DATA "pmix.grp.endpt" (pmix_byte_object_t)</pre>
33	Data collected to be shared during group construction
34	PMIX_GROUP_OPTIONAL "pmix.grp.opt" (bool)
35	Participation is optional - do not return an error if any of the specified processes terminate

without having joined. The default is false

```
PMIX_RANGE "pmix.range" (pmix_data_range_t)
 1
 2
                    Value for calls to publish/lookup/unpublish or for monitoring event notifications.
 3
              The following attributes may be included in the host's response:
              PMIX GROUP ID "pmix.grp.id" (char*)
 4
 5
                    User-provided group identifier
              PMIX_GROUP_MEMBERSHIP "pmix.grp.mbrs" (pmix_data_array_t*)
6
 7
                    Array of group member ID's
              PMIX_GROUP_CONTEXT_ID "pmix.grp.ctxid" (size_t)
8
9
                    Context identifier assigned to the group by the host RM.
10
              PMIX_GROUP_ENDPT_DATA "pmix.grp.endpt" (pmix_byte_object_t)
                    Data collected to be shared during group construction
11
              Description
12
13
              Perform the specified operation across the identified processes, plus any special actions included in
              the directives. Return the result of any special action requests in the callback function when the
14
              operation is completed. Actions may include a request (PMIX GROUP ASSIGN CONTEXT ID
15
              ) that the host assign a unique numerical (size t) ID to this group - if given, the PMIX RANGE
16
17
              attribute will specify the range across which the ID must be unique (default to
18
              PMIX RANGE SESSION).
    11.2.28 pmix_server_fabric_fn_t
20
              Summary
21
              Request fabric-related operations (e.g., information on a fabric) on behalf of a tool or other process.
22
              Format
   PMIx v4.0
23
              typedef pmix status t (*pmix server fabric fn t)(
                                                 const pmix proc t *requestor,
24
25
                                                 pmix_fabric_operation_t op,
                                                 const pmix_info_t directives[],
26
27
                                                 size_t ndirs,
```

28

pmix_info_cbfunc_t cbfunc, void *cbdata);

1	IN requestor
2	<pre>pmix_proc_t identifying the requestor (handle)</pre>
3	IN op
4 5	<pre>pmix_fabric_operation_t value indicating operation the host is requested to perform (integer)</pre>
6	IN directives
7	Array of info structures (array of handles)
8	IN ndirs
9	Number of elements in the <i>info</i> array (integer)
10 11	IN cbfunc Callback function pmix_info_cbfunc_t (function reference)
12	IN cbdata
13	Data to be passed to the callback function (memory reference)
14	Returns one of the following:
15 16 17	• PMIX_SUCCESS, indicating that the request is being processed by the host environment - result will be returned in the provided <i>cbfunc</i> . Note that the library must not invoke the callback function prior to returning from the API.
18 19	• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will not be called
20 21 22	• PMIX_ERR_NOT_SUPPORTED , indicating that the host environment does not support the request, even though the function entry was provided in the server module - the <i>cbfunc</i> will not be called
23 24	• a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will not be called
	▼
25 26	The following directives are required to be supported by all hosts to aid users in identifying the fabric and (if applicable) the device to whom the operation references:
27 28	PMIX_FABRIC_VENDOR "pmix.fab.vndr" (string) Name of fabric vendor (e.g., Amazon, Mellanox, Cray, Intel)
29 30	<pre>PMIX_FABRIC_IDENTIFIER "pmix.fab.id" (string) An identifier for the fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1)</pre>
31 32 33 34 35	PMIX_FABRIC_PLANE "pmix.fab.plane" (char*) ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request for information, specifies the plane whose information is to be returned. When used directly in a request, returns a pmix_data_array_t of string identifiers for all fabric planes in the system.

PMIX_FABRIC_DEVICE_INDEX "pmix.fabdev.idx" (uint32_t) System-unique index of a particular fabric device (NIC).

Description

Perform the specified operation. Return the result of any requests in the callback function when the operation is completed. Operations may, for example, include a request for fabric information. See <code>pmix_fabric_t</code> for a list of expected information to be included in the response. Note that requests for device index are to be returned in the callback function's array of <code>pmix_info_t</code> using the <code>PMIX_FABRIC_DEVICE_INDEX</code> attribute.

CHAPTER 12

Fabric Support Definitions

As the drive for performance continues, interest has grown in both scheduling algorithms that take into account network locality of the allocated resources, and in optimizing collective communication patterns by structuring them to follow fabric topology. Several interfaces have been defined that are specifically intended to support WLMs (also known as *schedulers*) by providing access to information of potential use to scheduling algorithms - e.g., information on communication costs between different points on the fabric.

In contrast, hierarchical collective operations require each process have global information about both its peers and the fabric. For example, one might aggregate the contribution from all processes on a node, then again across all nodes on a common switch, and finally across all switches. Creating such optimized patterns relies on detailed knowledge of the fabric location of each participant.

PMIx supports these efforts by defining datatypes and attributes by which fabric coordinates for processes and devices can be obtained from the host SMS. When used in conjunction with the PMIx *instant on* methods, this results in the ability of a process to obtain the fabric coordinate of all other processes without incurring additional overhead associated with the publish/exchange of that information.

16 12.1 Fabric Support Constants

The following constants are defined for use in fabric-related events.

PMIX_FABRIC_UPDATE_PENDING The PMIx server library has been alerted to a change in the fabric that requires updating of one or more registered pmix_fabric_t objects.
PMIX_FABRIC_UPDATED The PMIx server library has completed updating the entries of all affected pmix_fabric_t objects registered with the library. Access to the entries of those objects may now resume.

12.2 Fabric Support Datatypes

Several datatype definitions have been created to support fabric-related operations and information.

12.2.1 Fabric Coordinate Structure

The pmix_coord_t structure describes the fabric coordinates of a specified process in a given view PMIx v4.0typedef struct pmix_coord { char *fabric; char *plane; pmix coord view t view; uint32 t *coord; size t dims; } pmix coord t;

All coordinate values shall be expressed as unsigned integers due to their units being defined in fabric devices and not physical distances. The coordinate is therefore an indicator of connectivity and not relative communication distance.

The fabric and plane fields are assigned by the fabric provider to help the user identify the fabric to which the coordinates refer. Note that providers are not required to assign any particular value to the fields and may choose to leave the fields blank. Example entries include {"Ethernet", "mgmt"} or {"infiniband", "data1"}.

———— Advice to PMIx library implementers ————

Note that the <code>pmix_coord_t</code> structure does not imply nor mandate any requirement on how the coordinate data is to be stored within the PMIx library. Implementers are free to store the coordinate in whatever format they choose.

A fabric coordinate is usually associated with a given fabric device - e.g., a particular NIC on a node. Thus, while the fabric coordinate of a device must be unique in a given view, the coordinate may be shared by multiple processes on a node. If the node contains multiple fabric devices, then either the device closest to the binding location of a process shall be used as its coordinate, or (if the process is unbound or its binding is not known) all devices on the node shall be reported as a <code>pmix_data_array_t</code> of <code>pmix_coord_t</code> structures.

Nodes with multiple fabric devices can also have those devices configured as multiple **fabric planes**. In such cases, a given process (even if bound to a specific location) may be associated with a coordinate on each plane. The resulting set of fabric coordinates shall be reported as a **pmix_data_array_t** of **pmix_coord_t** structures. The caller may request a coordinate from a specific fabric plane by passing the **PMIX_FABRIC_PLANE** attribute as a directive/qualifier to the **PMIx_Get** or **PMIx_Query_info_nb** call.

12.2.2 Fabric Coordinate Support Macros The following macros are provided to support the **pmix_coord_t** structure. 2 12.2.2.1 Initialize the pmix_coord_t structure 3 Initialize the pmix coord t fields 4 PMIx v4.0 5 PMIX COORD CONSTRUCT (m) IN 6 m 7 Pointer to the structure to be initialized (pointer to **pmix_coord_t**) 12.2.2.2 Destruct the pmix_coord_t structure Destruct the pmix_coord_t fields 9 PMIx v4.0PMIX COORD DESTRUCT (m) 10 11 IN Pointer to the structure to be destructed (pointer to pmix coord t) 12 12.2.2.3 Create a pmix_coord_t array 13 14 Allocate and initialize a pmix_coord_t array PMIx v4.0 15 PMIX COORD CREATE (m, n) INOUT m 16 Address where the pointer to the array of **pmix_coord_t** structures shall be stored (handle) 17 IN 18 Number of structures to be allocated (size t) 19 12.2.2.4 Release a pmix_coord_t array 20 21 Release an array of pmix_coord_t structures PMIx v4.0 22 PMIX COORD FREE (m, n) 23 IN 24 Pointer to the array of pmix_coord_t structures (handle) 25 IN Number of structures in the array (size t) 26

1 12.2.3 Fabric Coordinate Views

PMIx v4.0	C —
2	<pre>typedef uint8_t pmix_coord_view_t;</pre>
3	#define PMIX_COORD_VIEW_UNDEF 0x00
4	#define PMIX_COORD_LOGICAL_VIEW 0x01
5	#define PMIX_COORD_PHYSICAL_VIEW 0x02
	C
6 7	Fabric coordinates can be reported based on different <i>views</i> according to user preference at the time of request. The following views have been defined:
8 9	PMIX_COORD_VIEW_UNDEF The coordinate view has not been defined. PMIX_COORD_LOGICAL_VIEW The coordinates are provided in a <i>logical</i> view, typically
10	given in Cartesian (x,y,z) dimensions, that describes the data flow in the fabric as defined by
11	the arrangement of the hierarchical addressing scheme, fabric segmentation, routing domains,
12	and other similar factors employed by that fabric.
13	PMIX_COORD_PHYSICAL_VIEW The coordinates are provided in a <i>physical</i> view based on
14	the actual wiring diagram of the fabric - i.e., values along each axis reflect the relative
15	position of that interface on the specific fabric cabling.
	Advice to PMIx library implementers —————
16	PMIx library implementers are advised to avoid declaring the above constants as actual enum
17	values in order to allow host environments to add support for possibly proprietary coordinate views.
18	
19	If the requester does not specify a view, coordinates shall default to the <i>logical</i> view.
20 12.2.4	Fabric Link State
21	The pmix_link_state_t is a uint32_t type for fabric link states.
PMIx v4.0	C —
22	typedef uint8_t pmix_link_state_t;
	C
23	The following constants can be used to set a variable of the type pmix_link_state_t . All
24	definitions were introduced in version 4 of the standard unless otherwise marked. Valid link state
25	values start at zero.
26	PMIX_LINK_STATE_UNKNOWN The port state is unknown or not applicable.
27	PMIX_LINK_DOWN The port is inactive.
28	PMIX_LINK_UP The port is active.

12.2.5 Fabric Operation Constants

The pmix_fabric_operation_t structure is an enumerated type for specifying fabric 2 PMIx v4.03 operations used in the PMIx server module's pmix_server_fabric_fn_t API. All values 4 were originally defined in version 4 of the standard unless otherwise marked. 5 PMIX FABRIC REQUEST INFO Request information on a specific fabric - if the fabric isn't specified as per PMIx_Fabric_register, then return information on the system default 6 7 fabric. Information to be returned is described in pmix fabric t. Update information on a specific fabric - the index of the 8 PMIX FABRIC UPDATE INFO 9 fabric (**PMIX FABRIC INDEX**) to be updated must be provided. Request information on a specific NIC within the 10 PMIX_FABRIC_GET_VERTEX_INFO 11 identified fabric - the index of the device (PMIX FABRIC DEVICE INDEX) and of the fabric (PMIX FABRIC INDEX) must be provided. If the NIC identifier is not specified, 12 then return vertex info on all NICs in the fabric. Information to be included on each vertex is 13 14 described in pmix fabric t. Advice to users -15 Requesting information on every NIC in the fabric may be an expensive operation in terms of 16 both memory footprint and time. 17 PMIX FABRIC GET DEVICE INDEX Request the fabric-wide index (returned as PMIX FABRIC DEVICE INDEX) for a specific NIC within the identified fabric based on 18 19 the provided vertex information. The index of the fabric must be provided.

12.2.6 Fabric registration structure

The **pmix_fabric_t** structure is used by a WLM to interact with fabric-related PMIx interfaces, and to provide information about the fabric for use in scheduling algorithms or other purposes.

```
PMIx v4.0

23 typedef struct pmix_fabric_s {
24 char *name;
25 size_t index;
26 pmix_info_t *info;
27 size_t ninfo;
28 void *module;
29 } pmix fabric t;
```

20 21

ı	Note that in this structure.
2 3 4	• <i>name</i> is an optional user-supplied string name identifying the fabric being referenced by this struct. If provided, the field must be a NULL -terminated string composed of standard alphanumeric values supported by common utilities such as <i>strcmp</i> .;
5	• <i>index</i> is a PMIx-provided number identifying this object;
6 7	• <i>info</i> is an array of pmix_info_t containing information (provided by the PMIx library) about the fabric;
8	• <i>ninfo</i> is the number of elements in the <i>info</i> array
9	• module points to an opaque object reserved for use by the PMIx server library.
10 11 12 13	Note that only the <i>name</i> field is provided by the user - all other fields are provided by the PMIx library and must not be modified by the user. The <i>info</i> array contains a varying amount of information depending upon both the PMIx implementation and information available from the fabric vendor. At a minimum, it must contain (ordering is arbitrary):
	→
14 15	PMIX_FABRIC_VENDOR "pmix.fab.vndr" (string) Name of fabric vendor (e.g., Amazon, Mellanox, Cray, Intel)
16 17	PMIX_FABRIC_IDENTIFIER "pmix.fab.id" (string) An identifier for the fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1)
18 19 20	PMIX_FABRIC_NUM_VERTICES "pmix.fab.nverts" (size_t) Total number of NICs in the system - corresponds to the number of vertices (i.e., rows and columns) in the cost matrix
21	and may optionally contain one or more of the following:
	▼ Optional Attributes
22 23 24	<pre>PMIX_FABRIC_COST_MATRIX "pmix.fab.cm" (pointer) Pointer to a two-dimensional array of point-to-point relative communication costs expressed as uint16_t values</pre>
25 26 27 28 29	PMIX_FABRIC_GROUPS "pmix.fab.grps" (string) A string delineating the group membership of nodes in the system, where each fabric group consists of the group number followed by a colon and a comma-delimited list of nodes in that group, with the groups delimited by semi-colons (e.g., 0:node000,node002,node004,node006;1:node001,node003,node005,node007) PMIX_FABRIC_DIMS "pmix.fab.dims" (uint32_t)
	pinta tub (utilisa_o)

1 Number of dimensions in the specified fabric plane/view. If no plane is specified in a 2 request, then the dimensions of all planes in the system will be returned as a pmix data array t containing an array of uint32 t values. Default is to provide 3 4 dimensions in *logical* view. 5 PMIX_FABRIC_PLANE "pmix.fab.plane" (char*) ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request 6 7 for information, specifies the plane whose information is to be returned. When used directly 8 in a request, returns a pmix data array t of string identifiers for all fabric planes in 9 the system. 10 PMIX FABRIC SHAPE "pmix.fab.shape" (pmix data array t*) The size of each dimension in the specified fabric plane/view, returned in a 11 12 pmix data array t containing an array of uint32 t values. The size is defined as the number of elements present in that dimension - e.g., the number of NICs in one 13 14 dimension of a physical view of a fabric plane. If no plane is specified, then the shape of each plane in the system will be returned in an array of fabric shapes. Default is to provide 15 the shape in logical view. 16 PMIX_FABRIC_SHAPE_STRING "pmix.fab.shapestr" (string) 17 Network shape expressed as a string (e.g., "10x12x2"). 18 While unusual due to scaling issues, implementations may include an array of 19 **PMIX FABRIC DEVICE** elements describing the vertex information for each NIC in the system. 20 Each element shall contain a pmix_data_array_t of pmix_info_t values describing the 21 22 device. Each array may contain one or more of the following (ordering is arbitrary): 23 PMIX_FABRIC_DEVICE_NAME "pmix.fabdev.nm" (string) 24 The operating system name associated with the device. This may be a logical fabric interface 25 name (e.g. eth0 or eno1) or an absolute filename. PMIX_FABRIC_DEVICE_VENDOR "pmix.fabdev.vndr" (string) 26 Indicates the name of the vendor that distributes the NIC. 27 28 PMIX_FABRIC_DEVICE_ID "pmix.fabdev.devid" (string) 29 This is a vendor-provided identifier for the device or product. 30 PMIX_HOSTNAME "pmix.hname" (char*) 31 Name of the host (e.g., where a specified process is running, or a given device is located). PMIX_FABRIC_DEVICE_DRIVER "pmix.fabdev.driver" (string) 32 33 The name of the driver associated with the device 34 PMIX_FABRIC_DEVICE_FIRMWARE "pmix.fabdev.fmwr" (string) 35 The device's firmware version PMIX FABRIC_DEVICE_ADDRESS "pmix.fabdev.addr" (string) 36 37 The primary link-level address associated with the NIC, such as a Media Access Control (MAC) address. If multiple addresses are available, only one will be reported. 38

```
PMIX_FABRIC_DEVICE_MTU "pmix.fabdev.mtu" (size_t)
 1
 2
                     The maximum transfer unit of link level frames or packets, in bytes.
 3
               PMIX_FABRIC_DEVICE_SPEED "pmix.fabdev.speed" (size_t)
                     The active link data rate, given in bits per second.
 4
 5
               PMIX_FABRIC_DEVICE_STATE "pmix.fabdev.state" ( pmix_link_state_t )
6
                     The last available physical port state. Possible values are PMIX LINK STATE UNKNOWN,
                     PMIX_LINK_DOWN, and PMIX_LINK_UP, to indicate if the port state is unknown or not
 7
8
                     applicable (unknown), inactive (down), or active (up).
9
               PMIX_FABRIC_DEVICE_TYPE "pmix.fabdev.type" (string)
10
                     Specifies the type of fabric interface currently active on the device, such as Ethernet or
                     InfiniBand.
11
12
               PMIX_FABRIC_DEVICE_BUS_TYPE "pmix.fabdev.btyp" (string)
                     The type of bus to which the device is attached (e.g., "PCI", "GEN-Z").
13
14
               PMIX_FABRIC_DEVICE_PCI_DEVID "pmix.fabdev.pcidevid" (string)
                     A node-level unique identifier for a Peripheral Component Interconnect (PCI) device.
15
                     Provided only if the device is located on a PCI bus. The identifier is constructed as a
16
17
                     four-part tuple delimited by colons comprised of the PCI 16-bit domain, 8-bit bus, 8-bit
18
                     device, and 8-bit function IDs, each expressed in zero-extended hexadecimal form. Thus, an
19
                     example identifier might be "abc1:0f:23:01". The combination of node identifier (
20
                     PMIX HOSTNAME OF PMIX NODEID ) and PMIX FABRIC DEVICE PCI DEVID
                     shall be unique within the system.
21
    12.2.6.1
                Initialize the pmix fabric t structure
               Initialize the pmix_fabric_t fields
23
   PMIx v4.0
24
               PMIX FABRIC CONSTRUCT (m)
               IN
25
26
                    Pointer to the structure to be initialized (pointer to pmix_fabric_t)
```

1 12.3 Fabric Support Attributes

The following attributes are used by the library supporting the system's WLM to either access or 2 return fabric-related information (e.g., as part of the **pmix fabric t** structure). 3 PMIX SERVER SCHEDULER "pmix.srv.sched" (bool) 4 Server requests access to WLM-supporting features - passed solely to the 5 6 **PMIx_server_init** API to indicate that the library is to be initialized for scheduler 7 support. 8 PMIX_FABRIC_COST_MATRIX "pmix.fab.cm" (pointer) 9 Pointer to a two-dimensional array of point-to-point relative communication costs expressed 10 as uint16 t values PMIX FABRIC GROUPS "pmix.fab.grps" (string) 11 12 A string delineating the group membership of nodes in the system, where each fabric group consists of the group number followed by a colon and a comma-delimited list of nodes in 13 14 that group, with the groups delimited by semi-colons (e.g., 15 0:node000,node002,node004,node006;1:node001,node003,node005,node007) 16 The following attributes may be returned by calls to the scheduler-related APIs or in response to 17 queries (e.g., **PMIx_Get** or **PMIx_Query_info**) made by processes or tools. 18 PMIX FABRIC VENDOR "pmix.fab.vndr" (string) Name of fabric vendor (e.g., Amazon, Mellanox, Cray, Intel) 19 20 PMIX FABRIC IDENTIFIER "pmix.fab.id" (string) An identifier for the fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1) 21 22 PMIX_FABRIC_INDEX "pmix.fab.idx" (size_t) 23 The index of the fabric as returned in pmix fabric t PMIX_FABRIC_NUM_VERTICES "pmix.fab.nverts" (size_t) 24 25 Total number of NICs in the system - corresponds to the number of vertices (i.e., rows and 26 columns) in the cost matrix 27 PMIX FABRIC COORDINATE "pmix.fab.coord" (pmix data array t) 28 Fabric coordinate(s) of the specified process in the view and/or plane provided by the requester. If only one NIC has been assigned to the specified process, then the array will 29 30 contain only one address. Otherwise, the array will contain the coordinates of all NICs available to the process in order of least to greatest distance from the process (NICs equally 31 32 distant from the process will be listed in arbitrary order). 33 PMIX FABRIC VIEW "pmix.fab.view" (pmix coord view t) Fabric coordinate view to be used for the requested coordinate - see 34 pmix coord view t for the list of accepted values. 35 36 PMIX FABRIC DIMS "pmix.fab.dims" (uint32 t) Number of dimensions in the specified fabric plane/view. If no plane is specified in a 37 request, then the dimensions of all planes in the system will be returned as a 38 pmix data array t containing an array of uint32 t values. Default is to provide 39 40 dimensions in logical view. 41 PMIX_FABRIC_PLANE "pmix.fab.plane" (char*)

ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request 1 2 for information, specifies the plane whose information is to be returned. When used directly in a request, returns a pmix data array t of string identifiers for all fabric planes in 3 4 the system. 5 PMIX_FABRIC_SWITCH "pmix.fab.switch" (char*) 6 ID string of a fabric switch. When used as a modifier in a request for information, specifies 7 the switch whose information is to be returned. When used directly in a request, returns a 8 pmix_data_array_t of string identifiers for all fabric switches in the system. 9 PMIX_FABRIC_ENDPT "pmix.fab.endpt" (pmix_data_array_t) Fabric endpoints for a specified process. As multiple endpoints may be assigned to a given 10 process (e.g., in the case where multiple NICs are associated with a package to which the 11 process is bound), the returned values will be provided in a pmix_data_array_t - the 12 returned data type of the individual values in the array varies by fabric provider. 13 14 PMIX_FABRIC_SHAPE "pmix.fab.shape" (pmix_data_array_t*) The size of each dimension in the specified fabric plane/view, returned in a 15 pmix data array t containing an array of uint32 t values. The size is defined as 16 the number of elements present in that dimension - e.g., the number of NICs in one 17 dimension of a physical view of a fabric plane. If no plane is specified, then the shape of 18 19 each plane in the system will be returned in an array of fabric shapes. Default is to provide the shape in *logical* view. 20 PMIX_FABRIC_SHAPE_STRING "pmix.fab.shapestr" (string) 21 22 Network shape expressed as a string (e.g., "10x12x2"). PMIX SWITCH PEERS "pmix.speers" (string) 23 24 Comma-delimited string of peers that share the same switch as the process specified in the call to PMIx Get . Single-NIC environments will return a string. Multi-NIC environments 25 will return an array of results, each element consisting of an array containing the 26 PMIX_FABRIC_DEVICE_INDEX of the local NIC and the list of peers sharing the switch 27 to which that NIC is connected. 28 29 The following attributes are used to describe devices (a.k.a., NICs) attached to the fabric. PMIX_FABRIC_DEVICE "pmix.fabdev" (pmix_data_array_t) 30 31 An array of **pmix info t** describing a particular fabric device (NIC). PMIX FABRIC DEVICE INDEX "pmix.fabdev.idx" (uint32 t) 32 System-unique index of a particular fabric device (NIC). 33 PMIX_FABRIC_DEVICE_NAME "pmix.fabdev.nm" (string) 34 The operating system name associated with the device. This may be a logical fabric interface 35 name (e.g. eth0 or eno1) or an absolute filename. 36 PMIX_FABRIC_DEVICE_VENDOR "pmix.fabdev.vndr" (string) 37 Indicates the name of the vendor that distributes the NIC. 38 PMIX FABRIC DEVICE BUS TYPE "pmix.fabdev.btyp" (string) 39 40 The type of bus to which the device is attached (e.g., "PCI", "GEN-Z"). PMIX_FABRIC_DEVICE_ID "pmix.fabdev.devid" (string) 41 This is a vendor-provided identifier for the device or product. 42

PMIX FABRIC DEVICE DRIVER "pmix.fabdev.driver" (string)

```
1
                     The name of the driver associated with the device
2
               PMIX_FABRIC_DEVICE_FIRMWARE "pmix.fabdev.fmwr" (string)
 3
                     The device's firmware version
 4
               PMIX_FABRIC_DEVICE_ADDRESS "pmix.fabdev.addr" (string)
 5
                     The primary link-level address associated with the NIC, such as a MAC address. If multiple
                     addresses are available, only one will be reported.
6
7
               PMIX FABRIC DEVICE MTU "pmix.fabdev.mtu" (size t)
8
                     The maximum transfer unit of link level frames or packets, in bytes.
9
               PMIX_FABRIC_DEVICE_SPEED "pmix.fabdev.speed" (size_t)
10
                     The active link data rate, given in bits per second.
               PMIX_FABRIC_DEVICE_STATE "pmix.fabdev.state" ( pmix_link_state_t )
11
12
                     The last available physical port state. Possible values are PMIX_LINK_STATE_UNKNOWN,
13
14
                     PMIX_LINK_DOWN, and PMIX_LINK_UP, to indicate if the port state is unknown or not
                     applicable (unknown), inactive (down), or active (up).
15
               PMIX_FABRIC_DEVICE_TYPE "pmix.fabdev.type" (string)
16
                     Specifies the type of fabric interface currently active on the device, such as Ethernet or
17
18
                     InfiniBand.
19
               PMIX FABRIC DEVICE PCI DEVID "pmix.fabdev.pcidevid" (string)
                     A node-level unique identifier for a PCI device. Provided only if the device is located on a
20
21
                     PCI bus. The identifier is constructed as a four-part tuple delimited by colons comprised of
22
                     the PCI 16-bit domain, 8-bit bus, 8-bit device, and 8-bit function IDs, each expressed in
23
                     zero-extended hexadecimal form. Thus, an example identifier might be "abc1:0f:23:01". The
                     combination of node identifier ( PMIX_HOSTNAME or PMIX_NODEID ) and
24
25
                     PMIX_FABRIC_DEVICE_PCI_DEVID shall be unique within the system.
```

12.4 Fabric Support Functions

The following APIs allow the WLM to request specific services from the fabric subsystem via the PMIx library.

Advice to PMIx server hosts —

Due to their high cost in terms of execution, memory consumption, and interactions with other SMS components (e.g., a fabric manager), it is strongly advised that the underlying implementation of these APIs be restricted to a single PMIx server in a system that is supporting the SMS component responsible for the scheduling of allocations (i.e., the system <code>scheduler</code>). The <code>PMIX_SERVER_SCHEDULER</code> attribute can be used for this purpose to control the execution path. Clients, tools, and other servers utilizing these functions are advised to have their requests forwarded to the server supporting the scheduler using the <code>pmix_server_fabric_fn_t</code> server module function, as needed.

27

28

29

30

31 32

33

34

35

Summarv Register for access to fabric-related information. 3 **Format** PMIx v4.0 5 pmix status t PMIx_Fabric_register(pmix_fabric_t *fabric, 6 7 const pmix info t directives[], size t ndirs) 8 9 IN fabric 10 address of a pmix_fabric_t (backed by storage). User may populate the "name" field at will - PMIx does not utilize this field (handle) 11 IN directives 12 an optional array of values indicating desired behaviors and/or fabric to be accessed. If **NULL**, 13 then the highest priority available fabric will be used (array of handles) 14 IN ndirs 15 Number of elements in the *directives* array (integer) 16 17 Returns **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant. Required Attributes The following directives are required to be supported by all PMIx libraries to aid users in 18 identifying the fabric whose data is being sought: 19 PMIX_FABRIC_PLANE "pmix.fab.plane" (char*) 20 21 ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request 22 for information, specifies the plane whose information is to be returned. When used directly in a request, returns a pmix data array t of string identifiers for all fabric planes in 23 the system. 24 25 PMIX_FABRIC_IDENTIFIER "pmix.fab.id" (string) An identifier for the fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1) 26 PMIX FABRIC VENDOR "pmix.fab.vndr" (string) 27 Name of fabric vendor (e.g., Amazon, Mellanox, Cray, Intel) 28

12.4.1

PMIx Fabric register

Description

Register for access to fabric-related information, including the communication cost matrix. This call must be made prior to requesting information from a fabric. The caller may request access to a particular fabric using the vendor, type, or identifier, or to a specific **fabric plane** via the **PMIX_FABRIC_PLANE** attribute - otherwise, the default fabric will be returned.

For performance reasons, the PMIx library does not provide thread protection for accessing the information in the <code>pmix_fabric_t</code> structure. Instead, the PMIx implementation shall provide two methods for coordinating updates to the provided fabric information:

- Users may periodically poll for updates using the PMIx Fabric update API
- Users may register for PMIX_FABRIC_UPDATE_PENDING events indicating that an update to
 the cost matrix is pending. When received, users are required to terminate or pause any actions
 involving access to the cost matrix before returning from the event. Completion of the
 PMIX_FABRIC_UPDATE_PENDING event handler indicates to the PMIx library that the
 fabric object's entries are available for updating. This may include releasing and re-allocating
 memory as the number of vertices may have changed (e.g., due to addition or removal of one or
 more NICs). When the update has been completed, the PMIx library will generate a
 PMIX_FABRIC_UPDATED event indicating that it is safe to begin using the updated fabric
 object(s).

There is no requirement that the caller exclusively use either one of these options. For example, the user may choose to both register for fabric update events, but poll for an update prior to some critical operation.

12.4.2 PMIx_Fabric_register_nb

Summary

Register for access to fabric-related information.

Format

```
PMIx v4.0
```

pmix_status_t

IN fabric

address of a **pmix_fabric_t** (backed by storage). User may populate the "name" field at will - PMIx does not utilize this field (handle)

IN directives

an optional array of values indicating desired behaviors and/or fabric to be accessed. If **NULL**, then the highest priority available fabric will be used (array of handles)

IN 1 ndirs 2 Number of elements in the *directives* array (integer) 3 cbfunc 4 Callback function pmix_op_cbfunc_t (function reference) 5 IN cbdata 6 Data to be passed to the callback function (memory reference) 7 Returns one of the following: 8 • PMIX SUCCESS indicating that the request has been accepted for processing and the provided callback function will be executed upon completion of the operation. Note that the library must 9 not invoke the callback function prior to returning from the API. 10 • a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this 11 12 case, the provided callback function will not be executed **Description** 13 14 Non-blocking form of **PMIx Fabric register**. The caller is not allowed to access the 15 provided pmix_fabric_t until the callback function has been executed, at which time the fabric information will have been loaded into the provided structure. 16 12.4.3 PMIx Fabric update 17 18 Summary 19 Update fabric-related information. **Format** 20 PMIx v4.0 21 pmix_status_t PMIx Fabric_update(pmix_fabric_t *fabric) 22 IN 23 fabric address of a pmix fabric t (backed by storage) (handle) 24 25 Returns **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant. **Description** 26 Update fabric-related information. This call can be made at any time to request an update of the 27 28 fabric information contained in the provided **pmix_fabric_t** object. The caller is not allowed 29 to access the provided **pmix_fabric_t** until the call has returned. 12.4.4 PMIx Fabric update nb Summary 31 32 Update fabric-related information.

```
Format
 1
   PMIx v4.0
 2
               pmix status t
               PMIx Fabric update nb(pmix fabric t *fabric,
 3
                                            pmix op cbfunc t cbfunc, void *cbdata)
 4
               IN
                    fabric
 5
 6
                    address of a pmix fabric t (handle)
 7
               IN
 8
                    Callback function pmix op cbfunc t (function reference)
 9
               IN
                    cbdata
                    Data to be passed to the callback function (memory reference)
10
11
               Returns one of the following:
12
               • PMIX_SUCCESS indicating that the request has been accepted for processing and the provided
13
                 callback function will be executed upon completion of the operation. Note that the library must
                 not invoke the callback function prior to returning from the API.
14
15
               • a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this
                 case, the provided callback function will not be executed
16
17
               Description
               Non-blocking form of PMIx_Fabric_update . The caller is not allowed to access the provided
18
19
               pmix fabric t until the callback function has been executed.
               PMIx_Fabric_deregister
    12.4.5
20
21
               Summary
22
               Deregister a fabric object.
               Format
23
   PMIx v4.0
               pmix_status_t PMIx_Fabric_deregister(pmix_fabric_t *fabric)
24
25
               IN
                    fabric
26
                    address of a pmix fabric t (handle)
27
               Returns PMIX SUCCESS or a negative value corresponding to a PMIx error constant.
               Description
28
29
               Deregister a fabric object, providing an opportunity for the PMIx library to cleanup any information
               (e.g., cost matrix) associated with it. Contents of the provided pmix fabric t will be
30
31
               invalidated upon function return.
```

12.4.6 PMIx_Fabric_deregister_nb Summary Deregister a fabric object. 3 Format PMIx v4.0 5 pmix_status_t PMIx_Fabric_deregister_nb(pmix_fabric_t *fabric, 6 pmix op cbfunc t cbfunc, void *cbdata) IN fabric 7 address of a pmix_fabric_t (handle) 8 9 cbfunc 10 Callback function pmix op cbfunc t (function reference) IN cbdata 11 12 Data to be passed to the callback function (memory reference) 13 Returns one of the following: 14 • PMIX_SUCCESS indicating that the request has been accepted for processing and the provided callback function will be executed upon completion of the operation. Note that the library must 15 not invoke the callback function prior to returning from the API. 16 • a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this 17 case, the provided callback function will not be executed 18 Description 19 20 Non-blocking form of PMIx Fabric deregister. Provided fabric must not be accessed until after callback function has been executed. 21 12.4.7 PMIx Fabric get vertex info Summary 23 24 Given a communication cost matrix index for a specified fabric, return the corresponding vertex 25 info. **Format** 26 PMIx v4.027 pmix status t PMIx_Fabric_get_vertex_info(pmix_fabric_t *fabric, uint32_t index, 28 29 pmix info t **info, size t *ninfo)

1	IN fabric
2	address of a pmix_fabric_t (handle)
3	IN index
4	vertex index (i.e., communication cost matrix row or column number) (integer)
5	INOUT info
6 7	Address where a pointer to an array of pmix_info_t containing the results of the query can be returned (memory reference)
8	INOUT ninfo
9	Address where the number of elements in <i>info</i> can be returned (handle)
10	Returns one of the following:
11	• PMIX_SUCCESS, indicating return of a valid value.
12	• PMIX_ERR_BAD_PARAM, indicating that the provided index is out of bounds.
13	• a PMIx error constant indicating either an error in the input or that the request failed.
14	Description
15	Query information about a specified vertex (fabric device, or NIC) in the system. The returned
16	status indicates if requested data was found or not. The returned array of pmix_info_t will
17	contain information on the specified vertex - the exact contents will depend on the PMIx
18	implementation and the fabric vendor. At a minimum, it must contain sufficient information to
19	uniquely identify the device within the system (ordering is arbitrary):
	▼ Required Attributes
20	<pre>PMIX_HOSTNAME "pmix.hname" (char*)</pre>
21	Name of the host (e.g., where a specified process is running, or a given device is located).
22	The PMIX_NODEID may be returned in its place, or in addition to the hostname.
23	<pre>PMIX_FABRIC_DEVICE_NAME "pmix.fabdev.nm" (string)</pre>
24	The operating system name associated with the device. This may be a logical fabric interface
25	name (e.g. eth0 or eno1) or an absolute filename.
26	<pre>PMIX_FABRIC_DEVICE_VENDOR "pmix.fabdev.vndr" (string)</pre>
27	Indicates the name of the vendor that distributes the NIC.
28	<pre>PMIX_FABRIC_DEVICE_BUS_TYPE "pmix.fabdev.btyp" (string)</pre>
29	The type of bus to which the device is attached (e.g., "PCI", "GEN-Z").

PMIX_FABRIC_DEVICE_PCI_DEVID "pmix.fabdev.pcidevid" (string)

1 2 3	A node-level unique identifier for a PCI device. Provided only if the device is located on a PCI bus. The identifier is constructed as a four-part tuple delimited by colons comprised of the PCI 16-bit domain, 8-bit bus, 8-bit device, and 8-bit function IDs, each expressed in
4	zero-extended hexadecimal form. Thus, an example identifier might be "abc1:0f:23:01". The
5	combination of node identifier (PMIX_HOSTNAME or PMIX_NODEID) and
6	PMIX_FABRIC_DEVICE_PCI_DEVID shall be unique within the system. This item
7	should be included if the device bus type is PCI - the equivalent should be provided for any
8	other bus type.
9	The returned array may optionally contain one or more of the following:
	▼ Optional Attributes
10 11	<pre>PMIX_FABRIC_DEVICE_ID "pmix.fabdev.devid" (string) This is a vendor-provided identifier for the device or product.</pre>
12 13	<pre>PMIX_FABRIC_DEVICE_DRIVER "pmix.fabdev.driver" (string) The name of the driver associated with the device</pre>
14 15	<pre>PMIX_FABRIC_DEVICE_FIRMWARE "pmix.fabdev.fmwr" (string) The device's firmware version</pre>
16 17 18	<pre>PMIX_FABRIC_DEVICE_ADDRESS "pmix.fabdev.addr" (string) The primary link-level address associated with the NIC, such as a MAC address. If multiple addresses are available, only one will be reported.</pre>
19 20	<pre>PMIX_FABRIC_DEVICE_MTU "pmix.fabdev.mtu" (size_t) The maximum transfer unit of link level frames or packets, in bytes.</pre>
21 22	<pre>PMIX_FABRIC_DEVICE_SPEED "pmix.fabdev.speed" (size_t) The active link data rate, given in bits per second.</pre>
23	<pre>PMIX_FABRIC_DEVICE_STATE "pmix.fabdev.state" (pmix_link_state_t)</pre>
24	The last available physical port state. Possible values are PMIX_LINK_STATE_UNKNOWN,
25	PMIX_LINK_DOWN, and PMIX_LINK_UP, to indicate if the port state is unknown or not
26	applicable (unknown), inactive (down), or active (up).
27	<pre>PMIX_FABRIC_DEVICE_TYPE "pmix.fabdev.type" (string)</pre>
28	Specifies the type of fabric interface currently active on the device, such as Ethernet or
29	InfiniBand.

The caller is responsible for releasing the returned array.

12.4.8 PMIx_Fabric_get_vertex_info_nb Summary 2 3 Given a communication cost matrix index for a specified fabric, return the corresponding vertex 4 info. Format 5 PMIx v4.0 pmix status t 6 7 PMIx_Fabric_get_vertex_info_nb(pmix_fabric_t *fabric, uint32_t index, pmix info cbfunc t cbfunc, void *cbdata) 8 9 IN fabric 10 address of a pmix fabric t (handle) IN 11 vertex index (i.e., communication cost matrix row or column number) (integer) 12 cbfunc 13 14 Callback function **pmix** info **cbfunc** t (function reference) cbdata 15 IN Data to be passed to the callback function (memory reference) 16 17 Returns one of the following: 18 • PMIX SUCCESS indicating that the request has been accepted for processing and the provided callback function will be executed upon completion of the operation. Note that the library must 19 not invoke the callback function prior to returning from the API. 20 21 • a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this 22 case, the provided callback function will not be executed 23

Description

Non-blocking form of **PMIx Fabric get vertex info**. Data will be returned in the provided callback function.

12.4.9 PMIx Fabric get device index

Summary 27

24

25

28

Given vertex info, return the corresponding communication cost matrix index.

```
Format
 1
   PMIx v4.0
 2
               pmix status t
               PMIx_Fabric_get_device_index(pmix_fabric_t *fabric,
 3
                                           const pmix info t vertex[], size t ninfo,
 4
                                           uint32_t *index)
 5
 6
                    fabric
 7
                   address of a pmix fabric t (handle)
 8
               IN
                   vertex
 9
                   array of pmix info t containing info describing the vertex whose index is being queried
10
                   (handle)
               IN
                  ninfo
11
                   number of elements in vertex
12
13
               OUT index
                   pointer to the location where the index is to be returned (memory reference (handle))
14
15
               Returns one of the following:
16
               • PMIX_SUCCESS, indicating return of a valid value.
17
               • a PMIx error constant indicating either an error in the input or that the request failed.
18
               Description
19
               Query the index number of a vertex corresponding to the provided description. The description
20
               must provide adequate information to uniquely identify the target vertex. At a minimum, this must
21
               include identification of the node hosting the device using either the PMIX_HOSTNAME or
22
               PMIX_NODEID, plus a node-level unique identifier for the device (e.g., the
23
               PMIX_FABRIC_DEVICE_PCI_DEVID for a PCI device).
    12.4.10
                 PMIx_Fabric_get_device_index_nb
24
25
               Summary
               Given vertex info, return the corresponding communication cost matrix index.
26
               Format
27
   PMIx v4.0
28
               pmix_status_t
29
               PMIx_Fabric_get_device_index_nb(pmix_fabric_t *fabric,
30
                                           const pmix_info_t vertex[], size_t ninfo,
                                           pmix_info_cbfunc_t cbfunc, void *cbdata)
31
```

1	IN	fabric
2		address of a pmix_fabric_t (handle)
3	IN	vertex
4		array of pmix_info_t containing info describing the vertex whose index is being queried
5		(handle)
6	IN	ninfo
7		number of elements in <i>vertex</i>
8	IN	cbfunc
9		Callback function <pre>pmix_info_cbfunc_t</pre> (function reference)
10	IN	cbdata
11		Data to be passed to the callback function (memory reference)
12	Retu	urns one of the following:

Returns one of the following:

- PMIX_SUCCESS indicating that the request has been accepted for processing and the provided callback function will be executed upon completion of the operation. Note that the library must not invoke the callback function prior to returning from the API.
- a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this case, the provided callback function will not be executed

Description

Non-blocking form of PMIx_Fabric_get_device_index . Index will be returned in the provided callback function via the PMIX_FABRIC_INDEX attribute.

13

14

15 16

17

18

19

CHAPTER 13

Process Sets and Groups

PMIx supports two slightly related, but functionally different concepts known as *process sets* and *process groups*. This chapter these two concepts and describes how they are utilized, along with their corresponding APIs.

13.1 Process Sets

A PMIx *Process Set* is a user-provided label associated with a given set of application processes. Definition of a PMIx process set typically occurs at time of application execution - e.g., on a PRRTE command line:

\$ prun -n 4 --pset ocean myoceanapp : -n 3 --pset ice myiceapp

In this example, the processes in the first application will be labeled with a PMIX_PSET_NAME attribute of *ocean* while those in the second application will be labeled with an *ice* value. During the execution, application processes could lookup the process set attribute for any other process using PMIx_Get. Alternatively, other executing applications could utilize the PMIx_Query_info_nb API to obtain the number of declared process sets in the system, a list of their names, and other information about them. In other words, the *process set* identifier provides a label by which an application can derive information about a process and its application - it does *not*, however, confer any operational function.

Thus, process *sets* differ from process *groups* in several key ways:

- Process *sets* have no implied relationship between their members i.e., a process in a process set has no concept of a "pset rank" as it would in a process *group*
- Processes can only have one process *set* identifier, but can simultaneously belong to multiple process *groups*
- Process *set* identifiers are considered job-level information set at launch. No PMIx API is provided by which a user can change the process *set* value of a process on-the-fly. In contrast, PMIx process *groups* can only be defined dynamically by the application.

- Process *groups* can be used in calls to PMIx operations. Members of process *groups* that are involved in an operation are translated by their PMIx server into their *native* identifier prior to the operation being passed to the host environment. For example, an application can define a process group to consist of ranks 0 and 1 from the host-assigned namespace of 210456, identified by the group id of *foo*. If the application subsequently calls the PMIx_Fence API with a process identifier of {foo, PMIX_RANK_WILDCARD}, the PMIx server will replace that identifier with an array consisting of {210456, 0} and {210456, 1} the host-assigned identifiers of the participating processes prior to passing the request up to the host environment
- Process groups can request that the host environment assign a unique size_t PGCID to the
 group at time of group construction. An MPI library may, for example, use the PGCID as the
 MPI communicator identifier for the group.

The two concepts do, however, overlap in one specific area. Process *groups* are included in the process *set* information returned by calls to **PMIx_Query_info_nb**. Thus, a *process group* can effectively be considered an extended version of a *process set* that adds dynamic definition and operational context to the *process set* concept.

Advice to PMIx library implementers

PMIx implementations are required to include all active *group* identifiers in the returned list of process *set* names provided in response to the appropriate PMIx_Query_info_nb call.

13.2 Process Groups

PMIx *Groups* are defined as a collection of processes desiring a common, unique identifier for purposes such as passing events or participating in PMIx fence operations. As with processes that assemble via PMIx_Connect, each member of the group is provided with both the job-level information of any other namespace represented in the group, and the contact information for all group members. However, *groups* differ from PMIx_Connect assemblages in the following key areas:

- Relation to the host environment
 - Calls to PMIx_Connect are relayed to the host environment. This means that the host RM should treat the failure of any process in the specified assemblage as a reportable event and take appropriate action. However, the environment is not required to define a new identifier for the connected assemblage or any of its member processes, nor does it define a new rank for each process within that assemblage. In addition, the PMIx server does not provide any tracking support for the assemblage. Thus, the caller is responsible for addressing members of the connected assemblage using their RM-provided identifiers.

Destruct procedure

Calls to PMIx Group APIs are first processed within the local PMIx server. When constructed, the server creates a tracker that associates the specified processes with the user-provided group identifier, and assigns a new group rank based on their relative position in the array of processes provided in the call to PMIx_Group_construct. Members of the group can subsequently utilize the group identifier in PMIx function calls to address the group's members, using either PMIX_RANK_WILDCARD to refer to all of them or the group-level rank of specific members. The PMIx server will translate the specified processes into their RM-assigned identifiers prior to passing the request up to its host. Thus, the host environment has no visibility into the group's existence or membership.

Advice to users

User-provided group identifiers must be distinct from anything provided by the RM so as to avoid collisions between group identifiers and RM-assigned namespaces. This can usually be accomplished through the use of an application-specific prefix - e.g., "myapp-foo"

• Construction procedure

- PMIx_Connect calls require that every process call the API before completing i.e., it is modeled upon the bulk synchronous traditional MPI connect/accept methodology. Thus, a given application thread can only be involved in one connect/accept operation at a time, and is blocked in that operation until all specified processes participate. In addition, there is no provision for replacing processes in the assemblage due to failure to participate, nor a mechanism by which a process might decline participation.
- PMIx Groups are designed to be more flexible in their construction procedure by relaxing these constraints. While a standard blocking form of constructing groups is provided, the event notification system is utilized to provide a designated *group leader* with the ability to replace participants that fail to participate within a given timeout period. This provides a mechanism by which the application can, if desired, replace members on-the-fly or allow the group to proceed with partial membership. In such cases, the final group membership is returned to all participants upon completion of the operation.

Additionally, PMIx supports dynamic definition of group membership based on an invite/join model. A process can asynchronously initiate construction of a group of any processes via the <code>PMIx_Group_invite</code> function call. Invitations are delivered via a PMIx event (using the <code>PMIX_GROUP_INVITED</code> event) to the invited processes which can then either accept or decline the invitation using the <code>PMIx_Group_join</code> API. The initiating process tracks responses by registering for the events generated by the call to <code>PMIx_Group_join</code>, timeouts, or process terminations, optionally replacing processes that decline the invitation, fail to respond in time, or terminate without responding. Upon completion of the operation, the final list of participants is communicated to each member of the new group.

- Processes that assemble via PMIx_Connect must all depart the assemblage together i.e., no member can depart the assemblage while leaving the remaining members in it. Even the non-blocking form of PMIx_Disconnect retains this requirement in that members remain a part of the assemblage until all members have called PMIx_Disconnect_nb
- Members of a PMIx Group may depart the group at any time via the PMIx_Group_leave API. Other members are notified of the departure via the PMIX_GROUP_LEFT event to distinguish such events from those reporting process termination. This leaves the remaining members free to continue group operations. The PMIx_Group_destruct operation offers a collective method akin to PMIx_Disconnect for deconstructing the entire group.

Note that applications supporting dynamic group behaviors such as asynchronous departure take responsibility for ensuring global consistency in the group definition prior to executing group collective operations - i.e., it is the application's responsibility to either ensure that knowledge of the current group membership is globally consistent across the participants, or to register for appropriate events to deal with the lack of consistency during the operation.

In other words, members of PMIx Groups are *loosely coupled* as opposed to *tightly connected* when constructed via **PMIx_Connect**. The relevant APIs are explained below.

Advice to users

The reliance on PMIx events in the PMIx Group concept dictates that processes utilizing these APIs must register for the corresponding events. Failure to do so will likely lead to operational failures. Users are recommended to utilize the PMIX_TIMEOUT directive (or retain an internal timer) on calls to PMIx Group APIs (especially the blocking form of those functions) as processes that have not registered for required events will never respond.

13.2.1 Group Operation Constants

The **pmix_group_operation_t** structure is an enumerated type for specifying group operations. All values were originally defined in version 4 of the standard unless otherwise marked.

PMIX_GROUP_DECLINE Decline an invitation to join a PMIx group - provided for readability of user code

PMIX_GROUP_ACCEPT Accept an invitation to join a PMIx group - provided for readability of user code

PMIX_GROUP_CONSTRUCT Construct a group composed of the specified processes - used by a PMIx server library to direct host operation

PMIX_GROUP_DESTRUCT Destruct the specified group - used by a PMIx server library to direct host operation

13.2.2 PMIx_Group_construct Summary 2 Construct a PMIx process group 3 Format PMIx v4.0 5 pmix status t 6 PMIx Group construct(const char grp[], const pmix_proc_t procs[], size_t nprocs, 7 const pmix info t directives[], size t ndirs, 8 pmix info t **results, size t *nresults) 9 C 10 IN grp NULL-terminated character array of maximum size PMIX MAX NSLEN containing the 11 group identifier (string) 12 IN procs 13 Array of pmix proc t structures containing the PMIx identifiers of the member processes 14 (array of handles) 15 IN nprocs 16 Number of elements in the *procs* array (size_t) 17 18 IN directives 19 Array of pmix info t structures (array of handles) IN ndirs 20 Number of elements in the *directives* array (size_t) 21 **INOUT** results 22 23 Pointer to a location where the array of **pmix_info_t** describing the results of the operation is to be returned (pointer to handle) 24 **INOUT** nresults 25 Pointer to a size_t location where the number of elements in results is to be returned 26 27 (memory reference) 28 Returns one of the following: 29 • PMIX_SUCCESS, indicating that the request has been successfully completed 30 • PMIX_ERR_NOT_SUPPORTED The PMIx library and/or the host RM does not support this

• a PMIx error constant indicating either an error in the input or that the request failed to be

31

32

33

operation

completed

CHAPTER 13. PROCESS SETS AND GROUPS

	Required Attributes
1 2	The following attributes are <i>required</i> to be supported by all PMIx libraries that support this operation:
3 4	PMIX_GROUP_LEADER "pmix.grp.ldr" (bool) This process is the leader of the group
5 6 7	PMIX_GROUP_OPTIONAL "pmix.grp.opt" (bool) Participation is optional - do not return an error if any of the specified processes terminate without having joined. The default is false
8 9 10 11 12 13 14	PMIX_GROUP_LOCAL_ONLY "pmix.grp.lcl" (bool) Group operation only involves local processes. PMIx implementations are required to automatically scan an array of group members for local vs remote processes - if only local processes are detected, the implementation need not execute a global collective for the operation unless a context ID has been requested from the host environment. This can result in significant time savings. This attribute can be used to optimize the operation by indicating whether or not only local processes are represented, thus allowing the implementation to bypass the scan. The default is false
16	Host environments that support this operation are required to provide the following attributes:
17 18 19 20 21	PMIX_GROUP_ASSIGN_CONTEXT_ID "pmix.grp.actxid" (bool) Requests that the RM assign a new context identifier to the newly created group. The identifier is an unsigned, size_t value that the RM guarantees to be unique across the range specified in the request. Thus, the value serves as a means of identifying the group within that range. If no range is specified, then the request defaults to PMIX_RANGE_SESSION.
22 23 24	PMIX_GROUP_NOTIFY_TERMINATION "pmix.grp.notterm" (bool) Notify remaining members when another member terminates without first leaving the group. The default is false
	▼ Optional Attributes
25	The following attributes are optional for host environments that support this operation:
26 27 28 29	PMIX_TIMEOUT "pmix.timeout" (int) Time in seconds before the specified operation should time out (0 indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Advice to PMIx library implementers -

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Construct a new group composed of the specified processes and identified with the provided group identifier. The group identifier is a user-defined, **NULL**-terminated character array of length less than or equal to **PMIX_MAX_NSLEN**. Only characters accepted by standard string comparison functions (e.g., *strncmp*) are supported. Processes may engage in multiple simultaneous group construct operations so long as each is provided with a unique group ID. The *directives* array can be used to pass user-level directives regarding timeout constraints and other options available from the PMIx server.

If the PMIX_GROUP_NOTIFY_TERMINATION attribute is provided and has a value of true, then either the construct leader (if PMIX_GROUP_LEADER is provided) or all participants who register for the PMIX_GROUP_MEMBER_FAILED event will receive events whenever a process fails or terminates prior to calling PMIX_Group_construct – i.e. if a group leader is declared, only that process will receive the event. In the absence of a declared leader, all specified group members will receive the event.

The event will contain the identifier of the process that failed to join plus any other information that the host RM provided. This provides an opportunity for the leader or the collective members to react to the event – e.g., to decide to proceed with a smaller group or to abort the operation. The decision is communicated to the PMIx library in the results array at the end of the event handler. This allows PMIx to properly adjust accounting for procedure completion. When construct is complete, the participating PMIx servers will be alerted to any change in participants and each group member will receive an updated group membership (marked with the PMIX GROUP MEMBERSHIP attribute) as part of the *results* array returned by this API.

Failure of the declared leader at any time will cause a PMIX_GROUP_LEADER_FAILED event to be delivered to all participants so they can optionally declare a new leader. A new leader is identified by providing the PMIX_GROUP_LEADER attribute in the results array in the return of the event handler. Only one process is allowed to return that attribute, thereby declaring itself as the new leader. Results of the leader selection will be communicated to all participants via a PMIX_GROUP_LEADER_SELECTED event identifying the new leader. If no leader was selected, then the pmix_info_t provided to that event handler will include that information so the participants can take appropriate action.

Any participant that returns **PMIX_GROUP_CONSTRUCT_ABORT** from either the **PMIX_GROUP_MEMBER_FAILED** or the **PMIX_GROUP_LEADER_FAILED** event handler will

cause the construct process to abort, returning from the call with a **PMIX_GROUP_CONSTRUCT_ABORT** status.

If the PMIX_GROUP_NOTIFY_TERMINATION attribute is not provided or has a value of false, then the PMIx_Group_construct operation will simply return an error whenever a proposed group member fails or terminates prior to calling PMIx_Group_construct.

Providing the PMIX_GROUP_OPTIONAL attribute with a value of true directs the PMIx library to consider participation by any specified group member as non-required - thus, the operation will return PMIX_SUCCESS if all members participate, or PMIX_ERR_PARTIAL_SUCCESS if some members fail to participate. The results array will contain the final group membership in the latter case. Note that this use-case can cause the operation to hang if the PMIX_TIMEOUT attribute is not specified and one or more group members fail to call PMIx_Group_construct while continuing to execute. Also, note that no leader or member failed events will be generated during the operation.

Processes in a group under construction are not allowed to leave the group until group construction is complete. Upon completion of the construct procedure, each group member will have access to the job-level information of all namespaces represented in the group plus any information posted via PMIx_Put (subject to the usual scoping directives) for every group member.

——— Advice to PMIx library implementers —

At the conclusion of the construct operation, the PMIx library is *required* to ensure that job-related information from each participating namespace plus any information posted by group members via <code>PMIx_Put</code> (subject to scoping directives) is available to each member via calls to <code>PMIx_Get</code>.

Advice to PMIx server hosts —

The collective nature of this API generally results in use of a fence-like operation by the backend host environment. Host environments that utilize the array of process participants as a *signature* for such operations may experience potential conflicts should both a PMIx_Group_construct and a PMIx_Fence operation involving the same participants be simultaneously executed. As PMIx allows for such use-cases, it is therefore the responsibility of the host environment to resolve any potential conflicts.

13.2.3 PMIx_Group_construct_nb

Summary

Non-blocking form of PMIx_Group_construct

1	Format
PMIx v4.	
2	pmix_status_t
3	<pre>PMIx_Group_construct_nb(const char grp[],</pre>
4	<pre>const pmix_proc_t procs[], size_t nprocs,</pre>
5	<pre>const pmix_info_t directives[], size_t ndire</pre>
6	<pre>pmix_info_cbfunc_t cbfunc, void *cbdata)</pre>
	C —
7	IN grp
8	NULL-terminated character array of maximum size PMIX_MAX_NSLEN containing the
9	group identifier (string)
10	IN procs
11	Array of pmix_proc_t structures containing the PMIx identifiers of the member processes
12	(array of handles)
13	IN nprocs
14	Number of elements in the <i>procs</i> array (size_t)
15	IN directives
16	Array of pmix_info_t structures (array of handles)
17	IN ndirs
18	Number of elements in the <i>directives</i> array (size_t)
19	IN cbfunc
20	Callback function pmix_info_cbfunc_t (function reference)
21	IN cbdata
22	Data to be passed to the callback function (memory reference)
23	Returns one of the following:
24	• PMIX_SUCCESS indicating that the request has been accepted for processing and the provided
25	callback function will be executed upon completion of the operation. Note that the library must
26	not invoke the callback function prior to returning from the API.
27	• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
28	returned success - the cbfunc will not be called
	·
29	• PMIX_ERR_NOT_SUPPORTED The PMIx library does not support this operation - the <i>cbfunc</i>
30	will <i>not</i> be called
31	• a non-zero PMIx error constant indicating a reason for the request to have been rejected - the
32	cbfunc will not be called
33	If executed, the status returned in the provided callback function will be one of the following
34	constants:
35	• PMIX_SUCCESS The operation succeeded and all specified members participated.

1 2	 PMIX_ERR_PARTIAL_SUCCESS The operation succeeded but not all specified members participated - the final group membership is included in the callback function
3 4	• PMIX_ERR_NOT_SUPPORTED While the PMIx server supports this operation, the host RM does not.
5	• a non-zero PMIx error constant indicating a reason for the request's failure
	Required Attributes
6 7	PMIx libraries that choose not to support this operation <i>must</i> return PMIX_ERR_NOT_SUPPORTED when the function is called.
8 9	The following attributes are <i>required</i> to be supported by all PMIx libraries that support this operation:
10 11	PMIX_GROUP_LEADER "pmix.grp.ldr" (bool) This process is the leader of the group
12 13 14	<pre>PMIX_GROUP_OPTIONAL "pmix.grp.opt" (bool) Participation is optional - do not return an error if any of the specified processes terminate without having joined. The default is false</pre>
15 16 17 18 19 20 21	PMIX_GROUP_LOCAL_ONLY "pmix.grp.lcl" (bool) Group operation only involves local processes. PMIx implementations are required to automatically scan an array of group members for local vs remote processes - if only local processes are detected, the implementation need not execute a global collective for the operation unless a context ID has been requested from the host environment. This can result in significant time savings. This attribute can be used to optimize the operation by indicating whether or not only local processes are represented, thus allowing the implementation to bypass the scan. The default is false
23	Host environments that support this operation are required to provide the following attributes:
24 25 26 27 28	PMIX_GROUP_ASSIGN_CONTEXT_ID "pmix.grp.actxid" (bool) Requests that the RM assign a new context identifier to the newly created group. The identifier is an unsigned, size_t value that the RM guarantees to be unique across the range specified in the request. Thus, the value serves as a means of identifying the group within that range. If no range is specified, then the request defaults to PMIX_RANGE_SESSION.
29 30 31	PMIX_GROUP_NOTIFY_TERMINATION "pmix.grp.notterm" (bool) Notify remaining members when another member terminates without first leaving the group. The default is false

		▼ Optional Attributes
1		The following attributes are optional for host environments that support this operation:
2		<pre>PMIX_TIMEOUT "pmix.timeout" (int)</pre>
3		Time in seconds before the specified operation should time out (θ indicating infinite) in
4		error. The timeout parameter can help avoid "hangs" due to programming errors that prevent
5		the target process from ever exposing its data.
		^
		Advice to PMIx library implementers —
6		We recommend that implementation of the PMIX_TIMEOUT attribute be left to the host
7		environment due to race condition considerations between completion of the operation versus
8		internal timeout in the PMIx server library. Implementers that choose to support PMIX_TIMEOUT
9		directly in the PMIx server library must take care to resolve the race condition and should avoid
10		passing PMIX_TIMEOUT to the host environment so that multiple competing timeouts are not
11		created.
		Description
12		Description
13 14		Non-blocking version of the PMIx_Group_construct operation. The callback function will be called once all group members have called either PMIx_Group_construct or
15		PMIx_Group_construct_nb.
13		FMIX_GIOUP_CONSCIUCC_ND.
16	13.2.4	PMIx_Group_destruct
17		Summary
18		Destruct a PMIx process group

	Format
PMIx v4.0	▼
	pmix_status_t
	<pre>PMIx_Group_destruct(const char grp[],</pre>
	<pre>const pmix_info_t directives[], size_t ndirs)</pre>
	<u> </u>
	IN grp
	NULL-terminated character array of maximum size PMIX_MAX_NSLEN containing the
	identifier of the group to be destructed (string)
	IN directives
	Array of pmix_info_t structures (array of handles) IN ndirs
	Number of elements in the <i>directives</i> array (size_t)
	Returns one of the following:
	• PMIX_SUCCESS, indicating that the request has been successfully completed
	• PMIX_ERR_NOT_SUPPORTED The PMIx library and/or the host RM does not support this
	operation
	• a PMIx error constant indicating either an error in the input or that the request failed to be completed
	Required Attributes
	For implementations and host environments that support the operation, there are no identified
	required attributes for this API.
	A
	▼Optional Attributes
	The following attributes are optional for host environments that support this operation:
	PMIX_TIMEOUT "pmix.timeout" (int)
	Time in seconds before the specified operation should time out (θ indicating infinite) in
	error. The timeout parameter can help avoid "hangs" due to programming errors that prevent
	the target process from ever exposing its data.
	PMIx v4.0

Advice to PMIx library implementers -

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Destruct a group identified by the provided group identifier. Processes may engage in multiple simultaneous group destruct operations so long as each involves a unique group ID. The *directives* array can be used to pass user-level directives regarding timeout constraints and other options available from the PMIx server.

The destruct API will return an error if any group process fails or terminates prior to calling PMIx_Group_destruct or its non-blocking version unless the PMIX_GROUP_NOTIFY_TERMINATION attribute was provided (with a value of false) at time of group construction. If notification was requested, then the PMIX_GROUP_MEMBER_FAILED event will be delivered for each process that fails to call destruct and the destruct tracker updated to account for the lack of participation. The PMIX_Group_destruct operation will subsequently return PMIX_SUCCESS when the remaining processes have all called destruct – i.e., the event will serve in place of return of an error.

Advice to PMIx server hosts -

The collective nature of this API generally results in use of a fence-like operation by the backend host environment. Host environments that utilize the array of process participants as a *signature* for such operations may experience potential conflicts should both a PMIx_Group_destruct and a PMIx_Fence operation involving the same participants be simultaneously executed. As PMIx allows for such use-cases, it is therefore the responsibility of the host environment to resolve any potential conflicts.

13.2.5 PMIx_Group_destruct_nb

Summary

Non-blocking form of PMIx_Group_destruct

1	Format
PMIx v	4.0
2	<pre>pmix_status_t</pre>
3	<pre>PMIx_Group_destruct_nb(const char grp[],</pre>
4	<pre>const pmix_info_t directives[], size_t ndirs</pre>
5	<pre>pmix_op_cbfunc_t cbfunc, void *cbdata)</pre>
	C
6	IN grp
7	NULL-terminated character array of maximum size PMIX_MAX_NSLEN containing the
8	identifier of the group to be destructed (string)
9	IN directives
10	Array of pmix_info_t structures (array of handles)
11	IN ndirs
12	Number of elements in the <i>directives</i> array (size_t)
13	IN cbfunc
14	Callback function pmix_op_cbfunc_t (function reference)
15	IN cbdata
16	Data to be passed to the callback function (memory reference)
17	Returns one of the following:
18	• PMIX_SUCCESS, indicating that the request is being processed - result will be returned in the
19	provided cbfunc. Note that the library must not invoke the callback function prior to returning
20	from the API.
21	• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
22	returned <i>success</i> - the <i>cbfunc</i> will <i>not</i> be called
23	• PMIX_ERR_NOT_SUPPORTED The PMIx library does not support this operation - the <i>cbfunc</i>
24	will <i>not</i> be called
25	• a PMIx error constant indicating either an error in the input or that the request was immediately
26	processed and failed - the <i>cbfunc</i> will <i>not</i> be called
27 28	If executed, the status returned in the provided callback function will be one of the following constants:
20	Collstants.
29	• PMIX_SUCCESS The operation was successfully completed
30	• PMIX_ERR_NOT_SUPPORTED While the PMIx server supports this operation, the host RM
31	does not.
32	• a non-zero PMIx error constant indicating a reason for the request's failure

		Required Attributes
1 2 3		PMIx libraries that choose not to support this operation <i>must</i> return PMIX_ERR_NOT_SUPPORTED when the function is called. For implementations and host environments that support the operation, there are no identified required attributes for this API.
		▼Optional Attributes
4		The following attributes are optional for host environments that support this operation:
5 6 7 8		PMIX_TIMEOUT "pmix.timeout" (int) Time in seconds before the specified operation should time out (0 indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.
		Advice to PMIx library implementers —
9 10 11 12		We recommend that implementation of the PMIX_TIMEOUT attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support PMIX_TIMEOUT directly in the PMIx server library must take care to resolve the race condition and should avoid passing PMIX_TIMEOUT to the host environment so that multiple competing timeouts are not
14		created.
15 16 17 18		Description Non-blocking version of the PMIx_Group_destruct operation. The callback function will be called once all members of the group have executed either PMIx_Group_destruct or PMIx_Group_destruct_nb.
19	13.2.6	PMIx_Group_invite
20		Summary
21		Asynchronously construct a PMIx process group

1		Format
	<i>PMIx v4.0</i>	
2		pmix_status_t
3		<pre>PMIx_Group_invite(const char grp[],</pre>
4		<pre>const pmix_proc_t procs[], size_t nprocs,</pre>
5		<pre>const pmix_info_t directives[], size_t ndirs,</pre>
6		<pre>pmix_info_t **results, size_t *nresult)</pre>
		C
7		IN grp
8		NULL-terminated character array of maximum size PMIX_MAX_NSLEN containing the
9		group identifier (string)
10		IN procs
11		Array of pmix_proc_t structures containing the PMIx identifiers of the processes to be
12		invited (array of handles)
13		IN nprocs
14		Number of elements in the <i>procs</i> array (size_t)
15 16		IN directives Array of pmix_info_t structures (array of handles)
17		IN ndirs
18		Number of elements in the <i>directives</i> array (size_t)
19		INOUT results
20		Pointer to a location where the array of pmix_info_t describing the results of the
21		operation is to be returned (pointer to handle)
22		INOUT nresults
23		Pointer to a size_t location where the number of elements in <i>results</i> is to be returned
24		(memory reference)
25		Returns one of the following:
26		• PMIX_SUCCESS, indicating that the request has been successfully completed
27		• PMIX_ERR_NOT_SUPPORTED The PMIx library and/or the host RM does not support this
28		operation
29		• a PMIx error constant indicating either an error in the input or that the request failed to be
30		completed
		D. C. LANDE
		Required Attributes
31		The following attributes are required to be supported by all PMIx libraries that support this
32		operation:
33		PMIX_GROUP_OPTIONAL "pmix.grp.opt" (bool)
34		Participation is optional - do not return an error if any of the specified processes terminate
35		without having joined. The default is false

1	Host environments that support this operation are <i>required</i> to provide the following attributes:
2 3 4 5 6	PMIX_GROUP_ASSIGN_CONTEXT_ID "pmix.grp.actxid" (bool) Requests that the RM assign a new context identifier to the newly created group. The identifier is an unsigned, size_t value that the RM guarantees to be unique across the range specified in the request. Thus, the value serves as a means of identifying the group within that range. If no range is specified, then the request defaults to PMIX_RANGE_SESSION.
7 8 9	PMIX_GROUP_NOTIFY_TERMINATION "pmix.grp.notterm" (bool) Notify remaining members when another member terminates without first leaving the group. The default is false
	▼ Optional Attributes
10	The following attributes are optional for host environments that support this operation:
11 12 13 14	PMIX_TIMEOUT "pmix.timeout" (int) Time in seconds before the specified operation should time out (0 indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.
	Advice to PMIx library implementers
15 16 17 18 19 20	We recommend that implementation of the PMIX_TIMEOUT attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support PMIX_TIMEOUT directly in the PMIx server library must take care to resolve the race condition and should avoid passing PMIX_TIMEOUT to the host environment so that multiple competing timeouts are not created.

Description

 Explicitly invite the specified processes to join a group. The process making the
PMIx_Group_invite call is automatically declared to be the group leader. Each invited
process will be notified of the invitation via the PMIX_GROUP_INVITED event - the processes
being invited must therefore register for the PMIX_GROUP_INVITED event in order to be notified
of the invitation. Note that the PMIx event notification system caches events - thus, no ordering of
invite versus event registration is required.

The invitation event will include the identity of the inviting process plus the name of the group. When ready to respond, each invited process provides a response using either the blocking or non-blocking form of <code>PMIx_Group_join</code>. This will notify the inviting process that the invitation was either accepted (via the <code>PMIX_GROUP_INVITE_ACCEPTED</code> event) or declined (via the <code>PMIX_GROUP_INVITE_DECLINED</code> event). The <code>PMIX_GROUP_INVITE_ACCEPTED</code> event is captured by the PMIx client library of the inviting process – i.e., the application itself does not need to register for this event. The library will track the number of accepting processes and alert the inviting process (by returning from the blocking form of <code>PMIx_Group_invite</code> or calling the callback function of the non-blocking form) when group construction completes.

The inviting process should, however, register for the PMIX_GROUP_INVITE_DECLINED if the application allows invited processes to decline the invitation. This provides an opportunity for the application to either invite a replacement, declare "abort", or choose to remove the declining process from the final group. The inviting process should also register to receive PMIX_GROUP_INVITE_FAILED events whenever a process fails or terminates prior to responding to the invitation. Actions taken by the inviting process in response to these events must be communicated at the end of the event handler by returning the corresponding result so that the PMIx library can adjust accordingly.

Upon completion of the operation, all members of the new group will receive access to the job-level information of each other's namespaces plus any information posted via **PMIx_Put** by the other members.

The inviting process is automatically considered the leader of the asynchronous group construction procedure and will receive all failure or termination events for invited members prior to completion. The inviting process is required to provide a **PMIX_GROUP_CONSTRUCT_COMPLETE** event once the group has been fully assembled – this event is used by the PMIx library as a trigger to release participants from their call to **PMIx_Group_join** and provides information (e.g., the final group membership) to be returned in the *results* array.

Advice to users

Applications are not allowed to use the group in any operations until group construction is complete. This is required in order to ensure consistent knowledge of group membership across all participants.

Failure of the inviting process at any time will cause a PMIX_GROUP_LEADER_FAILED event to be delivered to all participants so they can optionally declare a new leader. A new leader is identified by providing the PMIX_GROUP_LEADER attribute in the results array in the return of the event handler. Only one process is allowed to return that attribute, declaring itself as the new leader. Results of the leader selection will be communicated to all participants via a PMIX_GROUP_LEADER_SELECTED event identifying the new leader. If no leader was selected, then the status code provided in the event handler will provide an error value so the participants can take appropriate action.

13.2.7 PMIx_Group_invite_nb

Returns one of the following:

1

2

4

5

6 7

8

34

10 Summary 11 Non-blocking form of PMIx_Group_invite **Format** 12 PMIx v4.013 pmix status t PMIx_Group_invite_nb(const char grp[], 14 const pmix_proc_t procs[], size_t nprocs, 15 const pmix_info_t directives[], size_t ndirs, 16 pmix info cbfunc t cbfunc, void *cbdata) 17 C IN 18 grp 19 NULL-terminated character array of maximum size PMIX MAX NSLEN containing the group identifier (string) 20 IN procs 21 Array of pmix proc t structures containing the PMIx identifiers of the processes to be 22 invited (array of handles) 23 24 IN nprocs Number of elements in the *procs* array (size_t) 25 26 IN directives 27 Array of pmix_info_t structures (array of handles) IN ndirs 28 29 Number of elements in the *directives* array (size_t) IN cbfunc 30 31 Callback function pmix_info_cbfunc_t (function reference) 32 IN cbdata 33 Data to be passed to the callback function (memory reference)

1 2 3	• PMIX_SUCCESS, indicating that the request is being processed - result will be returned in the provided <i>cbfunc</i> . Note that the library <i>must not</i> invoke the callback function prior to returning from the API.
4 5	• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will <i>not</i> be called
6 7	• PMIX_ERR_NOT_SUPPORTED The PMIx library does not support this operation - the <i>cbfunc</i> will <i>not</i> be called
8 9	• a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will <i>not</i> be called
10 11	If executed, the status returned in the provided callback function will be one of the following constants:
12	• PMIX_SUCCESS The operation succeeded and all specified members participated.
13 14	• PMIX_ERR_PARTIAL_SUCCESS The operation succeeded but not all specified members participated - the final group membership is included in the callback function
15 16	• PMIX_ERR_NOT_SUPPORTED While the PMIx server supports this operation, the host RM does not.
17	• a non-zero PMIx error constant indicating a reason for the request's failure
	▼ Required Attributes
18 19	The following attributes are <i>required</i> to be supported by all PMIx libraries that support this operation:
20 21 22	PMIX_GROUP_OPTIONAL "pmix.grp.opt" (bool) Participation is optional - do not return an error if any of the specified processes terminate without having joined. The default is false
23	Host environments that support this operation are <i>required</i> to provide the following attributes:
24 25 26 27 28	PMIX_GROUP_ASSIGN_CONTEXT_ID "pmix.grp.actxid" (bool) Requests that the RM assign a new context identifier to the newly created group. The identifier is an unsigned, size_t value that the RM guarantees to be unique across the range specified in the request. Thus, the value serves as a means of identifying the group within that range. If no range is specified, then the request defaults to PMIX_RANGE_SESSION.
29 30 31	PMIX_GROUP_NOTIFY_TERMINATION "pmix.grp.notterm" (bool) Notify remaining members when another member terminates without first leaving the group. The default is false

		The stricture of the st
1		The following attributes are optional for host environments that support this operation:
2		PMIX_TIMEOUT "pmix.timeout" (int)
3		Time in seconds before the specified operation should time out (0 indicating infinite) in
4		error. The timeout parameter can help avoid "hangs" due to programming errors that prevent
5		the target process from ever exposing its data.
		^
		Advice to PMIx library implementers
6		We recommend that implementation of the PMIX_TIMEOUT attribute be left to the host
7		environment due to race condition considerations between completion of the operation versus
8		internal timeout in the PMIx server library. Implementers that choose to support PMIX_TIMEOUT
9		directly in the PMIx server library must take care to resolve the race condition and should avoid
10		passing PMIX_TIMEOUT to the host environment so that multiple competing timeouts are not
11		created.
12		Description
13		Non-blocking version of the PMIx_Group_invite operation. The callback function will be
14		called once all invited members of the group (or their substitutes) have executed either
15		PMIx_Group_join or PMIx_Group_join_nb.
16	13.2.8	PMIx_Group_join
17		Summary
18		Accept an invitation to join a PMIx process group

1		Format
	<i>PMIx v4.0</i>	· · · · · · · · · · · · · · · · · · ·
2		pmix_status_t
3		<pre>PMIx_Group_join(const char grp[],</pre>
4		<pre>const pmix_proc_t *leader,</pre>
5		<pre>pmix_group_operation_t opt,</pre>
6		<pre>const pmix_info_t directives[], size_t ndirs,</pre>
7		<pre>pmix_info_t **results, size_t *nresult)</pre>
		C -
8		IN grp
9		NULL-terminated character array of maximum size PMIX_MAX_NSLEN containing the
10		group identifier (string)
11		IN leader
12		Process that generated the invitation (handle)
13		IN opt
14		Accept or decline flag (pmix_group_operation_t)
15		IN directives
16		Array of pmix_info_t structures (array of handles)
17		IN ndirs
18		Number of elements in the <i>directives</i> array (size_t)
19		INOUT results
20		Pointer to a location where the array of pmix_info_t describing the results of the
21		operation is to be returned (pointer to handle)
22		INOUT nresults
23		Pointer to a size_t location where the number of elements in <i>results</i> is to be returned
24		(memory reference)
25		Returns one of the following:
26		• PMIX_SUCCESS, indicating that the request has been successfully completed
27		• PMIX_ERR_NOT_SUPPORTED The PMIx library and/or the host RM does not support this
28		operation
29		• a PMIx error constant indicating either an error in the input or that the request failed to be
30		completed
		Required Attributes
		•
31		There are no identified required attributes for implementers.
		

Optional Attributes The following attributes are optional for host environments that support this operation: PMIX TIMEOUT "pmix.timeout" (int) Time in seconds before the specified operation should time out (θ indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data. Advice to PMIx library implementers ——— We recommend that implementation of the PMIX_TIMEOUT attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support PMIX_TIMEOUT directly in the PMIx server library must take care to resolve the race condition and should avoid passing PMIX TIMEOUT to the host environment so that multiple competing timeouts are not created. **Description** Respond to an invitation to join a group that is being asynchronously constructed. The process must have registered for the PMIX GROUP INVITED event in order to be notified of the invitation. When called, the event information will include the pmix_proc_t identifier of the process that generated the invitation along with the identifier of the group being constructed. When ready to respond, the process provides a response using either form of **PMIx Group** join. Advice to users Since the process is alerted to the invitation in a PMIx event handler, the process must not use the

blocking form of this call unless it first "thread shifts" out of the handler and into its own thread

the process *must not* block in the handler while waiting for the callback function to be called.

context. Likewise, while it is safe to call the non-blocking form of the API from the event handler,

1 2

3

4

5

6 7

8

9 10

11

12 13

14

15

16 17

18

19

20 21

CHAPTER 13. PROCESS SETS AND GROUPS

Calling this function causes the inviting process (aka the *group leader*) to be notified that the process has either accepted or declined the request. The blocking form of the API will return once the group has been completely constructed or the group's construction has failed (as described below) – likewise, the callback function of the non-blocking form will be executed upon the same conditions.

Failure of the leader during the call to PMIx_Group_join will cause a

PMIX_GROUP_LEADER_FAILED event to be delivered to all invited participants so they can optionally declare a new leader. A new leader is identified by providing the

PMIX_GROUP_LEADER attribute in the results array in the return of the event handler. Only one process is allowed to return that attribute, declaring itself as the new leader. Results of the leader selection will be communicated to all participants via a PMIX_GROUP_LEADER_SELECTED event identifying the new leader. If no leader was selected, then the status code provided in the event handler will provide an error value so the participants can take appropriate action.

Any participant that returns PMIX_GROUP_CONSTRUCT_ABORT from the leader failed event handler will cause all participants to receive an event notifying them of that status. Similarly, the leader may elect to abort the procedure by either returning PMIX_GROUP_CONSTRUCT_ABORT from the handler assigned to the PMIX_GROUP_INVITE_ACCEPTED or PMIX_GROUP_INVITE_DECLINED codes, or by generating an event for the abort code. Abort events will be sent to all invited participants.

13.2.9 PMIx_Group_join_nb

Summary

Non-blocking form of PMIx Group join

Format

PMIx v4.0

 pmix_status_t
PMIx_Group_join_nb(const char grp[],

const pmix_proc_t *leader,

pmix_group_operation_t opt,
 const pmix_info_t directives[], size_t ndirs,

pmix_info_cbfunc_t cbfunc, void *cbdata)

IN grp

NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the group identifier (string)

C

IN leader

Process that generated the invitation (handle)

1	na ope
2	Accept or decline flag (pmix_group_operation_t)
3	IN directives
4	Array of pmix_info_t structures (array of handles)
5	IN ndirs
6	Number of elements in the <i>directives</i> array (size_t)
7	IN cbfunc
8	Callback function pmix_info_cbfunc_t (function reference)
9	IN cbdata
10	Data to be passed to the callback function (memory reference)
11	Returns one of the following:
12 13 14	• PMIX_SUCCESS, indicating that the request is being processed - result will be returned in the provided <i>cbfunc</i> . Note that the library <i>must not</i> invoke the callback function prior to returning from the API.
15 16	• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will <i>not</i> be called
17 18	• PMIX_ERR_NOT_SUPPORTED The PMIx library does not support this operation - the <i>cbfunc</i> will <i>not</i> be called
19 20	• a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will <i>not</i> be called
21 22	If executed, the status returned in the provided callback function will be one of the following constants:
23 24	• PMIX_SUCCESS The operation succeeded and group membership is in the callback function parameters
25 26	• PMIX_ERR_NOT_SUPPORTED While the PMIx server supports this operation, the host RM does not.
27	• a non-zero PMIx error constant indicating a reason for the request's failure
_,	•
	Required Attributes
28	There are no identified required attributes for implementers.
	▼Optional Attributes
29	The following attributes are optional for host environments that support this operation:
30	PMIX_TIMEOUT "pmix.timeout" (int)
31	Time in seconds before the specified operation should time out (θ indicating infinite) in
32	error. The timeout parameter can help avoid "hangs" due to programming errors that prevent
33	the target process from ever exposing its data.
	· · · · · · · · · · · · · · · · ·

_______ Advice to PMIx library implementers We recommend that implementation of the PMIX_TIMEOUT attribute be left to the host 1 2 environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support PMIX TIMEOUT 3 directly in the PMIx server library must take care to resolve the race condition and should avoid 4 passing PMIX TIMEOUT to the host environment so that multiple competing timeouts are not 5 created. 6 **Description** 7 8 Non-blocking version of the PMIx_Group_join operation. The callback function will be called 9 once all invited members of the group (or their substitutes) have executed either PMIx_Group_join or PMIx_Group_join_nb. 10 13.2.10 PMIx Group leave 11 12 Summary 13 Leave a PMIx process group Format 14 PMIx v4.0 15 pmix status t PMIx Group leave(const char grp[], 16 const pmix_info_t directives[], size_t ndirs) 17 18 IN grp NULL-terminated character array of maximum size PMIX MAX NSLEN containing the 19 group identifier (string) 20 IN directives 21 22 Array of pmix info t structures (array of handles) 23 IN Number of elements in the *directives* array (size t) 24 25 Returns one of the following: 26 • PMIX SUCCESS, indicating that the request has been communicated to the local PMIx server • PMIX ERR NOT_SUPPORTED The PMIx library and/or the host RM does not support this 27 operation 28 29 • a PMIx error constant indicating either an error in the input or that the request is unsupported Required Attributes There are no identified required attributes for implementers. 30

Description

Leave a PMIx Group. Calls to PMIx_Group_leave (or its non-blocking form) will cause a PMIX_GROUP_LEFT event to be generated notifying all members of the group of the caller's departure. The function will return (or the non-blocking function will execute the specified callback function) once the event has been locally generated and is not indicative of remote receipt.

Advice to users

The PMIx_Group_leave API is intended solely for asynchronous departures of individual processes from a group as it is not a scalable operation – i.e., when a process determines it should no longer be a part of a defined group, but the remainder of the group retains a valid reason to continue in existence. Developers are advised to use PMIx_Group_destruct (or its non-blocking form) for all other scenarios as it represents a more scalable operation.

13.2.11 PMIx Group leave nb

Summary

Non-blocking form of PMIx_Group_leave

Format

PMIx v4.0

1

3

5

6 7

8

9

10

12

13

14

15

16

17

18

19

20

21

22

23 24

25

26

27 28

29

30

31

32

33

C

IN grp

NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the group identifier (string)

IN directives

Array of pmix_info_t structures (array of handles)

IN ndirs

Number of elements in the *directives* array (size t)

IN cbfunc

Callback function **pmix_op_cbfunc_t** (function reference)

IN cbdata

Data to be passed to the callback function (memory reference)

Returns one of the following:

• PMIX_SUCCESS, indicating that the request is being processed - result will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning from the API.

1 2	• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will <i>not</i> be called
3 4	• PMIX_ERR_NOT_SUPPORTED The PMIx library does not support this operation - the <i>cbfunc</i> will <i>not</i> be called
5 6	• a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will <i>not</i> be called
7 8	If executed, the status returned in the provided callback function will be one of the following constants:
9 0	 PMIX_SUCCESS The operation succeeded - i.e., the PMIX_GROUP_LEFT event was generated
1 2	• PMIX_ERR_NOT_SUPPORTED While the PMIx library supports this operation, the host RM does not.
3	• a non-zero PMIx error constant indicating a reason for the request's failure
	▼ Required Attributes
4	There are no identified required attributes for implementers.
5	Description
6	Non-blocking version of the PMIx_Group_leave operation. The callback function will be
7	called once the event has been locally generated and is not indicative of remote receipt.

CHAPTER 14

Tools and Debuggers

The term *tool* widely refers to non-computational programs executed by the user or system administrator on a command line. Tools almost always interact with either the SMS, user applications, or both to perform administrative and support functions. For example, a debugger tool might be used to remotely control the processes of a parallel application, monitoring their behavior on a step-by-step basis. Historically, such tools were custom-written for each specific host environment due to the customized and/or proprietary nature of the environment's interfaces.

The advent of PMIx offers the possibility for creating portable tools capable of interacting with multiple RMs without modification. Possible use-cases include:

- querying the status of scheduling queues and estimated allocation time for various resource options
- job submission and allocation requests
- querying job status for executing applications
- launching and monitoring applications

Enabling these capabilities requires some extensions to the PMIx Standard (both in terms of APIs and attributes), and utilization of client-side APIs for more tool-oriented purposes.

This chapter defines specific APIs related to tools, provides tool developers with an overview of the support provided by PMIx, and serves to guide RM vendors regarding roles and responsibilities of RMs to support tools. As the number of tool-specific APIs and attributes is fairly small, the bulk of the chapter serves to provide a "theory of operation" for tools and debuggers. Description of the APIs themselves is therefore deferred to the Section 14.5 later in the chapter.

14.1 Connection Mechanisms

The key to supporting tools lies in providing mechanisms by which a tool can connect to a PMIx server. Application processes are able to connect because their local RM daemon provides them with the necessary contact information upon execution. A command-line tool, however, isn't spawned by an RM daemon, and therefore lacks the information required for rendezvous with a PMIx server.

Support for tools requires that the PMIx server be initialized with an appropriate attribute indicating that tool connections are to be allowed. Separate attributes are provided to "fine-tune" this permission by allowing the environment to independently enable (or disable) connections from

10

11

12

13

14

15

16

17

18 19

20

21 22

23

24

25

26

tools executing on nodes other than the one hosting the server itself. The PMIx server library shall provide an opportunity for the host environment to authenticate and approve each connection request from a specific tool by calling the **pmix server tool connection fn t** "hook" provided in the server module for that purpose. Servers in environments that do not provide this "hook" shall automatically reject all tool connection requests.

Tools can connect to any local or remote PMIx server provided they are either explicitly given the required connection information, or are able to discover it via one of several defined rendezvous protocols. Connection discovery centers around the existence of rendezvous files containing the necessary connection information, as illustrated in Fig. 14.1.

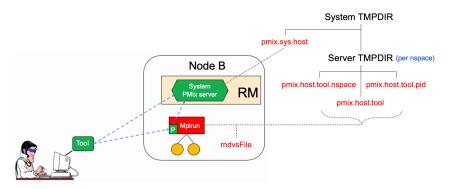


Figure 14.1.: Tool rendezvous files

The contents of each rendezvous file are specific to a given PMIx implementation, but should at least contain the namespace and rank of the server along with its connection URI. Note that tools linked to one PMIx implementation are therefore unlikely to successfully connect to PMIx server libraries from another implementation.

The top of the directory tree is defined by either the **PMIX_SYSTEM_TMPDIR** attribute (if given) or the **TMPDIR** environmental variable. PMIx servers that are designated as *system servers* by including the PMIX_SERVER_SYSTEM_SUPPORT attribute when calling **PMIx_server_init** will create a rendezvous file in this top-level directory. The filename will be of the form *pmix.sys.hostname*, where *hostname* is the string returned by the **gethostname** system call. Note that only one PMIx server on a node can be designated as the system server.

Non-system PMIx servers will create a set of three rendezvous files in the directory defined by either the PMIX SERVER TMPDIR attribute or the TMPDIR environmental variable:

- pmix.host.tool.nspace where host is the string returned by the **gethostname** system call and nspace is the namespace of the server
- pmix.host.tool.pid where host is the string returned by the **gethostname** system call and pid is the PID of the server
- *pmix.host.tool* where *host* is the string returned by the **gethostname** system call. Note that

servers which are not given a namespace-specific **PMIX_SERVER_TMPDIR** attribute may not generate this file due to conflicts should multiple servers be present on the node.

The files are identical and may be implemented as symlinks to a single instance. The individual file names are composed so as to aid the search process should a tool wish to connect to a server identified by its namespace or PID.

Servers will additionally provide a rendezvous file in any given location if the path (either absolute or relative) and filename is specified either during **PMIx_server_init** using the **PMIX_LAUNCHER_RENDEZVOUS_FILE** attribute, or by the

PMIX_LAUNCHER_RENDEZVOUS_FILE environmental variable prior to executing the process containing the server. This latter mechanism may be the preferred mechanism for tools such as debuggers that need to fork/exec a launcher (e.g., "mpiexec") and then rendezvous with it. This is described in more detail in Section 14.2.2.

Rendezvous file ownerships are set to the User ID (UID) and Group ID (GID) of the server that created them, with permissions set according to the desires of the implementation and/or system administrator policy. All connection attempts are first governed by read access privileges to the target rendezvous file - thus, the combination of permissions, UID, and GID of the rendezvous files act as a first-level of security for tool access.

Tools are limited to only one connection to a server at a time. This is done to avoid confusion when the tool calls an API as to which server should service the request. Tools are, however, allowed to disconnect from one server and reconnect to another as many times as desired. Note that standing requests (e.g., event registrations) with a server will be lost when transferring to another server - PMIx implementors are not required to transfer state across tool-server connections.

Tool process identifiers are assigned by one of the following methods:

- If **PMIX_TOOL_NSPACE** is given, then the namespace of the tool will be assigned that value
- If PMIX_TOOL_RANK is given, then the rank of the tool will be assigned that value. Passing PMIX_TOOL_NSPACE without also specifying the rank will result in the rank being set to a default value of zero
- If a process ID has not been provided, then one will be assigned by the host environment upon connection to a server. Users should note that the tool's process ID will be *invalid* until a connection has been established

Tool process identifiers remain constant when connection is transferred between servers. Thus, it is critical that a system-wide unique namespace be provided if the tool itself sets the identifier, and that host environments provide a system-wide unique identifier in the case where the identifier is set by the server upon connection.

For simplicity, the following descriptions will refer to the:

• **PMIX_SYSTEM_TMPDIR** as the directory specified by either the **PMIX_SYSTEM_TMPDIR** attribute (if given) or the **TMPDIR** environmental variable.

Rendezvousing with a local server

Connection to a local PMIx server is pursued according to the following precedence chain based on attributes contained in the call to the PMIx_tool_init API. Except where noted, the PMIx library will return an error if the specified file cannot be found, the caller lacks permissions to read it, or the server specified within the file does not respond to or accept the connection — the library will not proceed to check for other connection options as the user specified a particular one to use.

- If PMIX_TOOL_ATTACHMENT_FILE is given, then the tool will attempt to read the specified file and connect to the server based on the information contained within it.
- If PMIX_SERVER_URI or PMIX_TCP_URI is given, then connection will be attempted to the server at the specified URI. Note that it is an error for both of these attributes to be specified.
 PMIX_SERVER_URI is the preferred method as it is more generalized PMIX_TCP_URI is provided for those cases where the user specifically wants to use the Transmission Control Protocol (TCP) transport for the connection and wants to error out if it isn't available or cannot be used.
- If **PMIX_SERVER_PIDINFO** was provided, then the tool will search for a rendezvous file created by a PMIx server of the given PID in the **PMIX_SERVER_TMPDIR** directory..
- If **PMIX_SERVER_NSPACE** is given, then the tool will search for a rendezvous file created by a PMIx server of the given namespace in the **PMIX SERVER TMPDIR** directory.
- If **PMIX_CONNECT_TO_SYSTEM** is given, then the tool will search for a system-level rendezvous file created by a PMIx server in the **PMIX_SYSTEM_TMPDIR** directory.
- If **PMIX_CONNECT_SYSTEM_FIRST** is given, then the tool will look for a system-level rendezvous file created by a PMIx server in the **PMIX_SYSTEM_TMPDIR** directory. If found, then the tool will attempt to connect to it. In this case, no error will be returned if the rendezvous file is not found or connection is refused the PMIx library will silently continue to the next option
- By default, the tool will search the directory tree under the **PMIX_SERVER_TMPDIR** directory for rendezvous files of PMIx servers, attempting to connect to each it finds until one accepts the connection. If no rendezvous files are found, or all contacted servers refuse connection, then the PMIx library will return an error.

Note that there can be multiple local servers - one from the system plus others from launchers and active jobs. The PMIx tool connection search method is not guaranteed to pick a particular server unless directed to do so.

14.1.2 Connecting to a remote server

Connecting to remote servers is complicated due to the lack of access to the previously-described rendezvous files. Two methods are required to be supported, both based on the caller having explicit knowledge of either connection information or a path to a local file that contains such information:

- If PMIX_TOOL_ATTACHMENT_FILE is given, then the tool will attempt to read the specified file and connect to the server based on the information contained within it.
- If PMIX_SERVER_URI or PMIX_TCP_URI is given, then connection will be attempted to the server at the specified URI. Note that it is an error for both of these attributes to be specified.
 PMIX_SERVER_URI is the preferred method as it is more generalized PMIX_TCP_URI is provided for those cases where the user specifically wants to use the TCP transport for the connection and wants to error out if it isn't available or cannot be used.

Additional methods may be provided by particular PMIx implementations. For example, the tool may use *ssh* to launch a *probe* process onto the remote node so that the probe can search the **PMIX_SYSTEM_TMPDIR** and **PMIX_SERVER_TMPDIR** directories for rendezvous files, relaying the discovered information back to the requesting tool. If sufficient information is found to allow for remote connection, then the tool can use it to establish the connection. Note that this method is not required to be supported - it is provided here as an example and left to the discretion of PMIx implementors.

19 14.1.3 Attaching to running jobs

When attaching to a running job, the tool must connect to a PMIx server that is associated with that job - e.g., a server residing in the host environment's local daemon that spawned one or more of the job's processes, or the server residing in the launcher that is overseeing the job. Identifying an appropriate server can sometimes prove challenging, particularly in an environment where multiple job launchers may be in operation, possibly under control of the same user.

In cases where the user has only the one job of interest in operation on the local node (e.g., when engaged in an interactive session on the node from which the launcher was executed), the normal rendezvous file discovery method can often be used to successfully connect to the target job, even in the presence of jobs executed by other users. The permissions and security authorizations can, in many cases, reliably ensure that only the one connection can be made. However, this is not guaranteed in all cases.

The most common method, therefore, for attaching to a running job is to specify either the PID of the job's launcher or the namespace of the launcher's job (note that the launcher's namespace frequently differs from the namespace of the job it has launched). Unless the application processes themselves act as PMIx servers, connection must be to the servers in the daemons that oversee the application. This is typically either daemons specifically started by the job's launcher process, or daemons belonging to the host environment, that are responsible for starting the application's processes and oversee their execution.

Identifying the correct PID or namespace can be accomplished in a variety of ways, including:

- Using typical OS or host environment tools to obtain a listing of active jobs and perusing those to
 find the target launcher
 - Using a PMIx-based tool attached to a system-level server to query the active jobs and their command lines, thereby identifying the application of interest and its associated launcher
 - Manually recording the PID of the launcher upon starting the job

Once the namespace and/or PID of the target server has been identified, either of the previous methods can be used to connect to it.

14.1.4 Tool initialization attributes

The following attributes are passed to the **PMIx_tool_init** API for use when initializing the PMIx library.

```
PMIX_TOOL_NSPACE "pmix.tool.nspace" (char*)
```

Name of the namespace to use for this tool.

```
PMIX_TOOL_RANK "pmix.tool.rank" (uint32_t)
```

Rank of this tool.

```
PMIX_LAUNCHER "pmix.tool.launcher" (bool)
```

Tool is a launcher and needs rendezvous files created

17 14.1.5 Tool connection attributes

These attributes are defined to assist PMIx-enabled tools to connect with a PMIx server by passing them into either the PMIx_tool_init or the PMIx_tool_connect_to_server APIs - thus, they are not typically accessed via the PMIx_Get API.

```
PMIX_SERVER_PIDINFO "pmix.srvr.pidinfo" (pid_t)
```

PID of the target PMIx server for a tool.

```
PMIX CONNECT TO SYSTEM "pmix.cnct.sys" (bool)
```

The requestor requires that a connection be made only to a local, system-level PMIx server.

```
PMIX_CONNECT_SYSTEM_FIRST "pmix.cnct.sys.first" (bool)
```

Preferentially, look for a system-level PMIx server first.

```
PMIX_SERVER_URI "pmix.srvr.uri" (char*)
```

URI of the PMIx server to be contacted.

PMIX_SERVER_HOSTNAME "pmix.srvr.host" (char*)

Host where target PMIx server is located.

PMIX CONNECT MAX RETRIES "pmix.tool.mretries" (uint32 t)

Maximum number of times to try to connect to PMIx server.

PMIX CONNECT RETRY DELAY "pmix.tool.retry" (uint32 t)

Time in seconds between connection attempts to a PMIx server.

```
PMIX_TOOL_DO_NOT_CONNECT "pmix.tool.nocon" (bool)
```

The tool wants to use internal PMIx support, but does not want to connect to a PMIx server.

PMIX_TOOL_CONNECT_OPTIONAL "pmix.tool.conopt" (bool)

3

4

5

6

7

11

12 13

14

15

16

18

19

20 21

22 23

24 25

26

27 28

29

30

31 32

33

34 35

36

The tool shall connect to a server if available, but otherwise continue to operate unconnected PMIX_RECONNECT_SERVER "pmix.tool.recon" (bool) Tool is requesting to change server connections - superseded by the PMIx_tool_connect_to_server API PMIX_TOOL_ATTACHMENT_FILE "pmix.tool.attach" (char*) File containing connection info to be used for attaching to server PMIX LAUNCHER RENDEZVOUS FILE "pmix.tool.lncrnd" (char*) Pathname of file where the tool is to store its connection info

14.2 Launching Applications with Tools

Tool-directed launches can take two modes:

- *Direct launch* where the RM is directly responsible for launching all processes, including debugger daemons, under the direction of the tool itself i.e., there is no intermediate launcher such as *mpiexec*; and
- *Indirect launch* where all processes are started via an intermediate launcher such as *mpiexec* and the tool itself is not directly involved in launching application processes or debugger daemons. Note that the intermediate launcher may utilize the RM to launch processes and/or daemons under the launcher's direction.

Either of these methods can be executed interactively or by a batch script. Note that not all host environments will support the direct launch method.

14.2.1 Direct launch

In the direct-launch use-case (Fig. 14.2), the tool itself performs the role of the launcher. Once invoked, the tool connects to a system-level PMIx server, typically hosted by the RM. The tool is responsible for assembling the description of the application to be launched (e.g., by parsing its command line) into a spawn request containing an array of <code>pmix_app_t</code> applications and <code>pmix_info_t</code> job-level information. An allocation of resources may or may not have been made in advance – if not, then the spawn request must include allocation request information. The tool then calls the <code>PMIx_Spawn</code> API so the PMIx library can communicate the spawn request to the server.

Upon receipt, the system-level PMIx server library passes the spawn request to its host RM daemon for processing via the <code>pmix_server_spawn_fn_t</code> server module function. If this callback was not provided, then the PMIx server library will return the <code>PMIX_ERR_NOT_SUPPORTED</code> error status.

If an allocation must be made, then the host RM daemon is responsible for communicating the request to its associated scheduler. Once resources are available, the host RM initiates the launch process to start the application.

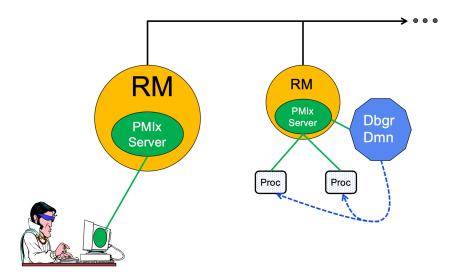


Figure 14.2.: Direct Launch

The RM must parse the spawn request for relevant directives, returning an error if any required directive cannot be supported. Whether the parsing occurs on the initial daemon, or on the remote compute-node daemons, is left to the discretion of the RM. Optional directives may be ignored if they cannot be supported.

14.2.2 Indirect launch

Controlling the launcher with "wait for directives"

In the indirect-launch use-case, the resource manager (RM) itself is not involved in launching application processes or debugger daemons. Indeed, in some cases the RM has no actual visibility of those processes, nor knowledge of their existence. Instead, processes are started via an intermediate launcher such as mpiexec (which we will use for this example). In turn, the intermediate launcher starts its own network of daemons (e.g., mpid) that assume responsibility for launching and supporting the job. The intermediate launcher may use the RM to launch the daemons, or ssh, depending on the precise implementation, environment, and user preferences.

A primary objective during the design of this operational mode is to avoid any requirement that the debugger parse and/or understand the command line of mpiexec. Thus, the focus is on cleanly passing all non-debugger options from the initial command line to mpiexec, using the PMIx tool-to-server connection to communicate any other directives.

In this operational mode, the user invokes a tool (typically on a non-compute, or "head", node) that in turn uses mpiexec to launch their application – a typical command line might look like the following:

\$ dbgr -dbgoption mpiexec -n 32 ./myapp

The tool may subsequently invoke mpiexec by simply executing it from a command line (e.g., using the Posix "system" function), or it may fork/exec it, or may request that it be started by the RM using the PMIx_Spawn API. The above illustration uses the last method. Regardless of how it is started, the debugger sets the PMIX_LAUNCHER_PAUSE_FOR_TOOL in the environment of mpiexec or in the pmix_info_t array in the spawn command. This instructs mpiexec to pause after initialization so it can receive further instructions from the debugger. This might include a request to co-spawn debugger daemons along with the application, or further directives relating to the startup of the application (e.g., to LD_PRELOAD a library, or replace the launcher's local spawn agent with one provided by the debugger).

As mpiexed starts up, it calls PMIx_server_init to setup its PMIx server. The server initialization includes writing a server-level rendezvous file that allows other processes (such as the originating debugger) to connect to the server. It then pauses, awaiting further instructions from the debugger.

Armed with the pid (returned by fork/exec or the "system" command) or the namespace (returned by PMIx_Spawn) of the executing mpiexec, the debugger tool utilizes the PMIx_tool_switch_server API to complete the connection to the mpiexec server. Note that:

PMIx does not allow servers to initiate connections – thus, the debugger tool must initiate the connection to the mpiexec server. tools can only be connected to one server at a time. Therefore, if connected to the system-level server to use PMIx_Spawn to launch mpiexec, the debugger tool will be disconnected from that server and connected to the PMIx server in mpiexec

At this point, the debugger can execute any PMIx operation, including:

query mpiexec capabilities; pass directives to configure application behavior – e.g., specifying the desired pause point where application processes shall wait for debugger release; request launch of debugger daemons, providing the appropriate pmix_app_t description specify a replacement fork/exec agent; and define/modify standard environmental variables for the application

Once ready to launch, mpiexec parses its command line to obtain a description of the desired job. An allocation of resources may or may not have been made in advance (either by the user, or by the tool prior to starting mpiexec)- if not, then mpiexec may itself utilize the PMIx_Alloc API to obtain one from the system-level PMIx server. Once resources are available, mpiexec initiates the launch process by first spawning its daemon network across the allocation – in the above diagram, this is done via ssh commands. After the daemons have launched and wired up, mpiexec sends an application launch command to its daemons, which then start their local client processes and debugger daemons, providing the latter with all information required for them to attach to their targets.

The following example illustrates how a debugger tool would execute an indirect launch using the mpiexec launcher from some supporting MPI library. There are a few points worth noting:

the debugger tool itself doesn't need to know if mpiexec can co-launch the debugger daemons, or must launch the application and debugger daemons as separate operations. We require that mpiexec notify the tool when the entire spawn is completed the mpiexec launcher is required to provide each

debugger daemon with the nspace of the target application it is to debug. This is done via a job-level PMIx attribute that the debugger daemon can query upon startup. Once the daemon has the target nspace, it can obtain the local (and complete, if desired) table of process pid's and hostnames (commonly called the proctable) by querying it from the local PMIx server hosted in the local mpid the debugger tool command-line parser does not need to identify the application to be executed. It can parse only its own options, taking everything else as being an opaque array of argy to be passed along the job-level information provided by mpiexec to the debugger daemons must include the mechanism by which the daemon can release the target application processes. This could include release via PMIx event notification (the precise notification code must be given), use of a specific signal, or some other mechanism. The debugger is free to terminate the job if it cannot support the given mechanism

14.2.3 Tool spawn-related attributes

Tools are free to utilize the spawn attributes available to applications (see 3.4.20) when constructing a spawn request, but can also utilize the following attributes that are specific to tool-based spawn operations:

PMIX_FWD_STDIN "pmix.fwd.stdin" (bool)

Forward **stdin** from this process to the designated process.

PMIX FWD STDOUT "pmix.fwd.stdout" (bool)

Forward **stdout** from spawned processes to this process.

PMIX FWD STDERR "pmix.fwd.stderr" (bool)

Forward **stderr** from spawned processes to this process.

PMIX_FWD_STDDIAG "pmix.fwd.stddiag" (bool)

If a diagnostic channel exists, forward any output on it from the spawned processes to this process

25 14.2.4 Tool spawn-related constants

PMIX_LAUNCH_DIRECTIVE Launcher directives have been received from a PMIx-enabled tool

PMIX_LAUNCHER_READY Application launcher (e.g., *mpiexec*) is ready to receive directives from a PMIx-enabled tool

PMIX_LAUNCH_COMPLETE A job has been launched - the namespace of the launched job will be included in the notification

PMIX_NOHUP "pmix.nohup" (bool)

Any descendants of the calling tool (or the specified namespace, if such specification is included in the list of attributes) should continue after the tool disconnects from its server

1 14.3 Debugger Support

36

37

2 3 4 5 6 7		DESCRIBE THE DEBUGGER SUPPORT OPTIONS - HOW TO START DEBUGGER DAEMONS (THE DEBUGGER-SPECIFIC FLAGS TO PASS AND WHAT THEY MEAN - REF THE CO-SPAWN SECTION), HOW TO GET PROC TABLES, LOCAL VS GLOBAL PROC TABLES, PAUSE FOR DEBUGGER AT DIFFERENT POINTS, HOW TO QUERY LEVELS OF SUPPORT, HOW TO DETACH WHILE LEAVING THE APP TO RUN, HOW TO REATTACH LATER
8	14.3.1	Co-Spawn of Debugger Daemons
9	14.3.2	Debugger attributes
10 11 12		Attributes used to assist debuggers - these are values that can be passed to the PMIx_Spawn or PMIx_Init APIs. Some may be accessed using the PMIx_Get API with the PMIX_RANK_WILDCARD rank.
13 14 15 16		PMIX_DEBUG_STOP_ON_EXEC "pmix.dbg.exec" (bool) Passed to PMIx_Spawn to indicate that the specified application is being spawned under a debugger, and that the launcher is to pause the resulting application processes on first instruction for debugger attach.
17 18 19 20		PMIX_DEBUG_STOP_IN_INIT "pmix.dbg.init" (bool) Passed to PMIx_Spawn to indicate that the specified application is being spawned under a debugger, and that the PMIx client library is to pause the resulting application processes during PMIx_Init until debugger attach and release.
21 22 23 24		PMIX_DEBUG_WAIT_FOR_NOTIFY "pmix.dbg.notify" (bool) Passed to PMIx_Spawn to indicate that the specified application is being spawned under a debugger, and that the resulting application processes are to pause at some application-determined location until debugger attach and release.
25 26 27		PMIX_DEBUG_JOB "pmix.dbg.job" (char*) Namespace of the job to be debugged - provided to the debugger upon launch (replaced by PMIX_DEBUG_TARGET)
28 29 30		PMIX_DEBUG_WAITING_FOR_NOTIFY "pmix.dbg.waiting" (bool) Job to be debugged is waiting for a release - this is not a value accessed using the PMIx_Get API.
31 32		PMIX_DEBUG_JOB_DIRECTIVES "pmix.dbg.jdirs" (pmix_data_array_t*) Array of job-level directives to be passed to debugger daemons.
33 34 35		<pre>PMIX_DEBUG_APP_DIRECTIVES "pmix.dbg.adirs" (pmix_data_array_t*)</pre>

Identifier of process(es) to be debugged - a rank of PMIX_RANK_WILDCARD indicates that

all processes in the specified namespace are to be included

14.4 IO Forwarding

Underlying the operation of many tools is a common need to forward stdin from the tool to targeted processes, and to return stdout/stderr from those processes to the tool (e.g., for display on the user's console). Historically, each tool developer was responsible for creating their own IO forwarding subsystem. However, the introduction of PMIx as a standard mechanism for interacting between applications and the host environment has made it possible to relieve tool developers of this burden.

This section defines functions by which tools can request forwarding of input/output to/from other processes and serves as a design guide to:

- provide tool developers with an overview of the expected behavior of the PMIx IO forwarding support;
- guide RM vendors regarding roles and responsibilities expected of the RM to support IO forwarding; and
- provide insight into the thinking of the PMIx community behind the definition of the PMIx IO forwarding APIs

Note that the forwarding of IO via PMIx requires that both the host environment and the tool support PMIx, but does not impose any similar requirements on the application itself.

Advice to PMIx server hosts —

The responsibility of the host environment in forwarding of IO falls into the following areas:

- Capturing output from specified child processes
- Forwarding that output to the host of the PMIx server library that requested it
- Delivering that payload to the PMIx server library via the PMIx_server_IOF_deliver API for final dispatch

— Advice to PMIx library implementers ————

It is the responsibility of the PMIx library to buffer, format, and deliver the payload to the requesting client. This

requires caching of output until forwarding registration received. Add attribute for setting cache side - it will be total number of messages to hold, intended to allow more flexible memory management. Not per source, but total across all jobs within the scope of that set of RM daemons - probably need to clarify the scope. Oldest message shall be dropped to make room for latest message, if required.

14.4.1 Forwarding stdout/stderr

At an appropriate point in its operation (usually during startup), a tool will utilize the **PMIx_tool_init** function to connect to a PMIx server. The PMIx server can be hosted by an SMS daemon or could be embedded in a library-provided starter program such as *mpiexec* - in terms of IO forwarding, the operations remain the same either way. For purposes of this discussion, we will assume the server is in an SMS daemon and that the application processes are directly launched by the SMS (see diagram at right).

Once the tool has connected to the target server, it can request that processes be spawned on its behalf and/or that output from a specified set of existing processes in a given executing application be forwarded to it. Requests to forward output from processes being spawned by the tool should be included in calls to the PMIx_Spawn API using the PMIX_FWD_STDOUT and/or PMIX_FWD_STDERR attributes. This alleviates

Two modes are supported for the latter case of catching output from existing processes:

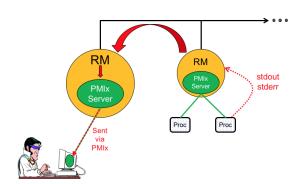


Figure 14.3.: Forwarding stdout/stderr

- PMIX_IOF_COPY deliver a copy of the output to the tool, letting the stream continue to also be delivered to the default location. This allows the tool to "tap" into the output stream without redirecting it from its current final destination
- PMIX_IOF_REDIRECT intercept the output stream and deliver to the requesting tool instead of its current final destination. This might be used, for example, during a debugging procedure to avoid "polluting" the application's results file. The original output stream destination is restored upon termination of the tool

Application processes are the children of the local SMS (typically, the local RM daemon) and not directly related to the PMIx server itself. Thus, it is the responsibility of the local SMS to collect the child's output – usually done by capturing the relevant file descriptors at fork/exec of the child process – and the PMIx server on the remote nodes is not involved in this process. Once captured, the host SMS is responsible for returning the output to the SMS daemon serving the tool. This typically will be the daemon co-located with the tool, but this isn't required.

Once the output reaches the serving SMS daemon, the daemon passes the output to its embedded PMIx server via the PMIx_IOF_push function whose parameter list includes the identifier of the source process and the IOF channel of the provided data. The PMIx server will transfer the data to the tool's client library, which will in turn output it to the screen.

When registering to receive output, the tool can specify several formatting options to be used on the resulting output stream. These include:

PMIX_IOF_TAG – output is prefixed with the nspace,rank of the source and a string identifying the channel (stdout, stderr, etc.) PMIX_IOF_TIMESTAMP – output is marked with the time at which the data was received by the tool (note that this will differ from the time at which it was actually output by the source) PMIX_IOF_XML_OUTPUT – output is to be formatted in XML

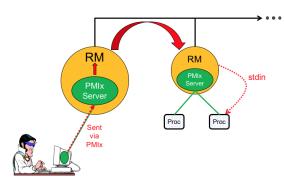
The PMIx client in the tool will format the output stream. Note that output from multiple processes will often be interleaved due to variations in arrival time - ordering of output is *not* guaranteed across processes and/or nodes.

14.4.2 Forwarding stdin

A tool is not necessarily a child of the RM as it may have been started directly from the command line. Thus, provision must be made for the tool to collect its stdin and pass it to the host RM (via the PMIx server) for forwarding. Two methods of support for forwarding of stdin are defined:

- internal collection by the PMIx tool library itself. This is requested via the PMIX_IOF_PUSH_STDIN attribute in the PMIx_IOF_push call. When this mode is selected, the tool library begins collecting all stdin data and internally passing it to the local server for distribution to the specified target processes. All collected data is sent to the same targets until stdin is closed, or a subsequent call to PMIx_IOF_push is made that includes the PMIX_IOF_COMPLETE attribute indicating that forwarding of stdin is to be terminated.
- external collection directly by the tool. It is assumed that the tool will provide its own code/mechanism for collecting its stdin as the tool developers may choose to insert some filtering and/or editing of the stream prior to forwarding it. In addition, the tool can directly control the targets for the data on a per-call basis i.e., each call to PMIx_IOF_push can specify its own set of target recipients for that particular "blob" of data. Thus, this method provides maximum flexibility, but requires that the tool developer provide their own code to capture stdin.

Note that it is the responsibility of the resource manager to forward data to the host where the target process(es) are executing, and for the host daemon on that node to deliver the data to the stdin of target process(es) via the typical pipe. The PMIx server on the remote node is *not* involved in this process. Systems that do not support forwarding of stdin shall



1	return PMIX _	_ERR_	_NOT_	_SUPPORTED
2	in response to	o a for	wardi	ng request.

-Advice to PMIx library implementers-

It is recognized that scalable forwarding

of stdin represents a significant challenge.

A high quality implementation will at least

handle a "send-to-1" model whereby stdin

is forwarded to a single identified process, and an additional "send-to-all" model where stdin is forwarded to all processes in the application. Other models (e.g., forwarding stdin to an arbitrary subset of processes) are left to the discretion of the implementor.

Advice to users

Stdin buffering by the RM and/or PMIx library can be problematic. If the targeted recipient is slow reading data (or decides never to read data), then the data must be buffered in some intermediate daemon or the local PMIx server itself. Thus, piping a large amount of data into stdin can result in a very large memory footprint in the system management stack. This is further exacerbated when targeting multiple recipients as the buffering problem, and hence the resulting memory footprint, is compounded. Best practices, therefore, typically focus on reading of input files by application processes as opposed to forwarding of stdin.

7 14.4.3 IO Forwarding attributes

PMIx v3.0 Attributes used to control IO forwarding behavior

PMIX IOF CACHE SIZE "pmix.iof.csize" (uint32 t)

The requested size of the server cache in bytes for each specified channel. By default, the server is allowed (but not required) to drop all bytes received beyond the max size.

PMIX_IOF_DROP_OLDEST "pmix.iof.old" (bool)

In an overflow situation, drop the oldest bytes to make room in the cache.

PMIX_IOF_DROP_NEWEST "pmix.iof.new" (bool)

In an overflow situation, drop any new bytes received until room becomes available in the cache (default).

PMIX_IOF_BUFFERING_SIZE "pmix.iof.bsize" (uint32_t)

Controls grouping of IO on the specified channel(s) to avoid being called every time a bit of IO arrives. The library will execute the callback whenever the specified number of bytes becomes available. Any remaining buffered data will be "flushed" upon call to deregister the respective channel.

PMIX_IOF_BUFFERING_TIME "pmix.iof.btime" (uint32_t)

Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering size, this prevents IO from being held indefinitely while waiting for another payload to arrive.

```
1
              PMIX_IOF_COMPLETE "pmix.iof.cmp" (bool)
2
                    Indicates whether or not the specified IO channel has been closed by the source.
 3
              PMIX_IOF_TAG_OUTPUT "pmix.iof.tag" (bool)
4
                    Tag output with the channel it comes from.
5
              PMIX IOF TIMESTAMP OUTPUT "pmix.iof.ts" (bool)
6
                    Timestamp output
7
              PMIX IOF XML OUTPUT "pmix.iof.xml" (bool)
8
                    Format output in XML
9
              PMIX_IOF_PUSH_STDIN "pmix.iof.stdin" (()
10
11
              ool) Used by a tool to request that the PMIx library collect the tool's stdin and forward it to the
              processes specified in the PMIx_IOF_push call
12
    14.5
             Tool-Specific APIs
14
              Initialization and finalization routines for PMIx tools. Tools shall automatically connect to an
              available server (based on the prior discovery mechanisms) unless directed otherwise.
15
    14.5.1
               PMIx_tool_init
17
              Summary
              Initialize the PMIx library for operating as a tool, optionally connecting to a specified PMIx server
18
              Format
19
   PMIx v2.0
20
              pmix_status_t
              PMIx_tool_init(pmix_proc_t *proc,
21
22
                                 pmix_info_t info[], size_t ninfo)
                                             ____ C
              INOUT proc
23
                   pmix proc t structure (handle)
24
25
              IN info
                   Array of pmix info t structures (array of handles)
26
27
              IN ninfo
                   Number of element in the info array (size t)
28
```

Returns **PMIX SUCCESS** or a negative value corresponding to a PMIx error constant.

```
Required Attributes
              The following attributes are required to be supported by all PMIx libraries:
 1
 2
              PMIX TOOL NSPACE "pmix.tool.nspace" (char*)
                    Name of the namespace to use for this tool.
 3
 4
              PMIX_TOOL_RANK "pmix.tool.rank" (uint32_t)
                    Rank of this tool.
 5
 6
              PMIX_TOOL_DO_NOT_CONNECT "pmix.tool.nocon" (bool)
7
                    The tool wants to use internal PMIx support, but does not want to connect to a PMIx server.
8
              PMIX_TOOL_ATTACHMENT_FILE "pmix.tool.attach" (char*)
9
                    File containing connection info to be used for attaching to server
10
              PMIX SERVER URI "pmix.srvr.uri" (char*)
                    URI of the PMIx server to be contacted.
11
12
              PMIX_TCP_URI "pmix.tcp.uri" (char*)
13
                    The URI of the PMIx server to connect to, or a file name containing it in the form of
                    file: <name of file containing it>.
14
              PMIX_SERVER_PIDINFO "pmix.srvr.pidinfo" (pid_t)
15
                    PID of the target PMIx server for a tool.
16
17
              PMIX_SERVER_NSPACE "pmix.srv.nspace" (char*)
                    Name of the namespace to use for this PMIx server.
18
              PMIX CONNECT_TO_SYSTEM "pmix.cnct.sys" (bool)
19
20
                    The requestor requires that a connection be made only to a local, system-level PMIx server.
              PMIX_CONNECT_SYSTEM_FIRST "pmix.cnct.sys.first" (bool)
21
22
                    Preferentially, look for a system-level PMIx server first.
                 -----
                                             Optional Attributes
              The following attributes are optional for implementers of PMIx libraries:
23
24
              PMIX_CONNECT_RETRY_DELAY "pmix.tool.retry" (uint32_t)
25
                    Time in seconds between connection attempts to a PMIx server.
26
              PMIX CONNECT MAX RETRIES "pmix.tool.mretries" (uint32 t)
                    Maximum number of times to try to connect to PMIx server.
27
              PMIX_SOCKET_MODE "pmix.sockmode" (uint32_t)
28
                    POSIX mode t (9 bits valid) If the library supports socket connections, this attribute may
29
                    be supported for setting the socket mode.
30
31
              PMIX_TCP_REPORT_URI "pmix.tcp.repuri" (char*)
```

1 If provided, directs that the TCP URI be reported and indicates the desired method of 2 reporting: '-' for stdout, '+' for stderr, or filename. If the library supports TCP socket connections, this attribute may be supported for reporting the URI. 3 4 PMIX TCP IF INCLUDE "pmix.tcp.ifinclude" (char*) 5 Comma-delimited list of devices and/or CIDR notation to include when establishing the 6 TCP connection. If the library supports TCP socket connections, this attribute may be 7 supported for specifying the interfaces to be used. 8 PMIX TCP IF EXCLUDE "pmix.tcp.ifexclude" (char*) 9 Comma-delimited list of devices and/or CIDR notation to exclude when establishing the 10 TCP connection. If the library supports TCP socket connections, this attribute may be supported for specifying the interfaces that are *not* to be used. 11 12 PMIX TCP IPV4 PORT "pmix.tcp.ipv4" (int) The IPv4 port to be used. If the library supports IPV4 connections, this attribute may be 13 supported for specifying the port to be used. 14 15 PMIX TCP IPV6 PORT "pmix.tcp.ipv6" (int) 16 The IPv6 port to be used. If the library supports IPV6 connections, this attribute may be 17 supported for specifying the port to be used. PMIX TCP DISABLE IPV4 "pmix.tcp.disipv4" (bool) 18 19 Set to **true** to disable IPv4 family of addresses. If the library supports IPV4 connections, 20 this attribute may be supported for disabling it. PMIX_TCP_DISABLE_IPV6 "pmix.tcp.disipv6" (bool) 21 Set to **true** to disable IPv6 family of addresses. If the library supports IPV6 connections, 22 23 this attribute may be supported for disabling it. 24 PMIX EVENT BASE "pmix.evbase" (struct event base *) Pointer to libevent **base** to use in place of the internal progress thread. 25 26 Description 27 Initialize the PMIx tool, returning the process identifier assigned to this tool in the provided pmix_proc_t struct. The *info* array is used to pass user requests pertaining to the init and 28

subsequent operations. Passing a NULL value for the array pointer is supported if no directives are desired.

If called with the PMIX TOOL DO NOT CONNECT attribute, the PMIx tool library will fully initialize but not attempt to connect to a PMIx server. The tool can connect to a server at a later point in time, if desired. In all other cases, the PMIx tool library will attempt to connect to a PMIx server per the precedence chain described in Section 14.1.

29 30

31

32

33

¹http://libevent.org/

1 2 3 4	If successful, the function will return PMIX_SUCCESS and will fill the provided structure (if provided) with the sassigned namespace and rank of the tool. Note that each connection attempt in the above precedence chain will retry (with delay between each retry) a number of times according to the values of the corresponding attributes. Default is no retries.			
5 6	Note that the PMIx tool library is referenced counted, and so multiple calls to PMIx_tool_init are allowed.			
7 8 9 0	WHAT TO DO ABOUT CONNECTION ON SUBSEQUENT CALLS TO INIT? CAN ONE USE A CALL TO INIT TO CREATE A CONNECTION IF ONE TOLD PMIX "DO-NOT-CONNECT" ON PRIOR CALLS? CAN ONE TELL THE TOOL TO DISCONNECT WHILE REMAINING INITIALIZED? AND THEN CONNECT BY CALLING INIT AGAIN?			
1 14.	5.2 PMIx_tool_finalize			
2 3	Summary Finalize the PMIx library for a tool connection.			
4 PMIx	v2.0 Format C			
5 6	<pre>pmix_status_t PMIx_tool_finalize(void)</pre>			
7	Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.			
8 9 20 21	Description Finalize the PMIx tool library, closing the connection to the server. An error code will be returned if, for some reason, the connection cannot be cleanly terminated — in this case, the connection is dropped.			
2 14.	5.3 PMIx_tool_disconnect			
23 24 25	Summary Disconnect the PMIx tool from its current server connection while leaving the tool library initialized.			
e6 PMIx	Format C			
27 28	pmix_status_t PMIx_tool_disconnect(void)			
<u>.</u> 9	Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.			

Description 1 2 Close the current connection to the server, if one has been made, while leaving the PMIx library initialized. An error code will be returned if, for some reason, the connection cannot be cleanly 3 4 terminated - in this case, the connection is dropped, but the library will in either case remain 5 initialized. Note that all operations requiring support from a server will return the 6 PMIX ERR UNREACH error until the tool re-establishes a connection to a PMIx server. 14.5.4 PMIx_tool_connect_to_server Summary 8 9 Switch connection from the current PMIx server to another one, or initialize a connection to a specified server. 10 **Format** 11 PMIx v3.012 pmix status t PMIx tool connect to server(pmix proc t *proc, 13 pmix_info_t info[], size t ninfo) 14 Returns **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant. 15 Required Attributes _____ The following attributes are required to be supported by all PMIx libraries: 16 PMIX CONNECT TO SYSTEM "pmix.cnct.sys" (bool) 17 The requestor requires that a connection be made only to a local, system-level PMIx server. 18 19 PMIX CONNECT SYSTEM FIRST "pmix.cnct.sys.first" (bool) Preferentially, look for a system-level PMIx server first. 20 PMIX SERVER URI "pmix.srvr.uri" (char*) 21 URI of the PMIx server to be contacted. 22 23 PMIX SERVER NSPACE "pmix.srv.nspace" (char*) Name of the namespace to use for this PMIx server. 24 25 PMIX SERVER PIDINFO "pmix.srvr.pidinfo" (pid t) 26 PID of the target PMIx server for a tool. **▲**-----

1 Description

Switch connection from the current PMIx server to another one, or initialize a connection to a specified server. Closes the connection, if existing, to a server and establishes a connection to the specified server. This function can be called at any time by a PMIx tool to shift connections between servers. The process identifier assigned to this tool is returned in the provided <code>pmix_proc_t</code> struct. Passing a value of <code>NULL</code> for this parameter is allowed if the user wishes solely to connect to the PMIx server and does not require return of the identifier at that time.

Advice to PMIx library implementers -

PMIx tools and clients are prohibited from being connected to more than one server at a time to avoid confusion in subsystems such as event notification.

When a tool connects to a server that is under a different namespace manager (e.g., host RM) as the prior server, the identifier of the tool must remain unique in the namespaces. This may require the identifier of the tool to be changed on-the-fly, that is, the *proc* parameter would be filled (if non-NULL) with a different nspace/rank from the current tool identifier.

Advice to users ·

Passing a **NULL** value for the *info* pointer is not allowed and will result in returning an error.

Some PMIx implementations (for example, the current PRI) may not support connecting to a server that is not under the same namespace manager (e.g., host RM) as the tool.

14.5.5 PMIx_IOF_pull

Summary

Register to receive output forwarded from a set of remote processes.

1		Format				
	<i>PMIx v3.0</i>					
2		pmix_status_t				
3		<pre>PMIx_IOF_pull(const pmix_proc_t procs[], size_t nprocs,</pre>				
4		<pre>const pmix_info_t directives[], size_t ndirs,</pre>				
5		<pre>pmix_iof_channel_t channel, pmix_iof_cbfunc_t cbfunc,</pre>				
6		<pre>pmix_hdlr_reg_cbfunc_t regcbfunc, void *regcbdata)</pre>				
		C				
7		IN procs				
8		Array of proc structures identifying desired source processes (array of handles)				
9		IN nprocs				
10		Number of elements in the <i>procs</i> array (integer)				
11		IN directives				
12		Array of pmix_info_t structures (array of handles)				
13		IN ndirs				
14		Number of elements in the <i>directives</i> array (integer)				
15		IN channel				
16		Bitmask of IO channels included in the request (pmix_iof_channel_t)				
17		IN cbfunc				
18 19		Callback function for delivering relevant output (pmix_iof_cbfunc_t function				
20		reference) IN regcbfunc				
20 21		Function to be called when registration is completed (pmix_hdlr_reg_cbfunc_t				
22		function reference)				
23		IN regcbdata				
24		Data to be passed to the <i>regcbfunc</i> callback function (memory reference)				
25		Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant. In the event				
26		the function returns an error, the <i>regcbfunc</i> will <i>not</i> be called.				
		▼ Required Attributes				
27		The following attributes are required for PMIx libraries that support IO forwarding:				
28		<pre>PMIX_IOF_CACHE_SIZE "pmix.iof.csize" (uint32_t)</pre>				
29		The requested size of the server cache in bytes for each specified channel. By default, the				
30		server is allowed (but not required) to drop all bytes received beyond the max size.				
31		PMIX_IOF_DROP_OLDEST "pmix.iof.old" (bool)				
32		In an overflow situation, drop the oldest bytes to make room in the cache.				
		·				
33		PMIX_IOF_DROP_NEWEST "pmix.iof.new" (bool)				
34		In an overflow situation, drop any new bytes received until room becomes available in the				
35		cache (default).				

Optional Attributes The following attributes are optional for PMIx libraries that support IO forwarding: 1 2 PMIX_IOF_BUFFERING_SIZE "pmix.iof.bsize" (uint32_t) 3 Controls grouping of IO on the specified channel(s) to avoid being called every time a bit of 4 IO arrives. The library will execute the callback whenever the specified number of bytes becomes available. Any remaining buffered data will be "flushed" upon call to deregister the 5 6 respective channel. 7 PMIX_IOF_BUFFERING_TIME "pmix.iof.btime" (uint32_t) 8 Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering 9 size, this prevents IO from being held indefinitely while waiting for another payload to arrive. 10 11 PMIX_IOF_TAG_OUTPUT "pmix.iof.tag" (bool) 12 Tag output with the channel it comes from. 13 PMIX_IOF_TIMESTAMP_OUTPUT "pmix.iof.ts" (bool) 14 Timestamp output 15 PMIX IOF XML OUTPUT "pmix.iof.xml" (bool) 16 Format output in XML **Description** 17 Register to receive output forwarded from a set of remote processes. 18 Advice to users Providing a **NULL** function pointer for the *cbfunc* parameter will cause output for the indicated 19 20 channels to be written to their corresponding stdout/stderr file descriptors. Use of 21 PMIX_RANK_WILDCARD to specify all processes in a given namespace is supported but should 22 be used carefully due to bandwidth considerations.

3 14.5.6 PMIx_IOF_deregister

Summary

24

25

Deregister from output forwarded from a set of remote processes.

1	Format				
<i>PMIx v3.0</i>					
2	pmix_status_t				
3	PMIx_IOF_deregister(size_t iofhdlr,				
4	<pre>const pmix_info_t directives[], size_t ndirs,</pre>				
5	pmix_op_cbfunc_t cbfunc, void *cbdata)				
	- C				
6	IN iofhdlr				
7	Registration number returned from the pmix_hdlr_reg_cbfunc_t callback from the				
8	call to PMIx_IOF_pull (size_t)				
9	IN directives				
10	Array of pmix_info_t structures (array of handles)				
11	IN ndirs				
12	Number of elements in the <i>directives</i> array (integer)				
13	IN cbfunc				
14	Callback function to be called when deregistration has been completed. (function reference)				
15	IN cbdata				
16	Data to be passed to the <i>cbfunc</i> callback function (memory reference)				
17	Returns one of the following:				
18	• PMIX_SUCCESS, indicating that the request is being processed by the host environment - result				
19	will be returned in the provided cbfunc. Note that the library must not invoke the callback				
20	function prior to returning from the API.				
21	• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and				
22	returned <i>success</i> - the <i>cbfunc</i> will <i>not</i> be called				
23	a DMIv amon constant indicating either an amon in the input on that the request was immediately				
23 24	• a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will <i>not</i> be called				
-4	processed and raned - the cojunc will not be caned				
25	Description				
26	Deregister from output forwarded from a set of remote processes.				
	Advice to PMIx library implementers —				
	,				
27	Any currently buffered IO should be flushed upon receipt of a deregistration request. All received				
28	IO after receipt of the request shall be discarded.				
4455					
29 14.5.7	PMIx_IOF_push				
30	Summary				
					

Push data collected locally (typically from stdin or a file) to stdin of the target recipients.

30

1	Format					
PMI.						
2	pmix_status_t					
<pre>3 PMIx_IOF_push(const pmix_proc_t targets[], size_t ntargets,</pre>						
4	<pre>pmix_byte_object_t *bo,</pre>					
5	<pre>const pmix_info_t directives[], size_t ndirs,</pre>					
6	<pre>pmix_op_cbfunc_t cbfunc, void *cbdata)</pre>					
	C					
7	IN targets					
8	Array of proc structures identifying desired target processes (array of handles)					
9	IN ntargets					
10	Number of elements in the <i>targets</i> array (integer)					
11	IN bo					
12	Pointer to pmix_byte_object_t containing the payload to be delivered (handle)					
13	IN directives					
14	Array of pmix_info_t structures (array of handles)					
15	IN ndirs					
16	Number of elements in the <i>directives</i> array (integer)					
17	IN directives					
18	Array of pmix_info_t structures (array of handles)					
19	IN cbfunc					
20	Callback function to be called when operation has been completed. (pmix_op_cbfunc_t					
21	function reference)					
22	IN cbdata					
23	Data to be passed to the <i>cbfunc</i> callback function (memory reference)					
24	Returns one of the following:					
25	• PMIX_SUCCESS, indicating that the request is being processed by the host environment - result					
26	will be returned in the provided <i>cbfunc</i> . Note that the library <i>must not</i> invoke the callback					
27	function prior to returning from the API.					
28	• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and					
29	returned <i>success</i> - the <i>cbfunc</i> will <i>not</i> be called					
30	• a PMIx error constant indicating either an error in the input or that the request was immediately					
31	processed and failed - the <i>cbfunc</i> will <i>not</i> be called					
	· · · · · · · · · · · · · · · · · · ·					
	Required Attributes					
32	The following attributes are required for PMIx libraries that support IO forwarding:					
33	<pre>PMIX_IOF_CACHE_SIZE "pmix.iof.csize" (uint32_t)</pre>					
34	The requested size of the server cache in bytes for each specified channel. By default, the					
35	server is allowed (but not required) to drop all bytes received beyond the max size.					

1 2	<pre>PMIX_IOF_DROP_OLDEST "pmix.iof.old" (bool)</pre> In an overflow situation, drop the oldest bytes to make room in the cache.
۷	•
3	PMIX_IOF_DROP_NEWEST "pmix.iof.new" (bool)
4	In an overflow situation, drop any new bytes received until room becomes available in the
5	cache (default).
	→ Optional Attributes
6	The following attributes are optional for PMIx libraries that support IO forwarding:
7	<pre>PMIX_IOF_BUFFERING_SIZE "pmix.iof.bsize" (uint32_t)</pre>
8	Controls grouping of IO on the specified channel(s) to avoid being called every time a bit of
9	IO arrives. The library will execute the callback whenever the specified number of bytes
10	becomes available. Any remaining buffered data will be "flushed" upon call to deregister the
11	respective channel.
12	<pre>PMIX_IOF_BUFFERING_TIME "pmix.iof.btime" (uint32_t)</pre>
13	Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering
14	size, this prevents IO from being held indefinitely while waiting for another payload to
15	arrive.
16	Description
17	Push data collected locally (typically from stdin or a file) to stdin of the target recipients.
	Advice to users
18	Execution of the <i>cbfunc</i> callback function serves as notice that the PMIx library no longer requires
19	the caller to maintain the <i>bo</i> data object - it does <i>not</i> indicate delivery of the payload to the targets.
20	Use of PMIX_RANK_WILDCARD to specify all processes in a given namespace is supported but
21	should be used carefully due to bandwidth considerations.
	onound of acts that if the to build main conditional.

APPENDIX A

Python Bindings

While the PMIx Standard is defined in terms of C-based APIs, there is no intent to limit the use of PMIx to that specific language. Support for other languages is captured in the Standard by describing their equivalent syntax for the PMIx APIs and native forms for the PMIx datatypes. This Appendix specifically deals with Python interfaces, beginning with a review of the PMIx datatypes. Support is restricted to Python 3 and above - i.e., the Python bindings do not support Python 2.

Note: the PMIx APIs have been loosely collected into three Python classes based on their PMIx "class" (i.e., client, server, and tool). All processes have access to a basic set of the APIs, and therefore those have been included in the "client" class. Servers can utilize any of those functions plus a set focused on operations not commonly executed by an application process. Finally, tools can also act as servers but have their own initialization function.

A.1 Design Considerations

Several issues arose during design of the Python bindings:

3 A.1.1 Error Codes vs Python Exceptions

The C programming language reports errors through the return of the corresponding integer status codes. PMIx has defined a range of negative values for this purpose. However, Python has the option of raising *exceptions* that effectively operate as interrupts that can be trapped if the program appropriately tests for them. The PMIx Python bindings opted to follow the C-based standard and return PMIx status codes in lieu of raising exceptions as this method was considered more consistent for those working in both domains.

A.1.2 Representation of Structured Data

PMIx utilizes a number of C-language structures to efficiently bundle related information. For example, the PMIx process identifier is represented as a struct containing a character array for the namespace and a 32-bit unsigned integer for the process rank. There are several options for translating such objects to Python – e.g., the PMIx process identifier could be represented as a two-element tuple (nspace, rank) or as a dictionary 'nspace': name, 'rank': 0. Exploration found no discernible benefit to either representation, nor was any clearly identifiable rationale developed that would lead a user to expect one versus the other for a given PMIx data type. Consistency in the translation (i.e., exclusively using tuple or dictionary) appeared to be the most important criterion. Hence, the decision was made to express all complex datatypes as Python dictionaries.

A.2 Datatype Definitions

PMIx defines a number of datatypes comprised of fixed-size character arrays, restricted range integers (e.g., uint32_t), and structures. Each datatype is represented by a named unsigned 16-bit integer (uint16_t) constant. Users are advised to use the named PMIx constants for indicating datatypes instead of integer values to ensure compatibility with future PMIx versions.

With only a few exceptions, the C-based PMIx datatypes defined in Chapter 3 on page 20 directly translate to Python. However, Python lacks the size-specific value definitions of C (e.g., uint8_t) and thus some care must be taken to protect against overflow/underflow situations when moving between the languages. Python bindings that accept values including PMIx datatypes shall therefore have the datatype and associated value checked for compatibility with their PMIx-defined equivalents, returning an error if:

- datatypes not defined by PMIx are encountered
- provided values fall outside the range of the C-equivalent definition e.g., if a value identified as **PMIX_UINT8** lies outside the **uint8_t**range

Note that explicit labeling of PMIx datatype, even when Python itself doesn't care, is often required for the Python bindings to know how to properly interpret and label the provided value when passing it to the PMIx library.

Table A.1 lists the correspondence between datatypes in the two languages.

Table A.1.: C-to-Python Datatype Correspondence

C-Definition	PMIx Name	Python Definition	Notes
bool	PMIX_BOOL	boolean	
byte	PMIX_BYTE	A single element byte array (i.e., a byte array of length one)	
char*	PMIX_STRING	string	
size_t	PMIX_SIZE	integer	
pid_t	PMIX_PID	integer	value shall be limited to the uint32_t range
<pre>int, int8_t, int16_t, int32_t, int64_t</pre>	PMIX_INT, PMIX_INT8, PMIX_INT16, PMIX_INT32, PMIX_INT64	integer	value shall be limited to its corresponding range
uint, uint8_t, uint16_t, uint32_t, uint64_t	PMIX_UINT, PMIX_UINT8, PMIX_UINT16, PMIX_UINT32, PMIX_UINT64	integer	value shall be limited to its corresponding range
float, double	PMIX_FLOAT, PMIX_DOUBLE	float	value shall be limited to its corresponding range
struct timeval	PMIX_TIMEVAL	{'sec': sec, 'usec': microsec}	each field is an integer value
time_t	PMIX_TIME	integer	limited to positive values
pmix_data_type_t	PMIX_DATA_TYPE	integer	value shall be limited to the uint16_t range
pmix_status_t	PMIX_STATUS	integer	
pmix_key_t	N/A	string	The string's length shall be limited to one less than the size of the <code>pmix_key_t</code> array (to reserve space for the terminating <code>NULL</code>)
pmix_nspace_t	N/A	string	The string's length shall be limited to one less than the size of the <code>pmix_nspace_t</code> array (to reserve space for the terminating <code>NULL</code>)

Table A.1.: C-to-Python Datatype Correspondence

C-Definition	PMIx Name	Python Definition	Notes
pmix_rank_t	PMIX_PROC_RANK	integer	value shall be limited to the uint32_t range excepting the reserved values near UINT32_MAX
pmix_proc_t	PMIX_PROC	{'nspace': nspace, 'rank': rank}	nspace is a Python string and rank is an integer value. The nspace string's length shall be limited to one less than the size of the pmix_nspace_t array (to reserve space for the terminating NULL), and the rank value shall conform to the constraints associated with pmix_rank_t
pmix_byte_object_t	PMIX_BYTE_OBJECT	{'bytes': bytes, 'size': size}	bytes is a Python byte array and size is the integer number of bytes in that array.
pmix_persistence_t	PMIX_PERSISTENCE	integer	value shall be limited to the uint8_t range
pmix_scope_t	PMIX_SCOPE	integer	value shall be limited to the uint8_t range
pmix_data_range_t	PMIX_RANGE	integer	value shall be limited to the uint8_t range
pmix_proc_state_t	PMIX_PROC_STATE	integer	value shall be limited to the uint8_t range
pmix_proc_info_t	PMIX_PROC_INFO	{'proc': {'nspace': nspace, 'rank': rank}, 'hostname': hostname, 'executable': executable, 'pid': pid, 'exitcode': exitcode, 'state': state}	proc is a Python proc dictionary; hostname and executable are Python strings; and pid, exitcode, and state are Python integers

APPENDIX A. PYTHON BINDINGS

Table A.1.: C-to-Python Datatype Correspondence

C-Definition	PMIx Name	Python Definition	Notes
<pre>pmix_data_array_t</pre>	PMIX_DATA_ARRAY	{'type': type, 'array': array}	type is the PMIx type of object in the array and array is a Python list containing the individual array elements. Note that array can consist of any PMIx types, including (for example) a Python info object that itself contains an array value
pmix_info_directives_t	PMIX_INFO_DIRECTIVES	integer	value shall be limited to the uint32_t range
pmix_alloc_directive_t	PMIX_ALLOC_DIRECTIVE	integer	value shall be limited to the uint8_t range
pmix_iof_channel_t	PMIX_IOF_CHANNEL	integer	value shall be limited to the uint16_t range
pmix_envar_t	PMIX_ENVAR	{'envar': envar, 'value': value, 'separator': separator}	envar and value are Python strings, and separator a single-character Python string
pmix_value_t	PMIX_VALUE	{'value': value, 'val_type': type}	type is the PMIx datatype of value, and value is the associated value expressed in the appropriate Python form for the specified datatype
pmix_info_t	PMIX_INFO	{'key': key, 'flags': flags, value': value, 'val_type': type}	key is a Python string key, flags is a bitmask of info directives, type is the PMIx datatype of value, and value is the associated value expressed in the appropriate Python form for the specified datatype
pmix_pdata_t	PMIX_PDATA	{'proc': {'nspace': nspace, 'rank': rank}, 'key': key, 'value': value, 'val_type': type}	proc is a Python proc dictionary; key is a Python string key; type is the PMIx datatype of value; and value is the associated value expressed in the appropriate Python form for the specified datatype

Table A.1.: C-to-Python Datatype Correspondence

C-Definition	PMIx Name	Python Definition	Notes
pmix_app_t	PMIX_APP	{'cmd': cmd, 'argv':	cmd is a Python string; argv and env are
		[argv], 'env': [env],	Python <i>lists</i> containing Python strings;
		'maxprocs': maxprocs,	maxprocs is an integer; and info is a
		'info': [info]}	Python list of info values
pmix_query_t	PMIX_QUERY	{'keys': [keys],	keys is a Python list of Python strings, and
		'qualifiers': [info]}	qualifiers is a Python list of info values
pmix_regattr_t	PMIX_REGATTR	{'name': name, 'key':	name and string are Python strings; type is
		key, 'type': type, 'info':	the PMIx datatype for the attribute's value;
		[info], 'description':	<i>info</i> is a Python <i>list</i> of info values;
		[desc]}	and description is a list of Python strings
			describing the attribute
pmix_link_state_t	PMIX_LINK_STATE	integer	value shall be limited to the uint8_t
			range

A.2.1 Example

2 Converting a C-based program to its Python equivalent requires translation of the relevant 3 datatypes as well as use of the appropriate API form. An example small program may help 4 illustrate the changes. Consider the following C-based program snippet: 5 #include <pmix.h> 6 . . . 7 8 pmix info t info[2]; 9 10 PMIX_INFO_LOAD(&info[0], PMIX_PROGRAMMING_MODEL, "TEST", PMIX_STRING) 11 PMIX_INFO_LOAD(&info[1], PMIX_MODEL_LIBRARY_NAME, "PMIX", PMIX_STRING) 12 13 rc = PMIx_Init(&myproc, info, 2); 14 15 PMIX_INFO_DESTRUCT(&info[0]); // free the copied string 16 PMIX_INFO_DESTRUCT(&info[1]); // free the copied string 17 18 Moving to the Python version requires that the pmix info t be translated to the Python info 19 equivalent, and that the returned information be captured in the return parameters as opposed to a 20 pointer parameter in the function call, as shown below: 21 import pmix 22 23 24 myclient = PMIxClient() 25 info = [{'key':PMIX_PROGRAMMING_MODEL, 26 'value':'TEST', 'val_type':PMIX_STRING}, 27 {'key':PMIX_MODEL_LIBRARY_NAME, 28 'value':'PMIX', 'val_type':PMIX_STRING}]

Note the use of the **PMIX_STRING** identifier to ensure the Python bindings interpret the provided string value as a PMIx "string" and not an array of bytes.

A.3 Callback Function Definitions

(rc,myproc) = myclient.init(info)

A.3.1 IOF Delivery Function

Summary

29

30 31

32

35

36

Callback function for delivering forwarded IO to a process

```
Format
1
                                                     Python -
   PMIx v4.0
2
              def iofcbfunc(iofhdlr:integer, channel:integer,
                                source:dict, payload:dict, info:list)
 3
                                                     Python ——
4
              IN
                  iofhdlr
                   Registration number of the handler being invoked (integer)
5
6
              IN channel
7
                   Python channel bitmask identifying the channel the data arrived on (integer)
8
              IN
                   source
9
                   Python proc identifying the namespace/rank of the process that generated the data (dict)
10
              IN payload
                   Python byteobject containing the data (dict)
11
              IN
                  info
12
13
                   List of Python info provided by the source containing metadata about the payload. This
                   could include PMIX IOF COMPLETE (list)
14
              Returns: nothing
15
16
              See pmix iof cbfunc t for details
   A.3.2 Event Handler
18
              Summary
              Callback function for event handlers
19
              Format
20
                                                     Pvthon
   PMIx v4.0
21
              def evhandler(evhdlr:integer, status:integer,
                                source:dict, info:list, results:list)
22
                                                     Python -
              IN
23
                  iofhdlr
24
                   Registration number of the handler being invoked (integer)
25
              IN
26
                   Status associated with the operation (integer)
              IN source
27
                   Python proc identifying the namespace/rank of the process that generated the event (dict)
28
29
              IN info
                   List of Python info provided by the source containing metadata about the event (list)
30
31
              IN
                   results
32
                   List of Python info containing the aggregated results of all prior evhandlers (list)
33
              Returns:
```

1 • rc - Status returned by the event handler's operation (integer) • results - List of Python info containing results from this event handler's operation on the event 2 3 (list) See pmix_notification_fn_t for details 4 A.3.3 **Server Module Functions** The following definitions represent functions that may be provided to the PMIx server library at 6 7 time of initialization for servicing of client requests. Module functions that are not provided default to returning "not supported" to the caller. 8 9 A.3.3.1 **Client Connected** 10 Summarv 11 Notify the host server that a client connected to this server. **Format** 12 Python PMIx v4.013 def clientconnected(proc:dict is not None) Python 14 IN proc Python **proc** identifying the namespace/rank of the process that connected (dict) 15 16 Returns: 17 • rc - PMIX_SUCCESS or a PMIx error code indicating the connection should be rejected 18 (integer) 19 See pmix_server_client_connected_fn_t for details A.3.3.2 Client Finalized 20 21 Summary 22 Notify the host environment that a client called **PMIx_Finalize**. Format 23 Python PMIx v4.0 def clientfinalized(proc:dict is not None): 24 **Python** 25 IN proc Python proc identifying the namespace/rank of the process that finalized (dict) 26 27 Returns: nothing 28 See pmix server client finalized fn t for details

1 A. 3	.3.3 Client Aborted
2	Summary Notify the host environment that a local client called PMIx_Abort .
4 <i>PMI</i> :	v4.0 ► Python — Python
5	def clientaborted(args:dict is not None)
6 7	IN args Python dictionary containing:
8	• 'caller': Python proc identifying the namespace/rank of the process calling abort (dict)
9	• 'status': PMIx status to be returned on exit (integer)
10	• 'msg': Optional string message to be printed (string)
11 12	• 'targets': Optional list of Python proc identifying the namespace/rank of the processes to be aborted (list)
13	Returns:
14	• rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
15	See pmix_server_abort_fn_t for details
16 A. 3	.3.4 Fence
17 18	Summary At least one client called either PMIx_Fence or PMIx_Fence_nb
19 <i>PMI</i> :	v4.0 Python
20	def fence(args:dict is not None) Python
21 22	IN args Python dictionary containing:
23 24	 'procs': List of Python proc identifying the namespace/rank of the participating processes (list)
25 26	• 'directives': Optional list of Python info containing directives controlling the operation (list)
27	• 'data': Optional Python bytearray of data to be circulated during fence operation (bytearray)
28	Returns:

1		• rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
2		• data - Python bytearray containing the aggregated data from all participants (bytearray)
3 4	A.3.3.5	See pmix_server_fencenb_fn_t for details Direct Modex
5 6 7		Summary Used by the PMIx server to request its local host contact the PMIx server on the remote node that hosts the specified proc to obtain and return a direct modex blob for that proc.
8		Format Python
^	PMIx v4.0	
9		<pre>def dmodex(args:dict is not None)</pre>
10 11		IN args Python dictionary containing:
12		• 'proc': Python proc of process whose data is being requested (dict)
13 14		 'directives': Optional list of Python info containing directives controlling the operation (list)
15		Returns:
16		• <i>rc</i> - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
17		• data - Python bytearray containing the data for the specified process (bytearray)
18 19	A.3.3.6	See pmix_server_dmodex_req_fn_t for details Publish
20 21		Summary Publish data per the PMIx API specification.
22	PMIx v4.0	Format Python —
23	Time vi.o	def publish(args:dict is not None) Python
24 25		IN args Python dictionary containing:
26		• 'proc': Python proc dictionary of process publishing the data (dict)
27		• 'directives': List of Python info containing data and directives (list)
28		Returns:
29		• <i>rc</i> - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
30		See pmix_server_publish_fn_t for details

```
A.3.3.7 Lookup
 1
 2
               Summary
 3
               Lookup published data.
               Format
                                                        Python
   PMIx v4.0
 5
               def lookup(args:dict is not None)
                                                        Python
               IN
 6
                    args
 7
                    Python dictionary containing:
                    • 'proc': Python proc of process seeking the data (dict)
 8
 9
                    • 'keys': List of Python strings (list)
10
                    • 'directives': Optional list of Python info containing directives (list)
               Returns:
11
               • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
12
               • pdata - List of pdata containing the returned results (list)
13
14
               See pmix server lookup fn t for details
    A.3.3.8
               Unpublish
15
               Summary
16
               Delete data from the data store.
17
               Format
18
                                                        Python
   PMIx v4.0
19
               def unpublish(args:dict is not None)
                                                        Python
               IN
20
                   args
                    Python dictionary containing:
21
                    • 'proc': Python proc of process unpublishing data (dict)
22
23
                    • 'keys': List of Python strings (list)
24
                    • 'directives': Optional list of Python info containing directives (list)
               Returns:
25
26
               • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
27
               See pmix server unpublish fn t for details
```

1	A.3.3.9	Spawn
2		Summary Spawn a set of applications/processes as per the PMIx_Spawn API.
4		Format
•	PMIx v4.0	Python —
5		def spawn(args:dict is not None)
		Python —
6 7		IN args Python dictionary containing:
8		• 'proc': Python proc of process making the request (dict)
9		• 'jobinfo': Optional list of Python info job-level directives and information (list)
0		• 'apps': List of Python app describing applications to be spawned (list)
1		Returns:
2		• rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
3		• nspace - Python string containing namespace of the spawned job (str)
4		See pmix_server_spawn_fn_t for details
5	A.3.3.10	Connect
6 7		Summary Record the specified processes as <i>connected</i> .
8		Format
	PMIx v4.0	Python —
9		def connect(args:dict is not None) Python
20 21		IN args Python dictionary containing:
22		 'procs': List of Python proc identifying the namespace/rank of the participating processes (list)
24 25		• 'directives': Optional list of Python info containing directives controlling the operation (list)
26		Returns:
27		• rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
28		See pmix_server_connect_fn_t for details

A.3.3.11 Disconnect 1 2 Summary 3 Disconnect a previously connected set of processes. **Format** Python PMIx v4.0 5 def disconnect(args:dict is not None) Python IN 6 args 7 Python dictionary containing: • 'procs': List of Python proc identifying the namespace/rank of the participating 8 9 processes (list) • 'directives': Optional list of Python **info** containing directives controlling the operation 10 11 (list) 12 Returns: 13 • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer) 14 See pmix server disconnect fn t for details A.3.3.12 Register Events 15 Summary 16 17 Register to receive notifications for the specified events. **Format** 18 Python PMIx v4.0 def register_events(args:dict is not None) 19 Python IN 20 args 21 Python dictionary containing: 22 • 'codes': List of Python integers (list) • 'directives': Optional list of Python **info** containing directives controlling the operation 23 (list) 24 25 Returns: 26 • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer) 27 See pmix_server_register_events_fn_t for details

A.3.3.13 Deregister Events 1 2 Summary 3 Deregister to receive notifications for the specified events. Format Python -PMIx v4.0 5 def deregister_events(args:dict is not None) Python IN 6 args 7 Python dictionary containing: • 'codes': List of Python integers (list) Returns: 9 10 • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer) See pmix server deregister events fn t for details 11 A.3.3.14 Notify Event 12 Summary 13 14 Notify the specified range of processes of an event. 15 Format Python PMIx v4.0def notify_event(args:dict is not None) 16 Python 17 IN args Python dictionary containing: 18 19 • 'code': Python integer pmix_status_t (integer) 20 • 'source': Python proc of process that generated the event (dict) • 'range': Python range in which the event is to be reported (integer) 21 22 • 'directives': Optional list of Python info directives (list) 23 Returns: 24 • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer) 25 See pmix_server_notify_event_fn_t for details A.3.3.15 Query 26 Summary 27 28 Query information from the resource manager.

```
Format
 1
                                                       Python
   PMIx v4.0
 2
               def query(args:dict is not None)
                                                       Python
               IN
 3
                    args
                    Python dictionary containing:
                    • 'source': Python proc of requesting process (dict)
 5
                    • 'queries': List of Python query directives (list)
 6
               Returns:
 7
               • rc - PMIX SUCCESS or a PMIx error code indicating the operation failed (integer)
 8
 9
               • info - List of Python info containing the returned results (list)
               See pmix_server_query_fn_t for details
10
    A.3.3.16 Tool Connected
11
12
               Summary
13
               Register that a tool has connected to the server.
               Format
14
                                                       Python
   PMIx v4.0
15
               def tool connected(args:dict is not None)
                                                       Python
16
               IN
                    args
                    Python dictionary containing:
17
18
                    • 'directives': Optional list of Python info info on the connecting tool (list)
               Returns:
19
20
               • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
               • proc - Python proc containing the assigned namespace:rank for the tool (dict)
21
22
               See pmix server tool connection fn t for details
    A.3.3.17 Log
23
               Summary
24
25
               Log data on behalf of a client.
```

1		Format
	PMIx v4.0	Python —
2		<pre>def log(args:dict is not None)</pre>
0		
3 4		IN args Python dictionary containing:
5		• 'source': Python proc of requesting process (dict)
6		• 'data': Optional list of Python info containing data to be logged (list)
7		• 'directives': Optional list of Python info containing directives (list)
8		Returns:
9		• rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
10		See pmix_server_log_fn_t for details
11	A.3.3.18	Allocate Resources
12 13		Summary Request allocation operations on behalf of a client.
14		Format Python
15	PMIx v4.0	def allocate(args:dict is not None)
13		Python
16 17		IN args Python dictionary containing:
18		• 'source': Python proc of requesting process (dict)
19		• 'action': Python allocdir specifying requested action (integer)
20		• 'directives': Optional list of Python info containing directives (list)
21		Returns:
22		• <i>rc</i> - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
23		• refarginfo - List of Python info containing results of requested operation (list)
24		See <pre>pmix_server_alloc_fn_t</pre> for details
25	A.3.3.19	Job Control
26 27		Summary Execute a job control action on behalf of a client.

```
Format
 1
                                                         Python
   PMIx v4.0
 2
               def job control(args:dict is not None)
                                                         Python
               IN
 3
                     args
                    Python dictionary containing:
                    • 'source': Python proc of requesting process (dict)
 5
                    • 'targets': List of Python proc specifying target processes (list)
 6
                    • 'directives': Optional list of Python info containing directives (list)
 8
               Returns:
 9
               • rc - PMIX SUCCESS or a PMIx error code indicating the operation failed (integer)
10
               See pmix_server_job_control_fn_t for details
    A.3.3.20 Monitor
11
               Summary
12
13
               Request that a client be monitored for activity.
               Format
14
                                                         Python
   PMIx v4.0
15
               def monitor(args:dict is not None)
                                                         Python
16
               IN
                     args
                    Python dictionary containing:
17
18
                     • 'source': Python proc of requesting process (dict)
                     • 'monitor': Python info attribute indicating the type of monitor being requested (dict)
19
                    • 'error': Status code to be used when generating an event notification (integer) alerting that
20
21
                       the monitor has been triggered.
22
                    • 'directives': Optional list of Python info containing directives (list)
23
               Returns:
24
               • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
25
               See pmix server monitor fn t for details
    A.3.3.21 Get Credential
26
27
               Summary
28
               Request a credential from the host environment
```

1	PMIx v4.0	Format Python
2	1 W11x V4.0	def get_credential(args:dict is not None)
		Python —
3 4		IN args Python dictionary containing:
5		• 'source': Python proc of requesting process (dict)
6		• 'directives': Optional list of Python info containing directives (list)
7		Returns:
8		• rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
9		• cred - Python byteobject containing returned credential (dict)
10		• <i>info</i> - List of Python info containing any additional info about the credential (list)
11		See pmix_server_get_cred_fn_t for details
12	A.3.3.22	Validate Credential
13 14		Summary Request validation of a credential
15	PMIx v4.0	Format Python
15 16	PMIx v4.0	
	PMIx v4.0	Python def validate_credential(args:dict is not None)
16 17	PMIx v4.0	Python def validate_credential(args:dict is not None) Python IN args
16 17 18	PMIx v4.0	Python def validate_credential(args:dict is not None) Python IN args Python dictionary containing:
16 17 18 19	PMIx v4.0	Python def validate_credential(args:dict is not None) Python IN args Python dictionary containing: • 'source': Python proc of requesting process (dict)
16 17 18 19 20	PMIx v4.0	Python def validate_credential(args:dict is not None) Python IN args Python dictionary containing: • 'source': Python proc of requesting process (dict) • 'credential': Python byteobject containing credential (dict)
16 17 18 19 20 21	PMIx v4.0	Python def validate_credential(args:dict is not None) Python IN args Python dictionary containing: • 'source': Python proc of requesting process (dict) • 'credential': Python byteobject containing credential (dict) • 'directives': Optional list of Python info containing directives (list)
16 17 18 19 20 21 22	PMIx v4.0	Python def validate_credential(args:dict is not None) Python IN args Python dictionary containing: • 'source': Python proc of requesting process (dict) • 'credential': Python byteobject containing credential (dict) • 'directives': Optional list of Python info containing directives (list) Returns:
16 17 18 19 20 21 22 23	PMIx v4.0	Python def validate_credential(args:dict is not None) Python IN args Python dictionary containing: • 'source': Python proc of requesting process (dict) • 'credential': Python byteobject containing credential (dict) • 'directives': Optional list of Python info containing directives (list) Returns: • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer) • info - List of Python info containing any additional info from the credential (list) See pmix_server_validate_cred_fn_t for details
16 17 18 19 20 21 22 23 24	PMIx v4.0 A.3.3.23	Python def validate_credential(args:dict is not None) Python IN args Python dictionary containing: • 'source': Python proc of requesting process (dict) • 'credential': Python byteobject containing credential (dict) • 'directives': Optional list of Python info containing directives (list) Returns: • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer) • info - List of Python info containing any additional info from the credential (list) See pmix_server_validate_cred_fn_t for details

```
Format
 1
                                                        Python
   PMIx v4.0
 2
               def iof pull(args:dict is not None)
                                                         Python
               IN
 3
                     args
                    Python dictionary containing:
                    • 'sources': List of Python proc of processes whose IO is being requested (list)
 5
                     • 'channels': Bitmask of Python channel identifying IO channels to be forwarded (integer)
 6
                    • 'directives': Optional list of Python info containing directives (list)
 7
 8
               Returns:
 9
               • rc - PMIX SUCCESS or a PMIx error code indicating the operation failed (integer)
               See pmix server iof fn t for details
10
    A.3.3.24 IO Push
11
               Summary
12
13
               Pass standard input data to the host environment for transmission to specified recipients.
               Format
14
                                                         Python
   PMIx v4.0
15
               def iof push(args:dict is not None)
                                                         Python
16
               IN
                     args
                    Python dictionary containing:
17
18
                     • 'source': Python proc of process whose input is being forwarded (dict)
                    • 'payload': Python byteobject containing input bytes (dict)
19
                    • 'targets': List of proc of processes that are to receive the payload (list)
20
                    • 'directives': Optional list of Python info containing directives (list)
21
22
               Returns:
23
               • rc - PMIX SUCCESS or a PMIx error code indicating the operation failed (integer)
24
               See pmix_server_stdin_fn_t for details
    A.3.3.25 Group Operations
25
               Summary
26
27
               Request group operations (construct, destruct, etc.) on behalf of a set of processes.
```

1		Format Python
_	PMIx v4.0	
2		def group(args:dict is not None) Python
3		IN args
4		Python dictionary containing:
5		• 'op': Operation host is to perform on the specified group (integer)
6		• 'group': String identifier of target group (str)
7		• 'procs': List of Python proc of participating processes (dict)
8		• 'directives': Optional list of Python info containing directives (list)
9		Returns:
10		• rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
11		• refarginfo - List of Python info containing results of requested operation (list)
12		See pmix_server_grp_fn_t for details
13	A.3.3.26	Fabric Operations
14 15		Summary Request fabric-related operations (e.g., information on a fabric) on behalf of a tool or other process.
16	PMIx v4.0	Format Python
17		<pre>def fabric(args:dict is not None)</pre>
18 19		IN args Python dictionary containing:
20		• 'source': Python proc of requesting process (dict)
21		• 'op': Operation host is to perform on the specified fabric (integer)
22		• 'directives': Optional list of Python info containing directives (list)
23		Returns:
24		• rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
25		• refarginfo - List of Python info containing results of requested operation (list)
26		See pmix_server_fabric_fn_t for details

A.4 PMIxClient

The client Python class is by far the richest in terms of APIs as it houses all the APIs that an 2 3 application might utilize. Due to the datatype translation requirements of the C-Python interface, 4 only the blocking form of each API is supported – providing a Python callback function directly to 5 the C interface underlying the bindings was not a supportable option.

Client.init A.4.1

Summary 7

8

11

12

13

14

15

21

22

23

Initialize the PMIx client library after obtaining a new PMIxClient object

_____ Python _____ PMIx v4.010 rc, proc = myclient.init(info:list) Python

IN info

Format

List of Python **info** dictionaries (list)

Returns:

- rc PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
- proc a Python proc dictionary (dict)

16 See PMIx_Init for description of all relevant attributes and behaviors

Client.initialized A.4.2

18 Format ----- Python -PMIx v4.0 rc = myclient.initialized() 19 Python

20 Returns:

> • rc - a value of 1 (true) will be returned if the PMIx library has been initialized, and 0 (false) otherwise (integer)

See PMIx Initialized for description of all relevant attributes and behaviors

A.4.3 Client.get version

```
Format
                                                    Python —
   PMIx v4.0
3
              vers = myclient.get_version()
                                                    Python
              Returns:
4
              • vers - Python string containing the version of the PMIx library (e.g., "3.1.4") (integer)
 5
6
              See PMIx_Get_version for description of all relevant attributes and behaviors
    A.4.4 Client.finalize
              Summary
8
              Finalize the PMIx client library.
9
              Format
10
                                                    Python -
   PMIx v4.0
              rc = myclient.finalize(info:list)
11
                                                    Python -
12
              IN
                   info
                   List of Python info dictionaries (list)
13
14
              Returns:
15
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
16
              See PMIx_Finalize for description of all relevant attributes and behaviors
              Client.abort
    A.4.5
              Summary
18
              Request that the provided list of procs be aborted
19
```

```
Format
1
                                                     Python -
   PMIx v4.0
2
               rc = myclient.abort(status:integer, msg:str, targets:list)
                                                      Python
               IN
3
                    status
4
                   PMIx status to be returned on exit (integer)
5
               IN
                   String message to be printed (string)
6
               IN
                   targets
7
                   List of Python proc dictionaries (list)
8
9
               Returns:
10
               • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
               See PMIx_Abort for description of all relevant attributes and behaviors
11
              Client.store internal
    A.4.6
               Summary
13
14
               Store some data locally for retrieval by other areas of the process
               Format
15
                                                      Python
   PMIx v4.0
               rc = myclient.store_internal(proc:dict, key:str, value:dict)
16
                                                      Python
17
               IN
                    proc
                   Python proc dictionary of the process being referenced (dict)
18
19
               IN
                   String key of the data (string)
20
               IN
                    value
21
22
                   Python value dictionary (dict)
23
               Returns:
24
               • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
25
               See PMIx_Store_internal for details
    A.4.7
              Client.put
               Summary
27
28
               Push a key/value pair into the client's namespace.
```

```
Format
1
                                                   Python —
   PMIx v4.0
2
              rc = myclient.put(scope:integer, key:str, value:dict)
                                                    Python
              IN
3
                   scope
4
                   Scope of the data being posted (integer)
5
              IN
                   String key of the data (string)
6
              IN
                  value
7
                   Python value dictionary (dict)
8
9
              Returns:
10
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
              See PMIx_Put for description of all relevant attributes and behaviors
11
    A.4.8
             Client.commit
              Summary
13
14
              Push all previously PMIxClient.put values to the local PMIx server.
              Format
15
                                   Python
   PMIx v4.0
16
              rc = myclient.commit()
                                                  Python
17
              Returns:
18
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
19
              See PMIx_Commit for description of all relevant attributes and behaviors
             Client.fence
    A.4.9
21
              Summary
22
              Execute a blocking barrier across the processes identified in the specified list
```

```
Format
1
                                                      Python
   PMIx v4.0
2
               rc = myclient.fence(peers:list, directives:list)
                                                      Python
               IN
3
                    peers
 4
                    List of Python proc dictionaries (list)
5
               IN
                  directives
                    List of Python info dictionaries (list)
6
               Returns:
 7
 8
               • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer)
9
               See PMIx Fence for description of all relevant attributes and behaviors
    A.4.10
                Client.get
               Summary
11
               Retrieve a key/value pair
12
13
               Format
                                                      Python
   PMIx v4.0
14
               rc, val = myclient.get(proc:dict, key:str, directives:list)
                                                      Python
               IN
15
                   proc
                    Python proc whose data is being requested (dict)
16
17
               IN
                    Python string key of the data to be returned (str)
18
19
               IN
                    directives
                    List of Python info dictionaries (list)
20
               Returns:
21
22
               • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
23
               • val - Python value containing the returned data (dict)
24
               See PMIx_Get for description of all relevant attributes and behaviors
    A.4.11
                Client.publish
25
               Summary
26
27
               Publish data for later access via PMIx Lookup.
```

```
Format
1
                                                      Python
   PMIx v4.0
2
               rc = myclient.publish(directives:list)
                                                       Python
               IN
3
                    directives
                    List of Python info dictionaries containing data to be published and directives (list)
               Returns:
 5
               • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer)
6
 7
               See PMIx_Publish for description of all relevant attributes and behaviors
    A.4.12
                Client.lookup
               Summary
9
               Lookup information published by this or another process with PMIx Publish.
10
               Format
11
                                                      Python
   PMIx v4.0
12
               rc,info = myclient.lookup(pdata:list, directives:list)
                                                       Python
13
               IN
                   pdata
                    List of Python pdata dictionaries identifying data to be retrieved (list)
14
                    directives
15
                    List of Python info dictionaries (list)
16
17
               Returns:
18
               • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
               • info - Python list of info containing the returned data (list)
19
20
               See PMIx_Lookup for description of all relevant attributes and behaviors
    A.4.13
                Client.unpublish
22
               Summary
               Delete data published by this process with PMIx Publish.
23
```

```
Format
1
                                                      Python
   PMIx v4.0
2
               rc = myclient.unpublish(keys:list, directives:list)
                                                      Python
               IN
3
                    keys
4
                   List of Python string keys identifying data to be deleted (list)
5
               IN
                   directives
                   List of Python info dictionaries (list)
6
               Returns:
 7
 8
               • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer)
9
               See PMIx Unpublish for description of all relevant attributes and behaviors
    A.4.14
                Client.spawn
               Summary
11
               Spawn a new job.
12
13
               Format
                                                     Python
   PMIx v4.0
               rc,nspace = myclient.spawn(jobinfo:list, apps:list)
14
                                                      Python
               IN
15
                    jobinfo
                   List of Python info dictionaries (list)
16
17
               IN
                    apps
                   List of Python app dictionaries (list)
18
19
               Returns:
20
               • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
21
               • nspace - Python nspace of the new job (dict)
22
               See PMIx_Spawn for description of all relevant attributes and behaviors
              Client.connect
    A.4.15
               Summary
24
25
               Connect namespaces.
```

```
Format
1
                                                    Python
   PMIx v4.0
2
              rc = myclient.connect(peers:list, directives:list)
                                                    Python
              IN
3
                   peers
 4
                   List of Python proc dictionaries (list)
 5
              IN
                 directives
                   List of Python info dictionaries (list)
6
              Returns:
 7
              • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer)
9
              See PMIx Connect for description of all relevant attributes and behaviors
    A.4.16
               Client.disconnect
              Summary
11
              Disconnect namespaces.
12
13
              Format
                                                    Python —
   PMIx v4.0
14
              rc = myclient.disconnect(peers:list, directives:list)
                                                    Python
              IN
15
                   peers
                   List of Python proc dictionaries (list)
16
17
              IN
                  directives
                   List of Python info dictionaries (list)
18
19
              Returns:
20
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
              See PMIx_Disconnect for description of all relevant attributes and behaviors
21
    A.4.17 Client.resolve_peers
              Summary
23
24
              Return list of processes within the specified nspace on the given node.
```

```
Format
1
                                                      Python
   PMIx v4.0
2
               rc,procs = myclient.resolve_peers(node:str, nspace:str)
                                                      Python
               IN
3
                    node
 4
                   Name of node whose processes are being requested (str)
5
               IN
                    nspace
                   Python nspace whose processes are to be returned (str)
6
 7
               Returns:
8
               • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer)
9
               • procs - List of Python proc dictionaries (list)
10
               See PMIx_Resolve_peers for description of all relevant attributes and behaviors
                Client.resolve nodes
    A.4.18
12
               Summary
13
               Return list of nodes hosting processes within the specified nspace.
               Format
14
                                                      Python –
   PMIx v4.0
               rc, nodes = myclient.resolve_nodes(nspace:str)
15
                                                      Python
16
               IN
                    nspace
17
                   Python nspace (str)
18
               Returns:
               • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
19
20
               • nodes - List of Python string node names (list)
21
               See PMIx Resolve nodes for description of all relevant attributes and behaviors
    A.4.19
                Client.query
23
               Summary
               Query information about the system in general
24
```

```
Format
1
                                                      Python
   PMIx v4.0
2
               rc,info = myclient.query(queries:list)
                                                      Python
               IN
3
                    queries
                    List of Python query dictionaries (list)
 5
               Returns:
               • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
 6
               • info - List of Python info containing results of the query (list)
               See PMIx Query info nb for description of all relevant attributes and behaviors
8
    A.4.20 Client.log
               Summary
10
               Log data to a central data service/store
11
               Format
12
                                                      Python –
   PMIx v4.0
13
               rc = myclient.log(data:list, directives:list)
                                                      Python
14
               IN
                   data
                    List of Python info (list)
15
               IN
                    directives
16
17
                    Optional list of Python info (list)
18
               Returns:
               • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
19
20
               See PMIx_Log for description of all relevant attributes and behaviors
    A.4.21
                Client.allocate
22
               Summary
23
               Request an allocation operation from the host resource manager.
```

```
Format
1
                                                     Python —
   PMIx v4.0
2
              rc,info = myclient.allocate(request:integer, directives:list)
                                                     Python
              IN
3
                   request
4
                   Python allocdir specifying requested operation (integer)
5
              IN
                  directives
                   List of Python info describing request (list)
6
              Returns:
 7
8
              • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer)
9
              • info - List of Python info containing results of the request (list)
10
              See PMIx Allocation request nb for description of all relevant attributes and behaviors
    A.4.22
               Client.job ctrl
              Summary
12
              Request a job control action
13
              Format
14
                                                     Python ————
   PMIx v4.0
              rc,info = myclient.job_ctrl(targets:list, directives:list)
15
                                                     Python
16
              IN
                   targets
17
                   List of Python proc specifying targets of requested operation (integer)
              IN
                   directives
18
                   List of Python info describing operation to be performed (list)
19
              Returns:
20
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
21
              • info - List of Python info containing results of the request (list)
22
23
              See PMIx_Job_control_nb for description of all relevant attributes and behaviors
    A.4.23 Client.monitor
              Summary
25
26
              Request that something be monitored
```

```
Format
1
                                                    Python —
   PMIx v4.0
2
              rc, info = myclient.monitor(monitor:dict, error_code:integer, directives:list
                                                     Python
              IN
3
                  monitor
4
                   Python info specifying specifying the type of monitor being requested (dict)
5
              IN error_code
                   Status code to be used when generating an event notification alerting that the monitor has
6
7
                   been triggered (integer)
                  directives
              IN
8
9
                   List of Python info describing request (list)
              Returns:
10
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
11
12
              • info - List of Python info containing results of the request (list)
13
              See PMIx Process monitor nb for description of all relevant attributes and behaviors
    A.4.24
               Client.get credential
              Summary
15
              Request a credential from the PMIx server/SMS
16
17
              Format
                                                     Python -
   PMIx v4.0
18
              rc,cred = myclient.get_credential(directives:list)
                                                     Python
              IN
19
                   directives
                   Optional list of Python info describing request (list)
20
21
              Returns:
22
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
23
              • cred - Python byteobject containing returned credential (dict)
24
              See PMIx_Get_credential for description of all relevant attributes and behaviors
    A.4.25
               Client.validate_credential
26
              Summary
27
              Request validation of a credential by the PMIx server/SMS
```

```
Format
1
                                          ——— Python ——————
   PMIx v4.0
              rc,info = myclient.validate_credential(cred:dict, directives:list)
2
                                                   Python —
              IN
3
                   cred
4
                   Python byteobject containing credential (dict)
              IN
                 directives
 5
6
                   Optional list of Python info describing request (list)
 7
              Returns:
8
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
9
              • info - List of Python info containing additional results of the request (list)
              See PMIx_Validate_credential for description of all relevant attributes and behaviors
10
    A.4.26
               Client.group construct
12
              Summary
              Construct a new group composed of the specified processes and identified with the provided group
13
14
              identifier
              Format
15
                                         ——— Python —————
   PMIx v4.0
              rc,info = myclient.construct_group(grp:string, members:list, directives:list
16
                                                   Python ———
17
              IN
                   grp
                   Python string identifier for the group (str)
18
              IN members
19
                  List of Python proc dictionaries identifying group members (list)
20
              IN
21
                   directives
22
                   Optional list of Python info describing request (list)
23
              Returns:
24
              • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer)
25
              • info - List of Python info containing results of the request (list)
26
              See PMIx_Group_construct for description of all relevant attributes and behaviors
    A.4.27 Client.group invite
              Summary
28
29
              Explicitly invite specified processes to join a group
```

446

```
Format
1
                                           ——— Python ——————
   PMIx v4.0
              rc, info = myclient.group_invite(grp:string, members:list, directives:list)
2
                                                     Python —
              IN
3
                   grp
4
                   Python string identifier for the group (str)
5
              IN
                  members
                   List of Python proc dictionaries identifying processes to be invited (list)
6
7
              IN
                  directives
                   Optional list of Python info describing request (list)
8
9
              Returns:
10
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
              • info - List of Python info containing results of the request (list)
11
12
              See PMIx Group invite for description of all relevant attributes and behaviors
    A.4.28 Client.group join
13
              Summarv
14
              Respond to an invitation to join a group that is being asynchronously constructed
15
              Format
16
                                           ——— Python ——————
   PMIx v4.0
              rc,info = myclient.group_join(grp:string, leader:dict, opt:integer, directiv
17
                                                     Pvthon
              IN
18
                   grp
                   Python string identifier for the group (str)
19
              IN
20
                   Python proc dictionary identifying process leading the group (dict)
21
22
              IN
                   One of the pmix group opt t values indicating decline/accept (integer)
23
24
              IN
                   directives
                   Optional list of Python info describing request (list)
25
26
              Returns:
27
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
              • info - List of Python info containing results of the request (list)
28
29
              See PMIx_Group_join for description of all relevant attributes and behaviors
```

A.4.29 Client.group_leave Summary 2 3 Leave a PMIx Group **Format** Python PMIx v4.0 5 rc = myclient.group_leave(grp:string, directives:list) Python -6 IN grp 7 Python string identifier for the group (str) IN directives 8 9 Optional list of Python **info** describing request (list) Returns: 10 11 • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer) 12 See PMIx_Group_leave for description of all relevant attributes and behaviors A.4.30 Client.group destruct Summary 14 Destruct a PMIx Group 15 16 Format Python -PMIx v4.0 17 rc = myclient.group_destruct(grp:string, directives:list) Python — IN 18 grp 19 Python string identifier for the group (str) directives IN 20 Optional list of Python **info** describing request (list) 21 22 Returns: 23 • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer) 24 See PMIx Group destruct for description of all relevant attributes and behaviors A.4.31 Client.register event handler 25 Summarv 26 Register an event handler to report events. 27

1	PMIx v4.0	Format Python —
2	FMIX V4.0	rc,id = myclient.register_event_handler(codes:list, directives:list, cbfunc)
_		Python — Python —
3 4 5 6 7 8 9 10		IN codes List of Python integer status codes that should be reported to this handler (llist) IN directives Optional list of Python info describing request (list) IN cbfunc Python evhandler to be called when event is received (func) Returns: • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer) • id - PMIx reference identifier for handler (integer)
12		See PMIx_Register_event_handler for description of all relevant attributes and behaviors
13	A.4.32	Client.deregister_event_handler
14 15		Summary Deregister an event handler
	PMIx v4.0	•
15	PMIx v4.0	Deregister an event handler Format
15 16	PMIx v4.0	Format Python myclient.deregister_event_handler(id:integer)
15161718	PMIx v4.0	Format Python myclient.deregister_event_handler(id:integer) Python Python
15 16 17 18 19	PMIx v4.0	Python myclient.deregister_event_handler(id:integer) Python Python N id PMIx reference identifier for handler (integer)
15 16 17 18 19 20 21		Format Python myclient.deregister_event_handler(id:integer) Python IN id PMIx reference identifier for handler (integer) Returns: None See PMIx_Deregister_event_handler for description of all relevant attributes and

```
Format
1
                                                      Python –
   PMIx v4.0
2
               rc = myclient.notify_event(status:integer, source:dict,
                                                  range:integer, directives:list)
3
                                                      Python
4
               IN
                  status
                   PMIx status code indicating the event being reported (integer)
 5
               IN
6
                    source
7
                   Python proc of the process that generated the event (dict)
8
               IN
                    range
9
                   Python range in which the event is to be reported (integer)
                    directives
10
               IN
                   Optional list of Python info dictionaries describing the event (list)
11
12
               Returns:
               • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
13
               See PMIx Notify event for description of all relevant attributes and behaviors
14
    A.4.34 Client.fabric_register
15
               Summary
16
               Register for access to fabric-related information, including communication cost matrix.
17
18
               Format
                                                — Python -
   PMIx v4.0
               rc, fabricinfo = myserver.fabric_register(directives:list)
19
                                                      Python
               IN
20
                    directives
                   Optional list of Python info containing directives (list)
21
22
               Returns:
23
               • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
24
               • fabricinfo - List of Python info containing fabric info (list)
25
               See PMIx_Fabric_register for details
    A.4.35
               Client.fabric_update
26
               Summarv
27
28
               Update fabric-related information, including communication cost matrix.
```

450

```
Format
1
                                                  Python
   PMIx v4.0
2
              rc, fabricinfo = myserver.fabric_update()
                                                  Python
3
              Returns:
 4
              • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer)
              • fabricinfo - List of Python info containing updated fabric info (list)
 5
              See PMIx Fabric update for details
 6
    A.4.36
              Client.fabric deregister
              Summary
8
9
              Deregister fabric
10
              Format
                                                  Python
   PMIx v4.0
              rc = myserver.fabric_deregister()
11
                                                  Python
              Returns:
12
13
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
              See PMIx_Fabric_deregister for details
14
    A.4.37 Client.fabric get vertex info
              Summary
16
              Given a communication cost matrix index for a specified fabric, return an array of information
17
              describing the corresponding NIC.
18
19
              Format
                                              PMIx v4.0
              rc,nicinfo = myserver.fabric_get_vertex_info(index:integer)
20
                                                  Python
21
              Returns:
22
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
23
              • nicinfo - List of Python info describing the referenced NIC (list)
24
              See PMIx_Fabric_get_vertex_info for details
```

A.4.38 Client.fabric get device index Summary 2 3 Given info describing a given vertex, return the corresponding communication cost matrix index **Format** Python -PMIx v4.0 5 rc,index = myserver.fabric_get_device_index(info:list) Python IN info 6 List of Python **info** containing vertex description (list) 7 8 Returns: • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer) 9 • *index* - Index of corresponding NIC (integer) 10 See PMIx_Fabric_get_device_index for details 11 A.4.39 Client.error_string 13 Summary Pretty-print string representation of pmix_status_t . 14 **Format** 15 Python — PMIx v4.0 rep = myclient.error_string(status:integer) 16 Python IN 17 status PMIx status code (integer) 18 Returns: 19 20 • rep - String representation of the provided status code (str) 21 See PMIx Error string for further details A.4.40 Client.proc state string

Summary

23 24

Pretty-print string representation of pmix_proc_state_t.

1	PMIx v4.0	Format Python
2		rep = myclient.proc_state_string(state:integer)
		Python —
3 4		IN state PMIx process state code (integer)
5		Returns:
6		• rep - String representation of the provided process state (str)
7		See PMIx_Proc_state_string for further details
8	A.4.41	Client.scope_string
9 10		Summary Pretty-print string representation of pmix_scope_t.
11		Format
	PMIx v4.0	Python —
12		<pre>rep = myclient.scope_string(scope:integer)</pre>
13 14		IN scope PMIx scope value (integer)
15		Returns:
16		• <i>rep</i> - String representation of the provided scope (str)
17		See PMIx_Scope_string for further details
18	A.4.42	Client.persistence_string
19 20		Summary Pretty-print string representation of pmix_persistence_t.
21		Format
	PMIx v4.0	Python —
22		<pre>rep = myclient.persistence_string(persistence:integer)</pre>
23 24		IN persistence PMIx persistence value (integer)
25		Returns:
26		• rep - String representation of the provided persistence (str)
27		See PMIx_Persistence_string for further details

A.4.43 Client.data range string Summary 3 Pretty-print string representation of pmix_data_range_t. Format Python PMIx v4.0 5 rep = myclient.data_range_string(range:integer) Python IN 6 range 7 PMIx data range value (integer) 8 Returns: 9 • rep - String representation of the provided data range (str) See PMIx_Data_range_string for further details 10 Client.info directives string A.4.44 Summary 12 13 Pretty-print string representation of **pmix_info_directives_t**. Format 14 Python -PMIx v4.0 15 rep = myclient.info_directives_string(directives:integer) Python — IN 16 directives PMIx info directives value (integer) 17 Returns: 18 • rep - String representation of the provided info directives (str) 19 20 See PMIx Info directives string for further details

21 A.4.45 Client.data type string

22 Summary

Pretty-print string representation of **pmix_data_type_t**.

```
Format
1
                                                   Python
   PMIx v4.0
2
              rep = myclient.data_type_string(dtype:integer)
                                                   Python
3
              IN
                   dtype
                   PMIx datatype value (integer)
 4
              Returns:
              • rep - String representation of the provided datatype (str)
6
              See PMIx_Data_type_string for further details
 7
    A.4.46
               Client.alloc directive string
8
              Summary
9
              Pretty-print string representation of pmix alloc directive t.
10
              Format
11
                                                    Python
   PMIx v4.0
12
              rep = myclient.alloc_directive_string(adir:integer)
                                                    Python
13
              IN
                   adir
                   PMIx allocation directive value (integer)
14
15
              Returns:
              • rep - String representation of the provided allocation directive (str)
16
17
              See PMIx Alloc directive string for further details
    A.4.47
               Client.iof channel string
18
              Summary
19
              Pretty-print string representation of pmix_iof_channel_t.
20
              Format
21
                                                   Python
   PMIx v4.0
22
              rep = myclient.iof_channel_string(channel:integer)
                                                   Python
              IN
23
                   channel
                   PMIx IOF channel value (integer)
24
25
              Returns:
26
              • rep - String representation of the provided IOF channel (str)
27
              See PMIx_IOF_channel_string for further details
```

A.4.48 Client.job state string Summary 3 Pretty-print string representation of pmix_job_state_t. Format Python PMIx v4.0 5 rep = myclient.job_state_string(state:integer) Python IN state 6 7 PMIx job state value (integer) 8 Returns: 9 • rep - String representation of the provided job state (str) See PMIx_Job_state_string for further details 10 A.4.49 Client.get attribute string Summary 12 13 Pretty-print string representation of a PMIx attribute. Format 14 _____ Python _ PMIx v4.0 15 rep = myclient.get_attribute_string(attribute:str) Python IN 16 attribute PMIx attribute name (string) 17 Returns: 18 • rep - String representation of the provided attribute (str) 19 20 See PMIx Get attribute string for further details

21 A.4.50 Client.get attribute name

22 Summary

Pretty-print name of a PMIx attribute corresponding to the provided string

```
Format
1
                                                     Python
   PMIx v4.0
2
               rep = myclient.get_attribute_name(attribute:str)
                                                      Python
3
               IN
                    attributestring
                   Attribute string (string)
               Returns:
 5
               • rep - Attribute name corresponding to the provided string (str)
6
               See PMIx_Get_attribute_name for further details
    A.4.51
                Client.link state string
9
               Summary
               Pretty-print string representation of pmix_link_state_t.
10
               Format
11
                                                     Python
   PMIx v4.0
12
               rep = myclient.link_state_string(state:integer)
                                                      Python
13
               IN
                   state
14
                   PMIx link state value (integer)
               Returns:
15
16
               • rep - String representation of the provided link state (str)
17
               See PMIx_Link_state_string for further details
           PMIxServer
   A.5
19
               The server Python class inherits the Python "client" class as its parent. Thus, it includes all client
               functions in addition to the ones defined in this section.
20
              Server.init
    A.5.1
22
               Summary
23
               Initialize the PMIx server library after obtaining a new PMIxServer object
```

```
Format
 1
                                                   Python
   PMIx v4.0
 2
              rc = myserver.init(directives:list, map:dict)
                                                   Python
              IN
 3
                   directives
 4
                   List of Python info dictionaries (list)
 5
              IN
                   Python dictionary key-function pairs that map server module callback functions to
 6
                   provided implementations (dict)
 7
              Returns:
 8
              • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer)
 9
10
              See PMIx server init for description of all relevant attributes and behaviors
    A.5.2
             Server.finalize
              Summary
12
              Finalize the PMIx server library
13
              Format
14
                                         ——— Python ——————
   PMIx v4.0
15
              rc = myserver.finalize()
                                                   Python
              Returns:
16
17
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
18
              See PMIx server finalize for details
    A.5.3
             Server.generate regex
20
              Summary
21
              Generate a regular expression representation of the input strings.
```

1		Format
	PMIx v4.0	Python —
2		<pre>rc,regex = myserver.generate_regex(input:list)</pre>
		- Fytholi
3 4		IN input List of Python strings (e.g., node names) (list)
5		Returns:
6		• rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
7 8		 regex - Python bytearray containing regular expression representation of the input list (bytearray)
9		See PMIx_generate_regex for details
10	A.5.4	Server.generate_ppn
11 12		Summary Generate a regular expression representation of the input strings.
13	PMIx v4.0	Format Python
14		rc,regex = myserver.generate_ppn(input:list) Python
15 16 17		IN input List of Python strings, each string consisting of a comma-delimited list of ranks on each node, with the strings being in the same order as the node names provided to "generate_regex" (list)
18		Returns:
19		• rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
20 21		 regex - Python bytearray containing regular expression representation of the input list (bytearray)
22		See PMIx_generate_ppn for details
23	A.5.5	Server.register_nspace
24 25		Setup the data about a particular namespace.

1	PMIx v4.0	Format Python ————————————————————————————————————
2	1 mix v4.0	rc = myserver.register_nspace(nspace:str,
3		nlocalprocs:integer,
4		directives:list)
		Python —
5		IN nspace
6		Python string containing the namespace (str)
7		IN nlocalprocs
8		Number of local processes (integer) IN directives
9 10		List of Python info dictionaries (list)
11		Returns:
12		• rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
13		See PMIx_server_register_nspace for description of all relevant attributes and behaviors
14	A.5.6	Server.deregister_nspace
15		Summary
16		Deregister a namespace.
17		Format
	PMIx v4.0	Python —
18		<pre>myserver.deregister_nspace(nspace:str)</pre>
		Python —
19		IN nspace
20		Python string containing the namespace (str)
21		Returns: None
22		See PMIx_server_deregister_nspace for details
23	A.5.7	Server.register_client
24		Summary
25		Register a client process with the PMIx server library

```
Format
1
                                         ——— Python ——————
   PMIx v4.0
2
              rc = myserver.register_client(proc:dict, uid:integer, gid:integer)
                                                   Python
              IN
3
                   proc
4
                  Python proc dictionary identifying the client process (dict)
5
              IN
                  Linux uid value for user executing client process (integer)
6
              IN
7
                  Linux gid value for user executing client process (integer)
8
9
              Returns:
10
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
              See PMIx_server_register_client for details
11
             Server.deregister_client
    A.5.8
              Summary
13
14
              Dergister a client process and purge all data relating to it
              Format
15
                                                   Python -
   PMIx v4.0
              myserver.deregister_client(proc:dict)
16
                                                   Python
17
              IN
                   proc
                  Python proc dictionary identifying the client process (dict)
18
19
              Returns: None
20
              See PMIx_server_deregister_client for details
    A.5.9
             Server.setup_fork
22
              Summary
23
              Setup the environment of a child process that is to be forked by the host
```

```
Format
1
                                                     Python
   PMIx v4.0
2
              rc = myserver.setup_fork(proc:dict, envin:dict)
                                                     Python
              IN
3
                  proc
 4
                   Python proc dictionary identifying the client process (dict)
              INOUT envin
5
                   Python dictionary containing the environment to be passed to the client (dict)
6
 7
              Returns:
              • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer)
9
              See PMIx server setup fork for details
    A.5.10
               Server.dmodex request
              Summary
11
              Function by which the host server can request modex data from the local PMIx server.
12
13
              Format
                                                     Python —
   PMIx v4.0
              rc,data = myserver.dmodex_request(proc:dict)
14
                                                     Python
              IN
15
                   proc
                   Python proc dictionary identifying the process whose data is requested (dict)
16
17
18
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
              • data - Python byteobject containing the returned data (dict)
19
20
              See PMIx_server_dmodex_request for details
               Server.setup_application
    A.5.11
22
              Summary
23
              Function by which the resource manager can request application-specific setup data prior to launch
24
              of a job.
```

1		Format Python
_	PMIx v4.0	
2		rc,info = myserver.setup_application(nspace:str, directives:list)
3 4 5 6 7 8		IN nspace Namespace whose setup information is being requested (str) IN directives Python list of info directives Returns: • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer) • info - Python list of info dictionaries containing the returned data (list)
10		See PMIx_server_setup_application for details
11	A.5.12	Server.register_attributes
12 13		Summary Register host environment attribute support for a function.
14		Format
	PMIx v4.0	Python —
15		<pre>rc = myserver.register_attributes(function:str, attrs:list)</pre>
16 17 18 19		 IN function Name of the function (str) IN attrs Python list of regattr describing the supported attributes
20		Returns:
21		• rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
22		See PMIx_Register_attributes for details
23	A.5.13	Server.setup_local_support
24 25 26		Summary Function by which the local PMIx server can perform any application-specific operations prior to spawning local clients of a given application

```
Format
1
                                                  Python —
   PMIx v4.0
2
              rc = myserver.setup_local_support(nspace:str, info:list)
                                                  Python
3
              IN
                  nspace
 4
                  Namespace whose setup information is being requested (str)
 5
              IN
6
                  Python list of info containing the setup data (list)
 7
              Returns:
8
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
9
              See PMIx_server_setup_local_support for details
   A.5.14 Server.iof_deliver
              Summary
11
              Function by which the host environment can pass forwarded IO to the PMIx server library for
12
13
              distribution to its clients.
              Format
14
                                         PMIx v4.0
15
              rc = myserver.iof_deliver(source:dict, channel:integer,
                                             data:dict, directives:list)
16
                                                  Python ———
              IN
17
                 source
18
                  Python proc dictionary identifying the process who generated the data (dict)
              IN
                  channel
19
                  Python channel bitmask identifying IO channel of the provided data (integer)
20
              IN
21
22
                  Python byteobject containing the data (dict)
                 directives
23
                  Python list of info containing directives (list)
24
25
26
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
27
              See PMIx_server_IOF_deliver for details
              Server.collect inventory
    A.5.15
29
              Summary
30
              Collect inventory of resources on a node
```

```
Format
1
                                                      Python -
   PMIx v4.0
2
               rc, info = myserver.collect_inventory(directives:list)
                                                      Python
               IN
3
                    directives
                   Optional Python list of info containing directives (list)
               Returns:
 5
               • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
 6
 7
               • info - Python list of info containing the returned data (list)
               See PMIx_server_collect_inventory for details
8
    A.5.16 Server.deliver inventory
10
               Summary
               Pass collected inventory to the PMIx server library for storage
11
12
                                                      Python
   PMIx v4.0
13
               rc = myserver.deliver_inventory(info:list, directives:list)
                                                      Python
               IN
14
                    info
                   - Python list of info dictionaries containing the inventory data (list)
15
               IN
16
                    directives
                   Python list of info dictionaries containing directives (list)
17
               Returns:
18
19
               • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
20
               See PMIx_server_deliver_inventory for details
    A.6 PMIxTool
22
               The tool Python class inherits the Python "server" class as its parent. Thus, it includes all client and
23
               server functions in addition to the ones defined in this section.
    A.6.1 Tool.init
25
               Summary
```

Initialize the PMIx tool library after obtaining a new PMIxTool object

```
Format
1
                                                      Python
   PMIx v4.0
               rc,proc = mytool.init(info:list)
2
                                                       Python
               IN
3
                    info
                    List of Python info directives (list)
 5
               Returns:
               • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer)
6
               • proc - a Python proc (dict)
 7
               See PMIx tool init for description of all relevant attributes and behaviors
8
    A.6.2
              Tool.finalize
               Summary
10
               Finalize the PMIx tool library, closing the connection to the server.
11
12
               Format
                                                      Python
   PMIx v4.0
13
               rc = mytool.finalize()
                                                      Python
14
               Returns:
               • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
15
16
               See PMIx_tool_finalize for description of all relevant attributes and behaviors
    A.6.3
              Tool.connect_to_server
18
               Summary
19
               Switch connection from the current PMIx server to another one, or initialize a connection to a
20
               specified server.
```

```
Format
1
                                                     Python –
   PMIx v4.0
2
               rc,proc = mytool.connect_to_server(info:list)
                                                     Python
               IN
                   info
3
 4
                   List of Python info dictionaries (list)
               Returns:
 5
               • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer)
6
 7
               • proc - a Python proc (dict)
               See PMIx tool connect to server for description of all relevant attributes and behaviors
8
    A.6.4 Tool.iof_pull
               Summary
10
               Register to receive output forwarded from a remote process.
11
               Format
12
                                            ---- Python -
   PMIx v4.0
               rc,id = mytool.iof_pull(sources:list, channel:integer, directives:list, cbfu
13
                                                      Python
               IN
14
                   sources
                   List of Python proc dictionaries of processes whose IO is being requested (list)
15
               IN channel
16
                   Python channel bitmask identifying IO channels to be forwarded (integer)
17
18
               IN directives
                   List of Python info dictionaries describing request (list)
19
               IN
20
                   cbfunc
                   Python iofcbfunc to receive IO payloads (func)
21
               Returns:
22
23
               • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer)
24
               • id - PMIx reference identifier for request (integer)
               See PMIx_IOF_pull for description of all relevant attributes and behaviors
25
              Tool.iof_deregister
    A.6.5
               Summary
27
28
               Deregister from output forwarded from a remote process.
```

```
Format
1
                                                      Python
   PMIx v4.0
2
               rc = mytool.iof_deregister(id:integer, directives:list)
                                                      Python
               IN
3
                    id
 4
                    PMIx reference identifier returned by pull request (list)
5
               IN
                   directives
                    List of Python info dictionaries describing request (list)
6
 7
               Returns:
               • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer)
9
               See PMIx IOF deregister for description of all relevant attributes and behaviors
              Tool.iof push
    A.6.6
               Summary
11
               Push data collected locally (typically from stdin) to stdin of target recipients
12
13
               Format
                                                      Python ———
   PMIx v4.0
               rc = mytool.iof_push(targets:list, data:dict, directives:list)
14
                                                      Python
               IN
                    sources
15
                    List of Python proc of target processes (list)
16
17
               IN
                    data
                    Python byteobject containing data to be delivered (dict)
18
               IN
                    directives
19
                    Optional list of Python info describing request (list)
20
               Returns:
21
22
               • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
23
               See PMIx_IOF_push for description of all relevant attributes and behaviors
```

A.7 Example Usage

The following examples are provided to illustrate the use of the Python bindings.

1 A.7.1 Python Client

2

3

5

The following example contains a client program that illustrates a fairly common usage pattern. The program instantiates and initializes the PMIxClient class, posts some data that is to be shared across all processes in the job, executes a "fence" that circulates the data, and then retrieves a value posted by one of its peers. Note that the example has been formatted to fit the document layout.

Python

```
from pmix import *
6
7
8
            def main():
9
                # Instantiate a client object
                myclient = PMIxClient()
10
                print("Testing PMIx ", myclient.get version())
11
12
                # Initialize the PMIx client library, declaring the programming model
13
14
                # as "TEST" and the library name as "PMIX", just for the example
                info = ['key':PMIX_PROGRAMMING_MODEL,
15
16
                          'value': 'TEST', 'val type': PMIX STRING,
                         'key': PMIX_MODEL_LIBRARY_NAME,
17
18
                          'value':'PMIX', 'val_type':PMIX_STRING]
19
                rc, myname = myclient.init(info)
                if PMIX SUCCESS != rc:
20
21
                    print("FAILED TO INIT WITH ERROR", myclient.error_string(rc))
22
                    exit(1)
23
                # try posting a value
24
25
                rc = myclient.put(PMIX_GLOBAL, "mykey",
                                   'value':1, 'val_type':PMIX_INT32)
26
27
                if PMIX SUCCESS != rc:
28
                    print("PMIx Put FAILED WITH ERROR", myclient.error string(rc))
29
                    # cleanly finalize
                    myclient.finalize()
30
31
                    exit(1)
32
                # commit it
33
                rc = myclient.commit()
34
                if PMIX_SUCCESS != rc:
35
                    print("PMIx_Commit FAILED WITH ERROR",
36
37
                           myclient.error_string(rc))
38
                    # cleanly finalize
39
                    myclient.finalize()
40
                    exit(1)
41
```

```
1
                # execute fence across all processes in my job
2
                procs = []
                info = []
3
4
                rc = myclient.fence(procs, info)
5
                if PMIX SUCCESS != rc:
6
                    print("PMIx_Fence FAILED WITH ERROR", myclient.error_string(rc))
7
                    # cleanly finalize
8
                    myclient.finalize()
9
                    exit(1)
10
11
                # Get a value from a peer
12
                if 0 != myname['rank']:
13
                    info = []
14
                    rc, get_val = myclient.get('nspace':"testnspace", 'rank': 0,
15
                                                 "mykey", info)
                    if PMIX_SUCCESS != rc:
16
17
                        print("PMIx_Commit FAILED WITH ERROR",
18
                               myclient.error_string(rc))
19
                        # cleanly finalize
20
                        myclient.finalize()
21
                        exit(1)
22
                    print("Get value returned: ", get val)
23
24
                # test a fence that should return not supported because
25
                # we pass a required attribute that the server is known
26
                # not to support
27
                procs = []
28
                info = ['key': 'ARBIT', 'flags': PMIX_INFO_REQD,
29
                          'value':10, 'val_type':PMIX_INT]
30
                rc = myclient.fence(procs, info)
31
                if PMIX_SUCCESS == rc:
32
                    print("PMIx_Fence SUCCEEDED BUT SHOULD HAVE FAILED")
33
                    # cleanly finalize
                    myclient.finalize()
34
35
                    exit(1)
36
37
                # Publish something
38
                info = ['key': 'ARBITRARY', 'value':10, 'val_type':PMIX_INT]
39
                rc = myclient.publish(info)
                if PMIX SUCCESS != rc:
40
41
                    print ("PMIx Publish FAILED WITH ERROR",
42
                          myclient.error string(rc))
43
                    # cleanly finalize
```

```
1
                    myclient.finalize()
2
                     exit(1)
3
4
                # finalize
5
                info = []
6
                myclient.finalize(info)
7
                print("Client finalize complete")
8
9
            # Python main program entry point
            if __name__ == '__main__':
10
11
                main()
                                            Python
```

12 A.7.2 Python Server

13

14

15

16 17 The following example contains a minimum-level server host program that instantiates and initializes the PMIxServer class. The program illustrates passing several server module functions to the bindings and includes code to setup and spawn a simple client application, waiting until the spawned client terminates before finalizing and exiting itself. Note that the example has been formatted to fit the document layout.

Python

```
from pmix import *
18
19
            import signal, time
20
            import os
            import select
21
22
            import subprocess
23
24
            def clientconnected(proc:tuple is not None):
25
                print("CLIENT CONNECTED", proc)
26
                return PMIX_OPERATION_SUCCEEDED
27
28
            def clientfinalized(proc:tuple is not None):
29
                print("CLIENT FINALIZED", proc)
30
                return PMIX_OPERATION_SUCCEEDED
31
32
            def clientfence(procs:list, directives:list, data:bytearray):
33
                # check directives
                if directives is not None:
34
                    for d in directives:
35
                         # these are each an info dict
36
37
                         if "pmix" not in d['key']:
38
                             # we do not support such directives - see if
```

```
1
                             # it is required
2
                             try:
                                 if d['flags'] & PMIX INFO REQD:
3
4
                                     # return an error
5
                                     return PMIX ERR NOT SUPPORTED
6
                             except:
7
                                 #it can be ignored
8
                                 pass
9
                return PMIX OPERATION SUCCEEDED
10
11
            def main():
12
                try:
13
                    myserver = PMIxServer()
14
                except:
15
                    print("FAILED TO CREATE SERVER")
16
                    exit(1)
17
                print("Testing server version ", myserver.get_version())
18
19
                args = ['key':PMIX_SERVER_SCHEDULER,
20
                          'value':'T', 'val_type':PMIX_BOOL]
21
                map = 'clientconnected': clientconnected,
22
                       'clientfinalized': clientfinalized,
23
                        'fencenb': clientfence
24
                my result = myserver.init(args, map)
25
26
                # get our environment as a base
27
                env = os.environ.copy()
28
29
                # register an nspace for the client app
30
                (rc, regex) = myserver.generate_regex("test000, test001, test002")
                (rc, ppn) = myserver.generate_ppn("0")
31
32
                kvals = ['key':PMIX_NODE_MAP,
33
                           'value':regex, 'val_type':PMIX_STRING,
                          'key':PMIX_PROC_MAP,
34
35
                           'value':ppn, 'val_type':PMIX_STRING,
36
                          'key':PMIX_UNIV_SIZE,
37
                           'value':1, 'val type':PMIX UINT32,
38
                          'key':PMIX JOB SIZE,
39
                           'value':1, 'val type':PMIX UINT32]
40
                rc = foo.register_nspace("testnspace", 1, kvals)
41
                print("RegNspace ", rc)
42
43
                # register a client
```

```
1
                uid = os.getuid()
2
                gid = os.getgid()
3
                rc = myserver.register_client('nspace':"testnspace", 'rank':0,
4
                                               uid, gid)
5
                print("RegClient ", rc)
6
                # setup the fork
7
                rc = myserver.setup fork('nspace':"testnspace", 'rank':0, env)
8
                print("SetupFrk", rc)
9
10
                # setup the client argv
                args = ["./client.py"]
11
12
                # open a subprocess with stdout and stderr
13
                # as distinct pipes so we can capture their
14
                # output as the process runs
                p = subprocess.Popen(args, env=env,
15
16
                    stdout=subprocess.PIPE, stderr=subprocess.PIPE)
17
                # define storage to catch the output
18
                stdout = []
19
                stderr = []
20
                # loop until the pipes close
                while True:
21
22
                    reads = [p.stdout.fileno(), p.stderr.fileno()]
23
                    ret = select.select(reads, [], [])
24
25
                    stdout done = True
26
                    stderr_done = True
27
28
                    for fd in ret[0]:
29
                         # if the data
                         if fd == p.stdout.fileno():
30
31
                             read = p.stdout.readline()
32
                             if read:
33
                                 read = read.decode('utf-8').rstrip()
                                 print('stdout: ' + read)
34
35
                                 stdout done = False
                         elif fd == p.stderr.fileno():
36
37
                             read = p.stderr.readline()
38
                             if read:
39
                                 read = read.decode('utf-8').rstrip()
40
                                 print('stderr: ' + read)
41
                                 stderr done = False
42
43
                    if stdout done and stderr done:
```

APPENDIX B

Acknowledgements

1 2 3 4		This document represents the work of many people who have contributed to the PMIx community. Without the hard work and dedication of these people this document would not have been possible. The sections below list some of the active participants and organizations in the various PMIx standard iterations.
5	B.1	Version 3.0
6		The following list includes some of the active participants in the PMIx v3 standardization process.
7		Ralph H. Castain, Andrew Friedley, Brandon Yates
8		• Joshua Hursey
9		Aurelien Bouteiller and George Bosilca
10		Dirk Schubert
11		Kevin Harms
12 13		The following institutions supported this effort through time and travel support for the people listed above.
14		• Intel Corporation
15		• IBM, Inc.
16		• University of Tennessee, Knoxville
17		• The Exascale Computing Project, an initiative of the US Department of Energy
18		National Science Foundation
19		Argonne National Laboratory
20		• Allinea (ARM)

₁ B.2 Version 2.0

2	The following list includes some of the active participants in the PMIx v2 standardization process.
3 4	 Ralph H. Castain, Annapurna Dasari, Christopher A. Holguin, Andrew Friedley, Michael Klemm and Terry Wilmarth
5	• Joshua Hursey, David Solt, Alexander Eichenberger, Geoff Paulsen, and Sameh Sharkawi
6	Aurelien Bouteiller and George Bosilca
7	Artem Polyakov, Igor Ivanov and Boris Karasev
8	Gilles Gouaillardet
9	Michael A Raymond and Jim Stoffel
10	• Dirk Schubert
11	Moe Jette
12	Takahiro Kawashima and Shinji Sumimoto
13	Howard Pritchard
14	• David Beer
15	Brice Goglin
16	Geoffroy Vallee, Swen Boehm, Thomas Naughton and David Bernholdt
17	Adam Moody and Martin Schulz
18	Ryan Grant and Stephen Olivier
19	Michael Karo
20 21	The following institutions supported this effort through time and travel support for the people listed above.
22	• Intel Corporation
23	• IBM, Inc.
24	University of Tennessee, Knoxville
25	• The Exascale Computing Project, an initiative of the US Department of Energy
26	National Science Foundation
27	• Mellanox, Inc.
28	Research Organization for Information Science and Technology
29	• HPE Co.

- 1 Allinea (ARM)
- SchedMD, Inc.
- Fujitsu Limited
- Los Alamos National Laboratory
- Adaptive Solutions, Inc.
- 6 INRIA
- Oak Ridge National Laboratory
- Lawrence Livermore National Laboratory
- Sandia National Laboratory
- 10 Altair

15 16

B.3 Version 1.0

- The following list includes some of the active participants in the PMIx v1 standardization process.
- Ralph H. Castain, Annapurna Dasari and Christopher A. Holguin
- Joshua Hursey and David Solt
 - Aurelien Bouteiller and George Bosilca
 - Artem Polyakov, Elena Shipunova, Igor Ivanov, and Joshua Ladd
- Gilles Gouaillardet
- Gary Brown
- Moe Jette
- The following institutions supported this effort through time and travel support for the people listed above.
- Intel Corporation
- IBM, Inc.
- University of Tennessee, Knoxville
- Mellanox, Inc.
- Research Organization for Information Science and Technology
- Adaptive Solutions, Inc.
- SchedMD, Inc.

Bibliography

[1] Ralph H. Castain, David Solt, Joshua Hursey, and Aurelien Bouteiller. PMIx: Process management for exascale environments. In *Proceedings of the 24th European MPI Users' Group Meeting*, EuroMPI '17, pages 14:1–14:10, New York, NY, USA, 2017. ACM.

Index

General terms and other items not induced in the other indices.

```
application, 10, 12, 15, 81, 82, 137, 194, 259, 263
fabric, 16
fabric plane, <u>16</u>, 350
fabric planes, 16, 339
fabrics, 16
host environment, \underline{16}
job, 10, 12, <u>15</u>, 81–83, 137, 139, 194, 252, 258, 259, 261, 263, 271, 273, 462
namespace, <u>15</u>
rank, <u>15</u>, 139, 263
resource manager, 16
scheduler, <u>16</u>, 348
session, 10, 12, <u>15</u>, 81, 82, 137, 194, 258
slot, <u>15</u>
slots, <u>15</u>
workflow, <u>15</u>
workflows, <u>15</u>, 114
```

Index of APIs

```
PMIx_Abort, 8, 24, 35, 157, 158, 284, 285, 422, 436
    PMIxClient.abort (Python), 435
PMIx Alloc directive string, 9, 119, 455
    PMIxClient.alloc directive string (Python), 455
PMIx Allocation request, 11–13, 95, 195, 196, 201
PMIx_Allocation_request_nb, 9, 95, 183, 199, 202, 444
    PMIxClient.allocate (Python), 443
PMIx Commit, 8, 112, 129, 130, 141, 141, 270, 290, 437
    PMIxClient.commit (Python), 437
PMIx_Connect, 8, 9, 24, 162, 168, 170, 172, 174, 360–362, 441
    PMIxClient.connect (Python), 440
PMIx_Connect_nb, 8, 170, 170
pmix_connection_cbfunc_t, 112, 311
pmix_credential_cbfunc_t, 113, 242, 326
PMIx_Data_copy, 9, 236
PMIx_Data_copy_payload, 9, 237
PMIx_Data_pack, 9, 232, 233, 249, 250
PMIx_Data_print, 9, 236
PMIx Data range string, 9, 118, 454
    PMIxClient.data range string (Python), 454
PMIx_Data_type_string, 9, <u>118</u>, 455
    PMIxClient.data_type_string (Python), 454
PMIx Data unpack, 9, 234
PMIx Deregister event handler, 9, 13, 224, 449
    PMIxClient.deregister event handler (Python), 449
PMIx Disconnect, 8, 9, 24, 170, 172, 174, 176, 362, 441
    PMIxClient.disconnect (Python), 441
PMIx_Disconnect_nb, 8, 174, 176, 362
pmix_dmodex_response_fn_t, 111, 270
PMIx_Error_string, 8, 117, 452
    PMIxClient.error_string (Python), 452
pmix_event_notification_cbfunc_fn_t, 107, 107, 109
pmix_evhdlr_reg_cbfunc_t, 107, 107, 222
PMIx_Fabric_deregister, 352, 353, 451
    PMIxClient.fabric deregister (Python), 451
PMIx Fabric deregister nb, 353
PMIx_Fabric_get_device_index, 356, 358, 452
    PMIxClient.fabric get device index (Python), 452
```

```
PMIx_Fabric_get_device_index_nb, 357
PMIx Fabric get vertex info, 353, 356, 451
    PMIxClient.fabric get vertex info (Python), 451
PMIx Fabric get vertex info nb, 356
PMIx_Fabric_register, 342, 349, 351, 450
    PMIxClient.fabric_register (Python), 450
PMIx Fabric register nb, 350
PMIx Fabric update, 350, 351, 352, 451
    PMIxClient.fabric update (Python), 450
PMIx_Fabric_update_nb, 351
PMIx_Fence, 3, 6, 8, 14, 128, 129, 142, 143, 145, 170, 174, 270, 285, 288, 360, 366, 371, 422, 438
    PMIxClient.fence (Python), 437
PMIx_Fence_nb, 8, 12, 105, 143, 145, 285, 288, 422
PMIx_Finalize, 8, 24, 27, 35, 86, 124, 125, 125, 167, 283, 284, 421, 435
    PMIxClient.finalize (Python), 435
PMIx_generate_ppn, 8, 249, 459
    PMIxServer.generate_ppn (Python), 459
PMIx_generate_regex, 8, 248, 250, 258, 459
    PMIxServer.generate_regex (Python), 458
PMIx_Get, 3, 8–10, 26, 42, 75–78, 83, 85–91, 93–95, 97, 98, 124, 130, 131, 133, 134, 136–138,
         141, 160–162, 164–167, 189, 194, 195, 251, 253, 279, 298–300, 339, 346, 347, 359, 366,
         392, 397, 438
    PMIxClient.get (Python), 438
PMIx Get attribute name, 120, 457
    PMIxClient.get_attribute_name (Python), 456
PMIx Get attribute string, 119, 456
    PMIxClient.get_attribute_string (Python), 456
PMIx_Get_credential, 11, 13, 99, 240, 327, 445
    PMIxClient.get_credential (Python), 445
PMIx_Get_credential_nb, 241
PMIx_Get_nb, 8, 18, 105, 106, 133
PMIx_Get_version, 8, 18, 122, 435
    PMIxClient.get_version (Python), 435
PMIx_Group_construct, 361, <u>363</u>, 365, 366, 369, 446
    PMIxClient.group_construct (Python), 446
PMIx_Group_construct_nb, 366, 369
PMIx Group destruct, 362, 369, 371, 373, 448
    PMIxClient.group destruct (Python), 448
PMIx Group destruct nb, 371, 373
PMIx_Group_invite, 361, 373, 376, 377, 379, 447
    PMIxClient.group invite (Python), 446
PMIx Group invite nb, 377
PMIx Group join, 65, 361, 376, 379, 379, 381, 382, 384, 447
```

```
PMIxClient.group_join (Python), 447
PMIx Group join nb, 379, 382, 384
PMIx Group leave, 362, 384, 385, 386, 448
    PMIxClient.group leave (Python), 448
PMIx Group leave nb, 385
pmix_hdlr_reg_cbfunc_t, 107, 116, 177, 179, 408, 410
pmix info cbfunc t, 102, 106, 106, 190, 199, 205, 208, 211, 212, 278, 312, 318, 320, 323, 334,
         336, 356, 358, 367, 377, 383
PMIx Info directives string, 9, 118, 454
    PMIxClient.info_directives_string (Python), 454
PMIx_Init, 9, 90, 122, 122, 124, 125, 161, 165, 282, 299, 397, 434
    PMIxClient.init (Python), 434
PMIx Initialized, 8, 121, 434
    PMIxClient.initialized (Python), 434
pmix_iof_cbfunc_t, 115, 177, 408, 420
PMIx_IOF_channel_string, 11, 119, 455
    PMIxClient.iof_channel_string (Python), 455
PMIx IOF deregister, 11, 13, 179, 409, 468
    PMIxTool.iof deregister (Python), 467
PMIx IOF pull, 11, 13, 177, 179, 407, 410, 467
    PMIxTool.iof pull (Python), 467
PMIx IOF push, 11, 13, 99, 100, 180, 400, 402, 410, 468
    PMIxTool.iof push (Python), 468
PMIx Job control, 11, 13, 97, 202, 204, 207, 208, 322
PMIx_Job_control_nb, 9, 97, 183, 202, 205, 258, 444
    PMIxClient.job ctrl (Python), 444
PMIx_Job_state_string, 119, 456
    PMIxClient.job_state_string (Python), 456
PMIx_Link_state_string, 120, 457
    PMIxClient.link_state_string (Python), 457
PMIx_Log, 11, 213, 215, 443
    PMIxClient.log (Python), 443
PMIx_Log_nb, 9, 93, 215, 218
PMIx_Lookup, 8, 54, 146, 149, 152, 153, 438, 439
    PMIxClient.lookup (Python), 439
pmix_lookup_cbfunc_t, 105, 105, 293
PMIx Lookup nb, 105, 152
pmix modex cbfunc t, 102, 103, 103, 286, 289
pmix notification fn t, 108, 108, 222, 421
PMIx_Notify_event, 9, 13, 225, 450
    PMIxClient.notify event (Python), 449
pmix op cbfunc t, 104, 104, 108, 111, 148, 155, 171, 175, 181, 216, 225, 226, 251, 266–268, 275,
         277, 279, 282, 283, 285, 290, 295, 301, 304, 306, 308, 309, 316, 321, 324, 330, 332,
```

```
351–353, 372, 385, 411
PMIx Persistence string, 9, 118, 453
    PMIxClient.persistence string (Python), 453
PMIx Proc state string, 9, 117, 453
    PMIxClient.proc state string (Python), 452
PMIx_Process_monitor, 11, 13, 208, 212
PMIx Process monitor nb, 9, 98, 183, 210, 212, 445
    PMIxClient.monitor (Python), 444
PMIx Publish, 8, 39, 40, 86, 146, 147–149, 291, 292, 439
    PMIxClient.publish (Python), 438
PMIx_Publish_nb, 8, 148, 149
PMIx_Put, 8, 38, 39, 42, 112, 129, 129, 130, 133, 141–143, 167, 194, 270, 290, 366, 376, 437
    PMIxClient.put (Python), 436
PMIx_Query_info, 185, 189, 194, 195, 346
PMIx_Query_info_nb, 9, 10, 60, 83, 91, 141, 167, 183, 189, 190, 195, 274, 339, 359, 360, 443
    PMIxClient.query (Python), 442
PMIx_Register_attributes, 13, 273, 463
    PMIxServer.register_attributes (Python), 463
PMIx Register event handler, 9, 13, 107, 183, 221, 449
    PMIxClient.register event handler (Python), 448
pmix_release_cbfunc_t, 102, 102
PMIx Resolve nodes, 8, 184, 442
    PMIxClient.resolve nodes (Python), 442
PMIx Resolve peers, 8, 183, 442
    PMIxClient.resolve_peers (Python), 441
PMIx Scope string, 9, 118, 453
    PMIxClient.scope_string (Python), 453
pmix_server_abort_fn_t, 284, 422
pmix_server_alloc_fn_t, 317, 429
pmix_server_client_connected_fn_t, 105, 239, 268, 281, 282, 421
pmix_server_client_finalized_fn_t, 283, 284, 421
PMIx_server_collect_inventory, 11, 277, 465
    PMIxServer.collect_inventory (Python), 464
pmix_server_connect_fn_t, 167, 301, 303, 305, 425
PMIx_server_deliver_inventory, 11, 278, 465
    PMIxServer.deliver_inventory (Python), 465
PMIx server deregister client, 8, 268, 461
    PMIxServer.deregister_client (Python), 461
pmix server deregister events fn t, 307, 427
PMIx_server_deregister_nspace, 8, 266, 269, 460
    PMIxServer.deregister_nspace (Python), 460
pmix server disconnect fn t, 303, 305, 426
pmix server dmodex req fn t, 10, 11, 103, 289, 423
```

```
PMIx_server_dmodex_request, 8, 111, 112, 269, 270, 462
    PMIxServer.dmodex request (Python), 462
pmix server fabric fn t, 335, 342, 348, 433
pmix server fencenb fn t, 12, 103, 285, 288, 423
PMIx server finalize, 8, 128, 458
    PMIxServer.finalize (Python), 458
pmix server get cred fn t, 326, 329, 431
pmix server grp fn t, 333, 433
PMIx server init, 8, 122, 125, 274, 280, 346, 388, 389, 458
    PMIxServer.init (Python), 457
PMIx_server_IOF_deliver, 11, 176, 276, 398, 464
    PMIxServer.iof_deliver (Python), 464
pmix_server_iof_fn_t, 329, 432
pmix_server_job_control_fn_t, 320, 430
pmix_server_listener_fn_t, 310
pmix_server_log_fn_t, 315, 429
pmix_server_lookup_fn_t, 292, 424
pmix_server_module_t, 126, 128, 274, 275, 280, 280
pmix_server_monitor_fn_t, 323, 430
pmix_server_notify_event_fn_t, 110, 309, 427
pmix_server_publish_fn_t, 290, 423
pmix server query fn t, 311, 428
PMIx server register client, 8, 239, 267, 268, 282, 284, 461
    PMIxServer.register client (Python), 460
pmix server register events fn t, 305, 426
PMIx_server_register_nspace, 8, 10, 18, 82, 105, 133, 249, 250, 258, 261, 460
    PMIxServer.register_nspace (Python), 459
PMIx_server_setup_application, 9, 12, 110, 111, 271, 276, 279, 463
    PMIxServer.setup_application (Python), 462
PMIx_server_setup_fork, 8, 269, 462
    PMIxServer.setup_fork (Python), 461
PMIx_server_setup_local_support, 9, 275, 464
    PMIxServer.setup_local_support (Python), 463
pmix_server_spawn_fn_t, 104, 296, 393, 425
pmix_server_stdin_fn_t, 332, 432
pmix_server_tool_connection_fn_t, 239, 314, 388, 428
pmix server unpublish fn t, 294, 424
pmix server validate cred fn t, 327, 431
pmix setup application cbfunc t, 110, 271
PMIx_Spawn, 8, 12, 57, 79, 89, 94, 95, <u>158</u>, 158, 159, 163, 164, 167, 199, 257, 258, 269, 296, 301,
         319, 393, 397, 399, 425, 440
    PMIxClient.spawn (Python), 440
pmix spawn cbfunc t, 104, 104, 163, 297
```

```
PMIx_Spawn_nb, 8, 57, 104, 163
PMIx Store internal, 8, 136, 436
    PMIxClient.store_internal (Python), 436
PMIx_tool_connect_to_server, 11, 392, 393, 406, 467
    PMIxTool.connect_to_server (Python), 466
pmix_tool_connection_cbfunc_t, 112, 314
PMIx tool disconnect, 405
PMIx_tool_finalize, 9, <u>405</u>, 466
    PMIxTool.finalize (Python), 466
PMIx_tool_init, 9, 122, 390, 392, 399, 402, 405, 466
    PMIxTool.init (Python), 465
PMIx_Unpublish, 8, 153, 155, 156, 440
    PMIxClient.unpublish (Python), 439
PMIx_Unpublish_nb, 8, 155
PMIx_Validate_credential, 11, 13, 243, 446
    PMIxClient.validate_credential (Python), 445
PMIx_Validate_credential_nb, 245
pmix_validation_cbfunc_t, 114, 246, 328
pmix_value_cbfunc_t, 18, 105, 105
```

Index of Support Macros

```
PMIX APP CONSTRUCT, 58
PMIX APP CREATE, 58
PMIX APP DESTRUCT, 58
PMIX APP FREE, 59
PMIX APP INFO CREATE, 10, 11, 59
PMIX APP RELEASE, 58
PMIX ARGV APPEND, 68
PMIX ARGV APPEND UNIQUE, 70
PMIX ARGV COPY, 72
PMIX_ARGV_COUNT, 72
PMIX_ARGV_FREE, 70
PMIX_ARGV_JOIN, 71
PMIX_ARGV_PREPEND, 69
PMIX_ARGV_SPLIT, 71
PMIX_BYTE_OBJECT_CONSTRUCT, 65
PMIX_BYTE_OBJECT_CREATE, 66
PMIX BYTE OBJECT DESTRUCT, 65
PMIX_BYTE_OBJECT_FREE, 66
PMIX BYTE OBJECT LOAD, 66
PMIX CHECK KEY, 29
PMIX CHECK NSPACE, 30
PMIX CHECK PROCID, 33
PMIX COORD CONSTRUCT, 340
PMIX COORD CREATE, 340
PMIX COORD DESTRUCT, 340
PMIX COORD FREE, 340
PMIX_DATA_ARRAY_CONSTRUCT, 40, 67
PMIX_DATA_ARRAY_CREATE, 41, 68
PMIX_DATA_ARRAY_DESTRUCT, 41, 67
PMIX DATA ARRAY FREE, 41, 68
PMIX_DATA_BUFFER_CONSTRUCT, 230, 233, 235
PMIX_DATA_BUFFER_CREATE, 230, 233, 235
PMIX_DATA_BUFFER_DESTRUCT, 231
PMIX DATA BUFFER LOAD, 231
PMIX DATA BUFFER RELEASE, 230
PMIX DATA BUFFER UNLOAD, 232, 249, 250
PMIX ENVAR CONSTRUCT, 53
PMIX ENVAR CREATE, 53
```

```
PMIX_ENVAR_DESTRUCT, 53
PMIX ENVAR FREE, 54
PMIX ENVAR LOAD, 54
PMIX FABRIC CONSTRUCT, 345
PMIx Heartbeat, 9, 212
PMIX_INFO_CONSTRUCT, 47
PMIX INFO CREATE, 47, 50, 51
PMIX INFO DESTRUCT, 47
PMIX INFO FREE, 47
PMIX_INFO_IS_END, 10, 12, 51
PMIX_INFO_IS_OPTIONAL, 51
PMIX_INFO_IS_REQUIRED, 49, 50, 51
PMIX INFO LOAD, 48
PMIX_INFO_OPTIONAL, 50
PMIX_INFO_REQUIRED, 49, 50
PMIX_INFO_TRUE, 49
PMIX_INFO_XFER, 48, 258
PMIX LOAD KEY, 29
PMIX LOAD NSPACE, 30
PMIX LOAD PROCID, 33, 34
PMIX_MULTICLUSTER_NSPACE_CONSTRUCT, 34
PMIX MULTICLUSTER NSPACE PARSE, 34
PMIX_PDATA_CONSTRUCT, <u>55</u>
PMIX PDATA CREATE, 55
PMIX_PDATA_DESTRUCT, 55
PMIX PDATA FREE, 56
PMIX_PDATA_LOAD, 56
PMIX PDATA RELEASE, 55
PMIX_PDATA_XFER, 57
PMIX_PROC_CONSTRUCT, 32
PMIX_PROC_CREATE, 32
PMIX_PROC_DESTRUCT, 32
PMIX_PROC_FREE, 33, 184
PMIX_PROC_INFO_CONSTRUCT, 37
PMIX_PROC_INFO_CREATE, 37
PMIX_PROC_INFO_DESTRUCT, 37
PMIX PROC INFO FREE, 38
PMIX PROC INFO RELEASE, 37
PMIX PROC LOAD, 33
PMIX_PROC_RELEASE, 32
PMIX QUERY CONSTRUCT, 60
PMIX QUERY CREATE, 60
PMIX QUERY DESTRUCT, 60
```

```
PMIX_QUERY_FREE, 61
PMIX_QUERY_QUALIFIERS_CREATE, 10, 11, 61
PMIX_QUERY_RELEASE, 61
PMIX_REGATTR_CONSTRUCT, 63
PMIX_REGATTR_CREATE, 63
PMIX_REGATTR_DESTRUCT, 63
PMIX REGATTR FREE, 64
PMIX_REGATTR_LOAD, 64
PMIX REGATTR XFER, 64
PMIX_SETENV, 72
PMIX_SYSTEM_EVENT, 27
PMIX_VALUE_CONSTRUCT, 43
PMIX_VALUE_CREATE, 43
PMIX_VALUE_DESTRUCT, 43
PMIX_VALUE_FREE, 44
PMIX_VALUE_GET_NUMBER, 46
PMIX_VALUE_LOAD, 44
PMIX_VALUE_RELEASE, 44
PMIX_VALUE_UNLOAD, 45
PMIX_VALUE_XFER, 45
```

Index of Data Structures

```
pmix alloc directive t, 52, 52, 75, 119, 318, 417, 455
pmix app t, 10, 11, 57, 57–59, 69, 70, 73, 159, 163, 297, 393, 418
pmix byte object t, 65, 65, 66, 74, 114, 181, 240, 244, 246, 277, 328, 332, 411, 416
pmix coord t, 75, 339, 339, 340
pmix coord view t, 341, 346
pmix_data_array_t, 10, 11, 40, 40-42, 67, 67, 68, 74, 92, 96, 188, 193, 195, 198, 201, 251-254,
         256, 261, 263, 264, 272, 313, 319, 336, 339, 344, 346, 347, 349, 417
pmix data buffer t, 229, 229-234, 238
pmix data range t, 39, 39, 74, 118, 226, 309, 416, 454
pmix_data_type_t, 40, 41, 44, 46, 48, 56, 64, 67, 68, 73, 73, 74, 118, 233, 235–237, 415, 454
pmix_envar_t, 52, 53, 54, 75, 417
pmix_fabric_operation_t, 336, 342, 342
pmix_fabric_t, 337, 338, 342, 342, 345, 346, 349–354, 356–358
pmix_group_operation_t, 333, 362, 362, 380, 383
pmix_group_opt_t, 65, 65, 447
pmix_info_directives_t, 49, 49, 50, 74, 118, 417, 454
pmix_info_t, 3, 9, 10, 12, 14, 29, 39, 46, 46–52, 59, 61, 62, 64, 74, 82, 93–97, 106, 108, 109, 115,
          116, 123–126, 128, 147, 151, 177, 179, 181, 185, 189, 195–199, 201, 203–205, 207–209,
         212, 215, 218, 226, 240, 242, 244, 246, 251, 252, 254, 258, 261, 263, 264, 272, 277–279,
         309, 314, 315, 317, 319, 320, 323, 324, 330, 337, 343, 344, 347, 354, 357, 358, 363, 365,
         367, 370, 372, 374, 377, 380, 383–385, 393, 402, 408, 410, 411, 417, 419
pmix iof channel t, 52, 52, 75, 116, 119, 177, 277, 330, 408, 417, 455
pmix job state t, 36, 36, 38, 38, 75, 119, 456
pmix key t, 28, 28, 64, 129, 131, 415
pmix link state t, 75, 120, 341, 341, 345, 348, 355, 418, 457
pmix nspace t, 29, 29, 30, 33, 34, 104, 415, 416
pmix_pdata_t, 54, 54-57, 105, 151, 417
pmix_persistence_t, 40, 40, 74, 118, 416, 453
pmix_proc_info_t, 36, 36-38, 74, 92, 188, 193, 313, 416
pmix proc state t, 35, 35, 74, 117, 416, 452
pmix_proc_t, 31, 31–34, 56, 64, 74, 81, 88, 109, 113, 116, 124, 133, 142–144, 157, 223, 226, 227,
         233, 234, 255, 267–270, 277, 282, 283, 285, 286, 289, 290, 293, 295, 297, 301, 304, 309,
         311, 316, 318, 321, 324, 326, 328, 330, 332, 333, 336, 363, 367, 374, 377, 381, 402, 404,
         407, 416
pmix query t, 10, 11, 60, 60, 61, 75, 187, 188, 192, 193, 195, 311, 314, 418
pmix rank t, 31, 31, 33, 34, 74, 416
pmix_regattr_t, 13, <u>62</u>, 62–64, 75, 100, 195, 274, 418
pmix scope t, 38, 38, 74, 118, 130, 416, 453
```

pmix_status_t, <u>22</u>, 22, 28, 45, 46, 68–70, 73, 74, 106–109, 111–115, 117, 221, 226, 306, 308, 309, 415, 427, 452
pmix_value_t, <u>42</u>, 42–46, 74, 106, 129, 130, 417

Index of Constants

```
PMIX ALLOC DIRECTIVE, 75
PMIX ALLOC EXTEND, 52
PMIX ALLOC EXTERNAL, 52
PMIX ALLOC NEW, 52
PMIX ALLOC REAQUIRE, 52
PMIX_ALLOC_RELEASE, 52
PMIX APP, 74
PMIX_APP_WILDCARD, 21
PMIX BOOL, 74
PMIX_BUFFER, 74
PMIX_BYTE, 74
PMIX_BYTE_OBJECT, 74
PMIX_COMMAND, 74
PMIX_COMPRESSED_STRING, 75
PMIX_CONNECT_REQUESTED, 25
PMIX_COORD, 75
PMIX COORD_LOGICAL_VIEW, 341
PMIX_COORD_PHYSICAL_VIEW, 341
PMIX COORD VIEW UNDEF, 341
PMIX DATA ARRAY, 74
PMIX DATA RANGE, 74
PMIX_DATA_TYPE, 74
PMIX DATA TYPE MAX, 75
PMIX DOUBLE, 74
PMIX ENVAR, 75
PMIX ERR BAD PARAM, 23
PMIX_ERR_COMM_FAILURE, 23
PMIX_ERR_CONFLICTING_CLEANUP_DIRECTIVES, 23
PMIX_ERR_DATA_VALUE_NOT_FOUND, 23
PMIX ERR DEBUGGER RELEASE, 22
PMIX_ERR_DUPLICATE_KEY, 25
PMIX_ERR_EVENT_REGISTRATION, 24
PMIX_ERR_GET_MALLOC_REQD, 26
PMIX ERR HANDSHAKE FAILED, 22
PMIX ERR IN ERRNO, 23
PMIX ERR INIT, 23
PMIX ERR INVALID ARG, 23
PMIX ERR INVALID ARGS, 23
```

```
PMIX_ERR_INVALID_CRED, 22
PMIX ERR INVALID KEY, 23
PMIX ERR INVALID KEY LENGTH, 23
PMIX ERR INVALID KEYVALP, 23
PMIX ERR INVALID LENGTH, 23
PMIX_ERR_INVALID_NAMESPACE, 23
PMIX ERR INVALID NUM ARGS, 23
PMIX ERR INVALID NUM PARSED, 23
PMIX ERR INVALID OPERATION, 25
PMIX_ERR_INVALID_SIZE, 23
PMIX_ERR_INVALID_TERMINATION, 24
PMIX_ERR_INVALID_VAL, 23
PMIX ERR INVALID VAL LENGTH, 23
PMIX_ERR_IOF_COMPLETE, 26
PMIX_ERR_IOF_FAILURE, 26
PMIX_ERR_JOB_ABORTED, 24, 26
PMIX_ERR_JOB_ABORTED_BY_SIG, 24, 26
PMIX ERR JOB ABORTED BY SYS EVENT, 27
PMIX_ERR_JOB_ALLOC_FAILED, 24, 27
PMIX_ERR_JOB_APP_NOT_EXECUTABLE, 24, 26
PMIX_ERR_JOB_CANCELED, 26
PMIX ERR JOB CANCELLED, 24
PMIX_ERR_JOB_CANNOT_LAUNCH, 24
PMIX ERR JOB FAILED TO LAUNCH, 24, 26
PMIX_ERR_JOB_FAILED_TO_MAP, 24, 26
PMIX ERR JOB FAILED TO START, 24
PMIX_ERR_JOB_KILLED_BY_CMD, 24, 26
PMIX ERR JOB NEVER LAUNCHED, 24
PMIX_ERR_JOB_NO_EXE_SPECIFIED, 24, 26
PMIX_ERR_JOB_NON_ZERO_TERM, 24, 27
PMIX_ERR_JOB_SENSOR_BOUND_EXCEEDED, 24, 27
PMIX_ERR_JOB_TERM_WO_SYNC, 24, 27
PMIX ERR JOB TERMINATED, 24
PMIX_ERR_LOST_CONNECTION_TO_CLIENT, 23
PMIX_ERR_LOST_CONNECTION_TO_SERVER, 23
PMIX_ERR_LOST_PEER_CONNECTION, 23
PMIX ERR NO PERMISSIONS, 23
PMIX ERR NODE DOWN, 27
PMIX ERR NODE OFFLINE, 27
PMIX ERR NOMEM, 23
PMIX ERR NOT FOUND, 23
PMIX ERR NOT IMPLEMENTED, 23
PMIX ERR NOT SUPPORTED, 23
```

```
PMIX_ERR_OUT_OF_RESOURCE, 23
PMIX ERR PACK FAILURE, 23
PMIX ERR PACK MISMATCH, 23
PMIX ERR PARTIAL SUCCESS, 25
PMIX_ERR_PROC_ABORTED, 22
PMIX_ERR_PROC_ABORTING, 22
PMIX ERR PROC CHECKPOINT, 22
PMIX ERR PROC ENTRY NOT FOUND, 22
PMIX ERR PROC MIGRATE, 22
PMIX_ERR_PROC_REQUESTED_ABORT, 22
PMIX_ERR_PROC_RESTART, 22
PMIX_ERR_READY_FOR_HANDSHAKE, 22
PMIX ERR REPEAT ATTR REGISTRATION, 26
PMIX_ERR_RESOURCE_BUSY, 23
PMIX_ERR_SERVER_FAILED_REQUEST, 22
PMIX_ERR_SERVER_NOT_AVAIL, 23
PMIX_ERR_SILENT, 22
PMIX ERR SYS BASE, 27
PMIX ERR SYS OTHER, 27
PMIX ERR TIMEOUT, 23
PMIX ERR TYPE MISMATCH, 22
PMIX ERR UNKNOWN DATA TYPE, 22
PMIX ERR UNPACK FAILURE, 23
PMIX ERR UNPACK INADEQUATE SPACE, 22
PMIX_ERR_UNPACK_READ_PAST_END_OF_BUFFER, 23
PMIX ERR UNREACH, 23
PMIX_ERR_UPDATE_ENDPOINTS, 24
PMIX_ERR_WOULD_BLOCK, 22
PMIX_ERROR, 22
PMIX_EVENT_ACTION_COMPLETE, 27
PMIX_EVENT_ACTION_DEFERRED, 27
PMIX_EVENT_JOB_END, 26
PMIX EVENT JOB START, 26
PMIX_EVENT_NO_ACTION_TAKEN, 27
PMIX_EVENT_PARTIAL_ACTION_TAKEN, 27
PMIX_EVENT_SESSION_END, 26
PMIX EVENT SESSION START, 26
PMIX EXISTS, 22
PMIX EXTERNAL ERR BASE, 27
PMIX_FABRIC_GET_DEVICE_INDEX, 342
PMIX FABRIC GET VERTEX INFO, 342
PMIX FABRIC REQUEST INFO, 342
PMIX FABRIC UPDATE INFO, 342
```

```
PMIX_FABRIC_UPDATE_PENDING, 338
PMIX FABRIC UPDATED, 338
PMIX FLOAT, 74
PMIX FWD ALL CHANNELS, 52
PMIX FWD NO CHANNELS, 52
PMIX_FWD_STDDIAG_CHANNEL, <u>52</u>
PMIX FWD STDERR CHANNEL, 52
PMIX FWD STDIN CHANNEL, 52
PMIX FWD STDOUT CHANNEL, 52
PMIX_GDS_ACTION_COMPLETE, 24
PMIX_GLOBAL, 38
PMIX_GROUP_ACCEPT, 65, 362
PMIX GROUP CONSTRUCT, 362
PMIX_GROUP_CONSTRUCT_ABORT, 25
PMIX_GROUP_CONSTRUCT_COMPLETE, 26
PMIX_GROUP_CONTEXT_ID_ASSIGNED, 26
PMIX_GROUP_DECLINE, 65, 362
PMIX GROUP DESTRUCT, 362
PMIX_GROUP_INVITE_ACCEPTED, 25
PMIX GROUP INVITE DECLINED, 25
PMIX GROUP INVITE FAILED, 25
PMIX GROUP INVITED, 25
PMIX GROUP LEADER FAILED, 26
PMIX GROUP LEADER SELECTED, 26
PMIX GROUP LEFT, 25
PMIX GROUP MEMBER FAILED, 25
PMIX_GROUP_MEMBERSHIP_UPDATE, 25
PMIX INFO, 74
PMIX_INFO_ARRAY_END, 50
PMIX INFO DIR RESERVED, 50
PMIX_INFO_DIRECTIVES, 74
PMIX_INFO_REQD, 50
PMIX INT, 74
PMIX_INT16, 74
PMIX_INT32, 74
PMIX_INT64, 74
PMIX INT8, 74
PMIX INTERNAL, 38
PMIX IOF CHANNEL, 75
PMIX JCTRL CHECKPOINT, 23
PMIX JCTRL CHECKPOINT COMPLETE, 23
PMIX JCTRL PREEMPT ALERT, 24
PMIX JOB STATE, 75
```

```
PMIX_JOB_STATE_CONNECTED, 36, 39
PMIX JOB STATE LAUNCH UNDERWAY, 36, 39
PMIX JOB STATE PREPPED, 36, 39
PMIX JOB STATE RUNNING, 36, 39
PMIX JOB STATE SUSPENDED, 36, 39
PMIX_JOB_STATE_TERMINATED, 36, 39
PMIX JOB STATE TERMINATED WITH ERROR, 36, 39
PMIX JOB STATE UNDEF, 36, 39
PMIX JOB STATE UNTERMINATED, 39
PMIX_KVAL, 74
PMIX_LAUNCH_COMPLETE, 396
PMIX_LAUNCH_DIRECTIVE, 396
PMIX LAUNCHER READY, 396
PMIX_LINK_DOWN, 341
PMIX_LINK_STATE, 75
PMIX_LINK_STATE_UNKNOWN, 341
PMIX_LINK_UP, 341
PMIX LOCAL, 38
PMIX_MAX_KEYLEN, 21
PMIX_MAX_NSLEN, 21
PMIX MODEL DECLARED, 24
PMIX MODEL RESOURCES, 25
PMIX MONITOR FILE ALERT, 24
PMIX MONITOR HEARTBEAT ALERT, 24
PMIX_NOTIFY_ALLOC_COMPLETE, 23
PMIX OPENMP PARALLEL ENTERED, 25
PMIX_OPENMP_PARALLEL_EXITED, 25
PMIX_OPERATION_IN_PROGRESS, 25
PMIX_OPERATION_SUCCEEDED, 25
PMIX PDATA, 74
PMIX_PERSIST, 74
PMIX_PERSIST_APP, 40
PMIX_PERSIST_FIRST_READ, 40
PMIX_PERSIST_INDEF, 40
PMIX_PERSIST_INVALID, 40
PMIX_PERSIST_PROC, 40
PMIX PERSIST SESSION, 40
PMIX_PID, 74
PMIX POINTER, 74
PMIX PROC, 74
PMIX PROC HAS CONNECTED, 25
PMIX PROC INFO, 74
PMIX PROC RANK, 74
```

```
PMIX_PROC_STATE, 74
PMIX PROC STATE ABORTED, 35
PMIX PROC STATE ABORTED BY SIG, 35
PMIX PROC STATE CALLED ABORT, 35
PMIX PROC STATE CANNOT RESTART, 35
PMIX_PROC_STATE_COMM_FAILED, 35
PMIX PROC STATE CONNECTED, 35
PMIX PROC STATE ERROR, 35
PMIX PROC STATE FAILED TO LAUNCH, 35
PMIX_PROC_STATE_FAILED_TO_START, 35
PMIX_PROC_STATE_HEARTBEAT_FAILED, 35
PMIX_PROC_STATE_KILLED_BY_CMD, 35
PMIX PROC STATE LAUNCH UNDERWAY, 35
PMIX_PROC_STATE_MIGRATING, 35
PMIX_PROC_STATE_PREPPED, 35
PMIX_PROC_STATE_RESTART, 35
PMIX PROC_STATE_RUNNING, 35
PMIX_PROC_STATE_SENSOR_BOUND_EXCEEDED, 35
PMIX_PROC_STATE_TERM_NON_ZERO, 35
PMIX PROC STATE TERM WO SYNC, 35
PMIX_PROC_STATE_TERMINATE, 35
PMIX PROC STATE TERMINATED, 35
PMIX PROC STATE UNDEF, 35
PMIX PROC STATE UNTERMINATED, 35
PMIX PROC TERMINATED, 24
PMIX QUERY, 75
PMIX_QUERY_PARTIAL_SUCCESS, 23
PMIX RANGE CUSTOM, 39
PMIX_RANGE_GLOBAL, 39
PMIX RANGE INVALID, 39
PMIX_RANGE_LOCAL, 39
PMIX_RANGE_NAMESPACE, 39
PMIX RANGE PROC LOCAL, 39
PMIX_RANGE_RM, 39
PMIX_RANGE_SESSION, 39
PMIX_RANGE_UNDEF, 39
PMIX RANK INVALID, 31
PMIX RANK LOCAL NODE, 31
PMIX RANK LOCAL PEERS, 31
PMIX_RANK_UNDEF, 31
PMIX RANK VALID, 31
PMIX RANK WILDCARD, 31
PMIX REGATTR, 75
```

PMIX_REGEX, 75 PMIX_REMOTE, 38 PMIX_SCOPE, 74 PMIX_SCOPE_UNDEF, 38 PMIX_SIZE, 74 PMIX_STATUS, 74 PMIX_STRING, 74 PMIX_SUCCESS, 22 PMIX_TIME, 74 PMIX_TIMEVAL, 74 PMIX_UINT, 74 PMIX_UINT16, 74 PMIX_UINT32, 74 PMIX_UINT64, <u>74</u> PMIX_UINT8, 74 PMIX_UNDEF, 74 PMIX_VALUE, 74

Index of Attributes

```
PMIX ADD ENVAR, 94
PMIX ADD HOST, 89, 160, 165, 298
PMIX ADD HOSTFILE, 89, 160, 165, 298
PMIX ALLOC BANDWIDTH, 96, 96, 198, 201, 272, 320
PMIX ALLOC CPU LIST, 95, 197, 200, 319
PMIX ALLOC FABRIC, 95, 95, 198, 201, 272, 319
PMIX ALLOC FABRIC ENDPTS, 95, 96, 96, 198, 201, 272, 319
PMIX ALLOC FABRIC_ENDPTS_NODE, 96, 96, 198, 201, 272
PMIX ALLOC FABRIC ID, 95, 95, 198, 201, 272, 319
PMIX_ALLOC_FABRIC_PLANE, 96, 96, 198, 201, 272, 319
PMIX_ALLOC_FABRIC_QOS, 96, 96, 198, 201, 272, 320
PMIX_ALLOC_FABRIC_SEC_KEY, 96, 96, 198, 201, 272, 320
PMIX_ALLOC_FABRIC_TYPE, 95, 96, 96, 198, 201, 272, 319
PMIX_ALLOC_ID, 12, 95, 199, 319
PMIX ALLOC MEM SIZE, 95, 198, 200, 319
PMIX_ALLOC_NETWORK (Deprecated), 95
PMIX ALLOC NETWORK ENDPTS (Deprecated), 96
PMIX_ALLOC_NETWORK_ENDPTS_NODE (Deprecated), 96
PMIX ALLOC NETWORK ID (Deprecated), 95
PMIX ALLOC NETWORK PLANE (Deprecated), 96
PMIX ALLOC NETWORK QOS (Deprecated), 96
PMIX ALLOC NETWORK SEC KEY (Deprecated), 96
PMIX ALLOC NETWORK TYPE (Deprecated), 96
PMIX ALLOC NODE LIST, 95, 197, 200, 319
PMIX ALLOC NUM CPU LIST, 95, 197, 200, 319
PMIX ALLOC NUM CPUS, 95, 197, 200, 319
PMIX_ALLOC_NUM_NODES, 95, 197, 200, 319
PMIX_ALLOC_QUEUE, 92, 95, 188, 193, 313
PMIX_ALLOC_REQ_ID, 12, 95, 197, 200
PMIX ALLOC TIME, 96, 197, 200, 272, 319
PMIX_ALLOCATED_NODELIST, 80, 252
PMIX_ANL_MAP, 87, 253
PMIX_APP_ARGV, 91, 254
PMIX APP INFO, 81, 132, 135, 139, 186, 191, 254
PMIX APP INFO ARRAY, 82, 82, 253, 263
PMIX APP MAP REGEX, 87
PMIX_APP_MAP_TYPE, 87
PMIX APP RANK, 80, 256
```

```
PMIX_APP_SIZE, 83, 139, 254, 263
PMIX APPEND ENVAR, 95
PMIX APPLDR, 80, 254, 263
PMIX APPNUM, 79, 81, 82, 132, 135, 139, 186, 191, 253, 256, 263
PMIX ARCH, 79
PMIX_ATTR_UNDEF, 75
PMIX AVAIL PHYS MEMORY, 84, 256
PMIX BINDTO, 89, 161, 165, 253, 299
PMIX CLEANUP EMPTY, 98, 204, 206
PMIX_CLEANUP_IGNORE, 98, 204, 206
PMIX_CLEANUP_LEAVE_TOPDIR, 98, 204, 207
PMIX_CLEANUP_RECURSIVE, 98, 204, 206
PMIX CLIENT ATTRIBUTES, 13, 100, 187, 192
PMIX_CLIENT_AVG_MEMORY, 84
PMIX_CLIENT_FUNCTIONS, 92, 100, 186, 191
PMIX_CLUSTER_ID, 79, 252
PMIX_CMD_LINE, 91
PMIX_COLLECT_DATA, 85, 142, 144, 287
PMIX COLLECTIVE ALGO, 9, 85, 142, 145, 169, 172, 287, 302
PMIX_COLLECTIVE_ALGO_REQD, 142, 145, 169, 172, 287, 302
PMIX_COLLECTIVE_ALGO_REQD (Deprecated), 85
PMIX CONNECT MAX RETRIES, 392, 403
PMIX CONNECT RETRY DELAY, 392, 403
PMIX CONNECT SYSTEM FIRST, 390, 392, 403, 406
PMIX_CONNECT_TO_SYSTEM, 390, 392, 403, 406
PMIX COSPAWN APP, 90
PMIX_CPU_LIST, 90, 162, 166, 300
PMIX CPUS PER PROC, 90, 161, 166, 300
PMIX_CPUSET, 79
PMIX CRED TYPE, 99, 327
PMIX_CREDENTIAL, 79
PMIX_CRYPTO_KEY, 99
PMIX DAEMON MEMORY, 84
PMIX_DATA_SCOPE, 86, 131, 134
PMIX_DEBUG_APP_DIRECTIVES, 397
PMIX_DEBUG_JOB (Deprecated), 397
PMIX DEBUG JOB DIRECTIVES, 397
PMIX DEBUG STOP IN INIT, 397
PMIX DEBUG STOP ON EXEC, 397
PMIX_DEBUG_TARGET, 397, 397
PMIX DEBUG WAIT FOR NOTIFY, 397
PMIX DEBUG WAITING FOR NOTIFY, 397
PMIX DEBUGGER DAEMONS, 90, 161, 166, 299
```

```
PMIX_DISPLAY_MAP, 89, 160, 165, 299
PMIX DSTPATH, 76
PMIX EMBED BARRIER, 86, 125
PMIX ENUM VALUE, 13, 62, 100
PMIX EVENT ACTION TIMEOUT, 88, 223
PMIX_EVENT_AFFECTED_PROC, 88, 223, 227
PMIX EVENT AFFECTED PROCS, 88, 223, 227
PMIX EVENT BASE, 75, 124, 127, 404
PMIX EVENT CUSTOM RANGE, 88, 223, 227
PMIX_EVENT_DO_NOT_CACHE, 88
PMIX_EVENT_HDLR_AFTER, 87, 222
PMIX_EVENT_HDLR_APPEND, 87, 223
PMIX EVENT HDLR BEFORE, 87, 222
PMIX_EVENT_HDLR_FIRST, 87, 222
PMIX_EVENT_HDLR_FIRST_IN_CATEGORY, 87, 222
PMIX_EVENT_HDLR_LAST, 87, 222
PMIX_EVENT_HDLR_LAST_IN_CATEGORY, 87, 222
PMIX EVENT HDLR NAME, 87, 222
PMIX EVENT HDLR PREPEND, 87, 223
PMIX EVENT NO TERMINATION, 88
PMIX EVENT NON DEFAULT, 88, 227
PMIX EVENT PROXY, 88
PMIX EVENT RETURN OBJECT, 88, 223
PMIX EVENT SILENT TERMINATION, 88, 223
PMIX EVENT TERMINATE JOB, 88, 223
PMIX EVENT TERMINATE NODE, 88, 223
PMIX_EVENT_TERMINATE_PROC, 88, 223
PMIX EVENT TERMINATE SESSION, 88, 223
PMIX_EVENT_TEXT_MESSAGE, 88
PMIX EVENT WANT TERMINATION, 88
PMIX_EXIT_CODE, 81
PMIX_FABRIC_COORDINATE, 346
PMIX FABRIC COST MATRIX, 343, 346
PMIX_FABRIC_DEVICE, 344, 347
PMIX FABRIC DEVICE ADDRESS, 344, 348, 355
PMIX_FABRIC_DEVICE_BUS_TYPE, 345, 347, 354
PMIX FABRIC DEVICE DRIVER, 344, 347, 355
PMIX_FABRIC_DEVICE_FIRMWARE, 344, 348, 355
PMIX FABRIC DEVICE ID, 344, 347, 355
PMIX FABRIC DEVICE INDEX, 337, 342, 347, 347
PMIX FABRIC DEVICE MTU, 345, 348, 355
PMIX FABRIC DEVICE NAME, 344, 347, 354
PMIX FABRIC DEVICE PCI DEVID, 345, 348, 348, 354, 355, 357
```

```
PMIX_FABRIC_DEVICE_SPEED, 345, 348, 355
PMIX FABRIC DEVICE STATE, 345, 348, 355
PMIX FABRIC DEVICE TYPE, 345, 348, 355
PMIX FABRIC DEVICE VENDOR, 344, 347, 354
PMIX FABRIC DIMS, 343, 346
PMIX_FABRIC_ENDPT, 347
PMIX FABRIC GROUPS, 343, 346
PMIX FABRIC IDENTIFIER, 336, 343, 346, 349
PMIX FABRIC INDEX, 342, 346, 358
PMIX_FABRIC_NUM_VERTICES, 343, 346
PMIX_FABRIC_PLANE, 336, 339, 344, 346, 349, 350
PMIX_FABRIC_SHAPE, 344, 347
PMIX FABRIC SHAPE STRING, 344, 347
PMIX_FABRIC_SWITCH, 347
PMIX_FABRIC_VENDOR, 336, 343, 346, 349
PMIX_FABRIC_VIEW, 346
PMIX_FIRST_ENVAR, 95
PMIX FORK EXEC AGENT, 91
PMIX FWD STDDIAG, 11, 396
PMIX FWD STDERR, 161, 166, 299, 315, 396, 399
PMIX_FWD_STDIN, 161, 166, 299, 315, <u>396</u>
PMIX FWD STDOUT, 161, 166, 299, 315, 396, 399
PMIX GDS MODULE, 78, 124
PMIX GET REFRESH CACHE, 86
PMIX GET STATIC VALUES, 86, 132, 135
PMIX GLOBAL RANK, 80, 256
PMIX_GROUP_ASSIGN_CONTEXT_ID, 101, 334, 335, 364, 368, 375, 378
PMIX GROUP CONTEXT ID, 101, 335
PMIX_GROUP_ENDPT_DATA, 101, 334, 335
PMIX GROUP FT COLLECTIVE, 101
PMIX_GROUP_ID, 101, 335
PMIX_GROUP_INVITE_DECLINE, 101
PMIX_GROUP_LEADER, 101, 364, 365, 368, 377, 382
PMIX_GROUP_LOCAL_ONLY, 101, 334, 364, 368
PMIX_GROUP_MEMBERSHIP, 101, 335, 365
PMIX_GROUP_NOTIFY_TERMINATION, 101, 364–366, 368, 371, 375, 378
PMIX GROUP OPTIONAL, 101, 334, 364, 366, 368, 374, 378
PMIX GRPID, 76, 115, 146, 148, 150, 152, 154, 156, 187, 192, 197, 200, 203, 206, 209, 211, 213,
        216, 241, 242, 244, 246, 291–294, 296, 298, 307, 312, 314, 316, 318, 322, 324, 326, 329,
        330, 333
PMIX HOST, 89, 160, 164, 298
PMIX HOST ATTRIBUTES, 13, 100, 187, 192, 195
PMIX HOST FUNCTIONS, 92, 100, 186, 191
```

```
PMIX_HOSTFILE, 89, 160, 164, 298
PMIX HOSTNAME, 80, 80–82, 132, 135, 140, 186, 191, 254–257, 344, 345, 348, 354, 355, 357
PMIX HOSTNAME ALIASES, 80, 255
PMIX HOSTNAME KEEP FQDN, 80, 253
PMIX HWLOC HOLE KIND, 85
PMIX_HWLOC_SHARE_TOPO, 85
PMIX HWLOC SHMEM ADDR, 84, 255
PMIX HWLOC SHMEM FILE, 85, 255
PMIX HWLOC SHMEM SIZE, 85, 255
PMIX_HWLOC_XML_V1, 85, 256
PMIX_HWLOC_XML_V2, 85, 256
PMIX_IMMEDIATE, 85, 131, 133, 134
PMIX INDEX ARGV, 90, 161, 166, 300
PMIX_IOF_BUFFERING_SIZE, 99, 178, 182, 331, 401, 409, 412
PMIX_IOF_BUFFERING_TIME, 99, 178, 182, 331, 401, 409, 412
PMIX_IOF_CACHE_SIZE, 99, 178, 182, 331, 401, 408, 411
PMIX_IOF_COMPLETE, 99, 116, 400, 402, 420
PMIX IOF DROP NEWEST, 99, 178, 182, 331, 401, 408, 412
PMIX_IOF_DROP_OLDEST, 99, 178, 182, 331, 401, 408, 412
PMIX IOF PUSH STDIN, 99, 400, 402
PMIX IOF STOP, 99
PMIX IOF TAG OUTPUT, 99, 178, 402, 409
PMIX IOF TIMESTAMP OUTPUT, 99, 178, 402, 409
PMIX IOF XML OUTPUT, 99, 179, 402, 409
PMIX JOB CONTINUOUS, 91, 162, 166, 300
PMIX JOB CTRL CANCEL, 97, 204, 207, 322
PMIX_JOB_CTRL_CHECKPOINT, 97, 204, 207, 322
PMIX JOB CTRL CHECKPOINT EVENT, 97, 204, 207, 322
PMIX_JOB_CTRL_CHECKPOINT_METHOD, 97, 204, 207, 323
PMIX JOB CTRL CHECKPOINT SIGNAL, 97, 204, 207, 322
PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT, 97, 204, 207, 322
PMIX_JOB_CTRL_ID, 97, 97, 203, 204, 206, 207, 322
PMIX JOB CTRL KILL, 97, 203, 206, 322
PMIX_JOB_CTRL_PAUSE, 97, 203, 206, 322
PMIX_JOB_CTRL_PREEMPTIBLE, 97, 204, 207, 323
PMIX_JOB_CTRL_PROVISION, <u>97</u>, 204, 207, 323
PMIX JOB CTRL PROVISION IMAGE, 97, 204, 207, 323
PMIX JOB CTRL RESTART, 97, 204, 207, 322
PMIX JOB CTRL RESUME, 97, 203, 206, 322
PMIX JOB CTRL SIGNAL, 97, 203, 206, 322
PMIX JOB CTRL TERMINATE, 97, 203, 206, 322
PMIX JOB INFO, 81, 131, 135, 138, 186, 191
PMIX JOB INFO ARRAY, 10, 82, 82, 83, 252, 261
```

```
PMIX_JOB_NUM_APPS, 83, 138, 253, 261
PMIX JOB RECOVERABLE, 91, 162, 166, 300
PMIX JOB SIZE, 10, 12, 83, 138, 252, 261, 262
PMIX JOB TERM STATUS, 86
PMIX_JOBID, <u>79</u>, 81, 82, 132, 135, 139, 186, 191, 252, 261
PMIX LAUNCHER, 392
PMIX LAUNCHER RENDEZVOUS FILE, 389, 393
PMIX LOCAL CPUSETS, 81, 255, 265
PMIX LOCAL PEERS, 80, 255, 264
PMIX_LOCAL_PROCS, 81, 255
PMIX_LOCAL_RANK, 80, 186, 187, 191, 192, 256
PMIX_LOCAL_SIZE, 83, 255
PMIX LOCAL TOPO, 84
PMIX_LOCALITY, 81
PMIX_LOCALITY_STRING, 84, 257
PMIX_LOCALLDR, 80, 255
PMIX_LOG_EMAIL, 94, 215, 218, 317
PMIX_LOG_EMAIL_ADDR, 94, 215, 218, 317
PMIX_LOG_EMAIL_MSG, 94, 215, 218, 317
PMIX LOG EMAIL SENDER ADDR, 94
PMIX_LOG_EMAIL_SERVER, 94
PMIX LOG EMAIL SRVR PORT, 94
PMIX LOG EMAIL SUBJECT, 94, 215, 218, 317
PMIX LOG GENERATE TIMESTAMP, 93, 214, 217
PMIX LOG GLOBAL DATASTORE, 94, 215, 218
PMIX LOG GLOBAL SYSLOG, 93, 214, 217
PMIX_LOG_JOB_RECORD, 94, 215, 218
PMIX_LOG_LOCAL_SYSLOG, 93, 214, 217
PMIX_LOG_MSG, 93, 317
PMIX LOG_ONCE, 93, 214, 217
PMIX_LOG_SOURCE, 93, 214, 217
PMIX_LOG_STDERR, 93, 214, 217, 316
PMIX_LOG_STDOUT, 93, 214, 217, 316
PMIX_LOG_SYSLOG, 93, 214, 217, 316
PMIX_LOG_SYSLOG_PRI, 93, 214, 217
PMIX_LOG_TAG_OUTPUT, 93, 214, 217
PMIX LOG TIMESTAMP, 93, 214, 217
PMIX LOG TIMESTAMP OUTPUT, 93, 214, 217
PMIX LOG XML OUTPUT, 93, 214, 217
PMIX_MAP_BLOB, 87
PMIX MAPBY, 89, 160, 165, 253, 299
PMIX MAPPER, 89, 89, 160, 165, 299
PMIX MAX PROCS, 12, 62, 83, 83, 140, 251, 252, 254, 256
```

```
PMIX_MAX_RESTARTS, 91, 162, 167, 300
PMIX MAX VALUE, 13, 62, 100
PMIX MERGE STDERR STDOUT, 90, 161, 166, 300
PMIX MIN VALUE, 13, 62, 100
PMIX MODEL AFFINITY POLICY, 77
PMIX_MODEL_CPU_TYPE, 77
PMIX MODEL LIBRARY NAME, 77, 254, 273
PMIX MODEL LIBRARY VERSION, 77, 254, 273
PMIX MODEL NUM CPUS, 77
PMIX_MODEL_NUM_THREADS, 77
PMIX_MODEL_PHASE_NAME, 77
PMIX_MODEL_PHASE_TYPE, 77
PMIX MONITOR APP CONTROL, 98, 209, 211, 325
PMIX_MONITOR_CANCEL, 98, 209, 211, 325
PMIX_MONITOR_FILE, 98, 209, 210, 212, 325
PMIX_MONITOR_FILE_ACCESS, 98, 209, 212, 325
PMIX_MONITOR_FILE_CHECK_TIME, 98, 210, 212, 325
PMIX MONITOR FILE DROPS, 98, 210, 212, 325
PMIX MONITOR FILE MODIFY, 98, 210, 212, 325
PMIX MONITOR FILE SIZE, 98, 209, 212, 325
PMIX MONITOR HEARTBEAT, 98, 209, 211, 325
PMIX MONITOR HEARTBEAT DROPS, 98, 209, 212, 325
PMIX MONITOR HEARTBEAT TIME, 98, 209, 211, 325
PMIX MONITOR ID, 98, 209, 211, 325
PMIX NO OVERSUBSCRIBE, 90, 162, 166, 300
PMIX NO PROCS ON HEAD, 90, 162, 166, 300
PMIX_NODE_INFO, 81, 132, 135, 140, 186, 191, 256
PMIX_NODE_INFO_ARRAY, 82, 82, 254, 262, 264
PMIX_NODE_LIST, 80
PMIX_NODE_MAP, 12, 86, 253, 261–263, 273
PMIX_NODE_RANK, 80, 256
PMIX_NODE_SIZE, 83, 140, 255
PMIX NODEID, 80, 80–82, 132, 135, 140, 186, 191, 254–256, 345, 348, 354, 355, 357
PMIX NOHUP, 396
PMIX NON PMI, 90, 161, 165, 299
PMIX NOTIFY COMPLETION, 85, 162
PMIX NOTIFY LAUNCH, 85, 86
PMIX NPROC OFFSET, 80, 253
PMIX NSDIR, 79, 79, 255, 257
PMIX NSPACE, 79, 81, 82, 91, 92, 132, 135, 139, 186–189, 191–194, 252, 262, 312, 313
PMIX NUM NODES, 84, 137-139, 261, 262
PMIX NUM SLOTS, 83
PMIX OPTIONAL, 86, 131, 133, 134
```

```
PMIX_OUTPUT_TO_DIRECTORY, 90
PMIX OUTPUT TO FILE, 90, 161, 166, 300
PMIX PACKAGE RANK, 80
PMIX PARENT ID, 81, 159, 164, 298
PMIX PERSISTENCE, 85, 147, 149, 291
PMIX PERSONALITY, 89, 160, 165, 299
PMIX PPR, 89, 160, 165, 299
PMIX PREFIX, 89, 160, 164, 298
PMIX PRELOAD BIN, 90, 160, 165, 298
PMIX_PRELOAD_FILES, 90, 160, 165, 298
PMIX_PREPEND_ENVAR, 94
PMIX_PROC_BLOB, 87
PMIX PROC DATA, 86
PMIX_PROC_INFO_ARRAY, 82, 86, 256, 263
PMIX_PROC_MAP, 12, 87, 253, 261, 262, 273
PMIX_PROC_PID, 80
PMIX_PROC_STATE_STATUS, 86
PMIX_PROC_TERM_STATUS, 86
PMIX_PROC_URI, 81, 189, 194
PMIX PROCDIR, 79, 257
PMIX_PROCID, <u>79</u>, 82, 92, 186–188, 191–193, 313
PMIX PROGRAMMING MODEL, 77, 254, 273
PMIX PSET NAME, 77, 254, 359
PMIX QUERY ALLOC STATUS, 92, 189, 193, 313
PMIX_QUERY_ATTRIBUTE_SUPPORT, 92, 186, 191, 195
PMIX QUERY AUTHORIZATIONS, 92
PMIX_QUERY_AVAIL_SERVERS, 93
PMIX OUERY DEBUG SUPPORT, 92, 188, 193, 313
PMIX_QUERY_JOB_STATUS, 91, 188, 193, 312
PMIX_QUERY_LOCAL_ONLY, 92, 313
PMIX_QUERY_LOCAL_PROC_TABLE, 92, 188, 193, 313
PMIX_QUERY_MEMORY_USAGE, 92, 188, 193, 313
PMIX_QUERY_NAMESPACE_INFO, 91
PMIX_QUERY_NAMESPACES, 91, 188, 193, 312
PMIX_QUERY_NUM_PSETS, 92
PMIX_QUERY_PROC_TABLE, 92, 188, 193, 313
PMIX QUERY PSET NAMES, 93
PMIX QUERY QUEUE LIST, 92, 188, 193, 312
PMIX QUERY QUEUE STATUS, 92, 188, 193, 312
PMIX_QUERY_REFRESH_CACHE, 91, 185, 189, 190, 194
PMIX QUERY REPORT AVG, 92, 188, 193, 313
PMIX QUERY REPORT MINMAX, 92, 189, 193, 313
PMIX QUERY SPAWN SUPPORT, 92, 188, 193, 313
```

```
PMIX_QUERY_STORAGE_LIST, 102
PMIX QUERY SUPPORTED KEYS, 91
PMIX QUERY SUPPORTED QUALIFIERS, 91
PMIX RANGE, 85, 147, 149, 150, 153, 154, 156, 210, 223, 291, 293, 296, 310, 335
PMIX_RANK, 80, 82, 92, 186–188, 191–193, 256, 313
PMIX_RANKBY, 89, 161, 165, 253, 299
PMIX RECONNECT SERVER (Deprecated), 393
PMIX REGISTER CLEANUP, 97, 203, 206
PMIX REGISTER CLEANUP DIR, 98, 203, 206
PMIX_REGISTER_NODATA, 86, 251
PMIX_REINCARNATION, 77, 256
PMIX_REPORT_BINDINGS, 90, 162, 166, 300
PMIX REQUESTOR IS CLIENT, 77, 159, 164
PMIX_REQUESTOR_IS_TOOL, 77, 159, 164
PMIX_REQUIRED_KEY, 87
PMIX_RM_NAME, 94
PMIX_RM_VERSION, 94
PMIX SEND HEARTBEAT, 98
PMIX SERVER ATTRIBUTES, 13, 100, 187, 192
PMIX SERVER ENABLE MONITORING, 76
PMIX SERVER FUNCTIONS, 92, 100, 186, 191
PMIX SERVER GATEWAY, 76
PMIX SERVER HOSTNAME, 392
PMIX SERVER INFO ARRAY, 82
PMIX SERVER NSPACE, 76, 126, 252, 390, 403, 406
PMIX SERVER PIDINFO, 390, 392, 403, 406
PMIX_SERVER_RANK, 76, 126, 252
PMIX SERVER REMOTE CONNECTIONS, 76, 127
PMIX_SERVER_SCHEDULER, 346, 348
PMIX SERVER SESSION SUPPORT, 76
PMIX_SERVER_START_TIME, 76
PMIX_SERVER_SYSTEM_SUPPORT, 76, 126, 388
PMIX SERVER TMPDIR, 76, 126, 388–390
PMIX_SERVER_TOOL_SUPPORT, 76, 126, 128, 239
PMIX_SERVER_URI, 189, 194, 390, 391, 392, 403, 406
PMIX_SESSION_ID, 80, 81, 82, 131, 135, 137, 186, 191, 252, 261
PMIX SESSION INFO, 81, 131, 135, 137, 185, 190, 251–253
PMIX_SESSION_INFO_ARRAY, 10, 82, 83, 251, 261
PMIX SET ENVAR, 94
PMIX SET SESSION CWD, 90, 160, 164, 298
PMIX SETUP APP ALL, 100, 271
PMIX SETUP APP ENVARS, 100, 271
PMIX SETUP APP NONENVARS, 100, 271
```

```
PMIX_SINGLE_LISTENER, 78, 123
PMIX SOCKET MODE, 78, 123, 127, 403
PMIX SPAWN TOOL, 91
PMIX SPAWNED, 79, 159, 164, 257, 298
PMIX_STDIN_TGT, 90, 161, 165, 299
PMIX_STORAGE_AVAIL_BW, 102
PMIX STORAGE BW, 102
PMIX STORAGE CAPACITY AVAIL, 102
PMIX STORAGE CAPACITY FREE, 102
PMIX_STORAGE_CAPACITY_LIMIT, 102
PMIX_STORAGE_ID, 101
PMIX_STORAGE_OBJECT_LIMIT, 102
PMIX STORAGE OBJECTS AVAIL, 102
PMIX_STORAGE_OBJECTS_FREE, 102
PMIX STORAGE PATH, 102
PMIX_STORAGE_TYPE, 102
PMIX_SWITCH_PEERS, 347
PMIX SYSTEM TMPDIR, 76, 126, 388, 389
PMIX TAG OUTPUT, 90, 161, 166, 299
PMIX TCP DISABLE IPV4, 78, 124, 127, 404
PMIX TCP DISABLE IPV6, 78, 124, 127, 404
PMIX TCP IF EXCLUDE, 78, 123, 127, 404
PMIX TCP IF INCLUDE, 78, 123, 127, 404
PMIX TCP IPV4 PORT, 78, 124, 127, 404
PMIX TCP IPV6 PORT, 78, 124, 127, 404
PMIX TCP REPORT URI, 78, 123, 127, 403
PMIX_TCP_URI, 78, 390, 391, 403
PMIX TDIR RMCLEAN, 79
PMIX_THREADING_MODEL, 77
PMIX TIME REMAINING, 92, 183, 189, 194, 313
PMIX_TIMEOUT, 3, 14, 85, 132, 133, 135, 136, 142, 143, 145, 147, 149–151, 153, 154, 156, 169,
        172, 173, 175, 176, 241, 243, 245, 247, 287, 290, 291, 294, 296, 300, 302, 305, 327, 329,
       362, 364–366, 369–371, 373, 375, 379, 381, 383, 384
PMIX_TIMEOUT_REPORT_STATE, 91
PMIX_TIMEOUT_STACKTRACES, 91
PMIX_TIMESTAMP_OUTPUT, 90, 161, 166, 300
PMIX TMPDIR, 79, 79, 255
PMIX TOOL ATTACHMENT FILE, 390, 391, 393, 403
PMIX TOOL ATTRIBUTES, 13, 100, 187, 192
PMIX TOOL CONNECT OPTIONAL, 392
PMIX TOOL DO NOT CONNECT, 392, 403, 404
PMIX TOOL FUNCTIONS, 92, 100, 186, 191
PMIX TOOL NSPACE, 389, 392, 403
```

```
PMIX_TOOL_RANK, 389, <u>392</u>, 403
PMIX_TOPOLOGY, <u>84</u>
PMIX_TOPOLOGY_FILE, <u>84</u>
PMIX_TOPOLOGY_SIGNATURE, <u>84</u>
PMIX_TOPOLOGY_XML, <u>84</u>
PMIX_UNIV_SIZE, 10, 12, <u>83</u>, 137, 251, 252, 261
PMIX_UNSET_ENVAR, <u>94</u>
PMIX_USERID, <u>76</u>, 115, 146, 148, 150, 152, 154, 156, 187, 192, 197, 200, 203, 206, 209, 211, 213, 216, 241, 242, 244, 246, 291–296, 298, 306, 312, 314, 316, 318, 321, 324, 326, 328, 330, 333
PMIX_USOCK_DISABLE, <u>77</u>, 123, 127
PMIX_VERSION_INFO, <u>76</u>
PMIX_WAIT, <u>85</u>, 150, 151, 153, 293
PMIX_WOIR, <u>89</u>, 159, 164, 254, 298
```