

1 *Consortium Specification*

2 **Interconnect Transport API (IT-API)**
3 **Version 2.0**

4 **The Interconnect Software Consortium**

5 in association with



6

7 Copyright © 2005, The Open Group

8 All rights reserved.

9 The copyright owner hereby grants permission for all or part of this publication to be reproduced, stored in a
10 retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying,
11 recording, or otherwise, provided that it remains unchanged and that this copyright statement is included in
12 all copies or substantial portions of the publication.

13 For any software code contained within this specification, permission is hereby granted, free-of-charge, to
14 any person obtaining a copy of this specification (the “Software”), to deal in the Software without restriction,
15 including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
16 copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the
17 above copyright notice and this permission notice being included in all copies or substantial portions of the
18 Software.

19 THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
20 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
21 FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. IN NO EVENT SHALL THE
22 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES, OR OTHER
23 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
24 OUT OF, OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
25 THE SOFTWARE.

26 Permission is granted for implementers to use the names, labels, etc. contained within the specification. The
27 intent of publication of the specification is to encourage implementations of the specification.

28 This specification has not been verified for avoidance of possible third-party proprietary rights. In
29 implementing this specification, usual procedures to ensure the respect of possible third-party intellectual
30 property rights should be followed.

31

32 Consortium Specification

33 **Interconnect Transport API (IT-API) Version 2.0**

34 ISBN: 1-931624-55-0

35 Document Number: C053

36

37 Published by The Open Group, March 2005.

38

39 Comments relating to the material contained in this document may be submitted to:

40 ogspecs@opengroup.org

Contents

41			
42	1	Introduction.....	1
43	1.1	Interface Adapters.....	1
44	1.2	Memory Management.....	2
45	1.3	Communication Endpoints	3
46	1.4	Work Requests and Data Transfer Operations	4
47	1.5	Events	5
48	2	Referenced Documents	7
49	3	Definitions.....	8
50	4	Global Behavior	28
51	4.1	Asynchronous versus Synchronous APIs	28
52	4.2	Thread Safety.....	28
53	4.3	Signal Handlers.....	32
54	4.4	Fork Semantics	33
55	4.5	Exec Semantics	33
56	4.6	Exit Semantics	33
57	4.7	Error Handling.....	33
58	4.8	IT Handle Management	34
59	4.9	Output Parameters	34
60	5	Connection Management	35
61	5.1	Overview	35
62	5.1.1	IRD/ORD Negotiation.....	36
63	5.2	Transport-Independent Interface.....	37
64	5.2.1	Overview	37
65	5.2.2	ULP Constraints for iWARP.....	38
66	5.2.3	API-Supported IRD/ORD Negotiation.....	38
67	5.2.4	Transport-Dependent Attributes.....	39
68	5.3	Transport-Dependent Interface (iWARP-Only)	40
69	5.3.1	Overview	40
70	5.3.2	IRD/ORD Negotiation.....	42
71	6	API Reference Pages.....	43
72		it_address_handle_create().....	46
73		it_address_handle_free().....	49
74		it_address_handle_modify().....	50
75		it_address_handle_query().....	52
76		it_convert_net_addr().....	54
77		it_ep_accept().....	56

78	it_ep_connect()	59
79	it_ep_disconnect().....	65
80	it_ep_free().....	67
81	it_ep_modify()	69
82	it_ep_query().....	71
83	it_ep_rc_create()	73
84	it_ep_reset()	78
85	it_ep_ud_create()	79
86	it_evd_create()	82
87	it_evd_dequeue().....	93
88	it_evd_free().....	95
89	it_evd_modify()	97
90	it_evd_post_se().....	100
91	it_evd_query().....	102
92	it_evd_wait().....	104
93	it_get_consumer_context().....	108
94	it_get_handle_type()	109
95	it_get_pathinfo()	110
96	it_handoff()	113
97	it_hton64().....	115
98	it_ia_create()	116
99	it_ia_free().....	118
100	it_ia_info_free()	119
101	it_ia_query().....	120
102	it_interface_list().....	121
103	it_listen_create()	124
104	it_listen_free().....	126
105	it_listen_query().....	127
106	it_lmr_create().....	129
107	it_lmr_free().....	134
108	it_lmr_modify().....	135
109	it_lmr_query().....	137
110	it_lmr_sync_rdma_read().....	140
111	it_lmr_sync_rdma_write()	142
112	it_make_rdma_addr_absolute()	144
113	it_make_rdma_addr_relative().....	145
114	it_post_rdma_read().....	146
115	it_post_rdma_read_to_rmr()	151
116	it_post_rdma_write().....	156
117	it_post_recv()	161
118	it_post_recvfrom()	166
119	it_post_send().....	169
120	it_post_sendto().....	174
121	it_pz_create()	178
122	it_pz_free().....	179
123	it_pz_query().....	180
124	it_reject().....	181
125	it_rmr_create()	183
126	it_rmr_free().....	185

127		it_rmr_link().....	186
128		it_rmr_query().....	192
129		it_rmr_unlink().....	194
130		it_set_consumer_context().....	197
131		it_socket_convert().....	198
132		it_srq_create().....	206
133		it_srq_free().....	209
134		it_srq_modify().....	210
135		it_srq_query().....	212
136		it_ud_service_reply().....	214
137		it_ud_service_request().....	217
138		it_ud_service_request_handle_create().....	220
139		it_ud_service_request_handle_free().....	223
140		it_ud_service_request_handle_query().....	224
141	7	Data Type Reference Pages	226
142		it_addr_mode_t.....	227
143		it_aevd_notification_event_t.....	229
144		it_affiliated_event_t.....	230
145		it_boolean_t.....	238
146		it_cm_msg_events.....	239
147		it_cm_req_events.....	250
148		it_conn_qual_t.....	253
149		it_context_t.....	255
150		it_dg_remote_ep_addr_t.....	256
151		it_dto_cookie_t.....	258
152		it_dto_events.....	259
153		it_dto_flags_t.....	263
154		it_dto_status_t.....	268
155		it_ep_attributes_t.....	275
156		it_ep_state_t.....	283
157		it_event_t.....	295
158		it_handle_t.....	300
159		it_ia_info_t.....	302
160		it_lmr_triplet_t.....	309
161		it_mem_priv_t.....	310
162		it_net_addr_t.....	312
163		it_path_t.....	314
164		it_rmr_triplet_t.....	318
165		it_rmr_type_t.....	319
166		it_software_event_t.....	320
167		it_status_t.....	321
168		it_unaffiliated_event_t.....	324
169	A	Implementer's Guide	326
170	B	Implementer's Guide to Connection Management for iWARP.....	341
171	B.1	Overview	341

172	B.2	Miscellaneous	342
173	B.3	Interoperability Considerations	342
174	B.4	MPA Marker Control.....	342
175	B.5	Transport-Independent Interface.....	343
176	B.6	Transport-Dependent Interface	350
177	C	Backwards Compatibility with Earlier IT-API Versions	357
178	D	Functional Changes (IT-API Version 2.0 Relative to IT-API Version 1.0)	361
179	E	IT-API 1.0 Errata	377
180	F	Header Files	394
181	F.1	it_api.h	394
182	F.2	it_api_os_specific.h	431
183			

184

List of Figures

185	Figure 1: Connection Establishment with the TII	37
186	Figure 2: Conversion Process with MPA Startup (TDI)	41
187	Figure 3: Conversion Process with MPA Startup Suppression (TDI)	42
188	Figure 4: Conversion Initiator with <i>flags</i> set to IT_SC_DEFAULT	201
189	Figure 5: Conversion Responder with <i>flags</i> set to IT_SC_DEFAULT	202
190	Figure 6: Conversion Initiator with the IT_SC_NO_REQ_REP bit set in <i>flags</i>	203
191	Figure 7: Conversion Responder with the IT_SC_NO_REQ_REP bit set in <i>flags</i>	204
192	Figure 8: Three Way Passive RC Endpoint State Diagram	290
193	Figure 9: Three Way Active RC Endpoint State Diagram	291
194	Figure 10: Two Way Active RC Endpoint State Diagram	292
195	Figure 11: Two Way Passive RC Endpoint State Diagram	293
196	Figure 12: Unreliable Datagram Endpoint State Diagram	293
197	Figure 13: Connection Establishment with MPA Startup (TII): RNIC without RTR State	343
198	Figure 14: Connection Establishment with MPA Startup (TII): RNIC with RTR State	345
199	Figure 15: IRD/ORD Header	346
200	Figure 16: RDMAC Initiator to Permissive IETF Responder	347
201	Figure 17: Permissive IETF Initiator to RDMAC Responder	348
202	Figure 18: RDMAC Initiator to Non-Permissive IETF Responder	349
203	Figure 19: Non-Permissive IETF Initiator to RDMAC Responder	350
204	Figure 20: Conversion Process with MPA Startup (TDI) - RNIC without RTR State	351
205	Figure 21: Conversion Process with MPA Startup Suppression (TDI): RNIC without RTR State	352
206	Figure 22: Permissive IETF Conversion Initiator to RDMAC Conversion Responder	353
207	Figure 23: RDMAC Conversion Initiator to Permissive IETF Conversion Responder	354
208	Figure 24: Non-Permissive IETF Conversion Initiator to RDMAC Conversion Responder	355
209	Figure 25: RDMAC Conversion Initiator to Non-Permissive IETF Conversion Responder	356
210		

211

List of Tables

212	Table 1: Thread Safety Models	29
213	Table 2: Thread-Safety Models Applied to IT-APIs	32
214	Table 3: Connection Management Event Definitions	243
215	Table 4: Event Management Event Fields	243
216	Table 5: reject_reason_code Descriptions	244
217	Table 6: UD Service Resolution Reply Event Definitions	245
218	Table 7: Service Resolution Reply Status	245
219	Table 8: InfiniBand reject_reason_code Mapping	246
220	Table 9: VIA reject_reason_code Mapping	246
221	Table 10: Communication Management Request Event Definitions	251
222		

223

Preface

224

The Interconnect Software Consortium

225
226
227

The purpose of the Interconnect Software Consortium is to develop and publish software specifications, guidelines, and compliance tests that enable the successful deployment of fast interconnects such as those defined by the Infiniband specification.

228

The Open Group

229
230
231
232
233
234
235
236
237

The Open Group is a vendor-neutral and technology-neutral consortium, whose vision of Boundaryless Information Flow will enable access to integrated information within and between enterprises based on open standards and global interoperability. The Open Group works with customers, suppliers, consortia, and other standards bodies. Its role is to capture, understand, and address current and emerging requirements, establish policies, and share best practices; to facilitate interoperability, develop consensus, and evolve and integrate specifications and Open Source technologies; to offer a comprehensive set of services to enhance the operational efficiency of consortia; and to operate the industry's premier certification service, including UNIX certification.

238

Further information on The Open Group is available at www.opengroup.org.

239
240
241

The Open Group has over 15 years' experience in developing and operating certification programs and has extensive experience developing and facilitating industry adoption of test suites used to validate conformance to an open standard or specification.

242

More information is available at www.opengroup.org/testing.

243
244
245
246

The Open Group publishes a wide range of technical documentation, the main part of which is focused on development of Technical and Product Standards and Guides, but which also includes white papers, technical studies, branding and testing documentation, and business titles. Full details and a catalog are available at www.opengroup.org/pubs.

247
248

Readers should note that updates – in the form of Corrigenda – may apply to any publication. This information is published at www.opengroup.org/corrigenda.

249

This Document

250
251
252

This document is the Technical Standard for the Interconnect Transport API (IT-API). It has been developed and approved by The Interconnect Software Consortium in association with The Open Group.

253
254

The version of the specification has the format $m.n$, where m and n denote the major version number and minor version number, respectively. The present publication corresponds to Version

255 2.0 of the IT-API. The previous publication – also known as IT-API Issue 1.0 – corresponds to
256 Version 1.0 of the IT-API.

257 As with all *live* documents, Technical Standards and Specifications require revision to align with
258 new developments and associated international standards. To distinguish between revised
259 specifications which are fully backwards-compatible and those which are not:

- 260 • An unchanged major version number (*m*) and a new minor version number (*n*) indicate
261 there is no change to the definitive information contained in the previous version of the
262 specification, but additions/extensions are included. The new version of the specification
263 is source code-compatible with the previous one and replaces the previous specification.
- 264 • A new major version number (*m*) and a minor version number (*n*) set to zero indicate
265 there is substantive change to the definitive information contained in the previous version
266 of the specification, and there may also be additions/extensions. The new version of the
267 specification is not source code-compatible with the previous one. The specifications
268 corresponding to both versions are maintained as current publications.

269 **Typographical Conventions**

270 The following typographical conventions are used throughout this document:

- 271 • Bold font is used in text for filenames and type names.
- 272 • Italic strings are used for emphasis. Italics in text also denote variable names, functions,
273 and data structures.
- 274 • Normal font is used for the names of constants and literals.
- 275 • Syntax and code examples are shown in fixed width font.
- 276 • Bold Italic is used for all terms defined in the Definitions section when they first appear.
277 IT-API objects are capitalized throughout the document (e.g., Interface Adapter,
278 Endpoint, etc.).

279 **Reference Page Conventions**

280 The following editorial conventions are used on all reference pages of this document:

- 281 • The SYNOPSIS section summarizes the syntax for an item.
- 282 • An APPLICABILITY section is provided for any API routine or data type which either
283 cannot be used for all service types supported by the IT-API or which is supported only
284 optionally.
- 285 • The DESCRIPTION section provides a summary description of an item.
- 286 • An EXTENDED DESCRIPTION section gives more detailed information about the use
287 of the item being documented, and about its behavior and features. It expands on subjects
288 discussed in the DESCRIPTION section, but with more detail and background
289 information. It also provides information on transport dependencies.

- 290 • A **BACKWARDS COMPATIBILITY** section is provided for API routines whose
291 signature has changed from the previous IT-API version. For API routines that were
292 renamed or removed, see Appendix D.
- 293 • The **RETURN VALUES** section (for API routines only) lists the returned values
294 including immediate errors for API routines, along with a description for each.
- 295 • The **ASYNCHRONOUS ERRORS** section describes errors that cannot be reported as
296 immediate errors via return values.
- 297 • The **APPLICATION USAGE** section provides additional guidance to application writers
298 on intended or typical usage of IT-API routines or objects, on using an IT-API routine or
299 object jointly with other routines or objects, and performance hints.

300

Trademarks

301

302

Boundaryless Information Flow™ is a trademark and UNIX® and The Open Group® are registered trademarks of The Open Group in the United States and other countries.

303

InfiniBand™ is a trademark of the InfiniBand™ Trade Association.

304

POSIX® is a registered trademark of the Institute of Electrical and Electronic Engineers, Inc.

305

306

307

308

The Open Group acknowledges that there may be other brand, company, and product names used in this document that may be covered by trademark protection and advises the reader to verify them independently.

309

310

Acknowledgements

311

The Interconnect Software Consortium gratefully acknowledges the contribution of:

Caitlin Bestler	Jay Rosser
Jim Hamrick	Heidi Scott
Arkady Kanevsky	Rajeev Sivaram
Martin Kirk	Richard Treumann
Bernard Metzler	Fred Worley
Fredy Neeser	Hanhong Xue
Matthew Pearson	

312

in the development of the IT-API Specification Version 2.0, and of:

Caitlin Bestler	Matthew Pearson
Edward Chang	Todd Pisek
Joe Cowan	Sherman Pun
Ellen Deleganes	Ashok Raj
David Ford	Kevin Reilly
Rama Govindaraju	Jim Roberts
Jim Hamrick	Jay Rosser
Al Hartmann	Sridharan Sakthivelu
Yaron Haviv	Heidi Scott
Carl Hensler	Steve Sistare
Jimmy Hill	Rajeev Sivaram
Peter Hochschild	Raja Srinivasan
Nobutaka Imamura	Tom Talpey
Arkady Kanevsky	Robert Teisberg
Ted Kim	Anthony Topper
John Kingman	Richard Treumann
Martin Kirk	Tom Tucker
Michael Krause	Andrew Twigger
Mike Moretti	Mark Wittle
Neil Moses	Fred Worley
Peter Ogilvie	Hanhong Xue

313

in the development of the IT-API Specification Version 1.0.

314 1 Introduction

315 The IT-API defines interfaces for direct interaction with Remote Direct Memory Access
316 (RDMA)-capable transports. This IT-API Specification Version 2.0 covers the *Reliable*
317 *Connection* and *Unreliable Datagram* services of the *InfiniBand Transport* [IB-R1.1] [IB-
318 R1.2], the *iWARP Transport* (which also provides a Reliable Connection service) [MPA-IETF]
319 [MPA-RDMAC], [DDP-IETF] [DDP-RDMAC] [RDMAP-IETF] [RDMAP-RDMAC], and *VIA*
320 networks [VIA-V1.0]. The specification includes:

- 321 • An Introduction (this section) (Chapter 1)
- 322 • A list of Referenced Documents (Chapter 2)
- 323 • A Glossary (Chapter 3)
- 324 • A section on Global Behavior (Chapter 4)
- 325 • A section on Connection Management (Chapter 5)
- 326 • Reference pages for 68 APIs and their supporting data type definitions (Chapters 6 and 7)
- 327 • Implementer’s Guides (Appendix A and B)
- 328 • A section on Backwards Compatibility with earlier versions of IT-API (Appendix C)
- 329 • A list of Functional Changes from IT-API Version 1.0 and the detailed Errata (Appendix
330 D and E)
- 331 • Two sample header files (Appendix F)

332 The introduction and all appendices, including implementer’s guides and sample header files, are
333 informative only; the remaining sections are the normative sections of the specification.

334 This overview describes the general architecture presented by the IT-API, reviews the significant
335 data structures that implement the architecture, and introduces key terminology used throughout
336 the API reference pages. It is not a complete description of all supporting interfaces provided by
337 the IT-API, nor does it include the level of descriptive detail provided by the reference pages. It
338 is an introduction to how to use the API. Separate implementer’s guides discuss issues related to
339 implementing the API on a specific transport.

340 1.1 Interface Adapters

341 RDMA-capable transports are implemented in a number of ways, on various hardware
342 platforms, and within different transport layering architectures. A vendor who provides the
343 hardware and software components that make up an RDMA transport implementation, also
344 called the *Implementation*, will enable the listing of the named instances of RDMA-capable
345 transports that are available within a system through the IT-API interface *it_interface_list*. The

346 application program that uses the IT-API to access an RDMA-capable transport is called the
347 **Consumer**. The Consumer may use the information returned by *it_interface_list* to identify an
348 appropriate transport resource. The Consumer then uses the *it_ia_create* call to create and
349 associate an IT **Interface Adapter** instance with the specified transport resource. The Interface
350 Adapter, also called an **IA**, is used to access the underlying RDMA transport.

351 When the Consumer creates an IA using the *it_ia_create* call, an *it_ia_handle_t* is returned. The
352 *it_ia_handle_t* is an opaque type reference **Handle** used by the Consumer to refer to a specific
353 instance of an **IT Object** created by the Implementation. The *it_ia_handle_t* is used as a
354 parameter to subsequent IT-API calls involving the IA. All IT-API interfaces that create an IT
355 Object return an opaque type reference Handle that the Consumer can use in subsequent IT-API
356 calls. It is the Consumer's responsibility to track these Handles, and use them appropriately.

357 The *it_ia_handle_t* is used both to query IA attributes and to create additional IT Objects used
358 for communication on the Interface Adapter. The Consumer can call *it_ia_query* to retrieve
359 attributes and transport-specific parameters associated with the IA, *it_ia_info_free* to release the
360 buffers allocated by *it_ia_query*, and *it_ia_free* to release the *it_ia_handle_t* and all IT Objects
361 associated with it. Most IT Objects follow the basic pattern of support for a standard set of
362 create, query, modify, and free interfaces that are used to manage the object. Additional
363 interfaces make use of each object's specific capabilities.

364 1.2 Memory Management

365 One of the key advantages of RDMA-capable transports is the ability for the transport
366 Implementation to directly access Consumer-defined message buffers. The IT-API provides
367 interfaces to manage the Interface Adapter's use of the Consumer's memory.

368 The Consumer creates a **Local Memory Region**, also called an **LMR**, which defines a region of
369 local memory to be used as a message buffer, as viewed from the perspective of the Interface
370 Adapter. The Consumer defines the LMR and associates it with an Interface Adapter using the
371 *it_lmr_create* call. The *it_lmr_create* call returns an *it_lmr_handle_t* that is used in subsequent
372 IT-API calls to manage the IA's use of the LMR. An LMR has access privileges that need to be
373 set depending on whether the LMR will be accessed by local read or write operations performed
374 by the Interface Adapter, or by incoming remote read (**RDMA Read**) or remote write (**RDMA**
375 **Write**) operations. Access privileges for the LMR can be set when the LMR is created. LMR
376 attributes can be queried and modified by using the *it_lmr_query* and *it_lmr_modify* calls,
377 respectively. The *it_lmr_free* call releases an LMR.

378 The Consumer may create a **Remote Memory Region**, also called an **RMR**, using the
379 *it_rmr_create* call, which returns an *it_rmr_handle_t*. The RMR can be linked to a segment or
380 subregion of an LMR through the *it_rmr_link* call, also referred to as an **RMR Link** operation.
381 Like an LMR, a linked RMR defines a region of local memory to be used as a message buffer, as
382 viewed from the perspective of the Interface Adapter. The one-level indirection provided by an
383 RMR offers flexibility and efficiency in defining message buffers that can be exposed to remote
384 Consumers as targets for subsequent RDMA **Data Transfer Operations**, also referred to as
385 **DTOs**. An RMR is exposed by advertizing to a remote Consumer the *it_rmr_context_t* identifier
386 returned by the *it_rmr_link* call. An RMR has access privileges that need to be set depending on
387 whether the RMR will be accessed through incoming remote read (RDMA Read) or remote
388 write (RDMA Write) operations. Access privileges for the RMR can be set when the RMR is

389 linked. RMR attributes can be queried through the *it_rmr_query* call. The *it_rmr_unlink* call –
390 i.e., an **RMR Unlink** operation – removes the link to the underlying LMR and revokes any
391 remote access rights of the RMR and its associated *it_rmr_context_t*. The *it_rmr_free* call
392 releases an RMR.

393 The Consumer need not create an RMR to expose message buffers for incoming RDMA Read or
394 RDMA Write operations. The Consumer may simply advertize the *it_rmr_context_t* identifier
395 implicitly created when remote access privileges are set in the call to *it_lmr_create*. The
396 *it_rmr_context_t* identifier may either be returned by *it_lmr_create* or may be retrieved through
397 *it_lmr_query*. However, repeatedly creating or modifying LMRs is a far less efficient operation
398 than linking or unlinking RMRs, if the RMRs are linked to underlying LMRs and the underlying
399 LMRs do not require frequent *it_lmr_create* or *it_lmr_modify* operations.

400 A **Protection Zone**, also called a **PZ**, is used to control access to memory when messages are
401 transferred and, more generally, access to IT Objects. Many IT Objects are associated with a PZ
402 when they are created. IT Objects involved in a Data Transfer Operation are required to have the
403 same Protection Zone for the operation to succeed. A Protection Zone is created through the
404 *it_pz_create* call, which returns an *it_pz_handle_t*. Attributes of the PZ can be queried through
405 the *it_pz_query* call. A PZ is released with the *it_pz_free* call.

406 1.3 Communication Endpoints

407 In order to communicate using an Interface Adapter, the Consumer must create a communication
408 **Endpoint**, also called an **EP**. An Endpoint is used to issue requests on the IA. The Endpoint also
409 provides a target for establishing connected communications, and can be associated with an
410 address for use with datagram communications.

411 The Consumer creates an RC-type Endpoint, for use with Reliable Connection communications
412 by calling *it_ep_rc_create*, or a UD-type Endpoint for use with Unreliable Datagram
413 communications by calling *it_ep_ud_create*. An EP can be queried and modified by using the
414 *it_ep_query* and *it_ep_modify* calls, respectively, and can be released with the *it_ep_free* call.

415 By default, when an Endpoint is created it has its own private **Receive Queue** for holding
416 pending requests for receiving data. The Consumer can, however, override this default when
417 creating an RC-type Endpoint and instead have the Endpoint use a **Shared Receive Queue**. A
418 Shared Receive Queue is created by calling *it_srq_create*. It can be queried and modified by
419 using the *it_srq_query* and *it_srq_modify* calls, respectively. It can be released with the
420 *it_srq_free* call. Shared Receive Queues allow the Consumer to conserve receive buffer
421 resources by sharing them across Endpoints.

422 For Reliable Connection communications, a Consumer may issue a request to connect a local
423 Endpoint to a remote Endpoint using the *it_ep_connect* call. In order to receive a **Connection**
424 **Request**, a Consumer creates an IT Listen Point object that is used to await Connection
425 Requests. The Listen Point is created by calling *it_listen_create*, which returns an
426 *it_listen_handle_t*. Attributes of the Listen Point can be queried by using the *it_listen_query* and
427 the Listen Point can be released with the *it_listen_free* call. A Consumer can accept or reject a
428 Connection Request using the *it_ep_accept* and *it_ep_reject* calls, respectively.

429 Alternatively, for the iWARP Transport only, a Consumer may perform a **Conversion** of an
430 unconnected RC-type Endpoint and a connected socket into a connected Endpoint through the
431 *it_socket_convert* call. This call is used for both initiating a Conversion and responding to a
432 Conversion request.

433 An existing Connection can be terminated with the *it_ep_disconnect* call. During the lifetime of
434 a connected communication session, an EP proceeds through successive stages of Connection
435 establishment via state transitions. These states and transitions are described in *it_ep_state_t*.

436 For a comprehensive overview of IT-API Connection management, see Chapter 5.

437 For Unreliable Datagram communications, an IT **Address Handle** object can be created for use
438 in defining and targeting specific remote Endpoints. An Address Handle is created through the
439 *it_address_handle_create* call, which returns an *it_address_handle_t*. Attributes of the Address
440 Handle can be queried and modified by using the *it_address_handle_query* and
441 *it_address_handle_modify* calls. The Address Handle can be released with the
442 *it_address_handle_free* call.

443 For Unreliable Datagram communications, the Consumer can create an IT Service Request
444 Handle that is used to store Destination address information. A Service Request Handle is
445 created through the *it_ud_service_request_handle_create* call, which returns an
446 *it_ud_svc_req_handle_t*. Attributes of the Service Request Handle can be queried through the
447 *it_ud_service_request_handle_query* call and the Service Request Handle can be released with
448 the *it_ud_service_request_handle_free* call. A Service Request Handle is used in the
449 *it_ud_service_request* call to provide addressing information for use in sending the reply
450 message sent by the *it_ud_service_reply* call.

451 1.4 Work Requests and Data Transfer Operations

452 The Consumer can queue different kinds of **Work Requests** to an Endpoint. Work Requests
453 include **Send**, **Receive**, **RDMA Read**, and **RDMA Write** Data Transfer Operations, as well as
454 RMR Link and RMR Unlink memory management operations. Work Requests are posted by
455 Data Transfer Operations to a **Work Queue**.

456 A DTO performs a data transfer from a local **Source** buffer to a remote Endpoint, from a remote
457 Endpoint to a local **Destination** buffer, or directly from a (local or remote) Source buffer into a
458 (remote or local) Destination buffer, as is the case for an RDMA DTO.

459 The Consumer may issue requests to send messages using either the *it_post_send* or
460 *it_post_sendto* interfaces, depending on whether the Endpoint used for communication is of the
461 RC or UD type, respectively. The Consumer issues requests to receive messages using either the
462 *it_post_rcv* or *it_post_rcvfrom* calls. RDMA operations are initiated through the
463 *it_post_rdma_read*, *it_post_rdma_read_to_rmr*, and *it_post_rdma_write* calls.

464 Completions of Work Requests posted to an Endpoint are reported to Consumers
465 asynchronously via **Events**.

466 1.5 Events

467 IT-API calls normally return program control immediately to the issuing Consumer. The call
468 return value indicates either success or immediate failure through an error indication. For some
469 calls, a successful return value means that an operation has been executed successfully, while for
470 other calls it indicates only that a Work Request has been accepted by the Implementation for
471 later execution. For the latter calls, the Consumer is notified of the completion of the Work
472 Request asynchronously via an *Event* mechanism. A Work Request completes either
473 successfully or with a *Completion Error*.

474 Common Event types include *Completion Events* for Work Requests (see *it_dto_status_t*;
475 includes any Completion Errors), *Communication Management Request Events* (see
476 *it_cm_req_events*), *Communication Management Message Events* (see *it_cm_msg_events*),
477 *Affiliated Asynchronous Events* or *Errors* (see *it_affiliated_event_t*), and *Unaffiliated*
478 *Asynchronous Events* or *Errors* (see *it_unaffiliated_event_t*).

479 Communication management Events include Connection management events for the RC service.
480 A transport error condition can manifest through different Event types depending on whether the
481 error condition can be associated with a particular IT Object.

482 Each Event surfaced by the IT-API has an associated Event object (see *it_event_t*) that contains
483 information about the Event, including its type. Event objects are created by the Implementation
484 and made available to the Consumer when an Event occurs. The Implementation enqueues these
485 Event objects on an *Event Dispatcher*, also called an *EVD*.

486 The Consumer creates an Event Dispatcher by calling the *it_evd_create* call, which returns an
487 *it_evd_handle_t*. Attributes of an EVD can be queried and modified by using the *it_evd_query*
488 and *it_evd_modify* calls, respectively. An EVD is released with the *it_evd_free* call. The
489 Consumer may reap a queued Event object using the *it_evd_dequeue* call, or by using the
490 *it_evd_wait* call, which provides a blocking interface for awaiting the next Event to occur, along
491 with a timeout value.

492 The IT-API supports two types of EVDs. A *Simple EVD*, also called an *SEVD*, enqueues only
493 Events of a single type. This simplifies the implementation of an SEVD and enhances its
494 performance characteristics. For many communication scenarios, this provides a Consumer with
495 the best performance option. An *Aggregate EVD*, also called an *AEVD*, can be used to collect
496 Events from a set of SEVDs. This allows the Consumer to create a single *Notification*
497 mechanism that will enqueue many different types of Events. In addition to these IT-API
498 Notification mechanisms, *it_evd_create* allows an Implementation-defined file descriptor to be
499 associated with each EVD. This allows the Consumer to use a file descriptor-based Notification
500 mechanism provided by the Implementation (e.g., POSIX *poll*) to collect both non-IT-API
501 Events and IT-API Events from multiple IAs.

502 Most Events are associated with Endpoints. These Events include Work Request completions
503 and communication management Events related to the Endpoint. When an Endpoint is created,
504 the Consumer associates EVDs with it. These EVDs collect a specific type of Event. One EVD
505 enqueues Events associated with the completion of Consumer-initiated Work Requests,
506 including the completion of RMR Link and Unlink operations, and outgoing Send or RDMA
507 DTOs. A second EVD enqueues Events that result when an incoming Send message that was
508 directed to an Endpoint matches with a corresponding Receive DTO. For Endpoints using

509 connected communications, a third EVD enqueues Events related to Connection management.
510 EVDs may be shared across multiple Endpoints.

511 Affiliated Asynchronous Events are associated with a particular Endpoint, EVD, or S-RQ, but
512 not with a particular Work Request. Consumers can receive Notification of these Events by
513 creating a separate EVD and specifying that it should enqueue this type of Event.

514 Unaffiliated Asynchronous Events are not associated with a particular Endpoint, but are
515 associated with a specific Interface Adapter. Consumers can receive Notification of these Events
516 by creating a separate EVD and specifying that it should enqueue this type of Event.

517 The IT-API gives the Consumer control over a number of aspects of Event handling. IT-API
518 interfaces used to initiate Work Requests include flags for enabling or suppressing the
519 generation of an Event object (also known as per-WR **Completion Suppression**), for enabling or
520 suppressing the associated Consumer Notification (also known as per-WR **Notification**
521 **Suppression**), and for requesting remote-side Endpoint Notification of message delivery. These
522 features are further described in [*it_dto_flags_t*](#).

523 Furthermore, EVDs provide a thresholding attribute used to batch the delivery of Event
524 Notification.

525 2 Referenced Documents

- 526 The following documents are referenced in this specification:
- 527 [DDP-IETF] Direct Data Placement over Reliable Transports, H. Shah et al,
528 IETF Internet Draft, February 2005.
529 (www.ietf.org/internet-drafts/draft-ietf-rddp-ddp-04.txt)
- 530 [DDP-RDMAC] Direct Data Placement over Reliable Transports (Version 1.0), H. Shah et al,
531 RDMA Consortium, October 2002.
532 (www.rdmaconsortium.org/home/draft-shah-iwarp-ddp-v1.0.pdf)
- 533 [IB-R1.1] InfiniBand Architecture Specification Volume 1, Release 1.1,
534 InfiniBand Trade Association, November 2002.
- 535 [IB-R1.2] InfiniBand Architecture Specification Volume 1, Release 1.2,
536 InfiniBand Trade Association, October 2004.
- 537 [INTEROP-IETF] RNIC Interoperability, J. Carrier & J. Pinkerton,
538 IETF Internet Draft, November 2004.
539 (www.ietf.org/internet-drafts/draft-carrier-rddp-rnic-interop-00.txt)
- 540 [IT-API-V1.0] Interconnect Transport API (IT-API) Issue 1.0,
541 The Interconnect Software Consortium, February 2004.
- 542 [MPA-IETF] Marker PDU Aligned Framing for TCP Specification, P. Culley et al,
543 IETF Internet Draft, February 2005.
544 (www.ietf.org/internet-drafts/draft-ietf-rddp-mpa-02.pdf)
- 545 [MPA-RDMAC] Marker PDU Aligned Framing for TCP Specification (Version 1.0),
546 P. Culley et al, RDMA Consortium, October 2002.
547 (www.rdmaconsortium.org/home/draft-culley-iwarp-mpa-v1.0.pdf)
- 548 [RDMAP-IETF] An RDMA Protocol Specification, R. Recio et al,
549 IETF Internet Draft, February 2005.
550 (www.ietf.org/internet-drafts/draft-ietf-rddp-rdmap-03.txt)
- 551 [RDMAP-RDMAC] An RDMA Protocol Specification (Version 1.0), R. Recio et al,
552 RDMA Consortium, October 2002.
553 (www.rdmaconsortium.org/home/draft-recio-iwarp-rdmap-v1.0.pdf)
- 554 [VERBS-RDMAC] RDMA Protocol Verbs Specification (Version 1.0), J. Hilland et al,
555 RDMA Consortium, April 2003.
556 (www.rdmaconsortium.org/home/draft-hilland-iwarp-verbs-v1.0-RDMAC.pdf)
- 557 [VIA-V1.0] Virtual Interface Architecture Specification Version 1.0,
558 Compaq Computer Corporation, Intel Corporation, & Microsoft Corporation,
559 December 1997.

560 **3 Definitions**

561 **Absolute Addressing**

562 Addressing data within an LMR or RMR using absolute addresses in some linear address space.

563 The *addr* attribute (requested starting address) of an LMR with Absolute Addressing is
564 interpreted as the Base Address of the LMR. When an LMR with Absolute Addressing is
565 accessed by a DTO, the *addr.abs* member of an LMR Triplet or the *rdma_addr* parameter
566 passed to an RDMA DTO is interpreted as the Base Address of the LMR plus a byte offset.

567 When an RMR is linked to an LMR with Absolute Addressing selected for the RMR, the *addr*
568 parameter of the RMR Link operation and the *addr* attribute of the LMR (which must use
569 Absolute Addressing as well) are used for identifying the offset of the first byte of the RMR
570 from the first byte of the LMR.

571 The *addr* attribute (starting address) of a linked RMR with Absolute Addressing is interpreted as
572 the Base Address of the RMR. When a linked RMR with Absolute Addressing is accessed by a
573 DTO, the *addr.abs* member of an RMR Triplet or the *rdma_addr* parameter passed to an RDMA
574 DTO is interpreted as the Base Address of the RMR plus a byte offset.

575 See also **Relative Addressing**.

576 **Address Handle**

577 An object that contains the information necessary to transmit messages to a remote port over
578 Unreliable Datagram service. (It should be noted that an Address Handle is an IT Object, not a
579 Handle as defined later in this section.)

580 **AEVD**

581 See **Aggregation Event Dispatcher**.

582 **Affiliated Asynchronous Error**

583 An error which is associated with a specific Endpoint, EVD, or S-RQ, and which could not be
584 reported through an immediate error or Completion Error. Affiliated Asynchronous Errors are
585 reported through an Event Stream for Affiliated Asynchronous Events.

586 **Affiliated Asynchronous Event**

587 An Event associated with a specific Endpoint, EVD, or S-RQ, but not with a specific Work
588 Request.

- 589 **Affiliated Event**
- 590 See **Affiliated Asynchronous Event**.
- 591 **Aggregation Event Dispatcher (AEVD)**
- 592 An IT Object that conceptually merges Event completion Notifications from one or more Simple
593 Event Dispatchers. This provides the Consumer with a single point to receive Notification of
594 Event completions across multiple Event Streams.
- 595 **Asynchronous Error**
- 596 An error that could not be reported as an immediate error. Asynchronous Errors include
597 Completion Errors, Affiliated Asynchronous Errors, and Unaffiliated Asynchronous Errors.¹
- 598 **AV-RNIC**
- 599 API +Verbs-enabled RNIC. See **V-RNIC**. An implementation of an RDMA API (such as IT-
600 API) over a V-RNIC.
- 601 **AV-RNIC/IETF**
- 602 AV-RNIC based on a V-RNIC/IETF with API support for handling the MPA Startup. An AV-
603 RNIC/IETF based on a Permissive (Non-permissive) V-RNIC/IETF is called Permissive (Non-
604 permissive).
- 605 **AV-RNIC/IETF with MPA Startup Suppression**
- 606 An AV-RNIC/IETF, whose API is instructed to suppress MPA Startup for transitioning a
607 particular streaming-mode connection to RDMA mode. API support for suppressing the MPA
608 Startup is outside of [MPA-IETF] but is expected to be supported for interoperability with the
609 device class “AV-RNIC/RDMAC without MPA Startup”. The IT-API provides this capability
610 for the TDI (Conversion process), but not for the TII.
- 611 **AV-RNIC/RDMAC with MPA Startup**
- 612 AV-RNIC based on a V-RNIC/RDMAC and with API support for handling the MPA Startup.
613 according to [INTEROP-IETF], wherein such an AV-RNIC is designated a (Software) Upgraded
614 RDMAC RNIC. An AV-RNIC/RDMAC with MPA Startup has MPA Marker/CRC restrictions.
- 615 **AV-RNIC/RDMAC without MPA Startup**
- 616 AV-RNIC based on a V-RNIC/RDMAC without API support for handling the MPA Startup or
617 with suppressed MPA Startup. An AV-RNIC/RDMAC without MPA Startup also has MPA
618 Marker/CRC restrictions.

¹ This definition includes Completion Errors (which are also asynchronous by nature), in contrast to “Asynchronous Error” defined by [VERBS-RDMAC] or [IB-R1.2].

619	Base Address
620	The address corresponding to the first byte of an LMR or RMR in a given linear address space.
621	The Base Address of a linked RMR is within the address range of the underlying LMR.
622	Bind
623	See RMR Link .
624	Communication Management Message Events
625	The set of Event types related to the sequence of messages involved in RC Connection
626	establishment, normal disconnect, Connection error conditions, and Unreliable Datagram
627	Service Resolution Replies.
628	Communication Management Request Events
629	The set of Event types that result from messages received requesting RC Connection
630	establishment or Unreliable Datagram service. Normally these Events trigger state changes at the
631	receiving Endpoint.
632	Completion Error
633	A processing error that could not be reported through an immediate error and which is associated
634	with a specific Work Request. Completion Errors are reported through an Event Stream for
635	WR/DTO completions.
636	Completion Event
637	An Event indicating that a previously posted Work Request has completed.
638	Completion Suppression
639	An optional Work Request behavior specifying that no Event is to be generated upon successful
640	completion of the requested operation.
641	Connection
642	An association between a pair of Endpoints such that data posted via Data Transfer Operations
643	of either Endpoint arrives at the other Endpoint of the Connection.
644	Connection Qualifier
645	A value that allows an incoming Connection Request or Unreliable Datagram Service Resolution
646	Request to be associated with an entity that can provide that service.

647	Connection Reply
648	A message in response to a Connection Request message.
649	Connection Request
650	A message that requests RC Connection establishment.
651	Consumer
652	An application that utilizes the IT-API.
653	Context
654	A Consumer-supplied value that can be associated with an instance of an IT Object.
655	Conversion
656	The actions performed by <i>it_socket_convert</i> for converting an unconnected Endpoint of the RC
657	type and a connected socket to a connected, RDMA-enabled Endpoint. For the Conversion
658	Initiator, the Conversion includes the transmission of the Last ULP Streaming-Mode Message.
659	Conversion Initiator
660	A Consumer calling <i>it_socket_convert</i> with a Last ULP Streaming-Mode Message of length
661	greater than zero.
662	Conversion Responder
663	A Consumer calling <i>it_socket_convert</i> with a Last ULP Streaming-Mode Message of length
664	zero.
665	Data Transfer Operation (DTO)
666	A request submitted by the Consumer to the Implementation to move data between two
667	Endpoints. See also Work Request .
668	Deferred RDMA Transition
669	The enablement of RDMAP/DDP/MPA on top of a TCP connection after the connection has
670	been established and used by an ULP for some data exchange in streaming mode.
671	Destination
672	The Endpoint where a message is received.

673	DTO
674	See Data Transfer Operation .
675	DTO Cookie
676 677	A Consumer-supplied identifier for a Data Transfer Operation, Link, or Unlink operation that allows the Consumer to uniquely identify the operation when it completes.
678	Endpoint (EP)
679 680 681	The object to which Work Requests and Connection management operations are posted. An RC-type Endpoint is associated with a single Protection Zone. A UD-type Endpoint is associated with a single Protection Zone and a single Spigot.
682	Endpoint Hard High Watermark
683 684	The maximum number of Receive DTOs that an Endpoint with an associated Shared Receive Queue can have in progress.
685	Endpoint ID
686 687	An identifier for an Endpoint on a given Interface Adapter. This is used to help identify the particular Endpoint where a datagram is to be delivered.
688	Endpoint Key
689 690 691	A construct that some transports require to be associated with an outgoing datagram to allow the Receiver to validate that the sender of the datagram has permission to access the Receiver's Endpoint.
692	Endpoint Protection Zone
693	The Protection Zone associated with an Endpoint.
694	Endpoint Soft High Watermark
695 696 697	The maximum number of Receive DTOs that an Endpoint with an associated Shared Receive Queue can have in progress without causing an Endpoint Soft High Watermark Event to be generated.
698	Endpoint Soft High Watermark Event
699 700 701	An Affiliated Asynchronous Event that is generated when the Consumer has armed the Endpoint Soft High Watermark mechanism for an Endpoint and the total number of Receive DTOs in progress on that Endpoint exceeds the Endpoint Soft High Watermark.

702	EP
703	See Endpoint .
704	EVD
705	See Event Dispatcher .
706	Event
707	A structure or record that is delivered to the Consumer through an Event Dispatcher to provide notice of some kind. Types of Events include DTO completions, Connection state changes, Asynchronous Errors, and information passed through the it_evd_post_se interface that is generated by the Consumer.
708	
709	
710	
711	Event Dispatcher (EVD)
712	An IT Object that conceptually merges Event completion Notifications for the Consumer. The IT-API defines two types of EVDs: a Simple Event Dispatcher and an Aggregation Event Dispatcher.
713	
714	
715	Event Stream
716	A type of Event for an Event Dispatcher such as WR/DTO completions, Communication Management Request Events (e.g., Connection Requests), Communication Management Message Events (e.g., Connection reject Notifications, Connection establishment completion Notifications, disconnect Notifications, Connection errors, Connection Request timeouts), Affiliated Asynchronous Errors, Unaffiliated Asynchronous Errors, Consumer-generated Software Events, or AEVD Notifications. The Event Stream of an Event Dispatcher determines the type of Events that can be enqueued onto the Event Dispatcher by IT Objects.
717	
718	
719	
720	
721	
722	
723	Handle
724	An opaque data type used to reference an object.
725	IA
726	See Interface Adapter .
727	IANA
728	See Internet Address Naming Authority .
729	IANA Port Number
730	A specific port address as defined by IANA.

731	IB
732	See InfiniBand .
733	ICSC
734	See Interconnect Software Consortium .
735	IETF
736	See Internet Engineering Task Force .
737	Immediate RDMA Transition
738	The enablement of RDMAP/DDP/MPA on top of a TCP connection immediately after the
739	connection has been established; i.e., without any prior ULP use of the connection for data
740	exchange in streaming mode.
741	Implementation
742	The collection of software and hardware that combine to provide the service exported by the IT-
743	API.
744	Implementer's Guide
745	A non-normative section of the IT-API documentation set that contains information provided to
746	assist implementers of the IT-API.
747	Inbound RDMA Read Queue
748	An internal queue of an Endpoint that handles incoming RDMA Read Requests. The processing
749	of an incoming RDMA Read Request involves generating an RDMA Read Response.
750	InfiniBand (IB)
751	One of the transports that the IT-API supports. The host interface portion of InfiniBand is
752	defined in [IB-R1.1] and [IB-R1.2] for Releases 1.1 and 1.2, respectively.
753	InfiniBand Global Routing Header
754	A routing header that may be present in the first 40 bytes of a completed Unreliable Datagram
755	Receive operation. See the InfiniBand specification for a description of the format of this routing
756	header.
757	InfiniBand Transport
758	Transport services defined by the InfiniBand Architecture.

- 759 **Interconnect Software Consortium (ICSC)**
- 760 Standards organization that includes the ITWG. The *Interconnect Software Consortium* is
761 affiliated with *The Open Group*.
- 762 **Interconnect Transport Working Group (ITWG)**
- 763 The *ICSC Working Group* that created the IT-API.
- 764 **Internet Engineering Task Force (IETF)**
- 765 *Internet Engineering Task Force*.
- 766 **Internet Address Naming Authority (IANA)**
- 767 IETF Network Address naming authority.
- 768 **Interface**
- 769 A host resident device that transfers data to and from the host memory to which it is attached.
- 770 **Interface Adapter (IA)**
- 771 An instance of an Interface that is created by the *it_ia_create* call. An Interface Adapter may
772 contain one or more Spigots.
- 773 **IP**
- 774 The IETF Internet Protocol.
- 775 **IPv4**
- 776 The IETF Internet Protocol Version 4.
- 777 **IPv6**
- 778 The IETF Internet Protocol Version 6.
- 779 **IRD**
- 780 Inbound RDMA Read Queue Depth (IRD). The maximum number of inbound RDMA Read
781 requests that can be outstanding on an Endpoint.
- 782 **IRRQ**
- 783 See **Inbound RDMA Read Queue**.

784	IT-API
785	The data structures and routines that make up the Interconnect Transport Application
786	Programming Interface.
787	IT Handle
788	An opaque reference to an IT Object. An IT Handle is returned to the Consumer whenever an IT
789	Object is created for the Consumer's use. The IT Handle can be used to reference the IT Object
790	in subsequent calls into the IT-API.
791	IT Object
792	A software object created by the IT-API Implementation as a result of a Consumer call into the
793	IT-API, used to satisfy subsequent Consumer requests. When the IT Object is created, an opaque
794	reference to the object, called a Handle, is returned to the Consumer for use in subsequent calls
795	into the IT-API.
796	ITWG
797	See Interconnect Transport Working Group .
798	iWARP
799	The protocol suite enabling RDMA on top of TCP/IP or SCTP/IP, comprising MPA [MPA-
800	IETF] [MPA-RDMAC], DDP [DDP-IETF] [DDP-RDMAC], and RDMAP [RDMAP-IETF
801	RDMAP-RDMAC].
802	iWARP Transport
803	The iWARP protocol suite on top of TCP/IP or SCTP/IP.
804	Last Streaming-Mode Message
805	The last message sent in streaming mode prior to enabling RDMA mode on a connection. In the
806	presence of an MPA Startup, this message is the MPA Reply message.
807	Last ULP Streaming-Mode Message
808	The last ULP message sent in TCP streaming mode prior to enabling RDMA mode on a
809	connection, sent by an IT-API Consumer acting as the Conversion Initiator. Need not be the Last
810	Streaming-Mode Message since the Conversion of a TCP connection typically involves an MPA
811	Startup handshake that is invisible to the Consumers.
812	Link
813	See RMR Link .

814	Listen Point
815	An object capable of listening for incoming Connection Requests and of passing such requests to
816	a Simple Event Dispatcher. A Listen Point is associated with a single Spigot.
817	LLP
818	See Lower Layer Protocol .
819	LMR
820	See Local Memory Region .
821	LMR Triplet
822	A type used to specify a section of a Local Memory Region. An LMR Triplet specifies an LMR
823	Handle, an address, and a length.
824	Local Memory Region (LMR)
825	A contiguously addressable area of arbitrary size within a linear address space enabling local
826	access and optional remote access.
827	Lower Layer Protocol (LLP)
828	A protocol used in a layer of the underlying RDMA transport.
829	Marker PDU Aligned Framing for TCP (MPA)
830	The framing protocol used to support messaging over TCP streams. See [MPA-IETF] and
831	[MPA-RDMAC] .
832	MPA
833	See Marker PDU Aligned Framing for TCP .
834	MPA Startup
835	Exchange of MPA Request/Reply messages according to [MPA-IETF] , preferably with
836	additional support for interoperability according to [INTEROP-IETF] .
837	Narrow RMR
838	An RMR that can be referenced or accessed only via the associated Endpoint if the RMR is in
839	the linked state, where the associated Endpoint is the one through which the RMR was linked.
840	See also Wide RMR .

841	Network Address
842	An identifier that can be used to reach a particular Spigot attached to a network.
843	Non-Permissive AV-RNIC/IETF
844	An AV-RNIC/IETF that cannot support downgrading versions of RDMAP, DDP, and MPA. See
845	[INTEROP-IETF] .
846	Notification
847	An asynchronous mechanism for providing the Consumer with information about the completion
848	of a previously posted operation.
849	Notification Event
850	An Event in an Event Stream whose arrival triggers the Notification of the Event to a waiting
851	Consumer via either a wakeup from <i>it_evd_wait</i> , or via a higher-level Notification mechanism.
852	Notification Suppression
853	A Consumer-specified option for Data Transfer Operations that informs the Implementation that
854	no Notification Event should be created if the DTO completes successfully. Notification
855	Suppression has no effect on operations that complete in error – in this case the completion will
856	generate an error Event.
857	ORD
858	Outbound RDMA Read Queue Depth (ORD). The maximum number of outbound RDMA Read
859	Work Requests a Consumer can have outstanding on an Endpoint.
860	Organization Unique Identifier (OUI)
861	An OUI is a 24-bit globally unique number assigned by the Institute of Electrical and Electronics
862	Engineers (IEEE). The IT-API uses an OUI to map IETF IANA Port Numbers into the IB
863	Service ID space for use within the IT-API.
864	OUI
865	See Organization Unique Identifier .
866	Outstanding Operation
867	An operation is “Outstanding” until the Event for the operation completes, or for an operation
868	whose completion has been suppressed, until an operation posted subsequent to it completes.

869 **Path**

870 The collection of links, switches, and routers a message traverses from a Source Spigot to a
871 Destination Spigot. This is represented in the IT-API by the *it_path_t* structure.

872 **Permissive AV-RNIC/IETF**

873 An AV-RNIC/IETF that supports downgrading versions of RDMAP, DDP, and MPA. See
874 [\[INTEROP-IETF\]](#).

875 **Port Number**

876 See **IANA Port Number**.

877 **Private Data**

878 Consumer data that is opaque to the Implementation and is passed between the local and remote
879 Consumers by the Implementation's Connection establishment and UD service resolution
880 routines.

881 **Protection Zone (PZ)**

882 A mechanism for associating Endpoints and registered LMR and RMR memory of an Interface
883 Adapter that defines protection for local and remote memory accesses by DTO operations.

884 **PZ**

885 See **Protection Zone**.

886 **RC**

887 See **Reliable Connected**.

888 **RDMA**

889 See **Remote Direct Memory Access**.

890 **RDMA Initiator**

891 A Consumer calling *it_ep_connect* or *it_socket_convert* with a Last ULP Streaming-Mode
892 Message of length zero. Also used to identify the corresponding side of the connection. For the
893 RDMA Initiator, the active Endpoint states of the Endpoint finite-state machine are used.

894 **RDMA Responder**

895 A Consumer calling *it_listen_create* followed by *it_ep_accept* or *it_reject*, or a Consumer
896 calling *it_socket_convert* with a Last ULP Streaming-Mode Message of length greater than zero.

897 Also used to identify the corresponding side of the connection. For the RDMA Responder, the
898 passive Endpoint states of the Endpoint finite-state machine are used.

899 **RDMA Read**

900 The Data Transfer Operation (DTO) that is initiated by the *it_post_rdma_read* routine.

901 **RDMA Read Request**

902 A message used by a transport to request the transfer of data from a remote buffer to a local
903 buffer. An RDMA Read Request describes both the remote buffer (data source) and the local
904 buffer (data sink).

905 **RDMA Read Response**

906 A message used by a transport to respond to an RDMA Read Request message.

907 **RDMA Write**

908 The Data Transfer Operation (DTO) that is initiated by the *it_post_rdma_write* routine.

909 **RDMA Write Request**

910 A message used by a transport to request the transfer of data from a local buffer to a remote
911 buffer. An RDMA Write Request describes both the local buffer (data source) and the remote
912 buffer (data sink).

913 **Receive**

914 The Data Transfer Operation (DTO) that is initiated by the *it_post_recv* or *it_post_recvfrom*
915 routine.

916 **Receive Queue (RQ)**

917 An internal queue associated with an Endpoint on which Receive Work Requests are posted.
918 Alternatively, an Endpoint may be associated with a Shared Receive Queue, in which case
919 Receive Work Requests are posted to the Shared Receive Queue.

920 **Relative Addressing**

921 Addressing data within an LMR or RMR using byte offsets relative to the beginning of the LMR
922 or RMR, respectively.

923 The *addr* attribute (requested starting address) of an LMR with Relative Addressing is
924 interpreted as the Base Address of the LMR, unless it equals *IT_NO_ADDR*. This attribute can
925 be used for specifying or registering an LMR but is ignored in case of Relative Addressing when
926 the LMR is accessed by a DTO. When an LMR with Relative Addressing is accessed by a DTO,

927 the *addr.rel* member of an LMR Triplet or the *rdma_addr* parameter passed to an RDMA DTO
928 is interpreted as a byte offset relative to the first byte of the LMR.

929 When an RMR is linked to an LMR, with Relative Addressing selected for the RMR, the *addr*
930 parameter of the RMR Link operation and the *addr* attribute of the LMR (which must use
931 Absolute Addressing) are used for identifying the offset of the first byte of the RMR from the
932 first byte of the LMR.

933 The *addr* attribute (starting address) of a linked RMR with Relative Addressing is interpreted as
934 the Base Address of the RMR. When a linked RMR with Relative Addressing is accessed by a
935 DTO, the *addr.rel* member of an RMR Triplet or the *rdma_addr* parameter passed to an RDMA
936 DTO is interpreted as a byte offset relative to the first byte of the RMR. The total offset relative
937 to the first byte of the underlying LMR is obtained by adding to this byte offset the difference
938 between the *addr* attributes of the RMR and the LMR established at RMR Link time.

939 See also **Absolute Addressing**.

940 **Reliable Connected (RC)**

941 A Transport Service Type in which an Endpoint is associated with only one other Endpoint, such
942 that messages transmitted from one Endpoint are reliably delivered to the other Endpoint,
943 uncorrupted in the absence of errors and in the order defined by the Reliable Connection
944 ordering rules. As such, each Endpoint is said to be “connected” to the opposite Endpoint.

945 **Reliable Connection**

946 A Connection type such that data of posted DTOs of either Endpoint of the Connection reliably
947 arrives at the other Endpoint of the Connection uncorrupted in the absence of errors and in the
948 order defined by the Reliable Connection ordering rules.

949 **Remote Direct Memory Access (RDMA)**

950 A method of accessing memory on a remote system without interrupting the processing of the
951 CPU(s) on that system.

952 **Remote Memory Region (RMR)**

953 A contiguously addressable area of arbitrary size within a linear address space enabling remote
954 access, or a placeholder for such an area. An RMR in unlinked state can be linked to a section of
955 an LMR through an RMR Link operation. An RMR in linked state can be unlinked through an
956 RMR Unlink operation.

957 **RMR**

958 See **Remote Memory Region**.

- 959 **RMR Context**
- 960 An opaque identifier generated by the Implementation to represent a contiguous memory region.
961 Used by remote Consumers in RDMA operations that target this region.
- 962 **RMR Link**
- 963 The memory management operation initiated by *it_rmr_link*, which associates an RMR with a
964 section of an LMR and thereby enables remote access to that section.
- 965 **RMR Unlink**
- 966 The memory management operation initiated by *it_rmr_unlink*, which destroys the association of
967 an RMR with a section of an LMR.
- 968 **RMR Triplet**
- 969 A type used to specify a section of a Remote Memory Region. An RMR Triplet specifies an
970 RMR Handle, an address, and a length.
- 971 **RNIC**
- 972 Hardware or software implementation of RDMA/DDP over MPA/TCP/IP or RDMA/DDP
973 over SCTP/IP.
- 974 **RQ**
- 975 See **Receive Queue**.
- 976 **SE**
- 977 See **Software Event**.
- 978 **Send**
- 979 The Data Transfer Operation (DTO) that is initiated by the *it_post_send* or *it_post_sendto*
980 routine.
- 981 **Send Queue (SQ)**
- 982 An internal queue of an Endpoint on which Send, RDMA Write, RDMA Read, and RMR
983 (Un)Link Work Requests are posted.
- 984 **Service Reply**
- 985 See **UD Service Reply**.

986	Service Request
987	See UD Service Request .
988	Service Request Handle
989	An IT Object that identifies a request to translate a Connection Qualifier associated with a
990	provider of a service into a Path, Endpoint ID, and Endpoint Key that can be used to
991	communicate with that provider via the Unreliable Datagram Transport Service Type.
992	Service Type
993	A class of transport service defining basic attributes of the communication; e.g., connected or
994	unconnected, reliable or unreliable.
995	SEVD
996	See Simple Event Dispatcher .
997	Shared Receive Queue (S-RQ)
998	A Work Queue that can be shared by multiple Endpoints and to which Receive DTOs (Work
999	Requests) can be posted.
1000	Simple Event Dispatcher (SEVD)
1001	An IT Object that conceptually merges Events from one or more Event Streams. These Events
1002	can be dequeued by the Consumer directly. The Consumer is notified that Events are available
1003	through the <i>it_evd_wait</i> interface, or through higher-level Notification mechanisms, such as the
1004	Aggregation Event Dispatcher. The Simple Event Dispatcher is responsible for completion of
1005	transport-specific fetching and handshaking for the Events it collects. Each Event is delivered to
1006	the Consumer exactly once.
1007	Socket Conversion
1008	See Conversion .
1009	Software Event (SE)
1010	An Event generated for a Simple Event Dispatcher by the Consumer, as opposed to those
1011	generated by the Interface Adapter.
1012	Solicited Wait
1013	A modifier for Send DTOs submitted to an Endpoint of the Connection. It specifies that the
1014	completion of matching Receive DTOs on the remote side of the Connection generate
1015	Notification Receive DTO Completion Events.

1016	Source
1017	The Endpoint where a message originates.
1018	Spigot
1019	A host resident device that transfers data to and from the host memory to which it is attached. A
1020	Spigot is associated with a single Interface. One or more Spigots may be associated with the
1021	same Interface.
1022	SQ
1023	See Send Queue .
1024	S-RQ
1025	See Shared Receive Queue .
1026	S-RQ Low Watermark
1027	The minimum number of Receive DTOs that can be sitting idle in a Shared Receive Queue
1028	without causing an S-RQ Low Watermark Event to be generated.
1029	S-RQ Low Watermark Event
1030	An Affiliated Asynchronous Event that is generated when the Consumer has armed the S-RQ
1031	Low Watermark mechanism for an S-RQ and the total number of Receive DTOs sitting idle in
1032	the S-RQ drops below the S-RQ Low Watermark.
1033	TCP
1034	See Transmission Control Protocol .
1035	Transmission Control Protocol (TCP)
1036	The IP-based, reliable, connection-oriented transport protocol.
1037	The Open Group
1038	<i>The Open Group</i> is a vendor-neutral and technology-neutral consortium, whose vision of
1039	Boundaryless Information Flow will enable access to integrated information within and between
1040	enterprises based on open standards and global interoperability.
1041	Transport-Dependent Interface (TDI)
1042	RDMA connection management service for iWARP only, allowing the Conversion of an
1043	unconnected Endpoint of the RC type and a connected socket to a connected Endpoint. The TDI
1044	includes the <i>it_socket_convert</i> and <i>it_ep_disconnect</i> calls.

1045	Transport-Independent Interface (TII)
1046	Unified RDMA connection management service abstraction for any RC transport supported by
1047	the IT-API. The TII includes the <i>it_ep_connect</i> , <i>it_listen_create</i> , <i>it_ep_accept</i> , <i>it_reject</i> , and
1048	<i>it_ep_disconnect</i> calls.
1049	Transport Service Type
1050	See Service Type .
1051	UD
1052	See Unreliable Datagram .
1053	UD Service Reply
1054	A reply message sent via the Unreliable Datagram service in response to a UD Service Request.
1055	UD Service Request
1056	A request message sent via the Unreliable Datagram service requesting service resolution.
1057	ULP
1058	See Upper Layer Protocol .
1059	Unaffiliated Asynchronous Error
1060	An error which is associated only with an Interface Adapter, rather than a specific Endpoint,
1061	EVD, or S-RQ, and which could not be reported through an immediate error or Completion
1062	Error. Unaffiliated Asynchronous Errors are reported through an Event Stream for Unaffiliated
1063	Asynchronous Events.
1064	Unaffiliated Asynchronous Event
1065	An Event that is associated only with an Interface Adapter, rather than a specific Endpoint,
1066	EVD, S-RQ, or Work Request.
1067	Unaffiliated Event
1068	See Unaffiliated Asynchronous Event .
1069	Unbind
1070	See RMR Unlink .

1071	Unreliable Datagram (UD)
1072	A Transport Service Type in which an Endpoint may transmit and Receive single-packet messages to/from any other Endpoint that supports that Service Type. Ordering and delivery are not guaranteed, and the Receiver may drop delivered packets.
1073	
1074	
1075	Upper Layer Protocol (ULP)
1076	A protocol taking advantage of RDMA services as provided by the IT-API.
1077	VIA
1078	See Virtual Interface Architecture .
1079	Virtual Interface Architecture (VIA)
1080	One of the transports that the IT-API supports [VIA-V1.0].
1081	V-RNIC
1082	Verbs-enabled RNIC (equivalent to RI defined in [VERBS-RDMAC]).
1083	V-RNIC/IETF
1084	V-RNIC compliant with [MPA-IETF] and exposing Verbs that are slightly extended relative to [VERBS-RDMAC] for MPA Marker/CRC control according to [MPA-IETF]. A V-RNIC/IETF can be Permissive or Non-permissive [INTEROP-IETF].
1085	
1086	
1087	V-RNIC/RDMAC
1088	V-RNIC compliant with [MPA-RDMAC] and [VERBS-RDMAC].
1089	Wide RMR
1090	An RMR that can be referenced or accessed via all Endpoints in the Protection Zone of the RMR. See also Narrow RMR .
1091	
1092	Work Queue (WQ)
1093	A queue on which Work Requests are posted. Endpoints commonly have two internal Work Queues: a Send Queue and a Receive Queue. Alternatively, an Endpoint may be associated with a Shared Receive Queue, in which case the Endpoint has no internal Receive Queue. Send, RDMA Read, RDMA Write, RMR Link, and RMR Unlink Work Requests are posted to an Endpoint's Send Queue. Receive Work Requests are posted either to an Endpoint's Receive Queue or to a Shared Receive Queue.
1094	
1095	
1096	
1097	
1098	

- 1099 **Work Request (WR)**
- 1100 A request to perform an operation, posted by the Consumer to an Endpoint. Work Requests
1101 include DTOs, RMR Link, and RMR Unlink operations.
- 1102 **WQ**
- 1103 See **Work Queue**.

1104 4 Global Behavior

1105 This section describes certain general aspects of the behavior of the IT-API. Behavior described
1106 in this section is applicable to all IT-API interfaces except where noted explicitly in individual
1107 reference pages.

1108 4.1 Asynchronous versus Synchronous APIs

1109 Many IT-API operations are made available through *asynchronous APIs* comprising an
1110 *asynchronous API request call* for requesting an operation and a *Notification mechanism* for
1111 asynchronously notifying the Consumer of the operation's completion. An asynchronous API
1112 request call may return success or failure, with call success indicating only that the request has
1113 been accepted for later execution. The Consumer can be notified subsequently through an Event
1114 whether the operation completed successfully or in error. Each Event surfaced by the IT-API has
1115 an associated Event object that contains information about the disposition and status of the
1116 Event. The calls for posting Work Requests to an Endpoint form one particularly important set
1117 of asynchronous API request calls.

1118 A few IT-API operations are made available through *synchronous APIs* comprising a single API
1119 call. Such a *synchronous API call* may return success or failure, with call success indicating that
1120 the operation was completed successfully.

1121 API calls for posting Work Requests to an Endpoint (e.g., *it_post_send*) or for reaping Events
1122 (e.g., *it_evd_dequeue*) typically return without blocking. These calls are often implemented
1123 through a *fast path* to or from underlying hardware or firmware.

1124 Some synchronous API calls as well as some asynchronous API request calls (such as
1125 *it_ep_connect*) may block the caller's execution waiting for a locally generated event or
1126 condition. A few synchronous API calls (such as *it_evd_wait* and *it_get_pathinfo*) may block the
1127 caller's execution waiting for a remotely generated event. All routines that can block waiting for
1128 a remotely generated event are explicitly noted in the appropriate reference pages.

1129 4.2 Thread Safety

1130 The IT-API supports multi-threaded applications through a variety of different thread safety
1131 models. The basic issue in thread-safety is to provide mutually exclusive access to a shared
1132 resource being accessed by multiple threads executing in parallel. Within a multi-threaded
1133 application, it is common to share data resources. A common solution to provide mutual
1134 exclusion is to serialize potentially conflicting accesses into a well-ordered succession of
1135 executions. For example, if two callers make a call at the same time the results of the two
1136 executions are as if the two calls were serialized in arbitrary order. Normally ensuring mutual
1137 exclusion introduces some performance reduction, and so is only desirable when needed.

1138 To support multi-threaded applications, the IT-API defines three models of thread safety.
 1139 Briefly, these three models are described as:

- 1140 • Strongly Thread-Safe
- 1141 • Efficiently Thread-Safe
- 1142 • Not Thread-Safe

1143 While none of the three models is completely thread-safe, each provides a different degree of
 1144 thread-safety. Each of the models is appropriate for a different Consumer programming model.
 1145 The thread safety models are described in more detail below.

1146 Implementations of the IT-API may support one or more thread safety models. Which model or
 1147 models a particular Implementation supports, and how that thread safety support is
 1148 communicated to the Consumer, is beyond the scope of the IT-API. One potential mechanism
 1149 would have the Implementation vendor associate a specific thread safety model with a particular
 1150 library Implementation of the IT-API. In this scheme, different libraries would support different
 1151 thread safety models. The IT-API defines the thread safety models described in Table 1.

Model	Description
Strongly Thread Safe	<p>Nearly all routines are thread-safe. This model assumes that the Consumer wants the Implementation to provide considerable thread-safety. This thread-safety model may introduce some performance cost.</p> <p>All IT-API routines are thread-safe except for object destruction routines. The Consumer is required to ensure that an IT Object is not in use when a call is made to free or destroy that Object.</p>
Efficiently Thread Safe	<p>Some routines are thread-safe, and some are not thread-safe. This model assumes that the Consumer primarily wants the Implementation to provide high performance, and the Consumer is willing to take some responsibility for providing thread-safety.</p> <p>The Implementation provides thread-safety for routines that are not critical to the performance of the I/O code paths, and for routines where thread-safety cannot be managed by the Consumer. This includes thread-safety for routines that harvest and post Events.</p> <p>Routines that are critical to the performance of the I/O code paths, and object management routines for objects that are central to the function of the I/O code paths are not thread-safe. The Consumer is required to ensure that I/O operations are suspended when IT Object management routines are invoked on objects associated with the I/O code path. In addition, object destruction routines are not thread-safe. The Consumer is required to ensure that an IT Object is not in use when a call is made to free or destroy that Object.</p>
Not Thread-Safe	<p>No routines are thread-safe. This model assumes that the Consumer is single-threaded, or that the Consumer is managing mutual exclusion access control to IT Objects.</p>

1152 **Table 1: Thread Safety Models**

1153 For the Strongly Thread-Safe and Efficiently Thread-Safe models, the IT-API defines thread
 1154 safety on a routine-by-routine basis, and applies thread safety to IT Objects according to specific
 1155 rules.

1156 Thread-safety means that the routine:

1157 1. Provides well-defined results without imposing any restrictions on other IT-API
1158 routines called by other threads in the system

1159 2. Provides well-defined results without regard to which other IT-API routines
1160 currently have threads of execution within them

1161 Not thread-safe means that the results of the routine can possibly be *not* well-defined if another
1162 in-progress not thread-safe routine is called with the same primary (i.e., first) call argument, and
1163 none of the calls is an object destructor routine. (For *it_rmr_link* the *rmr_handle*, *lmr_handle*,
1164 and *ep_handle* arguments must be treated as primary call arguments for thread-safety purposes.
1165 For *it_rmr_unlink* the *rmr_handle* and *ep_handle* arguments, and the *lmr_handle* of the LMR to
1166 which the RMR is linked must be treated as primary call arguments for thread-safety purposes.)

1167 This definition of thread-safe and not thread-safe routines allows simultaneous execution of not
1168 thread-safe calls involving different instances of an IT Object (e.g., two different EPs) or
1169 involving IT Objects that are related (e.g., an EVD and an EP), so long as the primary call
1170 argument is not the same for the two routines.

1171 For the not-thread-safe model, the Implementation does not provide thread-safety for any data
1172 structures whatsoever.

1173 The IT-API applies these thread safety models on a routine-by-routine basis. IT-API routines can
1174 be classified into five groups according to their basic function. These groups determine their
1175 thread-safety under each thread safety model:

- 1176 • Non-performance critical routines: These routines create objects and manage and query
1177 the state of objects that do interact with the I/O code paths.
- 1178 • Event harvesting and posting routines: These routines retrieve Events from the
1179 Implementation, and invoke software Events.
- 1180 • Performance critical routines: These routines invoke Data Transfer Operations and RMR
1181 Operations.
- 1182 • Object management routines: These routines modify and query the state of objects that
1183 interact with the I/O code paths.
- 1184 • Object destructor routines: These routines free and/or destroy IT Objects.

1185 The thread-safety of each IT-API routine is given in Table 2.

Non-Performance Critical Routines	Strongly Thread-Safe Model	Efficiently Thread-Safe Model	Not Thread-Safe Model
<i>it_address_handle_create</i> <i>it_convert_net_addr</i> <i>it_ep_rc_create</i> <i>it_ep_ud_create</i> <i>it_evd_create</i> <i>it_get_handle_type</i> <i>it_get_pathinfo</i> <i>it_hton64, it_ntoh64</i> <i>it_ia_create</i> <i>it_ia_query</i> <i>it_interface_list</i> <i>it_listen_create</i> <i>it_listen_query</i> <i>it_lmr_create</i> <i>it_make_rdma_addr_absolute</i> <i>it_make_rdma_addr_relative</i> <i>it_pz_create</i> <i>it_pz_query</i> <i>it_rmr_create</i> <i>it_srq_create</i> <i>it_ud_service_request_handle_create</i>	Thread-safe		Not thread-safe
Event Harvesting and Posting Routines	Strongly Thread-Safe Model	Efficiently Thread-Safe Model	Not Thread-Safe Model
<i>it_evd_dequeue</i> <i>it_evd_post_se</i> <i>it_evd_wait</i>	Thread-safe		Not thread-safe
Performance Critical Routines	Strongly Thread-Safe Model	Efficiently Thread-Safe Model	Not Thread-Safe Model
<i>it_post_rdma_read</i> <i>it_post_rdma_read_to_rmr</i> <i>it_post_rdma_write</i> <i>it_post_rcv</i> <i>it_post_rcvfrom</i> <i>it_post_send</i> <i>it_post_sendto</i> <i>it_rmr_link</i> <i>it_rmr_unlink</i> <i>it_ud_service_reply</i> <i>it_ud_service_request</i>	Thread-safe	Not thread-safe	

Object Management Routines	Strongly Thread-Safe Model	Efficiently Thread-Safe Model	Not Thread-Safe Model
<i>it_address_handle_modify</i> <i>it_address_handle_query</i> <i>it_ep_connect</i> <i>it_ep_modify</i> <i>it_ep_query</i> <i>it_ep_reset</i> <i>it_evd_modify</i> <i>it_evd_query</i> <i>it_get_consumer_context</i> <i>it_lmr_modify</i> <i>it_lmr_query</i> <i>it_lmr_sync_rdma_read</i> <i>it_lmr_sync_rdma_write</i> <i>it_rmr_query</i> <i>it_set_consumer_context</i> <i>it_socket_convert</i> <i>it_srq_modify</i> <i>it_srq_query</i> <i>it_ud_service_request_handle_query</i>	Thread-safe	Not thread-safe	
Object Destructor Routines	Strongly Thread-Safe Model	Efficiently Thread-Safe Model	Not Thread-Safe Model
<i>it_address_handle_free</i> <i>it_ep_accept</i> <i>it_ep_disconnect</i> <i>it_ep_free</i> <i>it_evd_free</i> <i>it_handoff</i> <i>it_ia_free</i> <i>it_ia_info_free</i> <i>it_listen_free</i> <i>it_lmr_free</i> <i>it_reject</i> <i>it_ud_service_request_handle_free</i> <i>it_pz_free</i> <i>it_rmr_free</i> <i>it_srq_free</i>	Not thread-safe		

Table 2: Thread-Safety Models Applied to IT-APIs

1187

1188 **4.3 Signal Handlers**

1189 IT-API interfaces are not required to be safely executable from within a signal handler
1190 invocation.

1191 **4.4 Fork Semantics**

1192 Use of the POSIX *fork* family of calls is supported within the IT-API with the following
1193 semantics. After a fork call, the parent process' references to IT Objects are unchanged, and it
1194 may continue to use the references as it had before the *fork* call.

1195 The child process' IT Object references are invalid following the *fork* call (with one exception
1196 for file descriptors discussed below).

1197 The one exception to this behavior for the child is for file descriptors that were associated with
1198 EVDs before the *fork* call occurred. The Implementation supports the child's use of the *close* call
1199 to close the file descriptor.

1200 **4.5 Exec Semantics**

1201 The process' IT Object references are invalid following the POSIX *exec* family of calls.

1202 **4.6 Exit Semantics**

1203 Following an implicit or explicit call to the POSIX *exit*, all IT Objects associated with the
1204 process are destroyed and all references to them are invalid.

1205 **4.7 Error Handling**

1206 Error Notification is provided to Consumers of the IT-API in two ways:

- 1207 1. As error return values to interface calls
- 1208 2. As Events containing error status information for the earlier request

1209 In general, interface calls return an immediate error when a call argument is invalid or
1210 incompatible with a condition relevant to the request. However, some errors of this type are
1211 determined by the transport layer stack executing below the IT-API, and in this case the
1212 Consumer is notified of call parameter-related errors through an Event.

1213 For Work Requests posted to an RC-type Endpoint in the `IT_EP_STATE_CONNECTED` state,
1214 a completion status other than `IT_DTO_SUCCESS` will break the Connection by moving the
1215 Endpoint to the `IT_EP_STATE_NONOPERATIONAL` state and deliver an
1216 `IT_CM_MSG_CONN_BROKEN_EVENT` Event to the Connect EVD of the Endpoint. Once
1217 the Connection is broken, all outstanding and in-progress operations on the Connection will
1218 complete with an error status.

1219 Any posting to an Endpoint that is in the `IT_EP_STATE_NONOPERATIONAL` state will be
1220 flushed with completion status (*it_dto_status_t*) set to `IT_DTO_ERR_FLUSHED`.

1221 **4.8 IT Handle Management**

1222 IT-API interfaces that create an IT Object return an opaque type reference Handle that the
1223 Consumer can use in subsequent IT-API calls. It is the Consumer's responsibility to track these
1224 Handles, and use them appropriately. The Implementation will make its best effort to detect
1225 improper use of Handles by the Consumer and will return an invalid Handle immediate error
1226 whenever possible. However, it may not always be possible for the Implementation to detect
1227 improper use of a Handle, and improper use may manifest for the Consumer in Completion
1228 Errors for Work Requests, a broken Connection, data corruption, or other more severe errors.

1229 **4.9 Output Parameters**

1230 Many of the routines in the IT-API take an output parameter pointer that is used by the
1231 Implementation to copy information into an address specified by the Consumer. An example of
1232 such a routine is *it_evd_query*, which copies the state associated with an EVD into the data
1233 structure pointed to by the *params* pointer passed by the Consumer. The Implementation will not
1234 attempt to detect invalid pointers for output parameters, and passing such a pointer may result in
1235 severe errors such as data corruption or program termination.

1236 5 Connection Management

1237 5.1 Overview

1238 The IT-API provides RDMA services for several underlying RDMA transports, including
1239 InfiniBand and iWARP². The specifics of these RDMA transports as well as different
1240 application requirements led to the design of two different Connection management interfaces
1241 for Reliable Connected (RC) RDMA transports, as introduced below.

1242 The Transport-Independent Interface (TII) provides a unified RDMA Connection management
1243 service abstraction for any RC transport supported by the IT-API and is the legacy Connection
1244 management API from IT-API Version 1.0. The TII includes the *it_ep_connect*, *it_listen_create*,
1245 *it_ep_accept*, *it_reject*, and *it_ep_disconnect* calls³. The TII allows Consumers to negotiate the
1246 *Inbound RDMA Read Queue Depth* (IRD) and *Outbound RDMA Read Queue Depth* (ORD)
1247 parameters and to exchange Private Data during Connection establishment. Consumers are
1248 expected to preconfigure IRD and ORD prior to calling *it_ep_connect* or *it_ep_accept* by setting
1249 the corresponding Endpoint attributes *rdma_read_ird* and *rdma_read_ord*, respectively. From
1250 the Consumer's viewpoint, the RDMA service is available immediately after Connection
1251 establishment. No Lower Layer Protocol (LLP) handle is exposed to the Consumer. The usage of
1252 the TII is described in *Transport-Independent Interface*. Implementation aspects for the TII on
1253 iWARP are described in Appendix B.

1254 The Transport-Dependent Interface (TDI) is available for iWARP only and allows the
1255 Conversion of an unconnected Endpoint of the RC type and a connected socket into a connected
1256 Endpoint. The TDI includes the *it_socket_convert* and *it_ep_disconnect* calls. It enables
1257 Consumers to exchange an arbitrary (but non-zero) amount of streaming-mode data via a TCP
1258 socket (and possibly via an SCTP socket in the future) prior to performing a Conversion, which
1259 is a requirement for several iWARP-specific ULPs. This prior exchange of streaming-mode data
1260 occurs outside the IT-API. In contrast to Connection establishment with the TII, the
1261 *it_socket_convert* call neither supports the negotiation of IRD and ORD parameters, nor the
1262 exchange of Private Data. The main reason for this simplification is that the peers have an
1263 opportunity to negotiate IRD and ORD, to select the corresponding Endpoint attributes
1264 *rdma_read_ird* and *rdma_read_ord*, and to exchange Private Data prior to Conversion;
1265 moreover, they have an opportunity to modify ORD after the Conversion⁴; i.e., while already in
1266 RDMA mode. The TDI also allows the Consumer more detailed control of the Connection
1267 establishment process, including whether or not the MPA Request/Reply startup sequence is
1268 used. Potential uses for such control include interoperating with remote peers that are not using

² Version 2.0 of the IT-API supports TCP-based iWARP. SCTP support may be added in a later version.

³ The TII supports both two-way and three-way Connection establishment. Only two-way Connection establishment applies to all transports. The three-way Connection establishment model is useful on the InfiniBand Transport where the Initiating Consumer wishes to examine and adjust to the RDMA Responder IRD/ORD values.

⁴ There is no guaranteed opportunity to modify IRD after the Conversion because this is an optional capability according to [VERBS-RDMAC].

1269 the IT-API, interoperating with RNICs that have dissimilar feature sets, etc. The usage of the
1270 TDI is described in *Transport-Dependent Interface (iWARP-only)*. Implementation aspects for
1271 the TDI are described in Appendix B.

1272 5.1.1 IRD/ORD Negotiation

1273 Negotiation of IRD/ORD between RDMA Initiator and RDMA Responder should be done by a
1274 Consumer intending to use RDMA Read operations in their programming model. The Inbound
1275 RDMA Read Queue Depth, IRD, represents the maximum number of outstanding inbound
1276 RDMA Read operations supported for an Endpoint. The Outbound RDMA Read Queue Depth,
1277 ORD, represents the maximum number of emitted outbound RDMA Read operations the
1278 Endpoint may have outstanding.

1279 IRD and ORD must be negotiated such that each side of a Connection uses compatible values. If
1280 ORD on one side of a Connection exceeds IRD on the other, there is a potential for the
1281 Connection to be terminated if more RDMA Reads are issued than the advertised IRD can
1282 tolerate⁵.

1283 The IT-API supports two models of IRD/ORD negotiation: API-supported and Consumer-
1284 defined (i.e., via their own ULP in Private Data). The former, API-supported model is supported
1285 only by the TII. The latter approach, Consumer-defined, is needed for the TDI and optionally
1286 used by the Consumer for the TII (by disabling the API-supported mechanism).⁶ For the TII with
1287 IRD/ORD use suppressed, the Consumer must define their own scheme to convey the values.

1288 With either model, the RDMA Initiator should create and configure an Endpoint with desired
1289 IRD/ORD values prior to use. At the Responder, IRD/ORD values are also chosen per Endpoint
1290 with potentially differing criteria than at the Initiator.

1291 If the local ORD exceeds the remote IRD, then the local ORD must be reduced. If the local IRD
1292 exceeds the remote ORD, then resources can be saved by reducing the local IRD; whether IRD
1293 reduction is possible depends on the capabilities of the underlying device (see *it_ia_info_t*).

1294 The RDMA Initiator's IRD/ORD values (IRD_i/ORD_i) must be conveyed to and examined by the
1295 RDMA Responder.

1296 The RDMA Responder's IRD (IRD_r) can be reduced from its preset value to a value not
1297 exceeding ORD_i.

1298 The RDMA Responder's ORD (ORD_r) should be set to the minimum of its preset value and
1299 IRD_i.

1300 Finally, the RDMA Initiator adjusts ORD_i and optionally IRD_i after receiving the RDMA
1301 Responder's IRD_r and ORD_r.

⁵ For InfiniBand, such an error is a "Too many RDMA Read" error and manifests in the InfiniBand verbs as a Local Access Violation Work Queue Error. For iWARP, such an error is manifested in the verbs as an "Invalid MSN – too many RDMA Read Request Messages in process" terminate error (Terminate Code - 0x1203).

⁶ Disabling IRD/ORD use in the TII is accomplished on the active side through use of the flag `IT_CONNECT_SUPPRESS_IRD_ORD` (*it_cn_est_flags_t*) and on the passive side through use of the flag `IT_LISTEN_SUPPRESS_IRD_ORD` (*it_listen_flags_t*) settings.

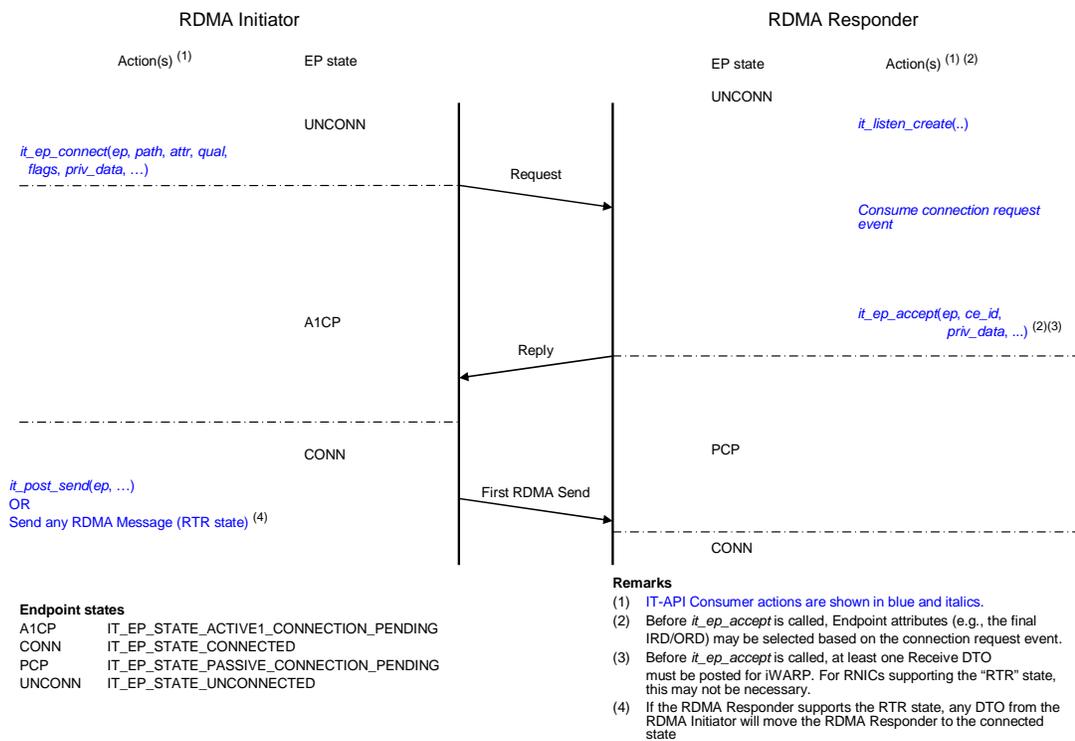
1302 **5.2 Transport-Independent Interface**

1303 **5.2.1 Overview**

1304 For the Transport-Independent Interface (TII), the Consumer calling *it_ep_connect* and the
 1305 corresponding side of the Connection is referred to as the *RDMA Initiator*, while the Consumer
 1306 calling *it_listen_create* followed by *it_ep_accept* (or *it_reject*) and the corresponding side of the
 1307 Connection is referred to as the *RDMA Responder*.

1308 The Endpoint finite-state machine uses the active (passive) Endpoint states for the RDMA
 1309 Initiator (Responder). See the *it_ep_state_t* reference page.

1310 Connection establishment for the TII is illustrated in Figure 1.



1311
 1312 **Figure 1: Connection Establishment with the TII**

1313 The RDMA Responder uses *it_listen_create* to create a Listen Point, which causes the (API)
 1314 Implementation to prepare for receiving an RDMA Connection Request. The RDMA Initiating
 1315 Consumer uses *it_ep_connect* to connect an Endpoint of the RC type, which causes the
 1316 Implementation to send an RDMA Connection Request.

1317 The Implementation of the RDMA Initiator generates and sends an RDMA Connection Request
1318 containing the local Endpoint IRD and ORD parameters⁷ and, amongst other things, the Private
1319 Data provided by the Consumer with *it_ep_connect*. After receiving a valid RDMA Connection
1320 reply from the Responding side, the Implementation generates a Connection established event
1321 for the RDMA Initiator providing the remote Consumer's Private Data.

1322 The Responding side Listen Point expects an incoming RDMA Connection Request. Upon
1323 receiving a valid request, it generates a Connection Request event for the Consumer containing
1324 the remote Endpoint's IRD and ORD parameters⁸ as well as the remote Consumer's Private
1325 Data. The Connection Request event is identified by the Connection Establishment ID
1326 (*cn_est_id*). The Consumer on the RDMA Responder side now invokes either *it_ep_accept* or
1327 *it_reject*.

1328 Calling *it_ep_accept* causes the Implementation to associate the Connection Establishment ID
1329 with an Endpoint. It then causes the Implementation to convey an RDMA Connection reply
1330 including the Private Data provided by the Consumer to the Initiating side. Finally, on receipt of
1331 a first RDMA message (see next paragraph), the Implementation generates a Connection
1332 established event for the Consumer and transitions the Endpoint into the
1333 IT_EP_STATE_CONNECTED state.

1334 5.2.2 ULP Constraints for iWARP

1335 Consumers using iWARP AV-RNICs must satisfy the following additional ULP constraints:

- 1336 • The RDMA Responder must post at least one Receive DTO prior to the calling
1337 *it_ep_accept*.
- 1338 • After reaping the Connection established event, the RDMA Initiator must post a Send
1339 DTO so that a Connection established Event will be generated for the RDMA Responder.

1340 In order to minimize such transport-dependent ULP constraints, the IT-API offers an alternative
1341 for RNICs that implement an extended QP state machine (i.e., they define an "RTR" state as
1342 footnoted in the above figure – see Appendix A for details). The Consumer may determine
1343 whether the underlying RNIC supports the extended QP state machine via the
1344 *extended_iwarp_qp_states* attribute found in the *it_ia_info_t* structure retrieved from
1345 *it_ia_query*.

1346 Consumers using iWARP AV-RNICs with "RTR" state support must satisfy the following ULP
1347 constraint only:

- 1348 • After reaping the Connection established event, the RDMA Initiator must post an RDMA
1349 or Send DTO so that a Connection established Event will be generated for the RDMA
1350 Responder.

1351 5.2.3 API-Supported IRD/ORD Negotiation

1352 Prior to calling *it_ep_connect*, the RDMA Initiator preconfigures the Endpoint to be used with
1353 IRD/ORD endpoint attributes (see *it_ep_attributes_t*), referred to as IRDi/ORDi.

⁷ If supported by the underlying transport and also if not disabled by the Consumer.

⁸ If supported by the underlying transport and also if not disabled by the Consumer.

1354 When *it_ep_connect* is called, IRDi/ORDi are filled into the Connection Request message (for
1355 InfiniBand, the CM REQ; for iWARP the MPA Request's IRD/ORD Header found in Private
1356 Data).

1357 When the Connection Request is received by the IT-API Listen Point (RDMA Responder side),
1358 a Connection Request event is generated for the ULP reporting the peer's IRDi/ORDi in the
1359 *rdma_read_ird* and *rdma_read_ord* fields respectively (see *it_cm_req_events*).

1360 The RDMA Responder now updates its IRDr/ORDr values, preferably through the algorithm:

```
1361 IRDr := MIN(IRDr, ORDi)  
1362 ORDr := MIN(ORDr, IRDi)
```

1363 and updates its Endpoint (which is still in the IT_EP_STATE_UNCONNECTED state) with the
1364 new IRDr/ORDr via *it_ep_modify*. Modification of IRD/ORD depends on the capabilities of the
1365 underlying device. All devices will support decrease of ORD. The *it_ia_info_t* attributes,
1366 *rdma_read_ird_modifiable* and *rdma_read_ord_increasable*, define any additional permitted
1367 operations on IRD/ORD for the RNIC.

1368 When the RDMA Responder calls *it_ep_accept* for the previous Connection Request,
1369 IRDr/ORDr are filled into the Connection Reply (for InfiniBand, the CM REP; for iWARP, the
1370 MPA Reply's IRD/ORD Header).

1371 When the Connection Reply is received by the RDMA Initiator side implementation, a
1372 Connection established event is generated reporting the peer's IRDr/ORDr.

1373 The RDMA Initiator may now update its IRDi/ORDi values, preferably through the algorithm:

```
1374 IRDi := MIN(IRDi, ORDr)  
1375 ORDi := MIN(ORDi, IRDr)
```

1376 and update its EP attributes *rdma_read_ird* and *rdma_read_ord* accordingly, as permitted by the
1377 underlying device.

1378 The Connection established event generated for the RDMA Responder contains *rdma_read_ird*
1379 and *rdma_read_ord* fields that represent the initial IRD/ORD sent by the Initiator. Since the
1380 Initiator optionally may have subsequently altered the values, the Consumer cannot rely on their
1381 contents.

1382 **5.2.4 Transport-Dependent Attributes**

1383 **5.2.4.1 iWARP**

1384 **5.2.4.1.1 Connection Qualifiers**

1385 Only the IT_IANA_PORT or IT_IANA_LR_PORT Connection Qualifiers are supported for the
1386 iWARP Transport.

1387 **5.2.4.1.2 Service Types**

1388 Only the IT_RC_SERVICE service type is supported for the iWARP Transport.

1389 **5.2.4.1.3 Handshake**

1390 The IT-API supports only two-way Connection establishment for iWARP.

1391 **5.3 Transport-Dependent Interface (iWARP-Only)**

1392 **5.3.1 Overview**

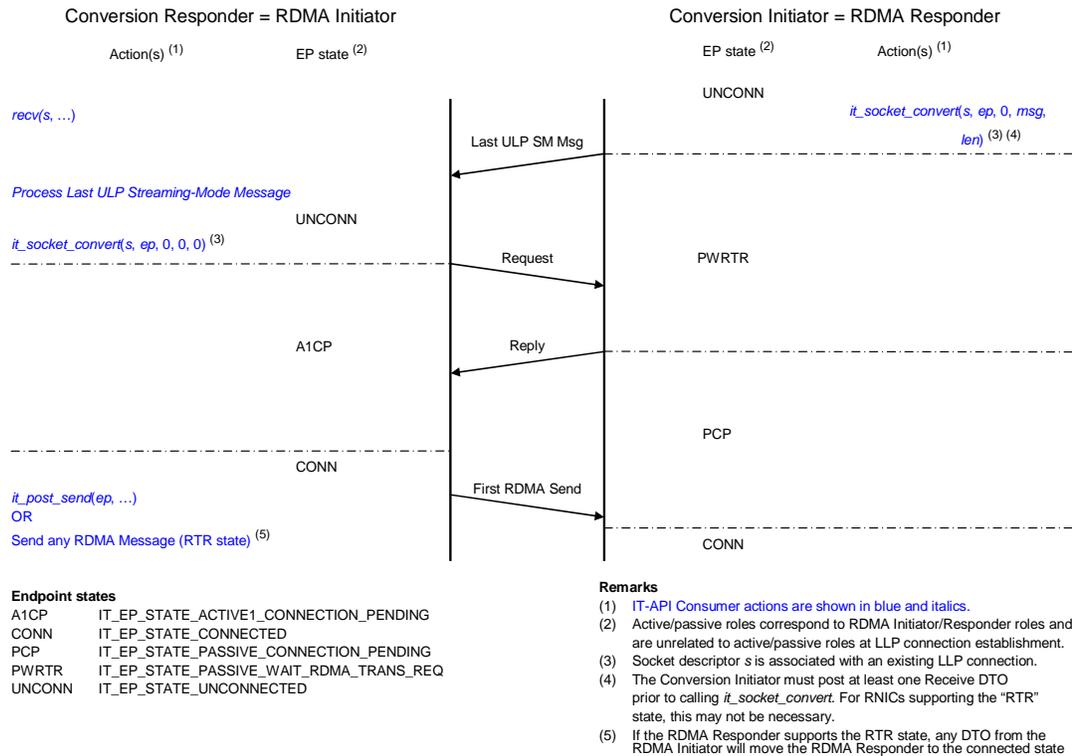
1393 The *Transport-Dependent Interface* (TDI) is defined to support iWARP-specific ULPs.

1394 For the *TDI*, the Consumer calling *it_socket_convert* with a Last ULP Streaming-Mode Message
1395 of length greater than zero is referred to as the *Conversion Initiator*, while the Consumer calling
1396 *it_socket_convert* with a Streaming Mode Message length of zero is referred to as the
1397 *Conversion Responder*.

1398 As for the *TII*, the Endpoint finite-state machine uses the active (passive) Endpoint states for the
1399 RDMA Initiator (Responder). In order to avoid an inconsistency in the meaning of Endpoint
1400 states between *TII* and *TDI*, the *TDI* uses the convention that the Conversion Initiator and the
1401 corresponding side of the Connection is in the RDMA Responder role, while the Conversion
1402 Responder and the corresponding side of the Connection is in the RDMA Initiator role. See the
1403 *it_socket_convert* reference page for the finite-state machine diagrams.

1404

The Conversion process is illustrated below.



1405

1406

Figure 2: Conversion Process with MPA Startup (TDI)

1407

1408

1409

1410

1411

1412

1413

The Conversion Initiator invokes *it_socket_convert*, passing an unconnected Endpoint of the RC type, a socket handle corresponding to an established LLP Connection, a flags argument, and a Last ULP Streaming-Mode Message of length greater than zero, indicating the Conversion Initiator role. Unless the *flags* argument includes *IT_SC_NO_REQ_REP*, an RDMA transition handshake (i.e., MPA Startup for MPA/TCP) shall be performed. The (API) Implementation sends the Last ULP Streaming-Mode Message over the Connection associated with the socket handle and expects to receive an MPA Request.

1414

1415

1416

1417

1418

1419

Upon receiving a valid MPA Request message, the Implementation generates an MPA Reply message without any Private Data. Finally, upon detection of the first iWARP payload, the Implementation generates a Connection established event for the Conversion Initiator providing neither IRD/ORD parameters nor any Private Data from the remote Consumer, and transitions the Endpoint into the *IT_EP_STATE_CONNECTED* state. For the TDI, IRD/ORD fields in the Connection established event will be zero and the Private Data present bit shall be clear.

1420

1421

1422

1423

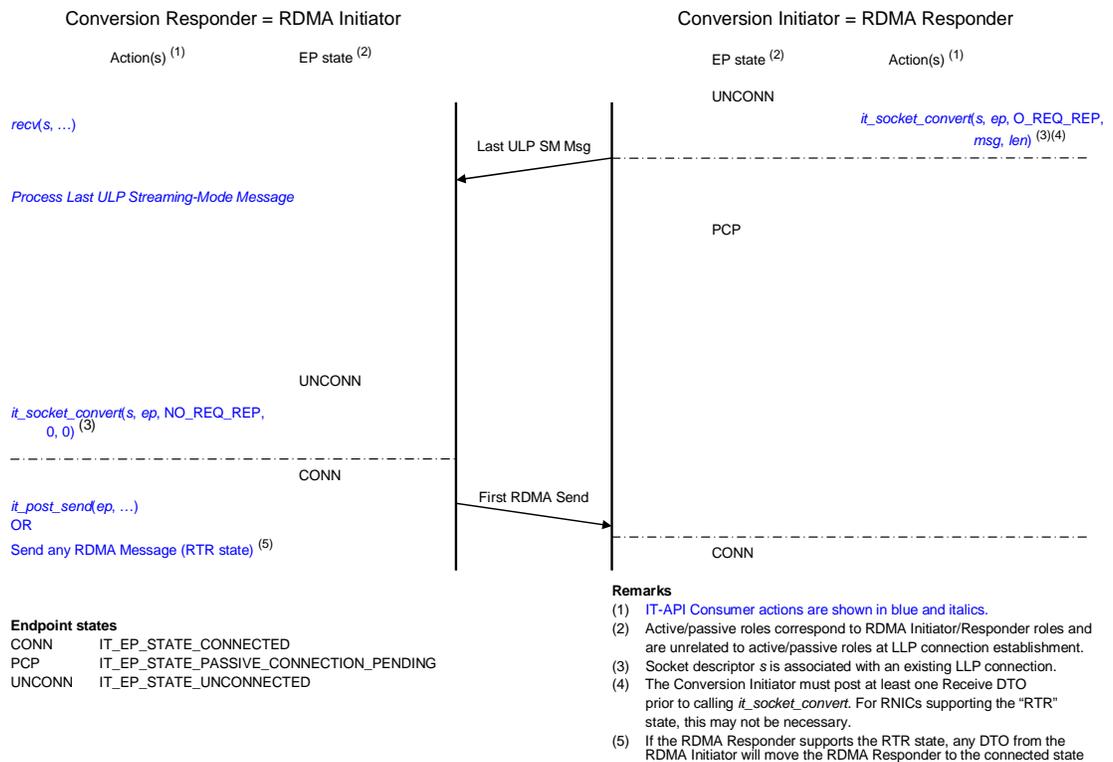
1424

The Conversion Responder expects to receive the Last ULP Streaming-Mode Message. Upon receiving that message, it invokes *it_socket_convert*, passing an unconnected Endpoint of the RC type, a socket handle corresponding to an established LLP Connection, a *flags* argument of zero, and a Streaming Mode Message length of zero indicating the Conversion Responder role. The Implementation now generates an MPA Request message without any Private Data, sends it over

1425 the Connection associated with the socket handle, and expects to receive an MPA Reply
 1426 message. After receiving a valid MPA Reply message with the MPA Reject bit not set, the
 1427 Implementation generates a Connection established event for the Consumer with IRD/ORD set
 1428 to zero and the Private Data present bit clear, and transitions the Endpoint into the
 1429 IT_EP_STATE_CONNECTED state.

1430 If the Conversion Responder receives an MPA Reply message with the MPA Reject bit set, an
 1431 IT_CM_MSG_CONN_PEER_REJECT_EVENT will be generated locally and the Endpoint will
 1432 be transitioned to the IT_EP_STATE_NONOPERATIONAL state.

1433 In order to provide interoperability with remote devices in the class AV-RNIC/RDMAC without
 1434 MPA Startup, the TDI allows suppressing the RDMA transition handshake in the Conversion
 1435 process by setting the flag IT_SC_NO_REQ_REP in the *it_socket_convert* call. The Conversion
 1436 process with MPA Startup suppression in case of an RNIC without RTR state is shown below.



1437
 1438 **Figure 3: Conversion Process with MPA Startup Suppression (TDI)**

1439 **5.3.2 IRD/ORD Negotiation**

1440 For the TDI (*it_socket_convert*), IRD/ORD must be negotiated either before or after Socket
 1441 Conversion.

<i>it_address_handle_create</i>	Create an Address Handle.
<i>it_address_handle_free</i>	Free an Address Handle.
<i>it_address_handle_modify</i>	Modify an Address Handle.
<i>it_address_handle_query</i>	Query an Address Handle.
<i>it_convert_net_addr</i>	Convert a Network Address from one format to another.
<i>it_ep_accept</i>	Accept an incoming Connection establishment request or reply.
<i>it_ep_connect</i>	Initiate an Endpoint Connection establishment request.
<i>it_ep_disconnect</i>	Disconnect an existing Endpoint-to-Endpoint Connection.
<i>it_ep_free</i>	Destroy an RC or UD Endpoint.
<i>it_ep_modify</i>	Modify parameters of an existing Endpoint.
<i>it_ep_query</i>	Query an existing Endpoint.
<i>it_ep_rc_create</i>	Create an Endpoint for Reliable Connection.
<i>it_ep_reset</i>	Reset a Reliable Connected Endpoint into the initial state.
<i>it_ep_ud_create</i>	Create an Endpoint for Unreliable Datagram.
<i>it_evd_create</i>	Create Simple or Aggregate Event Dispatcher.
<i>it_evd_dequeue</i>	Dequeue Events from Event Dispatcher.
<i>it_evd_free</i>	Destroy an Event Dispatcher.
<i>it_evd_modify</i>	Modify an existing Event Dispatcher.
<i>it_evd_post_se</i>	Post Software Event on Simple Event Dispatcher.
<i>it_evd_query</i>	Query an existing Simple or Aggregate Event Dispatcher.
<i>it_evd_wait</i>	Wait for Events on Event Dispatcher.
<i>it_get_consumer_context</i>	Get Consumer Context associated with an IT Object Handle.
<i>it_get_handle_type</i>	Return the Handle type value associated with an IT Object Handle.
<i>it_get_pathinfo</i>	Retrieve Paths used to communicate with a remote Network Address.
<i>it_handoff</i>	Hand-off an incoming Connection Request to another Connection Qualifier.
<i>it_hton64, it_ntoh64</i>	Convert 64-bit integers between host and network byte order.
<i>it_ia_create</i>	Create an Interface Adapter.

<i>it_ia_free</i>	Destroy an Interface Adapter Handle.
<i>it_ia_info_free</i>	Free an <i>it_ia_info_t</i> structure that was returned by <i>it_ia_query</i> .
<i>it_ia_query</i>	Retrieve attributes of given Interface Adapter and its Spigots.
<i>it_interface_list</i>	Retrieve information about the available Interface Adapters.
<i>it_listen_create</i>	Listen for an incoming Connection Request for a Connection Qualifier.
<i>it_listen_free</i>	Destroy a listening point for a Connection Qualifier.
<i>it_listen_query</i>	Query parameters associated with a listening point.
<i>it_lmr_create</i>	Create a Local Memory Region and register with an Interface Adapter.
<i>it_lmr_free</i>	Destroy a Local Memory Region.
<i>it_lmr_modify</i>	Modify selected attributes of a Local Memory Region.
<i>it_lmr_query</i>	Get attributes of a Local Memory Region.
<i>it_lmr_sync_rdma_read</i>	Make memory changes visible to an incoming RDMA Read operation.
<i>it_lmr_sync_rdma_write</i>	Make effects of an incoming RDMA Write operation visible.
<i>it_make_rdma_addr_absolute</i>	Make a platform-independent RDMA address for Absolute Addressing.
<i>it_make_rdma_addr_relative</i>	Make a platform-independent RDMA address for Relative Addressing.
<i>it_post_rdma_read</i>	Post an RDMA Read DTO to an RC-type Endpoint.
<i>it_post_rdma_read_to_rmr</i>	Post an RDMA Read DTO to an RC-type Endpoint.
<i>it_post_rdma_write</i>	Post an RDMA Write DTO to an RC-type Endpoint.
<i>it_post_recv</i>	Post a Receive DTO to an RC-type Endpoint.
<i>it_post_recvfrom</i>	Post a Receive DTO to a datagram Endpoint.
<i>it_post_send</i>	Post a Send DTO to an RC-type Endpoint.
<i>it_post_sendto</i>	Post a Send DTO to a datagram Endpoint.
<i>it_pz_create</i>	Create a new Protection Zone.
<i>it_pz_free</i>	Destroy a Protection Zone.
<i>it_pz_query</i>	Get attributes of a Protection Zone.
<i>it_reject</i>	Reject an incoming Connection establishment request or reply.
<i>it_rmr_link</i>	Post request to Link a Remote Memory Region to a memory range.
<i>it_rmr_create</i>	Create a Remote Memory Region (RMR).
<i>it_rmr_free</i>	Destroy a Remote Memory Region.
<i>it_rmr_query</i>	Get attributes of a Remote Memory Region.

<i>it_rmr_unlink</i>	Post operation to Unlink a Remote Memory Region from its memory range.
<i>it_set_consumer_context</i>	Associate a Consumer Context with an IT Object Handle.
<i>it_socket_convert</i>	Convert a connected socket into a connected IT-API Endpoint.
<i>it_srq_create</i>	Create a Shared Receive Queue (S-RQ).
<i>it_srq_free</i>	Destroy a Shared Receive Queue.
<i>it_srq_modify</i>	Modify selected attributes of a Shared Receive Queue.
<i>it_srq_query</i>	Get attributes of a Shared Receive Queue.
<i>it_ud_service_reply</i>	Return UD communication information.
<i>it_ud_service_request</i>	Request the recipient to return UD communication information.
<i>it_ud_service_request_handle_create</i>	Create a UD Service Request Handle.
<i>it_ud_service_request_handle_free</i>	Free a previously created <i>it_ud_svc_req_handle_t</i> .
<i>it_ud_service_request_handle_query</i>	Return <i>it_ud_svc_req_handle_t</i> information.

1443

1444

it_address_handle_create()

1445

1446 NAME

1447 `it_address_handle_create` – create an Address Handle

1448 SYNOPSIS

```
1449 #include <it_api.h>
1450
1451 it_status_t it_address_handle_create(
1452     IN          it_pz_handle_t      pz_handle,
1453     IN const    it_path_t           *destination_path,
1454     IN          it_ah_flags_t       ah_flags,
1455     OUT         it_addr_handle_t     *addr_handle
1456 );
1457
1458 typedef enum {
1459     IT_AH_PATH_COMPLETE = 0x1
1460 } it_ah_flags_t;
```

1461 APPLICABILITY

1462 `it_address_handle_create` is applicable only to the UD service type.

1463 DESCRIPTION

1464 *pz_handle* Handle for the Protection Zone to be associated with the created Address
1465 Handle. This implicitly identifies the Interface Adapter that the Address
1466 Handle will be associated with.

1467 *destination_path* The Path to use to create the Address Handle.

1468 *ah_flags* The bitwise OR of the set of operation modifier flags specified below.

1469 *addr_handle* Returned datagram Address Handle.

1470 `it_address_handle_create` creates an Address Handle, which is used when performing a Send
1471 DTO on an Unreliable Datagram Endpoint.

1472 The Protection Zone to associate with the newly created Address Handle is specified by
1473 *pz_handle*. An Address Handle can only be used to post a Send DTO on an Unreliable Datagram
1474 Endpoint that has a matching Protection Zone.

1475 The Source and Destination address information necessary to create the Address Handle are
1476 specified in the *destination_path* parameter. The Path can either be completely or incompletely
1477 specified. A completely specified Path is one that contains all the necessary information to create
1478 the Address Handle without the Implementation needing to consult a database of Path records.
1479 An incompletely specified Path does not contain enough information to create the Address
1480 Handle directly, but does contain enough information that the Implementation can determine the
1481 rest of the information needed by consulting a database of Path records. The Consumer should
1482 set the IT_AH_PATH_COMPLETE bit in *ah_flags* if the Path is completely specified, or clear it
1483 if it is incompletely specified.

1484 A completely specified Path that the Consumer can use to access a given remote network
1485 Endpoint can be obtained using the *it_get_pathinfo* routine. A Path returned from

1486
1487
1488

it_get_pathinfo can be used without modification by *it_address_handle_create*. If the Consumer wishes to have full control over the Path that datagrams sent using the created Address Handle will take, they should furnish a completely specified Path.

1489
1490
1491
1492

An incompletely specified Path is obtained from the Completion Event for a Receive operation on a datagram Endpoint. (See *it_dto_events* for details.) If an incompletely specified Path is supplied to *it_address_handle_create*, the routine will automatically choose the unspecified components of the Path required in order to reach the intended Destination.

1493
1494
1495
1496
1497
1498
1499
1500

The Consumer may also directly format the *destination_path* if they so desire. The *destination_path* actually contains more information than is necessary to create an Address Handle. The members of the *it_path_t* structure that are pertinent for creating an Address Handle using a completely specified Path are listed in the table below. For each member, whether the member is needed for an incompletely specified Path and the input modifier for the Infiniband “Create Address Handle” verb that the member corresponds to are also identified. For a detailed explanation of the semantics associated with each input modifier, see “Create Address Handle” in Chapter 11 of the Infiniband specification.

it_path_t Member	Needed for Incomplete Path?	IB Create Address Handle Input Modifier
<i>ib.sl</i>	Yes	Service level.
<i>ib.remote_port_lid</i>	Yes	Destination LID.
<i>ib.flow_label</i>	No	Flow label.
<i>ib.hop_limit</i>	No	Hop limit.
<i>ib.traffic_class</i>	No	Traffic class.
<i>ib.local_port_gid</i>	No	Source GID index. (The Implementation uses the <i>local_port_gid</i> to determine the appropriate Source GID index.)
<i>ib.remote_port_gid</i>	No	Destination’s GID.
<i>ib.packet_rate</i>	No	Maximum Static Rate.
<i>ib.local_port_lid</i>	No	Source Path Bits. (The low order bits of the supplied <i>local_port_lid</i> are used as the Source Path Bits.)
<i>ib.subnet_local</i>	No	Send InfiniBand Global Routing Header flag.
<i>spigot_id</i>	No	Physical Port.

1501
1502
1503
1504
1505
1506
1507
1508

If the Consumer chooses to directly format the Path, it is possible that the Implementation will decide that the resulting Path is one that the Consumer should not have access to. If so, a permission violation error will be returned. The Implementation will generally not return such a permission violation error if the Consumer instead uses a Path returned by *it_get_pathinfo* or from the Completion Event for a Receive operation. (It is still possible that a permission violation error could be returned if the network were reconfigured after the Path was returned but before *it_address_handle_create* was furnished with that Path.)

1509 **RETURN VALUE**

1510 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

1511	IT_ERR_RESOURCES	The requested operation failed due to insufficient resources.
1512	IT_ERR_INVALID_PZ	The Protection Zone Handle (<i>pz_handle</i>) was invalid.
1513	IT_ERR_INVALID_FLAGS	The flags (<i>ah_flags</i>) value was invalid.
1514 1515	IT_ERR_NO_PERMISSION	The Consumer did not have the proper permissions to perform the requested operation.
1516 1517	IT_ERR_INVALID_SOURCE_PATH	One of the components of the Source portion of the supplied Path was invalid.
1518	IT_ERR_INVALID_SPIGOT	An invalid Spigot ID was specified.
1519 1520 1521 1522	IT_ERR_IA_CATASTROPHE	The Interface Adapter has experienced a catastrophic error and is in the disabled state. None of the output parameters from this routine are valid. See <i>it_ia_info_t</i> for a description of the disabled state.
1523	SEE ALSO	
1524		<i>it_get_pathinfo()</i> , <i>it_path_t</i> , <i>it_address_handle_query()</i> , <i>it_address_handle_modify()</i> ,
1525		<i>it_address_handle_free()</i>
1526		

1527

it_address_handle_free()

1528 **NAME**

1529 *it_address_handle_free* – free an Address Handle

1530 **SYNOPSIS**

```
1531 #include <it_api.h>
1532
1533 it_status_t it_address_handle_free(
1534     IN it_addr_handle_t addr_handle
1535 );
```

1536 **APPLICABILITY**

1537 *it_address_handle_free* is applicable only to the UD service type.

1538 **DESCRIPTION**

1539 *addr_handle* Address Handle to free.

1540 *it_address_handle_free* removes an existing Address Handle and frees all associated underlying
1541 resources. Once *it_address_handle_free* returns, *addr_handle* can no longer be used. Consumers
1542 that wish to write code that is independent of the Implementation are therefore advised to allow
1543 all outstanding Send operations that reference an Address Handle to complete before freeing the
1544 Address Handle.

1545 **RETURN VALUE**

1546 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

1547 IT_ERR_INVALID_AH The Address Handle (*addr_handle*) was invalid.

1548 IT_ERR_IA_CATASTROPHE The Interface Adapter has experienced a catastrophic error and is
1549 in the disabled state. None of the output parameters from this
1550 routine are valid. See *it_ia_info_t* for a description of the disabled
1551 state.

1552 **SEE ALSO**

1553 [*it_address_handle_create\(\)*](#), [*it_address_handle_query\(\)*](#), [*it_address_handle_modify\(\)*](#)

1554

1555

it_address_handle_modify()

1556 NAME

1557 it_address_handle_modify – modify an Address Handle

1558 SYNOPSIS

```

1559 #include <it_api.h>
1560
1561 it_status_t it_address_handle_modify(
1562     IN          it_addr_handle_t      addr_handle,
1563     IN          it_addr_param_mask_t  mask,
1564     IN const    it_addr_param_t      *params
1565 );

```

1566 APPLICABILITY

1567 *it_address_handle_modify* is applicable only to the UD service type.

1568 DESCRIPTION

1569 *addr_handle* Address Handle to modify.

1570 *mask* Bitwise OR of flags for desired parameters to be modified.

1571 *params* Structure whose members contain the new parameter values.

1572 *it_address_handle_modify* changes selected attributes of the Address Handle *addr_handle*. If
1573 this routine returns success, all requested attributes are modified. If it does not return success,
1574 none of the requested attributes are modified.

1575 The Consumer should avoid calling this routine while a DTO that references this Address
1576 Handle is in progress. If the Consumer fails to abide by this restriction, the Destination that the
1577 DTO is sent to is undefined.

1578 The attributes to be modified are specified by the flags in *mask*. New values for the attributes are
1579 specified by the corresponding fields in the structure pointed to by *params*. Each field and the
1580 corresponding flag name that must appear in *mask* to modify the given field are shown below.
1581 (The flag name appears in a comment to the right of the field.) Note that attributes represented
1582 by fields of *it_addr_param_t* that are not shown below cannot be modified.

```

1583 typedef struct {
1584     ...
1585     it_path_t path; /* IT_ADDR_PARAM_PATH */
1586     ...
1587 } it_addr_param_t;
1588

```

1589 The table below defines the meaning of each member of the *it_addr_param_t* structure.

it_addr_param_t Member	Meaning
<i>path</i>	The new Path to be associated with this Address Handle. The Path will be associated with the Address Handle as a single unit. If the Consumer only wishes to modify a portion of the Path attributes, it can call <i>it_address_handle_query</i> to retrieve the current Path, modify the Path attributes as desired, and then call

it_addr_param_t Member	Meaning
	<i>it_address_handle_modify</i> with the resulting Path. See the <i>it_address_handle_create</i> reference page for details about which portions of the Path are relevant for Address Handles.

1590
1591
1592
1593

The Consumer may not be allowed to access the Path that the requested modification to the Address Handle would imply. If that is the case, a permission violation error will be returned by this routine.

1594 **RETURN VALUE**

1595 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

1596	IT_ERR_INVALID_AH	The Address Handle (<i>addr_handle</i>) was invalid.
1597	IT_ERR_INVALID_MASK	The <i>mask</i> contained invalid flag values.
1598 1599	IT_ERR_NO_PERMISSION	The Consumer did not have the proper permissions to perform the requested operation.
1600 1601	IT_ERR_INVALID_SOURCE_PATH	One of the components of the Source portion of the supplied Path was invalid.
1602	IT_ERR_INVALID_SPIGOT	An invalid Spigot ID was specified.
1603 1604 1605 1606	IT_ERR_IA_CATASTROPH	The Interface Adapter has experienced a catastrophic error and is in the disabled state. None of the output parameters from this routine are valid. See <i>it_ia_info_t</i> for a description of the disabled state.

1607 **SEE ALSO**

1608 *it_address_handle_create()*, *it_address_handle_query()*, *it_address_handle_free()*

1609

1610

it_address_handle_query()

1611 **NAME**

1612 `it_address_handle_query` – query an Address Handle

1613 **SYNOPSIS**

```

1614 #include <it_api.h>
1615
1616 it_status_t it_address_handle_query(
1617     IN  it_addr_handle_t      addr_handle,
1618     IN  it_addr_param_mask_t  mask,
1619     OUT it_addr_param_t      *params
1620 );
1621
1622 typedef enum {
1623     IT_ADDR_PARAM_ALL      = 0x0001,
1624     IT_ADDR_PARAM_IA      = 0x0002,
1625     IT_ADDR_PARAM_PZ      = 0x0004,
1626     IT_ADDR_PARAM_PATH    = 0x0008
1627 } it_addr_param_mask_t;
1628
1629 typedef struct {
1630     it_ia_handle_t  ia;    /* IT_ADDR_PARAM_IA */
1631     it_pz_handle_t  pz;    /* IT_ADDR_PARAM_PZ */
1632     it_path_t       path; /* IT_ADDR_PARAM_PATH */
1633 } it_addr_param_t;

```

1634 **APPLICABILITY**

1635 `it_address_handle_query` is applicable only to the UD service type.

1636 **DESCRIPTION**

1637 *addr_handle* Address Handle to query.

1638 *mask* Bitwise OR of flags for desired parameters.

1639 *params*: Structure whose members are written with the desired parameters.

1640 `it_address_handle_query` returns the desired attributes of the Address Handle *addr_handle* in the structure pointed to by *params*. On return, each field of *params* is only valid if the corresponding flag as shown above in the comment to the right of the field is set in the *mask* argument. The *mask* value `IT_ADDR_PARAM_ALL` causes all fields to be returned.

1644 The table below defines the meaning of each member of the `it_addr_param_t` structure.

it_addr_param_t Member	Meaning
<i>ia</i>	The Handle for the IA with which this Address Handle is associated.
<i>pz</i>	The Handle for the Protection Zone with which this Address Handle is associated.
<i>path</i>	The Path that is associated with this Address Handle. Not all

it_addr_param_t Member	Meaning
	fields in the Path are relevant for an Address Handle; see the it_address_handle_create reference page for details. All fields in the it_path_t identified as pertinent to Address Handles as documented on the it_address_handle_create reference page are returned by it_address_handle_query .

1645 **RETURN VALUE**

1646 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

1647	IT_ERR_INVALID_AH	The Address Handle (<i>addr_handle</i>) was invalid.
1648	IT_ERR_INVALID_MASK	The <i>mask</i> contained invalid flag values.
1649	IT_ERR_IA_CATASTROPHE	The Interface Adapter has experienced a catastrophic error and is in the disabled state. None of the output parameters from this routine are valid. See it_ia_info_t for a description of the disabled state.

1653 **SEE ALSO**

1654 [it_address_handle_create\(\)](#), [it_address_handle_modify\(\)](#), [it_address_handle_free\(\)](#)

1655

1656

it_convert_net_addr()

1657 NAME

1658 it_convert_net_addr – convert a Network Address from one format to another

1659 SYNOPSIS

```

1660 #include <it_api.h>
1661
1662 it_status_t it_convert_net_addr(
1663     IN  const  it_net_addr_t      *source_addr,
1664     IN          it_net_addr_type_t  addr_type,
1665     OUT          it_net_addr_t      *destination_addr
1666 );

```

1667 DESCRIPTION

1668 *source_addr* The input Network Address that is to be converted.

1669 *addr_type* The new type of address to convert the *source_addr* address to.

1670 *destination_addr* The returned Network Address.

1671 The *it_convert_net_addr* routine is used to convert one form of Network Address into another.
1672 The type of Network Address desired is specified by *addr_type*, and upon successful return from
1673 this routine *destination_addr* will contain an address of that type. If this routine does not return
1674 success, the contents of *destination_addr* are undefined.

1675 The Implementation might not support the requested Network Address conversion. If it does not,
1676 an error will be returned.

1677 The set of Network Addresses that are associated with a given Spigot is dynamic, and can
1678 change over time. (For example, a link on a switch or router could become inoperative, thus
1679 decreasing the set of Network Addresses by which a given Spigot can be reached.) There is
1680 therefore no guarantee that given the same input parameters two different invocations of
1681 *it_convert_net_addr* will return the same results. The Network Address returned by
1682 *it_convert_net_addr* is chosen from amongst the Network Addresses that match the selection
1683 criteria at the time of the call. In addition, since multiple Network Addresses of a given type can
1684 be associated with the same Spigot, the Implementation may return a different Network Address
1685 for two different invocations of *it_convert_net_addr* regardless of the state of the network.

1686 This call may block the caller's execution waiting for a remotely generated event.

1687 RETURN VALUE

1688 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

1689 IT_ERR_INVALID_ADDRESS The Network Address specified in *source_addr* was invalid.

1690 IT_ERR_INVALID_NETADDR The type of the Network Address specified in *source_addr*
1691 was not recognized.

1692 IT_ERR_INVALID_CONVERSION The requested Network Address conversion either was not
1693 supported by the Implementation, or was supported but could

1694		not be performed by the Implementation for the particular
1695		<i>source_addr</i> that was input.
1696	IT_ERR_INTERRUPT	The call was unblocked by a signal.
1697	IT_ERR_IA_CATASTROPHE	The Interface Adapter has experienced a catastrophic error
1698		and is in the disabled state. None of the output parameters
1699		from this routine are valid. See <i>it_ia_info_t</i> for a description
1700		of the disabled state.

1701 **APPLICATION USAGE**

1702 When a Consumer receives an incoming Connection establishment attempt Event, the

1703 *source_addr_info* field in that Event will contain the Network Address of the initiator of the

1704 Connection establishment attempt. The type of Network Address contained within

1705 *source_addr_info* is Implementation-specific, and therefore may not be one that the Consumer

1706 wishes to deal with. The Consumer can use *it_convert_net_addr* to convert the Network Address

1707 supplied in *source_addr_info* into a preferred type.

1708 **SEE ALSO**

1709 *it_listen_create()*, *it_net_addr_t*

1710

it_ep_accept()

1711

1712 NAME

1713 `it_ep_accept` – accept an incoming Connection Request or Connection Reply

1714 SYNOPSIS

```
1715 #include <it_api.h>
1716
1717 it_status_t it_ep_accept(
1718     IN          it_ep_handle_t          ep_handle,
1719     IN          it_cn_est_identifier_t  cn_est_id,
1720     IN const    unsigned char          *private_data,
1721     IN          size_t                  private_data_length
1722 );
1723
1724 typedef uint64_t it_cn_est_identifier_t;
```

1725 APPLICABILITY

1726 `it_ep_accept` is applicable only to Endpoints created for the RC service type.

1727 DESCRIPTION

1728 `ep_handle` Local Endpoint to be bound to the Connection Request being accepted.

1729 `cn_est_id` Connection Establishment Identifier associated with the Connection
1730 Request being accepted. The `cn_est_id` is obtained from the data delivered
1731 in the Connection Request Event (IT_CM_REQ_CONN_
1732 REQUEST_EVENT). See the [it_cm_req_events](#) reference page for details.

1733 `private_data` Opaque Private Data provided by the IT_CM_MSG_CONN_
1734 PEER_REJECT_EVENT Event delivered to the Remote Consumer that
1735 will be sent to the Remote Endpoint. If the Interface Adapter does not
1736 support Private Data, `private_data_length` must be zero. When `it_ep_accept`
1737 is called on the active side of the Connection (i.e., in three-way Connection
1738 Establishment), delivery of Private Data is unreliable.

1739 `private_data_length` Length of `private_data`. This field must be 0 if the IA does not support
1740 Private Data.

1741 `it_ep_accept` accepts an incoming Connection Request Event (IT_CM_REQ_CONN_
1742 REQUEST_EVENT) or Connection accept arrival Event (IT_CM_MSG_CONN_ACCEPT_
1743 ARRIVAL_EVENT). Calling `it_ep_accept` is the last Local Consumer step in establishing an
1744 Endpoint-to-Endpoint Connection for a three-way Connection Establishment. The Consumer is
1745 notified of an established Connection by an IT_CM_MSG_CONN_ESTABLISHED_EVENT
1746 Event being delivered on the connect EVD of the Endpoint. The Event is generated on both the
1747 active and passive side of the Connection establishment. See the Communication Management
1748 Message Event ([it_cm_msg_events](#)) reference page for details.

1749 If the initial [it_ep_connect](#) specified two-way Connection Establishment, then `it_ep_accept` is
1750 called only on the Passive side of the Endpoint-to-Endpoint Connection.

1751 If the initial [it_ep_connect](#) specified three-way Connection Establishment, then `it_ep_accept` is
1752 called on both the Active and the Passive sides of the Endpoint-to-Endpoint Connection.

1753 For two-way Connection Establishment, on the Passive side the Endpoint will transition into the
 1754 IT_EP_STATE_CONNECTED state when the Consumer calls *it_ep_accept*.

1755 For three-way Connection Establishment, on the Passive side the Endpoint will transition into
 1756 the IT_EP_STATE_PASSIVE_CONNECTION_PENDING state when the Consumer calls
 1757 *it_ep_accept*. An IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT Event will be delivered
 1758 to the Active-side Consumer after the Passive side calls *it_ep_accept* and will cause the Active
 1759 Endpoint eventually to transition into the IT_EP_STATE_ACTIVE2_
 1760 CONNECTION_PENDING state. Subsequently, the Passive side will transition into the
 1761 IT_EP_STATE_CONNECTED state after the Active side consumer calls *it_ep_accept*.

1762 *it_ep_accept* destroys the Connection Establishment Identifier, *cn_est_id*. After *it_ep_accept*
 1763 returns, *cn_est_id* is no longer valid and cannot be used.

1764 The Connection Establishment process cannot be successfully completed unless the attributes of
 1765 the Local and Remote Endpoints are compatible; see *it_cm_msg_events* for details. The
 1766 Consumer can call *it_ep_modify* to make the Local Endpoint attributes compatible before calling
 1767 *it_ep_accept*.

1768 **RETURN VALUE**

1769 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

1770	IT_ERR_PDATA_NOT_SUPPORTED	Private Data was supplied by the Consumer but this Interface Adapter does not support Private Data.
1771		
1772	IT_ERR_INVALID_PDATA_LENGTH	The Interface Adapter supports Private Data, but the length specified exceeded the Interface Adapter's capabilities.
1773		
1774		
1775	IT_ERR_INVALID_EP	The Endpoint Handle (<i>ep_handle</i>) was invalid.
1776	IT_ERR_INVALID_EP_STATE	The Endpoint was not in the proper state for the attempted operation. See <i>it_ep_state_t</i> reference page.
1777		
1778	IT_ERR_INVALID_EP_TYPE	The attempted operation was invalid for the Service Type of the Endpoint.
1779		
1780	IT_ERR_INVALID_CN_EST_ID	The Connection Establishment Identifier (<i>cn_est_id</i>) was invalid.
1781		
1782	IT_ERR_INVALID_EP_ATTR	The Local and Remote Endpoint attributes conflicted. Either the <i>max_message_size</i> , the number of <i>rdma_read_ird</i> , or the number of <i>rdma_read_ord</i> conflicted between the two Endpoints. This error will not be reported on the Passive-side accept of a three-way Connection Establishment.
1783		
1784		
1785		
1786		
1787		
1788	IT_ERR_EP_TIMEWAIT	The Endpoint provided to <i>it_ep_accept</i> was in the TimeWait condition, therefore the Connection could not be established. See <i>it_ep_rc_create</i> for details of the TimeWait condition.
1789		
1790		
1791		
1792	IT_ERR_IA_CATASTROPHE	The IA has experienced a catastrophic error and is in the disabled state. None of the output parameters from this
1793		

1794 routine are valid. See *it_ia_info_t* for a description of the
1795 disabled state.

1796 **ASYNCHRONOUS ERRORS**

1797 For the iWARP Transport, if the local side (Passive side, caller of *it_ep_accept*) uses an
1798 RDMAC AV-RNIC and the remote (Active side) uses a Non-permissive AV-RNIC/IETF, then
1799 the local Implementation for the side calling *it_ep_accept* will generate an IT_CM_MSG_
1800 NONPEER_REJECT_EVENT during MPA Startup, with an IT_CN_REJ_BAD_
1801 CONN_PARMS reject reason code indicating incompatible protocol versions (see [INTEROP-
1802 IETF]).

1803 **APPLICATION USAGE**

1804 Calls to routines such as *it_ep_accept*, *it_reject*, and *it_ep_disconnect* that pertain to the same
1805 Endpoint or Connection Establishment Identifier should be serialized by the Consumer. Failure
1806 to abide by this restriction may result in a segmentation violation or other error.

1807 If *it_ep_accept* returns the IT_ERR_EP_TIMEWAIT error, the Consumer can recover either by
1808 retrying the Connection Establishment after the TimeWait interval has elapsed, or by retrying the
1809 Connection Establishment using a different Endpoint that is not under a TimeWait condition.

1810 The Consumer should post at least one Receive buffer using the *it_post_recv* routine before
1811 calling *it_ep_accept*. Failure to do so can prevent a Connection from being established under
1812 certain circumstances on some transports.

1813 **SEE ALSO**

1814 *it_ep_connect()*, *it_reject()*, *it_ep_disconnect()*, *it_handoff()*, *it_ep_state_t*, *it_cm_req_events*,
1815 *it_cm_msg_events*

1816

it_ep_connect()

1817

1818 NAME

1819 it_ep_connect – initiate an Endpoint Connection establishment request

1820 SYNOPSIS

```
1821 #include <it_api.h>
1822
1823 it_status_t it_ep_connect(
1824     IN          it_ep_handle_t          ep_handle,
1825     IN const    it_path_t*             path,
1826     IN const    it_conn_attributes_t*   conn_attr,
1827     IN const    it_conn_qual_t*        connect_qual,
1828     IN          it_cn_est_flags_t       cn_est_flags,
1829     IN const    unsigned char*         private_data,
1830     IN          size_t                  private_data_length
1831 );
1832
1833 /* Transport-specific connection attributes for InfiniBand */
1834 typedef struct {
1835
1836     /* Remote CM Response Timeout, as defined in the REQ
1837      message for the IB CM protocol */
1838     uint8_t remote_cm_timeout : 5;
1839
1840     /* Local CM Response Timeout, as defined in the REQ
1841      message for the IB CM protocol */
1842     uint8_t local_cm_timeout : 5;
1843
1844     /* Retry Count, as defined in the REQ message for the
1845      IB CM protocol */
1846     uint8_t retry_count : 3;
1847
1848     /* RNR Retry Count, as defined in the REQ message for
1849      the IB CM protocol */
1850     uint8_t rnr_retry_count : 3;
1851
1852     /* Max CM retries, as defined in the REQ message for
1853      the IB CM protocol */
1854     uint8_t max_cm_retries : 4;
1855
1856     /* Local ACK Timeout, as defined in the REQ message
1857      for the IB CM protocol */
1858     uint8_t local_ack_timeout : 5;
1859 } it_ib_conn_attributes_t;
1860
1861 /* Transport-specific connection attributes for VIA */
1862 typedef struct {
1863
1864     /* VIA currently has no transport-specific connection
1865      attributes. A dummy entry is defined to allow ANSI
1866      compilation. */
1867
```

```

1868         void *unused;
1869     } it_via_conn_attributes_t;
1870
1871     /* Transport-specific connection attributes for iWARP */
1872     typedef struct {
1873         /* iWARP currently has no transport-specific connection
1874            attributes. A dummy entry is defined to allow ANSI
1875            compilation. */
1876         void *unused;
1877     } it_iwarp_conn_attributes_t;
1878
1879     /* Transport-specific connection attributes. This union is
1880        discriminated by the transport type being used to form the
1881        connection. This can be determined by examining
1882        the transport_type member in the it_ia_info_t that is
1883        associated with the IA that contains ep_handle. */
1884
1885     typedef union {
1886         it_ib_conn_attributes_t    ib;
1887         it_via_conn_attributes_t   via;
1888         it_iwarp_conn_attributes_t iwarp;
1889     } it_conn_attributes_t;
1890
1891     typedef enum {
1892         IT_CONNECT_FLAG_TWO_WAY      = 0x0001,
1893         IT_CONNECT_FLAG_THREE_WAY   = 0x0002,
1894         IT_CONNECT_SUPPRESS_IRD_ORD = 0x0004
1895     } it_cn_est_flags_t;

```

1899 **APPLICABILITY**

1900 *it_ep_connect* is applicable only to Endpoints created for the RC service type.

1901 **DESCRIPTION**

1902	<i>ep_handle</i>	Handle of the local Endpoint.
1903	<i>path</i>	Path for Connection establishment request.
1904	<i>conn_attr</i>	The transport-specific attributes for the Connection establishment attempt.
1905	<i>connect_qual:</i>	The Connection Qualifier for which the Consumer is initiating a
1906		Connection establishment request. See <i>it_conn_qual_t</i> for more information
1907		on Connection Qualifiers.
1908	<i>cn_est_flags</i>	Bitwise OR of flags for the Connection establishment request.
1909		

Features	Name	Bit Value	Description
Two-way Connection establishment	IT_CONNECT_FLAG_TWO_WAY	0x0001	The Connection is established once the passive side of the Connection establishment calls <i>it_ep_accept</i> .
Three-way Connection establishment	IT_CONNECT_FLAG_THREE_WAY	0x0002	The Connection is established once the active side of the Connection establishment calls <i>it_ep_accept</i> .
Suppress use of IRD/ORD	IT_CONNECT_SUPPRESS_IRD_ORD	0x0004	The active side of the Connection will not attempt to convey IRD/ORD values to the passive side. The IT_CONNECT_SUPPRESS_IRD_ORD bit is ignored if the <i>it_ia_info_t</i> value <i>ird_ord_ia_support</i> is IT_FALSE.

1910

1911 *private_data* Opaque Private Data to be sent in the IT_CM_REQ_CONN_
1912 REQUEST_EVENT Event delivered to the Remote Consumer. If the IA
1913 does not support Private Data, *private_data_length* must be zero.

1914 *private_data_length* Length of *private_data*. This field must be 0 if the IA does not support
1915 Private Data.

1916 The *it_ep_connect* routine initiates a Connection establishment request for an existing local
1917 Endpoint using an *it_path_t*. The *path* can be found by using the *it_get_pathinfo* function. This
1918 request generates a connect establishment request (IT_CM_REQ_CONN_REQUEST_EVENT)
1919 Event on the passive side based on the Path provided. Once the Connection establishment
1920 request has been initiated the active side Endpoint transitions into the
1921 IT_EP_STATE_ACTIVE1_CONNECTION_PENDING state.

1922 Consumers that wish to write portable code should pass IT_NO_ADDR (typically a NULL
1923 value) for the *conn_attr* parameter. If the Consumer passes a NULL value for this parameter, the
1924 Implementation will choose Implementation-dependent default values for the transport-specific
1925 Connection attributes that maximize the probability of the Connection being successfully
1926 established and maintained. If the Consumer wishes to have control over the transport-specific
1927 Connection attributes, they can pass a non-NULL value for the *conn_attr* parameter. If the
1928 Consumer passes a non-NULL value for this parameter and the Implementation determines that
1929 some portion of the transport-specific Connection attributes are invalid, it will return an error
1930 from this routine. What constitutes invalid transport-specific Connection attributes is
1931 Implementation-dependent. The Implementation will not return an error indicating some portion
1932 of the transport-specific Connection attributes are invalid if the Consumer passes a NULL value
1933 for the *conn_attr* parameter.

1934 The flags `IT_CONNECT_FLAG_THREE_WAY` and `IT_CONNECT_FLAG_TWO_WAY` of
1935 `it_cn_est_flags_t` select three-way and two-way Connection establishment, respectively, and
1936 their use in `cn_est_flags` is mutually exclusive. If the Interface Adapter attribute
1937 `three_way_handshake_support` equals `IT_TRUE`, the Consumer can select three-way or two-
1938 way Connection establishment; otherwise, two-way Connection establishment must be selected.

1939 The default behavior of the IT-API is to attempt to negotiate IRD/ORD between the active and
1940 passive sides as described in Chapter 5 where supported by the IA. For an IA where the
1941 `it_ia_info_t` values `ird_ord_ia_support` and `ird_ord_suppressible` are both `IT_TRUE`, the
1942 `IT_CONNECT_SUPPRESS_IRD_ORD` bit may be set by the Consumer to cause the IA not to
1943 perform IRD/ORD negotiation. For such an IA, if the `IT_CONNECT_SUPPRESS_IRD_ORD`
1944 bit is cleared in `cn_est_flags`, then the IRD/ORD Endpoint attributes will be negotiated. For an
1945 IA where `ird_ord_ia_support` is `IT_FALSE`, the `IT_CONNECT_SUPPRESS_IRD_ORD` bit is
1946 ignored. For an IA where `ird_ord_ia_support` is `IT_TRUE` but `ird_ord_suppressible` is
1947 `IT_FALSE`, attempting to set `IT_CONNECT_SUPPRESS_IRD_ORD` will yield an immediate
1948 error.

1949 The passive side Consumer can choose either to accept or reject the Connection Request. If the
1950 passive side chooses to reject the Connection by calling `it_reject`, then an
1951 `IT_CM_MSG_CONN_PEER_REJECT_EVENT` Event is generated on the active side and the
1952 active side Endpoint transitions into the `IT_EP_STATE_NONOPERATIONAL` state. If it
1953 chooses to accept the Connection by calling `it_ep_accept`, then the behavior is dependent on the
1954 type of Connection setup specified by the `cn_est_flags`.

1955 For three-way Connection establishments, an `IT_CM_MSG_CONN_ACCEPT_ARRIVAL_`
1956 `EVENT` Event is generated on the active side if the passive side Consumer accepts the
1957 Connection establishment request by calling `it_ep_accept`. The active Consumer can choose to
1958 either accept or reject the Connection by calling `it_ep_accept` or `it_reject` respectively. For a two-
1959 way Connection establishment, an `IT_CM_MSG_CONN_ESTABLISHED_EVENT` Event is
1960 generated on the active side after the passive side Consumer accepts the Connection and the
1961 Endpoint on the active side transitions into the (`IT_EP_STATE_CONNECTED`) connected
1962 state. See the `it_ep_state_t` reference page for a complete description on the Endpoint state
1963 diagram for both the three-way and two-way Connection establishment.

1964 Whenever an Endpoint transitions to the connected (`IT_EP_STATE_CONNECTED`) state the
1965 Consumer will Receive an `IT_CM_MSG_CONN_ESTABLISHED_EVENT` Event on the
1966 simple Event Dispatcher to which the Communication Management Message Event Stream is
1967 routed after the Endpoint transitions into the `IT_EP_STATE_CONNECTED` state. This Event is
1968 generated on both the active and passive side of the Connection establishment after the Endpoint
1969 state transition takes place.

1970 For a complete definition of Endpoint state and a more complete description of the state
1971 transitions see the `it_ep_state_t` reference page. If for any reason an Endpoint Connection fails to
1972 be established, the Endpoint will transition into the `IT_EP_STATE_NONOPERATIONAL` state
1973 and any Receive DTO operations that were successfully posted to the Endpoint will be
1974 completed with an `IT_DTO_ERR_FLUSHED` status.

1975 **EXTENDED DESCRIPTION**

1976 An Endpoint can only be connected to a different Endpoint. An Endpoint cannot be connected to
1977 itself.

1978 RETURN VALUE

1979 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

1980 1981	IT_ERR_NO_PERMISSION	The Consumer did not have the proper permissions to perform the requested operation.
1982	IT_ERR_RESOURCES	The operation failed due to resource limitations.
1983	IT_ERR_INVALID_CONN_QUAL	The Connection Qualifier is invalid.
1984 1985 1986	IT_ERR_PDATA_NOT_SUPPORTED	Private Data was supplied by the Consumer but this IA does not support Private Data. See <i>it_ia_query</i> for the IA's capabilities to support Private Data.
1987 1988	IT_ERR_INVALID_PDATA_LENGTH	The IA supports Private Data, but the length specified exceeds the IA's capabilities.
1989	IT_ERR_INVALID_SOURCE_PATH	The Source Path specified in the <i>it_path_t</i> was invalid.
1990	IT_ERR_INVALID_SPIGOT	The Spigot specified in the <i>it_path_t</i> was invalid.
1991	IT_ERR_INVALID_EP	The <i>ep_handle</i> was invalid.
1992	IT_ERR_INVALID_EP_STATE	The Endpoint is not in the proper state to be connected.
1993 1994	IT_ERR_INVALID_EP_TYPE	The Endpoint Service Type does not support this operation.
1995	IT_ERR_INVALID_CN_EST_FLAGS	The Connection establishment flags are invalid.
1996 1997 1998	IT_ERR_INVALID_RTIMEOUT	The <i>conn_attr.ib.remote_cm_timeout</i> value was invalid. The criteria for determining what constitutes an invalid value are Implementation-dependent.
1999 2000 2001	IT_ERR_INVALID_LTIMEOUT	The <i>conn_attr.ib.local_cm_timeout</i> value was invalid. The criteria for determining what constitutes an invalid value are Implementation-dependent.
2002 2003 2004	IT_ERR_INVALID_RETRY	The <i>conn_attr.ib.retry_count</i> value was invalid. The criteria for determining what constitutes an invalid value are Implementation-dependent.
2005 2006 2007	IT_ERR_INVALID_RNR_RETRY	The <i>conn_attr.ib.rnr_retry_count</i> value was invalid. The criteria for determining what constitutes an invalid value are Implementation-dependent.
2008 2009 2010	IT_ERR_INVALID_CM_RETRY	The <i>conn_attr.ib.max_cm_retries</i> value was invalid. The criteria for determining what constitutes an invalid value are Implementation-dependent.
2011 2012 2013	IT_ERR_INVALID_ETIMEOUT	The <i>conn_attr.ib.local_ack_timeout</i> value was invalid. The criteria for determining what constitutes an invalid value are Implementation-dependent.
2014	IT_ERR_INVALID_IANA_LR_PORT	The local port in <i>connect_qual</i> is already in use.

2015 2016 2017 2018	IT_ERR_IA_CATASTROPHE	The IA has experienced a catastrophic error and is in the disabled state. None of the output parameters from this routine are valid. See <i>it_ia_info_t</i> for a description of the disabled state.
2019 2020 2021 2022 2023 2024	ASYNCHRONOUS ERRORS	For the iWARP Transport, if one side (local or remote) uses a Non-permissive AV-RNIC/IETF and the other side (remote or local) uses an RDMAC AV-RNIC, then the local Implementation for the caller of <i>it_ep_connect</i> will generate an IT_CM_MSG_NONPEER_REJECT_EVENT during MPA Startup, with an IT_CN_REJ_BAD_CONN_PARMS reject reason code indicating incompatible protocol versions (see [INTEROP-IETF]).
2025 2026 2027 2028 2029 2030 2031 2032 2033	APPLICATION USAGE	It is possible that between the time the Consumer calls <i>it_get_pathinfo</i> to retrieve a valid Path and the time the Consumer passes that Path as the <i>path</i> parameter to this routine, the set of available Paths through the network for forming the Connection may change, rendering some portion of the given <i>path</i> invalid. If IT_ERR_INVALID_SOURCE_PATH or IT_ERR_INVALID_SPIGOT is returned from this routine and the <i>path</i> the Consumer provided is one that was returned from <i>it_get_pathinfo</i> and was not subsequently modified by the Consumer, the Consumer can attempt to recover from the error by calling <i>it_get_pathinfo</i> again to retrieve an up-to-date Path to form the Connection and retrying the call to <i>it_ep_connect</i> with the new Path.
2034 2035		On some transports, the passive side will not get a IT_CM_MSG_CONN_ESTABLISHED_EVENT Event until the active side first posts a Send operation.
2036 2037 2038 2039	SEE ALSO	<i>it_ep_accept()</i> , <i>it_reject()</i> , <i>it_ep_disconnect()</i> , <i>it_handoff()</i> , <i>it_ep_state_t</i> , <i>it_ia_query()</i> , <i>it_conn_qual_t</i>

2040

it_ep_disconnect()

2041 NAME

2042 it_ep_disconnect – disconnect an existing Endpoint-to-Endpoint Connection

2043 SYNOPSIS

```

2044 #include <it_api.h>
2045
2046 it_status_t it_ep_disconnect (
2047     IN          it_ep_handle_t  ep_handle,
2048     IN  const   unsigned char  *private_data,
2049     IN          size_t          private_data_length
2050 );

```

2051 APPLICABILITY

2052 *it_ep_disconnect* is applicable only to the RC service type.

2053 DESCRIPTION

2054 *ep_handle* Endpoint to be disconnected.

2055 *private_data* Opaque Private Data to be delivered in the IT_CM_MSG_CONN_ DISCONNECT_EVENT Event at the Remote Endpoint. If the IA does not support Private Data, *private_length* must be zero.

2058 *private_data_length*: Length of *private_data*. This field must be 0 if the IA does not support Private Data.

2060 *it_ep_disconnect* either breaks the existing Endpoint-to-Endpoint Connection or terminates Endpoint-to-Endpoint Connection in the process of being identified by the *ep_handle*. An IT_CM_MSG_CONN_DISCONNECT_EVENT Event will be generated on both the Local and Remote sides of the Connection. The generation of the Event on the remote Event is not guaranteed. If such an Event is not generated, Private Data will not be conveyed to the remote side. The Endpoints will transition into the IT_EP_STATE_NONOPERATIONAL state. See the [it_ep_reset](#) reference page for how to restore an Endpoint back into the IT_EP_STATE_UNCONNECTED state. *it_ep_disconnect* is ungraceful in the sense that the remote Endpoints transition directly into the IT_EP_STATE_NONOPERATIONAL state without the Consumer’s intervention.

2070 *it_ep_disconnect* may be successfully called in all states except IT_EP_STATE_UNCONNECTED.

2072 Once the Endpoint is in the IT_EP_STATE_NONOPERATIONAL state, any pending Data Transfer Operations or Link or Unlink operations on the Endpoint will be flushed and will generate Completion Events with a Status of IT_DTO_ERR_FLUSHED.

2075 See the [it_ep_state_t](#) reference page for a complete description of the Endpoint state behavior and transitions.

2077 RETURN VALUE

2078 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

2079	IT_ERR_INVALID_EP_STATE	The Endpoint was not in the proper state for the attempted operation.
2080		
2081	IT_ERR_INVALID_EP	The Endpoint Handle (<i>ep_handle</i>) was invalid.
2082	IT_ERR_INVALID_EP_TYPE	The attempted operation was invalid for the Service Type of the Endpoint.
2083		
2084	IT_ERR_PDATA_NOT_SUPPORTED	Private Data was supplied by the Consumer but this Interface Adapter does not support Private Data. See it_ia_query for the IA's capabilities to support Private Data.
2085		
2086		
2087		
2088	IT_ERR_INVALID_PDATA_LENGTH	The Interface Adapter supports Private Data, but the length specified exceeded the Interface Adapter's capabilities.
2089		
2090		
2091	IT_ERR_IA_CATASTROPHE	The IA has experienced a catastrophic error and is in the disabled state. None of the output parameters from this routine are valid. See it_ia_info_t for a description of the disabled state.
2092		
2093		
2094		

2095 **APPLICATION USAGE**

2096 The Consumer is responsible for coordinating the use of functions that free a Connection
 2097 Establishment Identifier (*cn_est_id*) such as [it_ep_accept](#), [it_reject](#), [it_ep_disconnect](#), and
 2098 [it_handoff](#). The behavior of functions that are passed as invalid Connection Establishment
 2099 Identifiers is indeterminate.

2100 The Consumer should be aware that successfully returning from this routine does not guarantee
 2101 that any interaction whatsoever will take place with the Remote Endpoint. If the Local
 2102 Consumer wishes to ensure that the Remote Consumer takes some action, an explicit message
 2103 should be sent to initiate that action before calling [it_ep_disconnect](#).

2104 With the three-way handshake Connection establishment method, there is also a potential race
 2105 condition between the Implementation generating the IT_CM_MSG_CONN_
 2106 ACCEPT_ARRIVAL_EVENT Event and the Consumer calling [it_ep_free](#). The Consumer
 2107 should not use the *cn_est_id* if the IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT Event
 2108 arrives after [it_ep_free](#) was called, regardless of whether the call returned yet, and regardless of
 2109 whether the Event was dequeued before or after the call was made. If the Consumer does use the
 2110 *cn_est_id*, then the Implementation may generate an IT_ERR_INVALID_CN_EST_ID error, or
 2111 it may generate a segmentation fault or other error.

2112 **SEE ALSO**

2113 [it_ep_accept\(\)](#), [it_reject\(\)](#), [it_ep_connect\(\)](#), [it_ep_state_t](#), [it_cm_msg_events](#), [it_ep_reset\(\)](#),
 2114 [it_ia_query\(\)](#)

it_ep_free()

2115

2116 NAME

2117 `it_ep_free` – destroy an RC or UD Endpoint

2118 SYNOPSIS

```
2119 #include <it_api.h>
2120
2121 it_status_t it_ep_free(
2122     IN it_ep_handle_t ep_handle
2123 );
```

2124 DESCRIPTION

2125 `ep_handle` Endpoint.

2126 `it_ep_free` destroys an Endpoint.

2127 An Endpoint cannot be destroyed if it has RMR(s) of type `IT_RMR_TYPE_NARROW` still
2128 bound to it. Otherwise, the Endpoint can be destroyed (freed) in any state. The freeing of an
2129 Endpoint also terminates the generation of Events to any of the EVDs associated with the
2130 Endpoint.

2131 Once `it_ep_free` returns, `ep_handle` may no longer be used.

2132 Freeing an Endpoint potentially means Events pertaining to that Endpoint might be lost on the
2133 `recv_sevd_handle` or `request_sevd_handle` SEVDs associated with the Endpoint. There is also
2134 potential to lose Events pertaining to that Endpoint on the `connect_sevd_handle` SEVD
2135 associated with the Endpoint. The Consumer should first drain these EVDs before calling
2136 `it_ep_free`.

2137 Freeing an RC-type Endpoint in the `IT_EP_STATE_CONNECTED` state while DTOs are in
2138 progress causes incoming DTOs to be ignored. All entries on the Endpoint `request_sevd_handle`,
2139 `recv_sevd_handle`, and `connect_sevd_handle` may or may not be on the EVDs after the Endpoint
2140 is destroyed.

2141 Freeing an RC-type Endpoint in the `IT_EP_STATE_ACTIVE1_CONNECTION_PENDING`,
2142 `IT_EP_STATE_ACTIVE2_CONNECTION_PENDING`, `IT_EP_STATE_PASSIVE_WAIT_`
2143 `RDMA_TRANS_REQ`, or `IT_EP_STATE_PASSIVE_CONNECTION_PENDING` state may
2144 cause a Connection establishment timeout or non-peer reject to be sent to the remote side.

2145 RETURN VALUE

2146 A successful call returns `IT_SUCCESS`. Otherwise, an error code is returned as described below:

2147 `IT_ERR_INVALID_EP` The Endpoint Handle (`ep_handle`) was invalid.

2148 `IT_ERR_EP_BUSY` An RMR of type `IT_RMR_TYPE_NARROW` is still bound to
2149 the Endpoint.

2150 `IT_ERR_IA_CATASTROPHE` The IA has experienced a catastrophic error and is in the
2151 disabled state. None of the output parameters from this routine
2152 are valid. See `it_ia_info_t` for a description of the disabled state.

2153 **ASYNCHRONOUS ERRORS**

2154 Freeing an RC-type Endpoint in the IT_EP_STATE_CONNECTED state while Work Requests
2155 are still active on the Send Queue or Receive Queue of the Endpoint may cause these Work
2156 Requests to complete with the IT_DTO_ERR_FLUSHED status.

2157 **APPLICATION USAGE**

2158 Since the Implementation may not immediately free underlying resources, the user must not rely
2159 upon being immediately able to reallocate an Endpoint that has been freed.

2160 The following method can be used to ensure that all Completion Events for Work Requests
2161 posted to a Send Queue or Receive Queue are dequeued prior to calling *it_ep_free* for an RC-
2162 type Endpoint:

- 2163 1. The Consumer should call *it_ep_disconnect* first.
- 2164 2. Then the Consumer should dequeue a disconnect, Connection broken, or Connection
2165 reject Event as appropriate from the Connection EVD associated with the Endpoint.
- 2166 3. If the last Work Request posted to the Send Queue of the Endpoint did not request a
2167 Completion Event be generated, the Consumer should post a Work Request set up as a
2168 “marker” that is flushed by the Implementation to *recv_evd_handle* or
2169 *request_evd_handle*. The Work Request is made a “marker” operation by setting the
2170 IT_COMPLETION_FLAG on the operation. (If the last Work Request posted to the Send
2171 Queue did request a Completion Event, that Completion Event can serve as the marker.)
- 2172 4. When the Consumer dequeues the completion of the marker from the EVD associated
2173 with the Send Queue of the Endpoint, it is guaranteed that all previously posted Work
2174 Request completions (including those posted with IT_COMPLETION_FLAG cleared) for
2175 the Endpoint have also been dequeued from that EVD.
- 2176 5. When the Consumer dequeues the completion for the last Receive Work Request posted to
2177 the Receive Queue of the Endpoint from the EVD associated with the Endpoint Receive
2178 Queue, it is guaranteed that all previously posted Receive completions for the Endpoint
2179 have also been dequeued from that EVD.
- 2180 6. After all of the previous steps, it is safe to destroy or reset the Endpoint without losing any
2181 completions or Connection Events.

2182 With the three-way handshake Connection establishment method, there is also a potential race
2183 condition between the Implementation generating the IT_CM_MSG_CONN_ACCEPT_
2184 ARRIVAL_EVENT Event and the Consumer calling *it_ep_disconnect*. The Consumer should
2185 not use *cn_est_id* if the IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT Event arrives
2186 after *it_ep_disconnect* was called, regardless of whether the call returned yet, and regardless of
2187 whether the Event was dequeued before or after the call was made. If the Consumer does use the
2188 *cn_est_id*, then the Implementation may generate an IT_ERR_INVALID_CN_EST_ID error, or
2189 it may generate a segmentation fault or other error.

2190 **SEE ALSO**

2191 *it_ep_rc_create()*, *it_ep_ud_create()*, *it_ep_modify()*, *it_ep_query()*, *it_ep_reset()*,
2192 *it_ep_disconnect()*, *it_ia_info_t*, *it_ep_state_t*, *it_dto_flags_t*, *it_post_send()*, *it_post_sendto()*,
2193 *it_post_recv()*, *it_post_recvfrom()*, *it_post_rdma_read()*, *it_post_rdma_write()*

2194

it_ep_modify()

2195

2196 NAME

2197 it_ep_modify – modify attributes of an existing Endpoint

2198 SYNOPSIS

```
2199 #include <it_api.h>
2200
2201 it_status_t it_ep_modify(
2202     IN          it_ep_handle_t      ep_handle,
2203     IN          it_ep_param_mask_t  mask,
2204     IN const    it_ep_attributes_t  *ep_attr
2205 );
```

2206 DESCRIPTION

2207 ep_handle Endpoint.

2208 mask Bitwise OR of flags for desired attributes to be modified.

2209 ep_attr Pointer to Consumer-allocated structure that contains new Consumer-
2210 requested Endpoint attributes.

2211 *it_ep_modify* changes selected attributes of the Endpoint *ep_handle*.

2212 Attributes to be modified are specified by flags in *mask*. New values for the attributes are
2213 specified by the corresponding fields in the structure pointed to by *ep_attr*. See
2214 [it_ep_attributes_t](#) for the definition of the structure.

2215 Flag values for the *mask* parameter are shown below. Note that attributes represented by fields of
2216 *ep_attr* for which no flag value is shown below cannot be modified. The requested attribute
2217 changes only affect the local Endpoint and have no effect on attributes of any remote Endpoint.

2218 Flag values for attributes that may be potentially modified:

```
2219 IT_EP_PARAM_MAX_PAYLOAD
2220 IT_EP_PARAM_MAX_REQ_DTO
2221 IT_EP_PARAM_MAX_RECV_DTO
2222 IT_EP_PARAM_RDMA_RD_ENABLE
2223 IT_EP_PARAM_RDMA_WR_ENABLE
2224 IT_EP_PARAM_MAX_IRD
2225 IT_EP_PARAM_MAX_ORD
2226 IT_EP_PARAM_EP_KEY
2227 IT_EP_PARAM_SOFT_HI_WATERMARK
2228 IT_EP_PARAM_HARD_HI_WATERMARK
```

2229 See [it_ep_attributes_t](#) for the definition of valid states in which each of the above attributes may
2230 be modified.

2231 Values for *mask* must be created as the bitwise OR of the Endpoint attributes flag values (above)
2232 that the Consumer desires to change. *it_ep_modify* must succeed in modifying all the requested
2233 attributes atomically; if the attempt to modify any of the requested attributes generates an error,
2234 none of the other attributes supplied to the call will be applied.

2235	RETURN VALUE	
2236		A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:
2237	IT_ERR_INVALID_EP	The Endpoint Handle (<i>ep_handle</i>) was invalid.
2238	IT_ERR_INVALID_MASK	The mask contained invalid flag values.
2239	IT_ERR_INVALID_EP_STATE	The Endpoint was not in the proper state for the attempted operation. See <i>it_ep_attributes_t</i> .
2240		
2241	IT_ERR_INVALID_EP_TYPE	The attempted operation was invalid for the Service Type of the Endpoint.
2242		
2243	IT_ERR_PAYLOAD_SIZE	The requested <i>max_dto_payload_size</i> exceeds the maximum payload size supported by the underlying transport.
2244		
2245		
2246	IT_ERR_RESOURCE_REQ_DTO	The underlying transport could not allocate the requested <i>max_req_dtos</i> resources at this time.
2247		
2248	IT_ERR_RESOURCE_RECV_DTO	The underlying transport could not allocate the requested <i>max_recv_dtos</i> resources at this time.
2249		
2250	IT_ERR_RESOURCE_IRD	The underlying transport could not allocate the requested <i>rdma_read_ird</i> resources at this time.
2251		
2252	IT_ERR_RESOURCE_ORD	The underlying transport could not allocate the requested <i>rdma_read_ord</i> resources at this time.
2253		
2254	IT_ERR_INVALID_EP_KEY	Invalid Endpoint Key value. The Consumer does not have local permissions to use the specified Endpoint Key.
2255		
2256	IT_ERR_SOFT_HI_WATERMARK	The IA associated with the Endpoint does not support the Endpoint Soft High Watermark mechanism and a non-zero Endpoint Soft High Watermark was supplied.
2257		
2258		
2259	IT_ERR_INVALID_WATERMARK	The Endpoint Soft High Watermark supplied was either larger than the maximum number of Receive DTOs that can be posted to the associated S-RQ, or was zero.
2260		
2261		
2262	IT_ERR_HARD_HI_WATERMARK	The IA associated with the Endpoint does not support the Endpoint Hard High Watermark mechanism and a non-zero Endpoint Hard High Watermark was supplied.
2263		
2264		
2265	IT_ERR_INVALID_RECV_DTO	An attempt was made to modify the <i>max_recv_dtos</i> attribute of an Endpoint that has an associated S-RQ.
2266		
2267	IT_ERR_IA_CATASTROPHE	The IA has experienced a catastrophic error and is in the disabled state. None of the output parameters from this routine are valid. See <i>it_ia_info_t</i> for a description of the disabled state.
2268		
2269		
2270		
2271	SEE ALSO	
2272		<i>it_ep_attributes_t</i> , <i>it_ep_rc_create()</i> , <i>it_ep_ud_create()</i> , <i>it_ep_query()</i> , <i>it_ep_free()</i> , <i>it_ia_info_t</i>

it_ep_query()

2273

2274 NAME

2275 `it_ep_query` – query an existing Endpoint

2276 SYNOPSIS

```
2277 #include <it_api.h>
2278
2279 it_status_t it_ep_query(
2280     IN    it_ep_handle_t    ep_handle,
2281     IN    it_ep_param_mask_t mask,
2282     OUT   it_ep_param_t    *params
2283 );
2284
2285 typedef struct {
2286     it_ia_handle_t    ia;                /* IT_EP_PARAM_IA */
2287     size_t            spigot_id;        /* IT_EP_PARAM_SPIGOT */
2288     it_ep_state_t    ep_state;         /* IT_EP_PARAM_STATE */
2289     it_transport_service_type_t service_type; /* IT_EP_PARAM_SERV_TYPE */
2290     it_path_t        dst_path;         /* IT_EP_PARAM_PATH */
2291     it_pz_handle_t    pz;              /* IT_EP_PARAM_PZ */
2292     it_evd_handle_t    request_sevd;   /* IT_EP_PARAM_REQ_SEVD */
2293     it_evd_handle_t    recv_sevd;     /* IT_EP_PARAM_RECV_SEVD */
2294     it_evd_handle_t    connect_sevd;  /* IT_EP_PARAM_CONN_SEVD */
2295     it_ep_attributes_t attr;           /* See it_ep_attributes_t
2296                                         for mask flags for attr */
2297 } it_ep_param_t;
```

2298 DESCRIPTION

2299 `ep_handle` Endpoint.

2300 `mask` Bitwise OR of flags for desired parameters and attributes.

2301 `params` Pointer to Consumer-allocated structure whose members are written with
2302 the desired Endpoint parameters and attributes.

2303 `it_ep_query` returns the desired parameters and attributes of the Endpoint `ep_handle` in the
2304 structure pointed to by `params`. On return, each field of `params` is only valid if the corresponding
2305 flag as shown below each `it_ep_param_t` member is set in the `mask` argument. The `mask` value
2306 `IT_EP_PARAM_ALL` causes all fields to be returned. The `it_ep_param_mask_t` enum is defined
2307 in `it_ep_attributes_t`.

2308 The definition of each field follows:

2309 `ia` Handle for the Interface Adapter specified to create the EP.

2310 `spigot_id` Spigot identifier. For RC Endpoints, this field is valid only if the Endpoint
2311 is not in the `IT_EP_STATE_UNCONNECTED` state. If the Endpoint is in
2312 the `IT_EP_STATE_UNCONNECTED` state, the value of this field is
2313 undefined.

2314 `ep_state` State of the Endpoint.

2315	<i>service_type</i>	Endpoint Service Type.
2316	<i>dst_path</i>	If the Endpoint is of the RC Service Type, the value of this field is undefined if the Endpoint state is IT_EP_STATE_UNCONNECTED. Otherwise, for an Endpoint of the RC Service Type on the active side of a Connection this is the Path that was specified to <i>it_ep_connect</i> ; on the passive side of a Connection this is the Path used by the Implementation to reach the requesting remote Endpoint. If the Endpoint is of the UD Service type, the value of this field is always undefined.
2317		
2318		
2319		
2320		
2321		
2322		
2323	<i>pz</i>	Handle for the Protection Zone specified while creating the EP.
2324	<i>request_sevd</i>	Handle for the IT_DTO_EVENT_STREAM Simple Event Dispatcher for DTO request Completion Events of the created Endpoint.
2325		
2326	<i>recv_sevd</i>	Handle for the IT_DTO_EVENT_STREAM Simple Event Dispatcher for DTO Receive Completion Events of the created Endpoint.
2327		
2328	<i>connect_sevd</i>	Handle for the IT_CM_MSG_EVENT_STREAM Simple Event Dispatcher for Connection Events of the created Endpoint. Invalid for UD Endpoint.
2329		
2330	<i>attr</i>	Attributes of Endpoint – definitions and mask values found in <i>it_ep_attributes_t</i> . Consumer ORs the appropriate mask values for each attribute field desired into the <i>mask</i> parameter to <i>it_ep_query</i> .
2331		
2332		
2333		

RETURN VALUE

2334 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

2335	IT_ERR_INVALID_EP	The Endpoint Handle (<i>ep_handle</i>) was invalid.
2336	IT_ERR_INVALID_MASK	The mask contained invalid flag values.
2337	IT_ERR_INVALID_EP_TYPE	The attempted operation was invalid for the Service Type of the Endpoint.
2338		
2339	IT_ERR_IA_CATASTROPHE	The IA has experienced a catastrophic error and is in the disabled state. None of the output parameters from this routine are valid. See <i>it_ia_info_t</i> for a description of the disabled state.
2340		
2341		
2342		

SEE ALSO

2344 *it_ep_attributes_t*, *it_ep_rc_create()*, *it_ep_ud_create()*, *it_ep_modify()*, *it_ep_free()*,
2345 *it_ia_info_t*
2346

2347

it_ep_rc_create()

2348 NAME

2349 it_ep_rc_create – create an Endpoint for Reliable Connection

2350 SYNOPSIS

```

2351 #include <it_api.h>
2352
2353 it_status_t it_ep_rc_create (
2354     IN          it_pz_handle_t          pz_handle,
2355     IN          it_evd_handle_t        request_sevd_handle,
2356     IN          it_evd_handle_t        recv_sevd_handle,
2357     IN          it_evd_handle_t        connect_sevd_handle,
2358     IN          it_ep_rc_creation_flags_t flags,
2359     IN const    it_ep_attributes_t     *ep_attr,
2360     OUT         it_ep_handle_t         *ep_handle
2361 );
2362
2363 typedef enum {
2364     IT_EP_NO_FLAG      = 0x00,
2365     IT_EP_REUSEADDR   = 0x01,
2366     IT_EP_SRQ         = 0x02
2367 } it_ep_rc_creation_flags_t;

```

2368 APPLICABILITY

2369 *it_ep_rc_create* is applicable only to create Endpoints of the RC Service Type.

2370 DESCRIPTION

2371 *pz_handle* Handle for the Protection Zone of the created Endpoint. Implicitly
2372 identifies the Interface Adapter to be used.

2373 *request_sevd_handle* Handle for the IT_DTO_EVENT_STREAM Simple Event Dispatcher for
2374 DTO request Completion Events of the created Endpoint.

2375 *recv_sevd_handle* Handle for the IT_DTO_EVENT_STREAM Simple Event Dispatcher for
2376 DTO Receive Completion Events of the created Endpoint.

2377 *connect_sevd_handle* Handle for the IT_CM_MSG_EVENT_STREAM Simple Event Dispatcher
2378 for Connection Events of the created Endpoint.

2379 *flags* Flags allowing the Consumer optionally to control behavior of the
2380 Implementation on Endpoint creation. Default is IT_EP_NO_FLAG.

2381 *ep_attr* Pointer to a structure that contains Consumer-requested Endpoint attributes.

2382 *ep_handle* Handle for the created Endpoint.

2383 *it_ep_rc_create* creates, on the Interface Adapter implicitly identified by *pz_handle*, a
2384 Connection Endpoint that is provided to the Consumer as *ep_handle*. The value of *ep_handle* is
2385 only defined if the return value of *it_ep_rc_create* is IT_SUCCESS.

2386 The Connection Endpoint is created in the IT_EP_STATE_UNCONNECTED state. See
2387 [it_ep_state_t](#) for details.

2388 The created Endpoint is not associated with an IA Spigot. An Endpoint is associated with a
2389 Spigot as part of Connection setup.

2390 The Protection Zone *pz_handle* allows Consumers to control what local memory the Endpoint
2391 can access for DTOs and what memory remote RDMA operations can access through the newly
2392 created Endpoint. Only memory referred to by LMRs and RMRs that match the Endpoint
2393 Protection Zone can be accessed through the Endpoint.

2394 *recv_sevd_handle* and *request_sevd_handle* are Event Dispatcher instances where the Consumer
2395 collects completion Notifications of DTOs and RMR operations. Completions of Receive DTOs
2396 are reported in *recv_sevd_handle* Event Dispatcher, and completions of Send, RDMA Read,
2397 RDMA Write DTOs, RMR Link, and RMR Unlink are reported in *request_sevd_handle*. It is
2398 permissible for *recv_sevd_handle* and *request_sevd_handle* to reference the same EVD. DTO
2399 and RMR operation Completion Events are defined in *it_dto_events*.

2400 The Consumer should not specify an SEVD in *recv_sevd_handle* or *request_sevd_handle* that is
2401 in overflowed state for use in the Endpoint creation call (see *it_evd_create* for more details on
2402 overflow). If the Consumer attempts to do so, the operation will fail with
2403 IT_ERR_INVALID_RECV_EVD_STATE or IT_ERR_INVALID_REQ_EVD_STATE.

2404 All Connection Events for the Endpoint are reported to the Consumer through the SEVD
2405 specified in *connect_sevd_handle*. For a complete list of Endpoint Connection Events, see
2406 *it_cm_msg_events*.

2407 The *flags* parameter allows the Consumer to control the behavior of the Implementation on
2408 Endpoint creation. Setting the IT_EP_REUSEADDR bit in *flags* allows the Consumer to specify
2409 that they allow the Implementation to return an Endpoint on creation that is possibly in the
2410 TimeWait state. Normally, the Implementation will only return Endpoints that are not in the
2411 TimeWait state. Setting the IT_EP_SRQ bit in *flags* allows the Consumer to specify that they
2412 wish to associate an S-RQ with the Endpoint that they are creating. If the Consumer sets this bit,
2413 then the *srq*, *soft_hi_watermark*, and *hard_hi_watermark* members of the input *ep_attr* structure
2414 are assumed to be valid; if this bit is cleared, these members of the input *ep_attr* structure shall
2415 be ignored by the Implementation.

2416 The TimeWait state exists for the purpose of preventing packets that were transmitted over one
2417 Connection from being inadvertently received in another subsequently established Connection.
2418 The TimeWait state is not a state of the Endpoint *per se*, but rather a state associated with a
2419 Connection the Endpoint had previously established. A Connection enters the TimeWait state
2420 when a disconnect is performed, and exits the TimeWait state after a TimeWait interval has
2421 elapsed. The duration of the TimeWait interval is transport-dependent, and for some transports it
2422 is also dependent upon network configuration parameters. This interval can be in the order of a
2423 minute or two in length.

2424 An Endpoint that is “in the TimeWait state” still has at least one Connection that it had
2425 previously established for which the TimeWait interval has not elapsed. (It is possible for an
2426 Endpoint to be in the TimeWait state with respect to multiple Connections it had previously
2427 established.) If an Endpoint attempts to establish a Connection that will use the same pair of
2428 Spigots that were involved in a previous Connection involving the Endpoint, and if that previous
2429 Connection is currently in the TimeWait state, the Connection establishment attempt may fail
2430 with an IT_ERR_EP_TIMEWAIT error; see *it_ep_accept*. This error will never be returned
2431 unless the Consumer sets the IT_EP_REUSEADDR bit in *flags* for the *it_ep_rc_create* routine.

2432 If the Consumer wishes to associate an S-RQ with the Endpoint being created, the following
2433 attributes in the *ep_attr* structure must be initialized appropriately:

- 2434 • *srq*
- 2435 • *soft_hi_watermark*
- 2436 • *hard_hi_watermark*

2437 See *it_ep_attributes* for details of how to initialize these attributes.

2438 The *ep_attr* parameter specifies the Consumer-requested attributes of the created Endpoint. The
2439 Implementation is required to satisfy all requested attributes or fail the operation. Hence, the
2440 Implementation must allocate all necessary resources to satisfy Consumer-requested attributes.
2441 The Implementation is allowed to allocate more resources than the Consumer requested in
2442 *ep_attr*. The Consumer can find the actual allocated resources by using *it_ep_query*. For detailed
2443 Endpoint attributes see the reference page for *it_ep_attributes*.

2444 RETURN VALUE

2445 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

2446	IT_ERR_INVALID_PZ	The Protection Zone Handle (<i>pz_handle</i>) was
2447		invalid.
2448	IT_ERR_INVALID_REQ_EVD	The Simple Event Dispatcher Handle for Data
2449		Transfer Operation request completions
2450		(<i>request_sevd_handle</i>) was invalid.
2451	IT_ERR_INVALID_RECV_EVD	The Simple Event Dispatcher Handle for Data
2452		Transfer Operation Receive completions
2453		(<i>recv_sevd_handle</i>) was invalid.
2454	IT_ERR_INVALID_CONN_EVD	The Connection Simple Event Dispatcher Handle
2455		was invalid.
2456	IT_ERR_INVALID_EVD_TYPE	The Event Stream type for the Event Dispatcher
2457		was invalid.
2458	IT_ERR_INVALID_REQ_EVD_STATE	The Simple Event Dispatcher for Data Transfer
2459		Operation request completions was in an unusable
2460		state.
2461	IT_ERR_INVALID_RECV_EVD_STATE	The Simple Event Dispatcher for Data Transfer
2462		Operation Receive completions was in an unusable
2463		state.
2464	IT_ERR_INVALID_FLAGS	The <i>flags</i> value was invalid.
2465	IT_ERR_RESOURCES	The requested operation failed due to insufficient
2466		resources.
2467	IT_ERR_PAYLOAD_SIZE	The requested <i>max_dto_payload_size</i> exceeds the
2468		maximum payload size supported by the
2469		underlying transport.

2470	IT_ERR_RESOURCE_REQ_DTO	The underlying transport could not allocate the requested <i>max_req_dtos</i> resources at this time.
2471		
2472	IT_ERR_RESOURCE_RECV_DTO	The underlying transport could not allocate the requested <i>max_recv_dtos</i> resources at this time.
2473		
2474	IT_ERR_RESOURCE_SSEG	The underlying transport could not allocate the requested <i>max_send_segments</i> resources at this time.
2475		
2476		
2477	IT_ERR_RESOURCE_RSEG	The underlying transport could not allocate the requested <i>max_recv_segments</i> resources at this time.
2478		
2479		
2480	IT_ERR_RESOURCE_RRSEG	The underlying transport could not allocate the requested <i>max_rdma_read_segments</i> resources at this time.
2481		
2482		
2483	IT_ERR_RESOURCE_RWSEG	The underlying transport could not allocate the requested <i>max_rdma_write_segments</i> resources at this time.
2484		
2485		
2486	IT_ERR_RESOURCE_IRD	The underlying transport could not allocate the requested <i>rdma_read_ird</i> resources at this time.
2487		
2488	IT_ERR_RESOURCE_ORD	The underlying transport could not allocate the requested <i>rdma_read_ord</i> resources at this time.
2489		
2490	IT_ERR_INVALID_SRQ	The S-RQ handle was invalid.
2491	IT_ERR_SOFT_HI_WATERMARK	The IA associated with the Endpoint does not support the Endpoint Soft High Watermark mechanism and a non-zero Endpoint Soft High Watermark was supplied.
2492		
2493		
2494		
2495	IT_ERR_INVALID_WATERMARK	The Endpoint Soft High Watermark supplied was larger than the maximum number of Receive DTOs that can be posted to the associated S-RQ.
2496		
2497		
2498	IT_ERR_HARD_HI_WATERMARK	The IA associated with the Endpoint does not support the Endpoint Hard High Watermark mechanism and a non-zero Endpoint Hard High Watermark was supplied.
2499		
2500		
2501		
2502	IT_ERR_IA_CATASTROPHE	The IA has experienced a catastrophic error and is in the disabled state. None of the output parameters from this routine are valid. See it_ia_info for a description of the disabled state.
2503		
2504		
2505		
2506	APPLICATION USAGE	
2507	Use of IT_EP_REUSEADDR	requires the Consumer to handle a potential
2508	IT_ERR_EP_TIMEWAIT	error from it_ep_accept if the Endpoint and an incoming Connection
2509	Request	are in the TimeWait state with respect to each other.

2510 Sometimes the required attribute values for an Endpoint depend on parameters in an incoming
2511 Connection Request and are not known at Endpoint creation time. The Consumer should specify
2512 these attributes at a later time using *it_ep_modify*; for example, before accepting an incoming
2513 Connection Request.

2514 Specifying an overflowed SEVD in *connect_sevd_handle* is recoverable but may result in
2515 connect Events being lost.

2516 **SEE ALSO**

2517 *it_ep_attributes_t, it_ep_ud_create(), it_ep_query(), it_ep_modify(), it_ep_free(), it_ep_accept(),*
2518 *it_cm_msg_events(), it_dto_events(), it_ia_info_t, it_srq_create()*

2519

it_ep_reset()

2520

2521 NAME

2522 `it_ep_reset` – reset a Reliable Connected Endpoint to the initial state

2523 SYNOPSIS

```
2524 #include <it_api.h>
2525
2526 it_status_t it_ep_reset(
2527     IN it_ep_handle_t ep_handle
2528 );
```

2529 APPLICABILITY

2530 `it_ep_reset` is applicable only to Endpoints created for the RC Service Type.

2531 DESCRIPTION

2532 `ep_handle` Reliable Connected Endpoint.

2533 `it_ep_reset` resets a Reliable Connected Endpoint into the `IT_EP_STATE_UNCONNECTED`
2534 state it had at original creation while maintaining the other attributes of the Endpoint in their
2535 current settings. `it_ep_reset` may only be applied to Reliable Connected Endpoints in the
2536 `IT_EP_STATE_NONOPERATIONAL` state. An Endpoint in the `IT_EP_STATE_`
2537 `NONOPERATIONAL` due to overflow of a DTO completion EVD cannot be reset.

2538 Upon return of this operation any Completion Events for the Endpoint not yet harvested by the
2539 Consumer may be dropped or not delivered to the EVD(s) associated with the Request or
2540 Receive Queue for the Endpoint. This operation is only needed if Consumers would like to re-
2541 use the Endpoint. Otherwise, they can just free the Endpoint using `it_ep_free`.

2542 RETURN VALUE

2543 A successful call returns `IT_SUCCESS`. Otherwise, an error code is returned as described below:

2544 `IT_ERR_INVALID_EP` The Endpoint Handle (`ep_handle`) was invalid.

2545 `IT_ERR_INVALID_EP_STATE` The Endpoint was not in the proper state for the
2546 attempted operation.

2547 `IT_ERR_INVALID_EP_TYPE` The attempted operation was invalid for the Service Type
2548 of the Endpoint.

2549 `IT_ERR_CANNOT_RESET` The Endpoint could not be reset due to an overflow of
2550 one of its Data Transfer Operation Event Stream Event
2551 Dispatchers.

2552 `IT_ERR_IA_CATASTROPHE` The IA has experienced a catastrophic error and is in the
2553 disabled state. None of the output parameters from this
2554 routine are valid. See `it_ia_info_t` for a description of the
2555 disabled state.

2556 SEE ALSO

2557 `it_ep_rc_create()`, `it_ep_disconnect()`, `it_ep_free()`, `it_ia_info_t`

it_ep_ud_create()

2558

2559 NAME

2560

ep_ud_create – create an Endpoint for Unreliable Datagram

2561 SYNOPSIS

2562

```
#include <it_api.h>
```

2563

```
it_status_t it_ep_ud_create (  
    IN          it_pz_handle_t      pz_handle,  
    IN          it_evd_handle_t     request_sevd_handle,  
    IN          it_evd_handle_t     rcv_sevd_handle,  
    IN const    it_ep_attributes_t  *ep_attr,  
    IN          size_t              spigot_id,  
    OUT         it_ep_handle_t      *ep_handle  
);
```

2571

2572 APPLICABILITY

2573

it_ep_ud_create is applicable only to create Endpoints of the UD Service Type.

2574 DESCRIPTION

2575

pz_handle Handle for the Protection Zone of the created Endpoint. Implicitly identifies the Interface Adapter.

2576

2577

request_sevd_handle: Handle for the IT_DTO_EVENT_STREAM Simple Event Dispatcher for DTO request Completion Events of the created Endpoint.

2578

2579

rcv_sevd_handle: Handle for the IT_DTO_EVENT_STREAM Simple Event Dispatcher for DTO Receive Completion Events of the created Endpoint.

2580

2581

ep_attr Pointer to a structure that contains Consumer-requested Endpoint Attributes.

2582

2583

spigot_id Interface Adapter Spigot identifier to use when creating Endpoint.

2584

ep_handle Handle for the created Endpoint.

2585

it_ep_ud_create creates, on the requested *spigot_id* of the Interface Adapter implicitly identified by *pz_handle*, an Unreliable Datagram Endpoint that is provided to the Consumer as *ep_handle*. The value of *ep_handle* is only defined if the return value is IT_SUCCESS.

2586

2587

2588

The Unreliable Datagram Endpoint is created in the IT_EP_STATE_UD_OPERATIONAL state. See *it_ep_state_t* for details.

2589

2590

Protection Zone *pz_handle* allows Consumers to control what local memory the Endpoint can access for DTOs. Only memory referred to by LMRs that match the Endpoint Protection Zone can be accessed by the Endpoint.

2591

2592

2593

rcv_sevd_handle and *request_sevd_handle* are Event Dispatcher instances where the Consumer collects completion Notifications of DTOs. Completions of Receive DTOs are reported in the *rcv_sevd_handle* Event Dispatcher, and completions of Send DTOs are reported in *request_sevd_handle*. It is permissible for *rcv_sevd_handle* and *request_sevd_handle* to reference the same EVD. DTO Completion Events are defined in *it_dto_events*.

2594

2595

2596

2597

2598 The Consumer should not specify an SEVD in *recv_sevd_handle* or *request_sevd_handle* that is
 2599 in overflowed state for use in the Endpoint creation call (see *it_evd_create* for more details on
 2600 overflow). If the Consumer attempts to do so, the operation will fail with
 2601 IT_ERR_INVALID_RECV_EVD_STATE or IT_ERR_INVALID_REQ_EVD_STATE.

2602 The *ep_attr* parameter specifies the Consumer-requested attributes of the created Endpoint. The
 2603 Implementation is required to satisfy all requested attributes or fail the operation. Hence, the
 2604 Implementation must allocate all necessary resources to satisfy Consumer-requested attributes.
 2605 The Implementation is allowed to allocate more resources than the Consumer requested in
 2606 *ep_attr*. The Consumer can find the actual allocated resources by using *it_ep_query*. For detailed
 2607 Endpoint attributes see the reference page for *it_ep_attributes_t*.

2608 **RETURN VALUE**

2609 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

2610	IT_ERR_INVALID_PZ	The Protection Zone Handle (<i>pz_handle</i>) was invalid.
2611		
2612	IT_ERR_INVALID_REQ_EVD	The Simple Event Dispatcher Handle for Data Transfer Operation request completions (<i>request_sevd_handle</i>) was invalid.
2613		
2614		
2615	IT_ERR_INVALID_RECV_EVD	The Simple Event Dispatcher Handle for Data Transfer Operation Receive completions (<i>recv_sevd_handle</i>) was invalid.
2616		
2617		
2618	IT_ERR_INVALID_EVD_TYPE -	The Event Stream Type for the Event Dispatcher was invalid.
2619		
2620	IT_ERR_INVALID_REQ_EVD_STATE	The Simple Event Dispatcher for Data Transfer Operation request completions was in an unusable state.
2621		
2622		
2623	IT_ERR_INVALID_RECV_EVD_STATE	The Simple Event Dispatcher for Data Transfer Operation Receive completions was in an unusable state.
2624		
2625		
2626	IT_ERR_INVALID_SPIGOT	An invalid Spigot ID was specified.
2627	IT_ERR_RESOURCES	The requested operation failed due to insufficient resources.
2628		
2629	IT_ERR_PAYLOAD_SIZE	The requested <i>max_dto_payload_size</i> exceeds the maximum payload size supported by the underlying transport.
2630		
2631		
2632	IT_ERR_RESOURCE_REQ_DTO	The underlying transport could not allocate the requested <i>max_req_dtos</i> resources at this time.
2633		
2634	IT_ERR_RESOURCE_RECV_DTO	The underlying transport could not allocate the requested <i>max_recv_dtos</i> resources at this time.
2635		
2636	IT_ERR_RESOURCE_SSEG	The underlying transport could not allocate the requested <i>max_send_segments</i> resources at this time.
2637		

2638	IT_ERR_RESOURCE_RSEG	The underlying transport could not allocate the requested <i>max_rcv_segments</i> resources at this time.
2639		
2640	IT_ERR_INVALID_EP_KEY	Invalid Endpoint Key value. The Consumer doesn't have local permissions to use the specified Endpoint Key.
2641		
2642		
2643	IT_ERR_IA_CATASTROPHE	The IA has experienced a catastrophic error and is in the disabled state. None of the output parameters from this routine are valid. See <i>it_ia_info_t</i> for a description of the disabled state.
2644		
2645		
2646		

2647 **SEE ALSO**

2648 *it_ep_attributes_t, it_ep_rc_create(), it_ep_query(), it_ep_modify(), it_ep_free(), it_ep_state_t,*
 2649 *it_dto_events, it_ia_info_t*

2650

it_evd_create()

2651

2652 NAME

2653 `it_evd_create` – create Simple or Aggregate Event Dispatcher

2654 SYNOPSIS

```
2655 #include <it_api.h>
2656
2657 it_status_t evd_create (
2658     IN      it_ia_handle_t   ia_handle,
2659     IN      it_event_type_t  event_number,
2660     IN      it_evd_flags_t   evd_flag,
2661     IN      size_t           sevd_queue_size,
2662     IN      size_t           sevd_threshold,
2663     IN      it_evd_handle_t  aevd_handle,
2664     OUT     it_evd_handle_t  *evd_handle,
2665     OUT     int              *fd
2666 );
2667
2668 #define IT_THRESHOLD_DISABLE 0
2669
2670 typedef enum {
2671     IT_EVD_DEQUEUE_NOTIFICATIONS = 0x01,
2672     IT_EVD_CREATE_FD             = 0x02,
2673     IT_EVD_OVERFLOW_DEFAULT      = 0x04,
2674     IT_EVD_OVERFLOW_NOTIFY       = 0x08,
2675     IT_EVD_OVERFLOW_AUTO_RESET   = 0x10
2676 } it_evd_flags_t;
```

2677 DESCRIPTION

2678	<i>ia_handle</i>	Handle for the Interface Adapter to which the created Event Dispatcher belongs.
2679		
2680	<i>event_number</i>	Identifier for Event Stream type that can be enqueued to the EVD.
2681	<i>evd_flag</i>	Bitwise OR of flag values for creation operation.
2682	<i>sevd_queue_size</i>	Minimum size of the Simple EVD Event queue. This parameter is ignored for an Aggregate EVD.
2683		
2684	<i>sevd_threshold</i>	Number of Events on the Simple EVD queue required for Notification of the associated AEVD or <i>fd</i> and for SEVD waiters unblocking. This parameter is ignored for Aggregate EVD.
2685		
2686		
2687	<i>aevd_handle</i>	Optional Handle to associate an Aggregate EVD with the Simple EVD. This parameter must be <code>IT_NULL_HANDLE</code> when the <code>IT_EVD_CREATE_FD</code> <i>evd_flag</i> is set. This parameter must also be <code>IT_NULL_HANDLE</code> when using <code>it_evd_create</code> to create an Aggregate EVD.
2688		
2689		
2690		
2691		
2692	<i>evd_handle</i>	Handle for the created Event Dispatcher.

2693 *fd* Pointer to an optional file descriptor corresponding to the Event Dispatcher.
2694 Only valid if return value is IT_SUCCESS and the IT_EVD_CREATE_FD
2695 *evd_flag* was set.

2696 *it_evd_create* creates an instance of an Event Dispatcher (EVD) that is provided to the
2697 Consumer as *evd_handle*. Two different types of EVDs are supported by the Implementation:
2698 Simple EVDs (SEVD) and Aggregate EVDs (AEVD). An SEVD is an EVD for a single Event
2699 Stream. An AEVD is an aggregation of SEVDs and thus can potentially return Events for more
2700 than one Event Stream type. *it_evd_create* can also optionally return a file descriptor (*fd*)
2701 associated with an EVD.

2702 The values of *evd_handle* and *fd* are only defined if the return value is IT_SUCCESS.

2703 The scope of an EVD is a single Interface Adapter identified by *ia_handle*.

2704 *event_number* identifies the type of Event Stream that the created EVD will handle. Multiple
2705 Event Streams of the same Event Stream type (such as DTO Completion Event Streams) can
2706 feed the EVD. Event Stream types are defined in *it_event_t*.

2707 To create an Aggregate EVD, the *event_number* must be set to IT_AEVD_NOTIFICATION_
2708 EVENT_STREAM; a Simple EVD (SEVD) is created otherwise. To create a Simple EVD the
2709 *event_number* can be any one of IT.DTO_EVENT_STREAM, IT_CM_REQ_
2710 EVENT_STREAM, IT_CM_MSG_EVENT_STREAM, IT_ASYNC_AFF_EVENT_STREAM,
2711 IT_ASYNC_UNAFF_EVENT_STREAM, or IT_SOFTWARE_EVENT_STREAM.

2712 A Simple EVD may feed only one Aggregate EVD. An Aggregate EVD may be fed by many
2713 Simple EVDs. The Consumer may create multiple AEVDs and SEVDs with the following two
2714 exceptions:

- 2715 1. Only one IT_ASYNC_AFF_EVENT_STREAM Simple EVD may be created per
2716 Interface Adapter instance. Subsequent calls to *it_evd_create* for the
2717 IT_ASYNC_AFF_EVENT_STREAM Event Stream, without intervening calls to
2718 *it_evd_free* the EVD, will fail with the error return IT_ERR_ASYNC_AFF_
2719 EVD_EXISTS.
- 2720 2. Only one IT_ASYNC_UNAFF_EVENT_STREAM Simple EVD may be created per
2721 Interface Adapter instance. Subsequent calls to *it_evd_create* for the
2722 IT_ASYNC_UNAFF_EVENT_STREAM Event Stream, without intervening calls to
2723 *it_evd_free* the EVD, will fail with the error return IT_ERR_ASYNC_UNAFF_
2724 EVD_EXISTS.

2725 For all Event Stream types except IT_SOFTWARE_EVENT_STREAM, IT_ASYNC_
2726 AFF_EVENT_STREAM, and IT_ASYNC_UNAFF_EVENT_STREAM, upon creation there is
2727 no Event Stream of *event_number* feeding Events to the created EVD. For an Aggregate EVD
2728 this means that there are no Simple EVDs associated with *evd_handle*. No Events are fed to
2729 *evd_handle* until *evd_handle* is associated with an object that feeds Events to it. For an
2730 Aggregate EVD this means that no Events are fed to *evd_handle* until *evd_handle* is associated
2731 with a Simple EVD. For a Simple EVD this means that no Events are fed to *evd_handle* until
2732 *evd_handle* is associated with an Endpoint, Listen Point, or UD Service Request Handle
2733 depending on the stream type.

2734 For the IT_ASYNC_AFF_EVENT_STREAM Event Stream type, the Simple EVD receives the
2735 Async Affiliated Events for the *ia_handle*. For the IT_ASYNC_UNAFF_EVENT_STREAM
2736 Event Stream type, the Simple EVD receives the Async Unaffiliated Events for the *ia_handle*.

2737 Multiple Event Streams of the same Event Stream type can be associated with the same EVD,
2738 with the exception of IT_ASYNC_AFF_EVENT_STREAM, IT_ASYNC_UNAFF_EVENT_
2739 STREAM, and IT_SOFTWARE_EVENT_STREAM Event Stream types. For the IT_AEVD_
2740 NOTIFICATION_EVENT_STREAM Event Stream type, multiple SEVDs can be associated
2741 with the same AEVD. For IT_DTO_EVENT_STREAM, multiple EPs can be associated with the
2742 same SEVD. For IT_CM_REQ_EVENT_STREAM, multiple Listens can be associated with the
2743 same SEVD. For IT_CM_MSG_EVENT_STREAM, multiple RC EPs and/or UD Service
2744 Requests can be associated with the same SEVD. For the IT_ASYNC_AFF_EVENT_STREAM,
2745 IT_ASYNC_UNAFF_EVENT_STREAM, and IT_SOFTWARE_EVENT_STREAM Event
2746 Stream types, only a single Event Stream feeds each EVD, respectively, and the corresponding
2747 Event Stream is created upon EVD creation. For IT_SOFTWARE_EVENT_STREAM, the
2748 Events are generated explicitly by the Consumer calling *it_evd_post_se*.

2749 When the Implementation attempts to enqueue more Events on an SEVD than the queue size of
2750 the SEVD will permit, the SEVD is said to overflow. An AEVD cannot overflow.

2751 Once an SEVD overflows, subsequent Events from the Event Stream will be dropped. For all
2752 Event Streams, with the exception of IT_DTO_EVENT_STREAM, Events will no longer be
2753 dropped once the Consumer makes more space available in the SEVD's Event queue. The
2754 Consumer can make room in an SEVD either by dequeuing an Event, or by using
2755 *it_evd_modify* to increase the queue size of the SEVD. For IT_DTO_EVENT_STREAM,
2756 however, Events will continue to be dropped; the overflow cannot be corrected.

2757 The behavior of an SEVD after an overflow depends upon the Event Stream associated with the
2758 SEVD, and upon whether the default overflow behavior has been configured for the SEVD. The
2759 reference page associated with each Event Stream type provides details of the default overflow
2760 behavior. The Consumer specifies default overflow behavior by setting the
2761 IT_EVD_OVERFLOW_DEFAULT *evd_flag* value.

2762 If default overflow behavior is not configured (IT_EVD_OVERFLOW_DEFAULT is cleared in
2763 *evd_flag*), then the Consumer can control two possible parameters: Whether the overflow
2764 occurrence causes generation (IT_EVD_OVERFLOW_NOTIFY flag value) of an overflow
2765 Event on the Affiliated or Unaffiliated SEVD, and, if configured, how the generation of the
2766 overflow Event is controlled (IT_EVD_OVERFLOW_AUTO_RESET flag value). Each
2767 subsequent SEVD Event that arrives after overflow of the SEVD initially occurs can potentially
2768 generate an overflow Event.

2769 The Consumer can request that an overflow Event be generated when an overflow occurs by
2770 setting IT_EVD_OVERFLOW_NOTIFY in *evd_flag*. For SEVDs associated with any Event
2771 Streams other than the IT_ASYNC_AFF_EVENT_STREAM or the IT_ASYNC_UNAFF_
2772 EVENT_STREAM, an IT_ASYNC_AFF_SEVD_ENQUEUE_FAILURE Event is enqueued on
2773 the affiliated asynchronous error Event Stream of *ia_handle*. For an SEVD associated with the
2774 IT_ASYNC_AFF_EVENT_STREAM, an IT_ASYNC_UNAFF_SEVD_ENQUEUE_FAILURE
2775 Event is enqueued on the unaffiliated asynchronous error Event Stream of *ia_handle*. The Event
2776 identifies the SEVD that overflowed. Overflow of the IT_ASYNC_UNAFF_EVENT_
2777 STREAM is never detected and no indication of such overflow is ever generated; however, no
2778 adverse consequences occur other than the dropping of some Unaffiliated Events.

2779 If an SEVD overflow has occurred, the *evd_overflowed* member of the *it_evd_param_t* structure
2780 (as returned by the *it_evd_query* routine) will have an IT_TRUE value until the condition is
2781 corrected or manually changed. When the Consumer creates an SEVD to hold Events of an
2782 Event Stream and has enabled generation of overflow Events on the SEVD

2783 (IT_EVD_OVERFLOW_NOTIFY flag value), the Consumer must chose one of two modes for
2784 generation of overflow Events using the IT_EVD_OVERFLOW_AUTO_RESET flag:
2785 automatic, or Consumer-controlled. In automatic mode, overflow Events may again be enqueued
2786 on the Affiliated or Unaffiliated SEVD as soon as the Consumer makes more space available in
2787 the EVD's Event queue. In Consumer-controlled generation, overflow Events are only again
2788 generated after the Consumer calls *it_evd_modify* to clear the *evd_overflowed* field. See
2789 *it_evd_modify* for more details.

2790 Note that even if overflow generation is disabled, the Consumer may still clear *evd_overflowed*
2791 using *it_evd_modify*. A subsequent overflow will again set the *evd_overflowed* member of the
2792 *it_evd_param_t* structure.

2793 For a newly created SEVD, the *evd_overflowed* member of the *it_evd_param_t* structure is not
2794 set.

2795 The *evd_flag* value of IT_EVD_DEQUEUE_NOTIFICATIONS applies only to AEVDs.

2796 When the IT_EVD_DEQUEUE_NOTIFICATIONS bit is set in *evd_flag*, then wait and dequeue
2797 operations on the AEVD will dequeue IT_AEVD_NOTIFICATION_EVENT_STREAM
2798 Events; such Events provide the SEVD Handle of the underlying SEVD that caused the
2799 Notification. To retrieve the underlying Event, the Consumer must call *it_evd_dequeue* on the
2800 SEVD Handle provided in the IT_AEVD_NOTIFICATION_EVENT_STREAM Event from the
2801 AEVD.

2802 When the IT_EVD_DEQUEUE_NOTIFICATIONS bit is cleared in *evd_flag*, then calling
2803 *it_evd_wait* on the AEVD directly returns the first Event from a notifying underlying SEVD
2804 (such as IT_DTO_EVENT_STREAM Events, etc.). The dequeue operation on the AEVD
2805 directly returns the first Event from an underlying SEVD. These Events will be of whatever
2806 Event Stream types that feed each of these associated SEVDs. The associated SEVD can be
2807 determined from the *evd_handle* found in every Event.

2808 If an underlying SEVD of an AEVD has been disabled, then the SEVD will no longer generate
2809 Notification Events for the AEVD until the SEVD is enabled (see *it_evd_modify*). Previously
2810 generated SEVD Notifications for the AEVD are unaffected by the enabling and disabling of
2811 SEVD.

2812 For a Simple EVD that does not have an associated AEVD, the Consumer can wait on and
2813 dequeue from the SEVD.

2814 If the SEVD has an associated AEVD with the IT_EVD_DEQUEUE_NOTIFICATIONS
2815 *evd_flag* cleared, then it is an error for the Consumer to wait on or dequeue from the SEVD.
2816 Attempting to wait on or dequeue from the SEVD will return IT_ERR_INVALID_EVD_
2817 STATE.

2818 If the SEVD has an associated AEVD with the IT_EVD_DEQUEUE_NOTIFICATIONS
2819 *evd_flag* set, then the Consumer can always dequeue from the SEVD, and the Consumer can
2820 wait on the SEVD, but only if they disable the SEVD first (see *it_evd_modify*). Attempting to
2821 wait on the SEVD when disallowed will return IT_ERR_INVALID_EVD_STATE.

2822 The *evd_flag* bit value of IT_EVD_CREATE_FD set indicates that the Consumer requests
2823 creation of a File Descriptor associated with the EVD (either SEVD or AEVD). If the EVD has
2824 an associated *fd*, then the Consumer can wait on the EVD if they disable the EVD first (see
2825 *it_evd_modify*). Attempting to wait on the EVD when disallowed will return

2826
2827

IT_ERR_INVALID_EVD_STATE. If the EVD has an associated *fd*, then the Consumer can dequeue from the feeding EVD.

2828
2829

Values for *evd_flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in `<it_api.h>`.

Flag Value	Description
IT_EVD_DEQUEUE_NOTIFICATIONS	Only applicable to AEVD. When set, wait and dequeue on the AEVD shall dequeue IT_AEVD_NOTIFICATION_EVENT_STREAM Events from the created AEVD. Otherwise, wait and dequeue on the AEVD will dequeue the underlying Events (of potentially various Event Stream types) from the SEVDs that feed the AEVD.
IT_EVD_CREATE_FD	The Implementation will allocate and return a file descriptor usable as a Notification object for this EVD. It is an error to set this flag as well as specify an AEVD for an SEVD.
IT_EVD_OVERFLOW_DEFAULT	Only applicable to an SEVD. When set, the overflow behavior for the SEVD will be the default behavior for the <i>event_number</i> as specified in the reference page for the Event Stream. When clear, the behavior is determined by how the IT_EVD_OVERFLOW_NOTIFY and IT_EVD_OVERFLOW_AUTO_RESET flags are set. It is an error to set this flag as well as IT_EVD_OVERFLOW_NOTIFY and/or IT_EVD_OVERFLOW_AUTO_RESET.
IT_EVD_OVERFLOW_NOTIFY	Only applicable to an SEVD. When clear, EVD overflow is ignored. When set, causes an IT_ASYNC_AFF_SEVD_ENQUEUE_FAILURE Event to be generated if the EVD overflows if <i>event_number</i> is anything other than IT_ASYNC_AFF_EVENT_STREAM or IT_ASYNC_UNAFF_EVENT_STREAM. Causes an IT_ASYNC_UNAFF_SEVD_ENQUEUE_FAILURE Event to be generated if the EVD overflows and <i>event_number</i> is IT_ASYNC_AFF_EVENT_STREAM. It is invalid to set this flag if <i>event_number</i> is IT_ASYNC_UNAFF_EVENT_STREAM. It is invalid to set both this flag and IT_EVD_OVERFLOW_DEFAULT.

Flag Value	Description
IT_EVD_OVERFLOW_AUTO_RESET	Only applicable to an SEVD. When set, this flag specifies that the Implementation will automatically reset overflow Event generation (i.e., when the Consumer makes space available, further Events that again overflow EVD will cause another overflow Event to attempt to be queued to the Affiliated or Unaffiliated SEVD); when clear, this flag specifies that the Consumer must manually reset the <i>evd_overflowed</i> state of the EVD (see <i>it_evd_modify</i>) and the Implementation shall not reset EVD overflow Event generation on its own. It is invalid to set both this flag and IT_EVD_OVERFLOW_DEFAULT. If IT_EVD_OVERFLOW_NOTIFY is not set, it is an error to set this flag. Since a DTO overflow cannot be corrected, it is an error to set this flag for the DTO Event Stream.

2830
2831
2832
2833
2834
2835

sevd_queue_size is only applicable for a Simple EVD. It defines the size of the Event queue that the Consumer requested. The Implementation is required to provide a queue size of at least *sevd_queue_size*, but is free to provide a larger queue size. The Consumer can determine the actual queue size by querying the created Simple Event Dispatcher. This parameter is ignored for Aggregate EVD.

2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847

The *sevd_threshold* is only applicable to an SEVD and allows the Consumer to request an accumulation of up to *sevd_threshold* number of enqueued “non-Notification Events” for the Simple EVD queue prior to waking up the Consumer or notifying *fd* or *aevd_handle*. A “non-Notification Event” is one of the following: An Event with *dto_status* of IT_DTO_SUCCESS corresponding to a non-Recv DTO that was posted with the IT_NOTIFY_FLAG bit cleared. An Event with *dto_status* of IT_DTO_SUCCESS corresponding to a Recv DTO that was posted with the IT_NOTIFY_FLAG bit cleared and with the IT_SOLICITED_WAIT_FLAG bit cleared in the corresponding remote Send. See *it_dto_flags_t* for more details. Only DTO Event Streams support non-notification Events; on all other Event Streams, every Event is a Notification Event (thus thresholds have no function on non-DTO Event Streams). Arrival of a “Notification Event” before *sevd_threshold* number of non-notification Events have arrived will cause wakeup or Notification.

2848

A “Notification Event” is one of the following:

2849
2850
2851
2852

- An Event corresponding to a DTO that was posted with the IT_NOTIFY_FLAG bit set
- An Event with a *dto_status* that is not IT_DTO_SUCCESS
- An Event corresponding to a Recv DTO with the IT_SOLICITED_WAIT_FLAG bit set in the corresponding remote Send

2853
2854
2855
2856

- Any Event of Event Stream other than IT_DTO_EVENT_STREAM

An SEVD is in the “notification criteria” when one of the following is true: There is a Notification Event queued on the SEVD. The number of Events on SEVD is larger or equal to the *sevd_threshold*.

2857 For SEVD, the *sevd_threshold* must be set to either the value `IT_THRESHOLD_DISABLE` or
2858 to a value between at least one and at most the actual Implementation-allocated *sevd_queue_size*
2859 (can be determined by querying the SEVD). Setting *sevd_threshold* to
2860 `IT_THRESHOLD_DISABLE` will cause *it_evd_wait* to return only for Notification Events
2861 (specifically not for a threshold number of Events). For AEVD, the *sevd_threshold* is ignored.

2862 An *aevd_handle* specified on creation of a Simple EVD allows a Consumer to consolidate
2863 Notifications from multiple Simple Event Dispatchers (from the same Interface Adapter) to a
2864 single higher-level Aggregate Event Dispatcher. For an SEVD the *aevd_handle* value of
2865 `IT_NULL_HANDLE` means that no AEVD is associated with nor fed by the created SEVD. For
2866 Aggregate EVD creation this parameter must be `IT_NULL_HANDLE`; otherwise, *it_evd_create*
2867 will return `IT_ERR_AEVD_NOT_ALLOWED`.

2868 Alternatively, if the `IT_EVD_CREATE_FD` *evd_flag* bit value is set, then the Implementation
2869 will return a new unique file descriptor associated with the EVD. The file descriptor is placed
2870 into the contents of the *fd* pointer. The *fd* may be used in *select()* or *poll()* system calls and will
2871 be identified as ready to read when a Notification occurs on the underlying EVD. It is up to a
2872 Consumer then to go and dequeue Events from the EVD which is one-to-one associated with the
2873 particular *fd*. It is the Consumer's responsibility to keep track of the one-to-one association of *fd*
2874 and EVD.

2875 For Simple EVD the use of a value other than `IT_NULL_HANDLE` for *aevd_handle* is mutually
2876 exclusive with use of the `IT_EVD_CREATE_FD` *evd_flag*. That is, specifying both an
2877 *aevd_handle* not equal to `IT_NULL_HANDLE` and the `IT_EVD_CREATE_FD` bit set in
2878 *evd_flag* in a call to *it_evd_create* will fail and return a value of `IT_ERR_MISMATCH_FD`.

2879 IT-API supports the following configurations: Simple EVD, Simple EVD with associated *fd*,
2880 Simple EVD feeding Aggregate EVD, and Simple EVD feeding Aggregate EVD that is
2881 associated with *fd*.

2882 The Aggregate EVD specified by *aevd_handle* or the *fd* will be notified by the Implementation
2883 when a Notification Event arrives or *sevd_threshold* value is reached when the EVD is enabled.

2884 When an SEVD feeds an AEVD or *fd*, control of the capability of the feeding EVD to notify the
2885 fed AEVD or *fd* is done by enabling or disabling the feeding SEVD (see *it_evd_modify*).

2886 By default, the created EVD is enabled. An enabled SEVD will cause *aevd_handle* or *fd* (if
2887 applicable) to be notified when an Event arrival causes Notification criteria to be reached on that
2888 SEVD. An enabled AEVD will cause *fd* (if applicable) to be notified when an Event arrival
2889 causes Notification criteria to be reached. Notification is done on *aevd_handle* by generating an
2890 `IT_AEVD_NOTIFICATION_EVENT` for the AEVD if the `IT_EVD_DEQUEUE_`
2891 `NOTIFICATIONS` *evd_flag* bit is set on AEVD creation. When Notification is necessary for an
2892 AEVD with the `IT_EVD_DEQUEUE_`
2893 `NOTIFICATIONS` *evd_flag* bit cleared, no `IT_AEVD_`
2894 `NOTIFICATION_EVENT` will be enqueued; rather, the AEVD Consumer will be unblocked
2895 with the underlying SEVD Event delivered to it. Notification is done on the *fd* by marking it as
ready to read.

2896 Consumers cannot wait on an enabled SEVD that feeds an AEVD or *fd*.

2897 A disabled feeding EVD will not generate Notification to the fed AEVD or *fd*. Consumers can
2898 wait on a disabled SEVD, unless it is associated with an AEVD with the `IT_EVD_`
2899 `DEQUEUE_`
`NOTIFICATIONS` *evd_flag* bit cleared.

2900 An SEVD preserves the order of Events within each individual Event Stream as provided by the
 2901 underlying Transport. No order is defined between Events of different Event Streams, even
 2902 when they are of the same Event Stream type. For IT_DTO_EVENT_STREAM Event Stream
 2903 type, the order of the Event completions is defined for each DTO and RMR post-operation on
 2904 the Endpoint. No order is defined between Events of Event Streams coming from different
 2905 SEVDs for an AEVD with the IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* bit cleared.
 2906 The order of Events of the IT_AEVD_NOTIFICATION_EVENT Event Stream is
 2907 Implementation-dependent.

2908 If the IT_EVD_DEQUEUE_NOTIFICATIONS bit is cleared in *evd_flag* on AEVD creation, the
 2909 Consumer, when blocked in *it_evd_wait* and an SEVD Notification occurs, is unblocked and
 2910 dequeues a lower-level Event from the same SEVD that caused Notification.

2911 Multiple SEVDs can feed the same AEVD. An SEVD generates a Notification for an AEVD
 2912 when an Event arriving on the SEVD causes the SEVD to reach Notification criteria (but only if
 2913 the SEVD is enabled).

2914 SEVD and AEVD can support multiple waiters. For an SEVD the *sevd_threshold* value must be
 2915 one for multiple waiters to be supported.

2916 An SEVD waiter will block when the SEVD queue is empty. An AEVD waiter will block when
 2917 all associated SEVDs are empty. An SEVD waiter may block when the SEVD is not in the
 2918 Notification criteria. An AEVD waiter may block when all associated SEVDs are not in the
 2919 Notification criteria. An SEVD waiter will return if there is a Notification Event on the queue or
 2920 if the number of Events on the SEVD is equal or larger then *threshold*. An AEVD waiter will
 2921 return if there is a Notification Event on any of the associated SEVDs or any of the associated
 2922 SEVDs has a number of Events larger or equal to its *sevd_threshold*.

2923 If an arriving Event causes an SEVD to reach Notification criteria, then an SEVD waiter will be
 2924 unblocked, if one exists. If there are multiple waiters on the SEVD, as many waiters as there are
 2925 Events available on the SEVD may be unblocked. If the SEVD is enabled and associated with an
 2926 AEVD or *fd*, then Notification will be generated for that AEVD or *fd*. In the AEVD case, as
 2927 many Notifications may be generated as there are Events available on all SEVDs of the AEVD.

2928 *it_evd_dequeue* from an SEVD will return an Event, if one exists, from the SEVD queue
 2929 regardless of whether there are waiters, except when the SEVD is associated with an AEVD with
 2930 the IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* bit cleared. In the latter case, dequeue
 2931 from the SEVD is not allowed. For an AEVD with the IT_EVD_DEQUEUE_NOTIFICATIONS
 2932 *evd_flag* bit cleared, dequeue from the AEVD will return an Event, if one exists, from any of its
 2933 associated SEVDs. For an AEVD with the IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag*
 2934 bit set, dequeue from the AEVD will return an IT_AEVD_NOTIFICATION_EVENT if any of
 2935 the associated enabled SEVDs is in Notification criteria or may return an
 2936 IT_AEVD_NOTIFICATION_EVENT if any of the associated enabled SEVD simply has an
 2937 Event.

2938 **RETURN VALUE**

2939 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

2940	IT_ERR_INVALID_IA	The Interface Adapter Handle (<i>ia_handle</i>) was invalid.
2941	IT_ERR_INVALID_EVD_TYPE	The Event Stream Type for the Event Dispatcher was
2942		invalid.

2943	IT_ERR_INVALID_FLAGS	The flags value was invalid.
2944 2945	IT_ERR_RESOURCE_QUEUE_SIZE	The underlying transport could not allocate the requested <i>sevd_queue_size</i> resources at this time.
2946 2947	IT_ERR_INVALID_THRESHOLD	An invalid value for the Simple Event Dispatcher threshold was specified.
2948 2949	IT_ERR_INVALID_AEVD	The Aggregation Event Dispatcher Handle (<i>aevd_handle</i>) was invalid.
2950 2951	IT_ERR_RESOURCES	The requested operation failed due to insufficient resources.
2952 2953	IT_ERR_MISMATCH_FD	An illegal request was made for both the File Descriptor and the Aggregation Event Dispatcher.
2954 2955 2956	IT_ERR_AEVD_NOT_ALLOWED	The <i>aevd_handle</i> was non-NULL and the <i>event_number</i> was IT_AEVD_NOTIFICATION_EVENT_STREAM.
2957 2958	IT_ERR_ASYNC_AFF_EVD_EXISTS	The Asynchronous Affiliated Event Dispatcher already exists.
2959 2960	IT_ERR_ASYNC_UNAFF_EVD_EXISTS	The Asynchronous Unaffiliated Event Dispatcher already exists.
2961 2962 2963 2964	IT_ERR_IA_CATASTROPHE	The IA has experienced a catastrophic error and is in the disabled state. None of the output parameters from this routine are valid. See <i>it_ia_info_t</i> for a description of the disabled state.

2965 **APPLICATION USAGE**

2966 Consumers may use SEVDs with a pure polling model. Consumers create SEVDs and dequeue
2967 from them directly. The Consumer threads never wait on the SEVDs and just dequeue Events
2968 when they are ready to process them.

2969 Alternatively, Consumers may create SEVDs and wait on and dequeue from them directly. This
2970 also potentially requires many waiting threads, one per SEVD.

2971 For the “non-thread-safe” Implementation the Consumer cannot have multiple threads calling on
2972 the same EVD Handle simultaneously. When multiple threads retrieve Events concurrently from
2973 the same SEVD, each Event will be retrieved exactly once but it is unpredictable which thread
2974 will retrieve any particular Event.

2975 The use of an AEVD can reduce the number of distinct waiting threads required for an
2976 application. EVDs must be enabled to generate Notifications for the AEVD.

2977 Consumers can wait on an AEVD that had been created with the
2978 IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* bit set and all feeding SEVDs enabled. When
2979 wait returns, a returned IT_AEVD_NOTIFICATION_EVENT_STREAM Event identifies the
2980 SEVD that caused the unblocking. Consumer can then dequeue Events directly from that SEVD
2981 or any other SEVD that feeds the AEVD. Thus, the Consumer can choose to service the SEVDs
2982 feeding the AEVD in any order they wish.

2983 Consumers can wait on an AEVD that had been created with the
2984 IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* bit cleared and all feeding SEVDs enabled.
2985 When wait returns, it provides the first Event from an SEVD that is in Notification status.
2986 Consumer can then dequeue Events only from the AEVD. This dequeuing will provide Events
2987 from all SEVDs that feed the AEVD.

2988 The order of returned Events from the AEVD is Implementation-dependent. If Events are
2989 retrieved from a given AEVD strictly by a single thread, the order of each Event from its
2990 underlying SEVDs is maintained, but the order in which SEVDs are selected by the AEVD is
2991 Implementation-dependent. If Events are retrieved from a given AEVD by more than one thread,
2992 no order guarantees are made.

2993 The use of a file descriptor can also reduce the number of distinct waiting threads. File
2994 descriptors also can be used to wait for Notification Events across multiple Interface Adapters or
2995 Events not generated by this API. EVDs must be enabled to generate Notifications for the file
2996 descriptor.

2997 Consumers can *select* or *poll* on multiple *fds* that are associated with EVDs. The return for the
2998 *select* or *poll* call identifies the notifying *fd*. It is the Consumer's responsibility to keep track of
2999 which EVD is associated with each *fd*. The Consumer can dequeue Events from the EVD one-to-
3000 one associated with that *fd* using *it_evd_dequeue*.

3001 Typically, if the Consumer chooses to use an AEVD, they are then prohibited from waiting on
3002 the underlying SEVDs (see the Description section above for exceptions) and also may be
3003 prohibited from dequeuing from the underlying SEVDs (again see the Description section
3004 above for details). If the Consumer chooses to use an *fd*, then they are prohibited from waiting
3005 on the underlying AEVD(s) or SEVD(s).

3006 Overflow may occur and may not be reported to the Consumer via Events if there is no Simple
3007 EVD for IT_ASYNC_AFF_EVENT_STREAM or IT_ASYNC_UNAFF_EVENT_STREAM.
3008 Additionally, an IA can enter catastrophic state and not notify the Consumer about it if there is
3009 no Simple EVD for IT_ASYNC_UNAFF_EVENT_STREAM or if it has overflowed. For the
3010 effect of catastrophic error, see *it_unaffiliated_event_t* and *it_ia_create*.

3011 When an IA supports Spigot *online* or *offline* Events the number of Events that can be generated
3012 for the Unaffiliated Asynchronous Event Stream is potentially unbounded, but the queuing
3013 capacity of an EVD is finite. This can potentially lead to Events that are generated for the
3014 Unaffiliated Asynchronous Event Stream being silently discarded by the Implementation. Events
3015 that are generated for the Affiliated (or Unaffiliated) Asynchronous Event Stream will be silently
3016 discarded by the Implementation until such time as an EVD is created to hold the Affiliated (or
3017 Unaffiliated) Asynchronous Event Stream. If the Consumer needs to know with certainty the
3018 state of an entity that can generate an Unaffiliated Asynchronous Event (e.g., a Spigot), it should
3019 query for that state itself rather than relying upon getting a state change Notification via the
3020 Unaffiliated Asynchronous Event Stream.

3021 IT_ASYNC_AFF_EVENT_STREAM and IT_ASYNC_UNAFF_EVENT_STREAM SEVDs
3022 store Events that notify users of errors and other conditions that affect IA operation. These
3023 Events are usually unpredictable, which can make determining an appropriate size for these
3024 queues a challenge. Users should consider the size and type of the fabric, their resource usage,
3025 and their message patterns when setting the *sevd_queue_size* parameter for these EVDs.

3026 **FUTURE DIRECTIONS**
3027 IT-API support for a callback routine being invoked when an Event is enqueued on an SEVD
3028 may be added in the future.
3029 Aggregate EVD support for multiple IAs may be added in the future.

3030 **SEE ALSO**
3031 *it_evd_post_se()*, *it_ep_rc_create()*, *it_ep_ud_create()*, *it_listen_create()*,
3032 *it_ud_service_request_handle_create()*, *it_evd_query()*, *it_evd_modify()*, *it_evd_wait()*,
3033 *it_evd_dequeue()*, *it_evd_free()*, *it_event_t*, *it_dto_flags_t*, *it_unaffiliated_event_t*,
3034 *it_ia_create()*, *it_ia_info_t*
3035

it_evd_dequeue()

3036

3037 NAME

3038 it_evd_dequeue – dequeue for Events from Event Dispatcher

3039 SYNOPSIS

```
3040 #include <it_api.h>
3041
3042 it_status_t it_evd_dequeue(
3043     IN  it_evd_handle_t  evd_handle,
3044     OUT it_event_t      *event
3045 );
```

3046 DESCRIPTION

3047 *evd_handle*: Handle for simple or aggregate Event Dispatcher.

3048 *event*: Pointer to the Consumer-allocated structure that the Implementation fills
3049 with the Event information.

3050 *it_evd_dequeue* removes the first Event from the Event Dispatcher Event queue and fills the
3051 Consumer-allocated *event* structure with Event information. For the Event information and *event*
3052 structure, see *it_event_t*. The Consumer should allocate an Event structure big enough to hold
3053 any Event that the Event Dispatcher can deliver.

3054 *it_evd_dequeue* returns the first Event from an EVD, if one exists, regardless of whether EVD
3055 has waiters.

3056 The return value for *event* is defined only if *it_evd_dequeue* returns IT_SUCCESS.

3057 For AEVD with the IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* bit clear, the operation
3058 dequeues the first Event from one of its associated SEVDs.

3059 For AEVD with the IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* bit set, the operation
3060 returns a Notification Event of the IT_AEVD_NOTIFICATION_EVENT Event Stream which
3061 identifies an *evd_handle* from one of its associated SEVDs. The order in which the associated
3062 SEVD's AEVD Notification Events are delivered is Implementation-dependent.

3063 For a Simple EVD that does not have an associated AEVD, the Consumer can dequeue from the
3064 SEVD.

3065 If the SEVD has an associated AEVD with the IT_EVD_DEQUEUE_NOTIFICATIONS
3066 *evd_flag* cleared, then it is an error for the Consumer to dequeue from the SEVD.

3067 If the SEVD has an associated AEVD with the IT_EVD_DEQUEUE_NOTIFICATIONS
3068 *evd_flag* set, then the Consumer may dequeue from the SEVD at will.

3069 Attempting to dequeue from the SEVD when disallowed will return IT_ERR_
3070 INVALID_EVD_STATE.

3071 The Consumer can always dequeue from the AEVD regardless of the IT_EVD_
3072 DEQUEUE_NOTIFICATIONS *evd_flag* value or associated *fd*. If the EVD is empty, then
3073 *it_evd_dequeue* will return IT_ERR_QUEUE_EMPTY.

3074 For IT_DTO_EVENT_STREAM Events when a Completion Event is returned for a given Send,
3075 RDMA Read, RDMA Write, RMR Link, or RMR Unlink operation that was posted to an
3076 Endpoint, the Implementation guarantees that all Send, RDMA Read, RDMA Write, RMR Link,
3077 and RMR Unlink operations that were posted to the Endpoint prior to the one whose Completion
3078 Event was returned have also completed regardless of their *dto_flag* value for
3079 IT_COMPLETION_FLAG.

3080 The SEVD *sevd_threshold* value has no effect on this operation.

3081 RETURN VALUE

3082 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

3083 IT_ERR_QUEUE_EMPTY There were no entries on the Event Dispatcher queue.

3084 IT_ERR_INVALID_EVD The Event Dispatcher Handle (*evd_handle*) was invalid.

3085 IT_ERR_INVALID_EVD_STATE The attempted operation was invalid for the current state
3086 of the Event Dispatcher.

3087 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the
3088 disabled state. None of the output parameters from this
3089 routine are valid. See *it_ia_info_t* for a description of the
3090 disabled state.

3091 APPLICATION USAGE

3092 For an AEVD with IT_EVD_DEQUEUE_NOTIFICATIONS set, receipt of an
3093 IT_AEVD_NOTIFICATION_EVENT Event indicates that the SEVD (identified by *evd_handle*
3094 in the Event) reached Notification status or has Events available. By the time the Consumer calls
3095 *it_evd_dequeue* on the returned SEVD it may be empty or may not be in the Notification Criteria
3096 any longer if there are multiple dequeuers from the SEVD.

3097 For the “non-thread-safe” Implementation, the Consumer cannot have multiple threads calling
3098 dequeue on the same EVD Handle simultaneously.

3099 When multiple threads retrieve Events concurrently from the same SEVD, each Event will be
3100 retrieved exactly once but it is unpredictable which thread will retrieve any particular Event.

3101 SEE ALSO

3102 *it_evd_create()*, *it_evd_wait()*, *it_event_t*

3103

it_evd_free()

3104

3105 NAME

3106 `it_evd_free` – destroy an Event Dispatcher

3107 SYNOPSIS

```
3108 #include <it_api.h>
3109
3110 it_status_t it_evd_free(
3111     IN it_evd_handle_t  evd_handle
3112 );
```

3113 DESCRIPTION

3114 `evd_handle` Handle to Simple or Aggregate Event Dispatcher.

3115 `it_evd_free` Destroys an Event Dispatcher.

3116 On successful completion, all Events on the queue of the specified Event Dispatcher are lost.

3117 `it_evd_free` will return `IT_ERR_EVD_BUSY` if the EVD is still associated with an active Event
3118 Stream feeding it for all Event Streams except `IT_ASYNC_AFF_EVENT_STREAM`,
3119 `IT_ASYNC_UNAFF_EVENT_STREAM`, and `IT_SOFTWARE_EVENT_STREAM`.
3120 `it_evd_free` may be called at any time for `IT_ASYNC_AFF_EVENT_STREAM`,
3121 `IT_ASYNC_UNAFF_EVENT_STREAM`, and `IT_SOFTWARE_EVENT_STREAM` Event
3122 Streams but Events may be lost.

3123 An AEVD with `IT_EVD_DEQUEUE_NOTIFICATIONS` set may be dissociated from its
3124 SEVDs through use of `it_evd_modify` on each SEVD or through use of `it_evd_free` on each
3125 SEVD. An AEVD with the `IT_EVD_DEQUEUE_NOTIFICATIONS evd_flag` bit clear may be
3126 dissociated from SEVDs through use of `it_evd_free` on each SEVD. DTO SEVDs may be
3127 disassociated from their DTO Event Streams through use of `it_ep_free` on each associated
3128 Endpoint. Communication Management Request SEVDs may be disassociated from their Event
3129 Streams through use of `it_listen_free` on each associated listen Handle. Communication
3130 Management Message SEVDs may be disassociated from their Event Streams through use of
3131 `it_ep_free` on each associated Endpoint.

3132 Once `it_evd_free` returns, `evd_handle` may no longer be used.

3133 This operation is applicable to both AEVD and SEVD Handles.

3134 RETURN VALUE

3135 A successful call returns `IT_SUCCESS`. Otherwise, an error code is returned as described below:

3136 `IT_ERR_INVALID_EVD` The Event Dispatcher Handle (`evd_handle`) was invalid.

3137 `IT_ERR_EVD_BUSY` The Event Dispatcher was still associated with active Event
3138 Streams.

3139 `IT_ERR_IA_CATASTROPHE` The IA has experienced a catastrophic error and is in the
3140 disabled state. None of the output parameters from this routine
3141 are valid. See `it_ia_info_t` for a description of the disabled state.

3142 **SEE ALSO**
3143 *it_evd_create(), it_evd_modify(), it_evd_query(), it_ep_free(), it_listen_free()*
3144

it_evd_modify()

3145

3146 NAME

3147 `it_evd_modify` – modify an existing Event Dispatcher

3148 SYNOPSIS

```
3149 #include <it_api.h>
3150
3151 it_status_t it_evd_modify(
3152     IN          it_evd_handle_t      evd_handle,
3153     IN          it_evd_param_mask_t mask,
3154     IN const    it_evd_param_t      *params
3155 );
```

3156 DESCRIPTION

3157 *evd_handle* Simple or Aggregate Event Dispatcher.

3158 *mask* Bitwise OR of flags for requested EVD parameters.

3159 *params* Pointer to Consumer-allocated structure that contains new Consumer-
3160 requested Event Dispatcher parameters.

3161 *it_evd_modify* changes the desired parameters of the Simple or Aggregate Event Dispatcher
3162 *evd_handle*. Parameters to be modified are specified by flags in *mask*. New values for the
3163 parameters are specified by the corresponding fields in the structure pointed to by *params*. Fields
3164 and their flag values are shown below. Note that parameters represented by fields of
3165 *it_evd_param_t* that are not shown below cannot be modified. See *it_evd_query* for definition of
3166 *it_evd_param_t* and *it_evd_param_mask_t*.

```
3167 typedef struct {
3168     ...
3169     size_t      sevd_queue_size; /* IT_EVD_PARAM_QUEUE_SIZE */
3170     size_t      sevd_threshold; /* IT_EVD_PARAM_THRESHOLD */
3171     it_evd_handle_t aevd; /* IT_EVD_PARAM_AEVD_HANDLE */
3172     ...
3173     it_boolean_t evd_enabled; /* IT_EVD_PARAM_ENABLED */
3174     it_boolean_t evd_overflowed; /* IT_EVD_PARAM_OVERFLOWED */
3175 } it_evd_param_t;
```

3176
3177 The definition of each field follows:

3178 *sevd_queue_size* Minimum size of the Simple EVD Event queue. Attempting to modify this
3179 field for an AEVD will return an `IT_ERR_INVALID_MASK` error code.

3180 *sevd_threshold* For Simple EVD only. Number of Events on a single Event Dispatcher queue
3181 required for Notification of the associated AEVD or FD and for SEVD
3182 waiters unblocking. Attempting to modify this field for an AEVD will return
3183 an `IT_ERR_INVALID_MASK` error code.

3184 *aevd* For Simple EVD only. The Handle for the new associated Aggregate EVD.
3185 Attempting to modify this field for an AEVD will return an
3186 `IT_ERR_INVALID_MASK` error code if the above criteria are not met.

3229 the IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* cleared, the AEVD can dequeue Events
3230 from the SEVD. The Consumer cannot wait on or dequeue from the SEVD that is associated
3231 with the AEVD that has the IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* cleared even
3232 when SEVD is disabled. The Consumer can wait on or dequeue from the SEVD that is
3233 associated with the AEVD that has the IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* set
3234 when SEVD is disabled.

3235 The Consumer can disable an AEVD so that the AEVD will not generate Notification for an
3236 associated *fd*. Disabling the AEVD allows the Consumer to wait on the AEVD.

3237 Disabling the disabled EVD has no effect.

3238 RETURN VALUE

3239 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

3240 IT_ERR_INVALID_EVD The Event Dispatcher Handle (*evd_handle*) was invalid.

3241 IT_ERR_INVALID_MASK The mask contained invalid flag values.

3242 IT_ERR_INVALID_EVD_STATE The attempted operation was invalid for the current state
3243 of the Event Dispatcher.

3244 IT_ERR_RESOURCE_QUEUE_SIZE The underlying transport could not allocate the requested
3245 *sevd_queue_size* resources at this time.

3246 IT_ERR_INVALID_QUEUE_SIZE The requested Simple Event Dispatcher queue size
3247 (*sevd_queue_size*) was less than the outstanding Events
3248 on the Event queue.

3249 IT_ERR_INVALID_THRESHOLD An invalid value for the Simple Event Dispatcher
3250 threshold was specified.

3251 IT_ERR_INVALID_AEVD The Aggregation Event Dispatcher Handle (*aevd*) was
3252 invalid.

3253 IT_ERR_RESOURCES The requested operation failed due to insufficient
3254 resources.

3255 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the
3256 disabled state. None of the output parameters from this
3257 routine are valid. See [it_ia_info_t](#) for a description of the
3258 disabled state.

3259 SEE ALSO

3260 [it_evd_create\(\)](#), [it_evd_query\(\)](#), [it_evd_free\(\)](#)

3261

3262

it_evd_post_se()

3263 NAME

3264 it_evd_post_se – post software Event on Simple Event Dispatcher

3265 SYNOPSIS

```
3266 #include <it_api.h>
3267
3268 it_status_t it_evd_post_se(
3269     IN          it_evd_handle_t  evd_handle,
3270     IN  const   void             *event
3271 );
```

3272 DESCRIPTION

3273 *evd_handle* Simple Event Dispatcher of IT_SOFTWARE_EVENT_STREAM Event
3274 Stream type.

3275 *event* Pointer to the Consumer-created Software Event.

3276 *it_evd_post_se* posts a software Event to the IT_SOFTWARE_EVENT_STREAM simple Event
3277 Dispatcher Event queue. This causes an Event to arrive on the Event Dispatcher Software Event
3278 Stream. The *event* pointer is opaque to the Implementation and release of the memory referenced
3279 by the *event* pointer in a software Event is the Consumer's responsibility.

3280 If the Event queue is full, the operation is completed unsuccessfully and returns
3281 IT_ERR_EVD_QUEUE_FULL. The *event* is not queued. Since the Event queue for software
3282 Events can never overflow, the Affiliated Asynchronous Event Dispatcher is not affected.

3283 *it_evd_post_se* can only be used to post software Events within the same process since
3284 *evd_handle* has the scope of a single IA.

3285 RETURN VALUE

3286 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

3287 IT_ERR_EVD_QUEUE_FULL The Simple Event Dispatcher queue was full.

3288 IT_ERR_INVALID_EVD The Event Dispatcher Handle (*evd_handle*) was invalid.

3289 IT_ERR_INVALID_SOFT_EVD The Simple Event Dispatcher Handle (*evd_handle*) was
3290 not an IT_SOFTWARE_EVENT_STREAM Event
3291 Dispatcher.

3292 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the
3293 disabled state. None of the output parameters from this
3294 routine are valid. See *it_ia_info_t* for a description of the
3295 disabled state.

3296 APPLICATION USAGE

3297 The Consumer can use this operation to unblock an AEVD waiter as well as passing specific
3298 instruction for the unblocked waiter. The SEVD for the Software Event should be associated
3299 with the AEVD. A software Event is a Notification Event and will unblock the waiter.

3300 **SEE ALSO**
3301 *it_evd_create(), it_software_event_t, it_evd_wait()*

it_evd_query()

3302

3303 NAME

3304 `it_evd_query` – query an existing Simple or Aggregate Event Dispatcher

3305 SYNOPSIS

```
3306 #include <it_api.h>
3307
3308 it_status_t it_evd_query(
3309     IN    it_evd_handle_t    evd_handle,
3310     IN    it_evd_param_mask_t mask,
3311     OUT   it_evd_param_t     *params
3312 );
3313
3314 typedef enum {
3315     IT_EVD_PARAM_ALL           = 0x000001,
3316     IT_EVD_PARAM_IA           = 0x000002,
3317     IT_EVD_PARAM_EVENT_NUMBER = 0x000004,
3318     IT_EVD_PARAM_FLAG        = 0x000008,
3319     IT_EVD_PARAM_QUEUE_SIZE  = 0x000010,
3320     IT_EVD_PARAM_THRESHOLD   = 0x000020,
3321     IT_EVD_PARAM_AEVD_HANDLE = 0x000040,
3322     IT_EVD_PARAM_FD          = 0x000080,
3323     IT_EVD_PARAM_BOUND       = 0x000100,
3324     IT_EVD_PARAM_ENABLED     = 0x000200,
3325     IT_EVD_PARAM_OVERFLOWED  = 0x000400
3326 } it_evd_param_mask_t;
3327
3328 typedef struct {
3329     it_ia_handle_t    ia;           /* IT_EVD_PARAM_IA */
3330     it_event_type_t  event_number; /* IT_EVD_PARAM_EVENT_NUMBER*/
3331     it_evd_flags_t   evd_flag;     /* IT_EVD_PARAM_FLAG */
3332     size_t           sev_queue_size; /* IT_EVD_PARAM_QUEUE_SIZE */
3333     size_t           sev_threshold; /* IT_EVD_PARAM_THRESHOLD */
3334     it_evd_handle_t aevid;         /* IT_EVD_PARAM_AEVD_HANDLE*/
3335     int              fd;           /* IT_EVD_PARAM_FD */
3336     it_boolean_t     evd_bound;    /* IT_EVD_PARAM_BOUND */
3337     it_boolean_t     evd_enabled;  /* IT_EVD_PARAM_ENABLED */
3338     it_boolean_t     evd_overflowed; /* IT_EVD_PARAM_OVERFLOWED */
3339 } it_evd_param_t;
```

3340 DESCRIPTION

3341 *evd_handle* Event Dispatcher.

3342 *mask* Bitwise OR of flags for requested EVD parameters.

3343 *params* Pointer to Consumer-allocated structure that the Implementation fills with
3344 Consumer-requested Event Dispatcher parameters.

3345 *it_evd_query* returns the desired parameters of the Simple or Aggregate Event Dispatcher
3346 *evd_handle* in the structure pointed to by *params*. On return, each field of *params* is only valid if
3347 the corresponding flag as shown adjacent to each field is set in the *mask* argument. The *mask*
3348 value `IT_EVD_PARAM_ALL` causes all fields to be returned.

3349		The definition of each field follows:
3350	<i>ia</i>	Handle for the Interface Adapter.
3351	<i>event_number</i>	Identifier for Event Stream type that can be enqueued to the EVD.
3352	<i>evd_flag</i>	Flags for Event Dispatcher. See it_evd_create for definitions and use of <i>evd_flag</i> .
3353		
3354	<i>sevd_queue_size</i>	Minimum size of the SEVD Event queue or zero for an AEVD.
3355	<i>sevd_threshold</i>	The number of non-notification Events on the Simple Event Dispatcher queue for Notification, unblocking.
3356		
3357	<i>aevd</i>	Handle for Aggregate EVD associated with SEVD or IT_NULL_HANDLE if none.
3358		
3359	<i>fd</i>	<i>File descriptor</i> corresponding to Event Dispatcher or -1 if none.
3360	<i>evd_bound</i>	When it has the value IT_TRUE, indicates that the EVD is tied to an Event Stream so Events can be queued on EVD. For an AEVD, indicates that SEVDs are tied to the AEVD.
3361		
3362		
3363	<i>evd_enabled</i>	When it has the value IT_TRUE, indicates: for an SEVD that it has been configured to notify an associated AEVD or <i>fd</i> when Notification criteria is reached; for an AEVD that it has been configured to notify an associated <i>fd</i> when it is notified by one of its associated SEVDs. See it_evd_modify .
3364		
3365		
3366		
3367	<i>evd_overflowed</i>	When it has the value IT_TRUE, indicates that the EVD has overflowed. See it_evd_create for more details.
3368		

3369 **RETURN VALUE**

3370 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

3371	IT_ERR_INVALID_EVD	The Event Dispatcher Handle (<i>evd_handle</i>) was invalid.
3372	IT_ERR_INVALID_MASK	The mask contained invalid flag values.
3373	IT_ERR_IA_CATASTROPHE	The IA has experienced a catastrophic error and is in the disabled state. None of the output parameters from this routine are valid. See it_ia_info_t for a description of the disabled state.
3374		
3375		
3376		

3377 **SEE ALSO**

3378 [it_evd_create\(\)](#), [it_evd_modify\(\)](#), [it_evd_free\(\)](#), [it_ia_info_t](#)

it_evd_wait()

3379

3380 NAME

3381 `it_evd_wait` – wait for Events on Event Dispatcher

3382 SYNOPSIS

```
3383 #include <it_api.h>
3384
3385 it_status_t it_evd_wait(
3386     IN  it_evd_handle_t  evd_handle,
3387     IN  uint64_t         timeout,
3388     OUT it_event_t       *event,
3389     OUT size_t           *nmore
3390 );
```

3391 DESCRIPTION

3392 *evd_handle* Handle for Simple or Aggregate Event Dispatcher.

3393 *timeout* The duration of time, in microseconds, that the Consumer is willing to wait
3394 for an Event.

3395 *event* Pointer to the Consumer-allocated structure that the Implementation fills
3396 with the Event information.

3397 *nmore* The snapshot of the number of Events queued on the EVD at the time of
3398 *it_evd_wait* return. Only applicable for SEVD.

3399 *it_evd_wait* removes the first Event from the Event Dispatcher Event queue and fills the
3400 Consumer-allocated *event* structure with Event information. For the Event information and *event*
3401 structure, see *it_event_t*. The Consumer should allocate an Event structure big enough to hold
3402 any Event that the Event Dispatcher can deliver.

3403 The return value for *event* is defined only if *it_evd_wait* returns IT_SUCCESS.

3404 The Consumer can wait on an EVD that is not associated with any higher-level object (AEVD or
3405 *fd*).

3406 The Consumer should not wait on an SEVD that has an associated AEVD with the
3407 IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* bit clear. An attempt by the Consumer to
3408 wait on *evd_handle* for that type of SEVD will result in routine failure with the return value of
3409 IT_ERR_INVALID_EVD_STATE.

3410 The Consumer should not wait on an EVD that is associated with and enabled for Notification to
3411 higher-level objects. An attempt by the Consumer to wait on *evd_handle* that is associated with
3412 and enabled for Notification to a higher-level object will result in routine failure with the return
3413 value of IT_ERR_INVALID_EVD_STATE. However, the Consumer can wait on the EVD
3414 associated with the higher-level object if the EVD is disabled for Notification (except if the
3415 object is an AEVD with the IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* bit clear, as
3416 stated above).

3417 An Implementation can support one or more simultaneous waiters on the same EVD (for thread-
3418 safety models see [Global Behavior](#)) if *sevd_threshold* value of *evd_handle* (see *it_evd_create*) is
3419 greater than one, then only a single waiter is supported. An attempt for more than one waiter to

3420 wait on the EVD will result in an immediate error with IT_ERR_WAITER_LIMIT return value.
3421 If *sevd_threshold* value of *evd_handle* is 1, then one or more simultaneous waiters can be
3422 supported for the SEVD.

3423 A waiter can be blocked. An SEVD waiter will block when the SEVD queue is empty. An
3424 AEVD waiter will block when all associated SEVDs are empty. An SEVD waiter may block
3425 when the SEVD has not reached the Notification criteria (see *it_evd_create* for the definition of
3426 the Notification criteria). An AEVD waiter may block when all associated SEVDs have not
3427 reached their Notification criteria.

3428 An SEVD waiter will return immediately if there is a Notification Event (see *it_evd_create* for
3429 the definition of the Notification Event) on the queue or if the number of Events on the SEVD is
3430 larger than or equal to *sevd_threshold*. An AEVD waiter with the IT_EVD_DEQUEUE_
3431 NOTIFICATIONS *evd_flag* bit cleared will return immediately if there is a Notification Event
3432 on any of the associated SEVDs or any of the associated SEVDs has a number of Events larger
3433 than or equal to its *sevd_threshold*. An AEVD waiter with the IT_EVD_DEQUEUE_
3434 NOTIFICATIONS *evd_flag* bit set will return immediately if there is an IT_AEVD_
3435 NOTIFICATION_EVENT available.

3436 If an arriving Event causes SEVD to reach Notification criteria, then SEVD waiter will be
3437 unblocked if one exists and if the SEVD is disabled and not associated with AEVD with the
3438 IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* bit cleared. As many waiters as there are
3439 Events available on the SEVD can be unblocked. If arriving Event causes the SEVD to reach
3440 Notification criteria and the SEVD is enabled for Notification to higher-level objects, then
3441 Notification will be generated for the associated AEVD or *fd*. If the associated AEVD has a
3442 waiter, then the waiter will be unblocked. As many Notifications can be generated as there are
3443 Events available on all SEVDs of the AEVD. As many waiters as there are Notifications can be
3444 unblocked. Which waiters will be woken and in what order they will be woken is
3445 Implementation-dependent.

3446 The *timeout* allows the Consumer to restrict the amount of time it will be blocked waiting for an
3447 Event arrival. The value of IT_TIMEOUT_INFINITE indicates that the Consumer will wait
3448 indefinitely for an Event arrival. Consumers should use caution in using this value because wait
3449 may never return if Notification is not generated. Consumers can use *signal* to unblock the
3450 waiter in this case.

3451 For IT_DTO_EVENT_STREAM Events, when a Completion Event is returned for a given
3452 Send, RDMA Read, RDMA Write, RMR Link, or RMR Unlink operation that was posted to an
3453 Endpoint, the Implementation guarantees that all Send, RDMA Read, RDMA Write, RMR Link,
3454 and RMR Unlink operations that were posted to the Endpoint prior to the one whose Completion
3455 Event was returned have also completed regardless of their *dto_flag* value for
3456 IT_COMPLETION_FLAG.

3457 For an SEVD, if the return value is neither IT_SUCCESS nor IT_ERR_TIMEOUT_EXPIRED,
3458 then the returned values of *nmore* and *Event* are undefined. If the return value is
3459 IT_ERR_TIMEOUT_EXPIRED, then the return value of *event* is undefined, but the return value
3460 of *nmore* is defined. If the return value is IT_SUCCESS, then the return values of both *nmore*
3461 and *event* are defined.

3462 For an AEVD *nmore* is undefined for all returns. If the return value is not IT_SUCCESS, then
3463 returned value *event* is undefined.

3464 The routine returns with return value `IT_ERR_INTERRUPT` when the waiter is unblocked by an
3465 OS signal.

3466 This call may block the caller's execution waiting for a remotely generated event.

3467 **RETURN VALUE**

3468 A successful call returns `IT_SUCCESS`. Otherwise, an error code is returned as described below:

3469 `IT_ERR_WAITER_LIMIT` No more waiters are permitted for the Event Dispatcher.

3470 `IT_ERR_INVALID_EVD` The Event Dispatcher Handle (*evd_handle*) was invalid.

3471 `IT_ERR_INVALID_EVD_STATE` The attempted operation was invalid for the current state
3472 of the Event Dispatcher.

3473 `IT_ERR_ABORT` The Event Dispatcher has been destroyed.

3474 `IT_ERR_INTERRUPT` The Event Dispatcher waiter was unblocked by a signal.

3475 `IT_ERR_TIMEOUT_EXPIRED` The operation timed out.

3476 `IT_ERR_IA_CATASTROPHE` The IA has experienced a catastrophic error and is in the
3477 disabled state. None of the output parameters from this
3478 routine are valid. See *it_ia_info_t* for a description of the
3479 disabled state.

3480 **APPLICATION USAGE**

3481 The Consumer should allocate an Event structure big enough to hold any Event that the Event
3482 Dispatcher can deliver. The Implementation is not able to check that the *event* that the Consumer
3483 provides is sufficient to hold a returned Event. As a result, a segmentation fault or memory
3484 corruption may occur if the Implementation overruns the user-specified memory.

3485 For an AEVD with `IT_EVD_DEQUEUE_NOTIFICATIONS` set, any `IT_AEVD_`
3486 `NOTIFICATION_EVENT` Event only indicates that the SEVD (identified by *evd_handle* in the
3487 Event) reached Notification criteria. The order in which the associated SEVD's AEVD
3488 Notification Events are delivered is Implementation-dependent. No restriction is imposed by the
3489 Implementation on dequeuing Events from the underlying SEVD. If other Consumer threads
3490 are independently dequeuing Events from the SEVD, the thread receiving the
3491 `IT_AEVD_NOTIFICATION_EVENT` may find the SEVD to be empty when it dequeues from
3492 the SEVD.

3493 For an AEVD with `IT_EVD_DEQUEUE_NOTIFICATIONS` set, receipt of an `IT_AEVD_`
3494 `NOTIFICATION_EVENT` Event indicates that the SEVD (identified by *evd_handle* in the
3495 Event) reached Notification status. If the Consumer fails to dequeue Events from the SEVD
3496 sufficient to remove it from Notification status, then an additional
3497 `IT_AEVD_NOTIFICATION_EVENT` Event for the SEVD will appear at the AEVD when the
3498 Consumer next calls *it_evd_wait* or *it_evd_dequeue*.

3499 For an AEVD with `IT_EVD_DEQUEUE_NOTIFICATIONS` set, receipt of an `IT_AEVD_`
3500 `NOTIFICATION_EVENT` Event indicates that the SEVD (identified by *evd_handle* in the
3501 Event) reached Notification status. By the time the Consumer calls *it_evd_dequeue* on the
3502 returned SEVD it may be empty or may not be in the Notification Criteria any longer if there are
3503 multiple dequeuers from the SEVD.

3504 The Consumer must be prepared to handle return from *it_evd_wait* with fewer than the expected
3505 number of Events or without any Notification Events on an EVD. This can occur for the
3506 following reasons:

- 3507 • The underlying Implementation does not support thresholding.
- 3508 • The underlying Implementation does not support IT_NOTIFY_FLAG.

3509 For *sevd_threshold* value of 1, if an Event is on the SEVD, then *it_evd_wait* will return
3510 immediately with IT_SUCCESS for the SEVD or the AEVD fed by the SEVD.

3511 For the “non-thread-safe” Implementation the Consumer should not have multiple threads
3512 calling on the same EVD Handle simultaneously. The Consumer should choose an
3513 Implementation that supports multi-threaded applications if they want to have multiple waiters.
3514 The Consumer should set the *sevd_threshold* to 1 for an SEVD if they want to use multiple
3515 waiters on the SEVD.

3516 When multiple threads retrieve Events concurrently from the same SEVD, each Event will be
3517 retrieved exactly once, but it is unpredictable which thread will retrieve any particular Event.

3518 The Consumer is advised not to destroy an EVD on which it is currently waiting. If the
3519 Consumer does so, the *it_evd_wait* routine may return IT_ERR_ABORT, or a segmentation
3520 violation may take place. Which behavior occurs is Implementation-dependent.

3521 **SEE ALSO**

3522 *it_evd_create()*, *it_event_t*, *it_post_send()*, *it_post_sendto()*, *it_post_rdma_read()*,
3523 *it_post_rdma_write()*, *it_rmr_link()*, *it_rmr_unlink()*, *it_dto_events*, *it_dto_flags_t*

3524

it_get_consumer_context()

3525

3526 NAME

3527 `it_get_consumer_context` – return the Consumer Context associated with an IT Object Handle

3528 SYNOPSIS

```
3529 #include <it_api.h>
3530
3531 it_status_t it_get_consumer_context(
3532     IN  it_handle_t  handle,
3533     OUT it_context_t *context
3534 );
```

3535 DESCRIPTION

3536 *handle* Handle of the IT-API object associated with the Consumer Context to be
3537 retrieved.

3538 *context* The address of the location where the retrieved Consumer Context is
3539 returned.

3540 *it_get_consumer_context* retrieves the Consumer Context associated with the specified *handle*. If
3541 the Consumer Context was never set (by a call to *it_set_consumer_context*), then the value of the
3542 returned Consumer Context is 0 and the immediate error IT_ERR_NO_CONTEXT is returned.

3543 The *handle* must be one of the IT-API Handle types, cast as an *it_handle_t*. See *it_handle_t* for a
3544 description of the valid Handle types.

3545 RETURN VALUE

3546 A successful call returns SUCCESS. Otherwise, an error code is returned as described below:

3547 IT_ERR_INVALID_HANDLE The *handle* was invalid.

3548 IT_ERR_NO_CONTEXT The *handle* does not have an associated Context.

3549 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the disabled
3550 state. None of the output parameters from this routine are valid.
3551 See *it_ia_info_t* for a description of the disabled state.

3552 EXAMPLES

3553 The following code example demonstrates the use of a cast in the call to
3554 *it_get_consumer_context*. The *lmr* object is cast to the generic *it_handle_t* type for the call.

```
3555 it_lmr_handle_t lmr;
3556 it_context_t    cxt;
3557 it_get_consumer_context( (it_handle_t) lmr, &cxt);
```

3558 SEE ALSO

3559 *it_set_consumer_context()*, *it_context_t*, *it_handle_t*

it_get_handle_type()

3560

3561 NAME

3562 `it_get_handle_type` – return the Handle type value associated with an IT Object Handle

3563 SYNOPSIS

```
3564 #include <it_api.h>
3565
3566 it_status_t it_get_handle_type(
3567     IN  it_handle_t      handle,
3568     OUT it_handle_type_enum_t *type_of_handle
3569 );
```

3570 DESCRIPTION

3571 *handle* Handle of an IT-API object.

3572 *type_of_handle* Type of the Handle of *handle*.

3573 The *it_get_handle_type* interface allows the Consumer to retrieve the type of an IT Object using
3574 its Handle. See [it_handle_t](#) for a description of the Handle types and associated enumeration
3575 values returned.

3576 RETURN VALUE

3577 A successful call returns `IT_SUCCESS`. Otherwise, an error code is returned as described below:

3578 `IT_ERR_INVALID_HANDLE` The *handle* was invalid.

3579 `IT_ERR_IA_CATASTROPHE` The IA has experienced a catastrophic error and is in the disabled
3580 state. None of the output parameters from this routine are valid.
3581 See [it_ia_info_t](#) for a description of the disabled state.

3582 SEE ALSO

3583 [it_handle_t](#)

3584

3627 Several different Network Address formats are supported; see the reference page for
3628 [it_net_addr_t](#) for details. The mechanism by which the Consumer determines the remote
3629 Network Address to target is outside the scope of the API. If the underlying transport is one that
3630 supports IP Network Addresses, existing APIs (such as *gethostbyname*) for translating host
3631 names into IP addresses can be used to convert a hostname into an IP address.

3632 The Consumer is responsible for allocating the storage necessary to hold the returned set of
3633 *paths*. Since the Consumer may not know how many Paths are available, it passes the number of
3634 Paths for which it has allocated storage in the *num_paths* parameter on input. This routine will
3635 return no more than that number of Paths to the Consumer. If more Paths are available than the
3636 Consumer has allocated space for, an arbitrary subset of the available Paths will be provided to
3637 the Consumer. A Consumer that does not wish to deal with Path selection can therefore avoid
3638 doing so by always specifying a value of 1 for the total number of Paths it wishes to have
3639 returned.

3640 The set of Paths that are available to reach a given remote Network Address is dynamic, and can
3641 change over time. (For example, a link on a switch or router could become inoperative, thus
3642 decreasing the set of available Paths.) There is therefore no guarantee that given the same input
3643 parameters two different invocations of *it_get_pathinfo* will return the same results. The
3644 information returned by *it_get_pathinfo* is a snapshot of the Paths available at the time of the
3645 call. In addition, if the Consumer asks for fewer Paths than are available, the API may return a
3646 different set of Paths for two different invocations of *it_get_pathinfo* regardless of the state of
3647 the network.

3648 It is possible that no Paths are available to reach the given remote Network Address. In that case,
3649 *it_get_pathinfo* will return IT_SUCCESS, but the total number of Paths available pointed to by
3650 *num_paths* will be zero.

3651 Once the Consumer has chosen one of the set of Paths returned, it can furnish that Path as input
3652 to the *it_ep_connect* routine. Consumers that wish to construct their own Path can also do so by
3653 populating the *it_path_t* data structure themselves, although this is inherently a transport-
3654 dependent programming practice. See the reference page for [it_path_t](#) for details on the internal
3655 structure of a Path.

3656 This call may block the caller's execution waiting for a remotely generated event.

3657 RETURN VALUE

3658 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

3659	IT_ERR_INVALID_IA	The Interface Adapter Handle (<i>ia_handle</i>) was invalid.
3660	IT_ERR_INVALID_SPIGOT	An invalid Spigot ID was specified.
3661	IT_ERR_INVALID_ADDRESS	The Network Address specified in <i>net_addr</i> was invalid.
3662	IT_ERR_INVALID_NETADDR	The type of the Network Address specified in <i>net_addr</i> was
3663		not recognized.
3664	IT_ERR_INTERRUPT	The call was unblocked by a signal.
3665	IT_ERR_IA_CATASTROPHE	The Interface Adapter has experienced a catastrophic error
3666		and is in the disabled state. None of the output parameters
3667		from this routine are valid. See it_ia_info_t for a description
3668		of the disabled state.

3669 **SEE ALSO**
3670 *it_interface_list(), it_ia_create(), it_ia_query(), it_ep_connect(), it_listen_create()*
3671

it_handoff()

3672

3673 NAME

3674 it_handoff – forward an incoming Connection Request to another Spigot and Connection
3675 Qualifier

3676 SYNOPSIS

```
3677 #include <it_api.h>
3678
3679 it_status_t it_handoff(
3680     IN const it_conn_qual_t      *conn_qual,
3681     IN      size_t                spigot_id,
3682     IN      it_cn_est_identifier_t cn_est_id
3683 );
3684
3685 typedef uint64_t it_cn_est_identifier_t;
```

3686 DESCRIPTION

3687 *conn_qual* The Connection Qualifier to which the Connection Request should be
3688 forwarded.

3689 *spigot_id* Interface Adapter Spigot to which the Connection Request should be
3690 forwarded.

3691 *cn_est_id* Connection establishment identifier associated with the Connection
3692 Request to be forwarded.

3693 *it_handoff* forwards a Connection Request to the specified Spigot and Connection Qualifier of
3694 the IA on which the Connection Request originally arrived. Specifying a *conn_qual* or *spigot_id*
3695 on any IA other than that on which the Connection Request originally arrived will yield
3696 IT_ERR_INVALID_CONN_QUAL or IT_ERR_INVALID_SPIGOT errors respectively. The
3697 forwarded Connection Request generates an IT_CM_REQ_CONN_REQUEST_EVENT Event
3698 at the listen point to which the request was forwarded. Forwarded Connection Request Events
3699 look identical to the original Events, therefore the Consumer cannot distinguish them. The
3700 connection establishment identifier, *cn_est_id*, is destroyed by this function.

3701 RETURN VALUE

3702 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

3703 ERR_INVALID_CN_EST_ID The connection establishment identifier (*cn_est_id*) was
3704 invalid.

3705 IT_ERR_INVALID_CONN_QUAL The Connection Qualifier (*conn_qual*) was invalid.

3706 IT_ERR_INVALID_SPIGOT An invalid Spigot ID was specified.

3707 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the
3708 disabled state. None of the output parameters from this
3709 routine are valid. See *it_ia_info_t* for a description of the
3710 disabled state.

3711 **APPLICATION USAGE**

3712 Calls to *it_reject*, *it_ep_accept*, and *it_handoff* which pertain to the same Endpoint should be
3713 serialized by the Consumer. Failure to abide by this restriction may result in a segmentation
3714 violation or other error.

3715 **SEE ALSO**

3716 *it_ep_connect()*, *it_reject()*, *it_ep_accept()*, *it_cm_req_events*

3717

it_hton64()

3718
3719 **NAME**
3720 `it_hton64`, `it_ntoh64` – convert 64-bit integers between host and network byte order

3721 **SYNOPSIS**
3722 `#include <it_api.h>`
3723
3724 `uint64_t it_hton64(`
3725 `uint64_t hostint`
3726 `);`
3727
3728 `uint64_t it_ntoh64(`
3729 `uint64_t netint`
3730 `);`

3731 DESCRIPTION

3732 *hostint* 64-bit integer stored in host byte order.

3733 *netint* 64-bit integer stored in network byte order.

3734 The `it_hton64` routine converts its input argument *hostint* from host byte order to network byte
3735 order and returns the result.

3736 `it_ntoh64` converts its input argument *netint* from network byte order to host byte order and
3737 returns the result.

3738 On some platforms, host byte order and network byte order are identical and these functions
3739 simply return their input argument.

3740 RETURN VALUE

3741 Both functions always succeed and return their converted input argument.

3742 APPLICATION USAGE

3743 The individual bytes of integer variables are stored in memory in an order that is platform-
3744 dependent, which is known as “host byte order”. To facilitate the exchange of integer variables
3745 between platforms having different host byte orders, a platform-independent byte order known
3746 as “network byte order” has been defined. To portably send an integer to a network peer, the
3747 Consumer should convert it from host to network byte order and send the network byte order
3748 value. The receiving peer then converts from network byte order to its own host byte order.

3749 Note that an integer must be stored in host byte order to be used correctly in normal arithmetic
3750 operations.

3751 SEE ALSO

3752 `htonl()` [Unix], `ntohl()` [Unix]

3753

it_ia_create()

3754

3755 NAME

3756 `it_ia_create` – create an Interface Adapter

3757 SYNOPSIS

```
3758 #include <it_api.h>
3759
3760 it_status_t it_ia_create(
3761     IN  const char      *name,
3762     IN  uint32_t        major_version,
3763     IN  uint32_t        minor_version,
3764     OUT it_ia_handle_t  *ia_handle
3765 );
```

3766 DESCRIPTION

3767	<i>name</i>	The name of the Interface for which to create an Interface Adapter.
3768	<i>major_version</i>	The IT-API major version that the Consumer will use in subsequent calls to the IA.
3769		
3770	<i>minor_version</i>	The IT-API minor version that the Consumer will use in subsequent calls to the IA.
3771		
3772	<i>ia_handle</i>	Upon successful return, points to an Interface Adapter Handle for the created Interface Adapter.
3773		

3774 *it_ia_create* is used to create an Interface Adapter. The Consumer identifies the Interface Adapter to be created by its Interface name, major and minor version numbers for the most recent version of the IT-API supported. The Consumer may select these parameters from the list returned by the *it_interface_list* call.

3778 IT-API Version 1.0 – also known as IT-API Issue 1.0 – has a major version number of 1 and a minor version number of 0. IT-API Version 2.0 has a major version number of 2 and a minor version number of 0. When a new IT-API specification is released, a unique combination of major and minor version numbers is associated with it. If the new specification is source code-compatible with the previous one, the major version number of the new specification will be the same as that of the previous one, and the minor version number will be incremented by one. If the new specification is not source code-compatible with the previous one, the major version number of the new specification will be incremented by one, and the minor version number will be zero.

3787 The latest version of the IT-API that an Implementation supports is returned from the *it_interface_list* call. For the *major_version* returned from that call, the Implementation shall support all minor versions less than or equal to the *minor_version* returned from that call. The Implementation is not required to support major versions of the IT-API previous to the one returned from *it_interface_list*. If the Implementation does not support conversing with the IA using the requested previous major version of the IT-API, an error will be returned from *it_ia_create*.

3794 If successful, this routine returns an Interface Adapter Handle. The returned Interface Adapter
3795 Handle may be passed to other IT-API routines that create and manage Interface Adapter objects
3796 such as Event Dispatchers and Local Memory Regions.

3797 **RETURN VALUE**

3798 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

3799	IT_ERR_RESOURCES	The requested operation failed due to insufficient
3800		resources.
3801	IT_ERR_INVALID_NAME	The specified <i>name</i> was invalid.
3802	IT_ERR_INVALID_MAJOR_VERSION	The requested IT-API major version number was not
3803		supported for this Interface Adapter.
3804	IT_ERR_INVALID_MINOR_VERSION	The requested IT-API minor version number was not
3805		supported for this Interface Adapter.

3806 **SEE ALSO**

3807 *it_interface_list(), it_ia_query(), it_ia_free()*

3808

it_ia_free()

3809

3810 NAME

3811 `it_ia_free` – free Interface Adapter Handle

3812 SYNOPSIS

```
3813 #include <it_api.h>
3814
3815 it_status_t it_ia_free(
3816     IN it_ia_handle_t ia_handle
3817 );
```

3818 DESCRIPTION

3819 *ia_handle* Identifies the Interface Adapter Handle to be freed.

3820 *it_ia_free* is used to free an Interface Adapter Handle.

3821 All IT Objects associated with the specified Interface Adapter Handle are freed before this
3822 routine returns. The documented semantics associated with freeing the various IT Objects are
3823 observed when these objects are freed by the call to *it_ia_free*. Further use by the Consumer of
3824 Handles for those freed IT Objects after this routine returns successfully may have unpredictable
3825 effects. All *it_ia_info_t* structures that were returned to the Consumer by *it_ia_query* that have
3826 not already been freed by the Consumer (via *it_ia_info_free*) are freed. Examining an
3827 *it_ia_info_t* that was associated with *ia_handle* after this routine returns may have unpredictable
3828 effects.

3829 All pending operations associated with the specified *ia_handle* will be terminated before this
3830 routine returns. Posted Data Transfer Operations that are currently in progress will be terminated
3831 before this routine returns. The completion status of such DTOs is indeterminate; if the
3832 Consumer wishes to know the completion status of the DTOs they have issued, they should
3833 dequeue the relevant Completion Events before freeing the IA. All callers blocked in *it_evd_wait*
3834 calls associated with the specified *ia_handle* will be unblocked.

3835 All Connections and pending Connection Requests associated with the specified *ia_handle* are
3836 terminated before this routine returns.

3837 RETURN VALUE

3838 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

3839 IT_ERR_INVALID_IA The Interface Adapter Handle (*ia_handle*) was invalid.

3840 SEE ALSO

3841 *it_ia_create()*, *it_ia_query()*

3842

3843

it_ia_info_free()

3844 **NAME**

3845 `it_ia_info_free` – free an *it_ia_info_t* structure that was returned by *it_ia_query*

3846 **SYNOPSIS**

```
3847 #include <it_api.h>
3848
3849 void it_ia_info_free(
3850     IN it_ia_info_t *ia_info
3851 );
```

3852 **DESCRIPTION**

3853 `ia_info` Points to an *it_ia_info_t* data structure that was previously returned from a
3854 call to *it_ia_query*.

3855 *it_ia_info_free* is used to free the memory for the data structure allocated and returned by the
3856 *it_ia_query* routine. The Consumer should use this routine rather than the *free* routine to
3857 deallocate the data structure pointed to by *ia_info*; unpredictable behavior can result if *free* is
3858 used. Since this routine deallocates the input data structure, the Consumer should not attempt to
3859 access it after successfully returning from this routine.

3860 This routine does not free any of the resources that are associated with the *it_ia_info_t* data
3861 structure; it only frees the data structure itself. In particular, calling this routine does not cause
3862 the EVD Handle associated with the EVD that contains the Affiliated Asynchronous Event
3863 Stream (if present) or the EVD Handle associated with the EVD that contains the Unaffiliated
3864 Asynchronous Event Stream (if present) to be freed.

3865 When an IA is freed (by calling *it_ia_free*), any *it_ia_info_t* structures that were returned by
3866 *it_ia_query* for that IA will also be freed. The Consumer can call *it_ia_info_free* to free an
3867 *it_ia_info_t* structure before the IA is freed. After the IA has been freed, calling *it_ia_info_free*
3868 to free an *it_ia_info_t* associated with that IA will have undefined results, and may result in
3869 memory corruption.

3870 **SEE ALSO**

3871 *it_ia_info_t()*, *it_ia_query()*

3872

it_ia_query()

3873

3874 NAME

3875 it_ia_query – retrieve attributes of given Interface Adapter and its Spigots

3876 SYNOPSIS

```
3877 #include <it_api.h>
3878
3879 it_status_t it_ia_query(
3880     IN  it_ia_handle_t  ia_handle,
3881     OUT it_ia_info_t    **ia_info
3882 );
```

3883 DESCRIPTION

3884 *ia_handle* Identifies the Interface Adapter to be queried.

3885 *ia_info* Points to a pointer to an *it_ia_info_t* structure upon successful return. The
3886 *it_ia_info_t* structure contains the attributes of the Interface Adapter and the
3887 identity of its Spigots.

3888 *it_ia_query* is used to retrieve the attributes of an Interface Adapter and its associated Spigots.
3889 See the reference page *it_ia_info_t* for details of the attributes structure.

3890 This routine allocates the storage necessary to hold the returned *it_ia_info_t* structure. The
3891 Consumer should free the allocated storage using the *it_ia_info_free* routine; if the Consumer
3892 fails to do so, the Implementation will free the storage when *it_ia_free* is called for *ia_handle*.

3893 If the query is unsuccessful, the return value will indicate failure, no *it_ia_info_t* structure will
3894 be allocated, and *ia_info* will point to a NULL pointer.

3895 RETURN VALUE

3896 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

3897 IT_ERR_INVALID_IA The Interface Adapter Handle (*ia_handle*) was invalid.

3898 IT_ERR_RESOURCES The requested operation failed due to insufficient resources.

3899 IT_ERR_IA_CATASTROPHE The Interface Adapter has experienced a catastrophic error
3900 and is in the disabled state. None of the output parameters
3901 from this routine are valid. See *it_ia_info_t* for a description
3902 of the disabled state.

3903 SEE ALSO

3904 *it_ia_create()*, *it_ia_free()*, *it_ia_info_t*, *it_ia_info_free()*

it_interface_list()

3905

3906 NAME

3907 `it_interface_list` – retrieve information about the available Interfaces

3908 SYNOPSIS

```
3909 #include <it_api.h>
3910
3911 void it_interface_list(
3912     OUT    it_interface_t  *interfaces,
3913     IN OUT size_t          *num_interfaces,
3914     IN OUT size_t          *total_interfaces
3915 );
3916
3917 typedef struct {
3918     /* Most recent major version number of the IT-API supported by the
3919     Interface. */
3920     uint32_t major_version;
3921
3922     /* Most recent minor version number of the IT-API supported by the
3923     Interface. */
3924     uint32_t minor_version;
3925
3926     /* The transport that the Interface uses, as defined in
3927     it_ia_info_t. */
3928     it_transport_type_t transport_type;
3929
3930     /* The name of the Interface, suitable for input to it_ia_create.
3931     The name is a string of maximum length IT_INTERFACE_NAME_SIZE,
3932     including the terminating NULL character. */
3933     char name[IT_INTERFACE_NAME_SIZE];
3934 } it_interface_t;
3935
```

3936 DESCRIPTION

3937	<i>interfaces</i>	An array allocated by the Consumer that contains the information returned for the Interface(s).
3938		
3939	<i>num_interface</i>	On input, points to the count of the maximum number of Interfaces for which the Consumer wishes to have information returned. On output, points to the count of the number of Interfaces for which information was actually returned, which is guaranteed to be less than or equal to the number that the Consumer requested.
3940		
3941		
3942		
3943		
3944	<i>total_interfaces</i>	Upon return, points to the number of Interfaces potentially available for Consumer use. A Consumer may specify NULL for this parameter if it does not wish to know how many Interfaces are potentially available.
3945		
3946		
3947		
3948		
3949		

it_interface_list is used to retrieve information about the set of available Interfaces. The Consumer may select an Interface from the returned set, and furnish the name and version number for that Interface as input to the *it_ia_create* call.

3950 The Consumer is responsible for allocating the storage necessary to hold the information for the
3951 returned set of Interfaces. Since the local Consumer may not know how many Interfaces are
3952 available, it passes the number of Interfaces for which it has allocated storage in the
3953 *num_interfaces* parameter on input. This routine will return information for no more than that
3954 number of Interfaces to the Consumer. If more Interfaces are available than the Consumer has
3955 allocated space for, information will be provided to the Consumer for only *num_interfaces* such
3956 Interfaces; which Interfaces information will be returned for is arbitrary in this case. Upon
3957 return, the value pointed to by *total_interfaces* is the total number of available Interfaces.

3958 The set of Interfaces available to the Consumer is dynamic, and can change over time. (For
3959 example, an Interface can become inoperative, thus decreasing the set of available Interfaces.)
3960 There is therefore no guarantee that given the same input parameters two different invocations of
3961 *it_interface_list* will return the same results. The information returned by *it_interface_list* is a
3962 snapshot of the Interfaces available at the time of the call. In addition, if the Consumer asks for
3963 fewer Interfaces than are available, the API may return information for a different set of
3964 Interfaces for two different invocations of *it_interface_list* regardless of the state of the
3965 Interfaces.

3966 It is possible that no Interfaces are available. In that case the total number of Interfaces available
3967 pointed to by *num_interfaces* will be zero.

3968 **EXAMPLES**

3969 The following example illustrates how the Consumer can check after it has created the IA to
3970 ensure that the information it retrieved from the *it_interface_list* call is still valid.

```
3971 it_interface_t  interface;
3972 size_t         num_interfaces;
3973 it_ia_handle_t ia;
3974 it_ia_info_t   *infop;
3975
3976 num_interfaces = 1;
3977 it_interface_list(&interface, &num_interfaces, NULL);
3978 if (num_interfaces != 1) {
3979
3980     /* Failed to find any IAs; */
3981 }
3982
3983 if (it_ia_create( interface.name, interface.major_version,
3984 interface.minor_version, &ia) != IT_SUCCESS) {
3985
3986     /* The IA wasn't found. Assuming sufficient resources were available,
3987        this can happen if the Interface that was retrieved by the
3988        it_interface_list call is no longer available. */
3989 }
3990
3991 if (it_ia_query(ia, &infop) != IT_SUCCESS) {
3992
3993     /* This can happen if the Implementation didn't have sufficient
3994        resources to return the information for the IA. */
3995 }
3996
3997 if ((infop->transport_type != interface.transport_type) {
3998
```

```
3999     /* This can happen if the Interface that was retrieved by the
4000     it_interface_list call is no longer available. (A different
4001     Interface with the same name as was retrieved by it_interface_list
4002     is available as it turns out. The most likely reason this happened
4003     is that a new Interface was added to the system between the time
4004     it_interface_list was called and the time that it_ia_create was
4005     called.) */
4006     }
4007
4008     /* Validation of the information returned by it_interface_list is
4009     complete. */
```

4010 **APPLICATION USAGE**

4011 The Interface Adapter associated with a given *name* can change between the time that the
4012 *it_interface_list* routine is called and the time that *it_ia_create* is called to actually create the IA.
4013 For that reason, the Consumer should check after it has created the IA to ensure that the
4014 information it retrieved from the *it_interface_list* call is still valid.

4015 **SEE ALSO**

4016 [*it_ia_create\(\)*](#), [*it_ia_info_t*](#)

it_listen_create()

4017

4018 NAME

4019 `it_listen_create` – create a Listen Point for incoming Connection Requests to a Connection
4020 Qualifier

4021 SYNOPSIS

```
4022 #include <it_api.h>
4023
4024 it_status_t it_listen_create(
4025     IN      it_ia_handle_t      ia_handle,
4026     IN      size_t              spigot_id,
4027     IN      it_evd_handle_t     connect_evd,
4028     IN      it_listen_flags_t   flags,
4029     IN OUT  it_conn_qual_t      *conn_qual,
4030     OUT     it_listen_handle_t  *listen_handle
4031 );
4032
4033 typedef enum {
4034     IT_LISTEN_NO_FLAG           = 0x0000,
4035     IT_LISTEN_CONN_QUAL_INPUT  = 0x0001,
4036     IT_LISTEN_SUPPRESS_IRD_ORD = 0x0002
4037 } it_listen_flags_t;
```

4038 DESCRIPTION

4039 *ia_handle*: Interface Adapter Handle.

4040 *spigot_id*: Interface Adapter Spigot identifier.

4041 *connect_evd*: The Handle of the Simple Event Dispatcher where Connection Request
4042 Events for this Listen Point will be posted. The Event Stream Type of the
4043 Simple Event Dispatcher must be `IT_CM_REQ_EVENT_STREAM`.

4044 *flags*: Bitwise OR of flag values. Can specify whether the Connection Qualifier is
4045 an input or output parameter and whether or not IRD/ORD values are to be
4046 used (if applicable).

4047 *conn_qual*: The Connection Qualifier for which the Consumer wants to listen for
4048 Connection Requests.

4049 *listen_handle*: Upon successful return points to a Handle to the created Listen Point.

4050 *it_listen_create* establishes a Listen Point for incoming Connection Requests for a particular
4051 Connection Qualifier on the Spigot identified. Incoming Connection Request Events will be
4052 posted to the Simple Event Dispatcher specified until the Listen Point is destroyed. The
4053 *listen_handle* returned can be passed to *it_listen_free* when the Listen Point is no longer needed.

4054 When the `IT_LISTEN_CONN_QUAL_INPUT` bit is set in *flags*, *conn_qual* is an input
4055 parameter. When this bit is clear, *conn_qual* is an output parameter and an available Connection
4056 Qualifier is returned through that parameter.

4057 The default behavior of the IT-API is to attempt to negotiate IRD/ORD between the active and
4058 passive sides as described in Chapter 5 where supported by the IA. For an IA where the

4059 *it_ia_info_t* values *ird_ord_ia_support* and *ird_ord_suppressible* are both IT_TRUE, the
4060 IT_LISTEN_SUPPRESS_IRD_ORD bit may be set by the Consumer in order to cause the IA
4061 not to perform IRD/ORD negotiation. For such an IA, if the
4062 IT_LISTEN_SUPPRESS_IRD_ORD bit is cleared in *flags*, then the IRD/ORD Endpoint
4063 attributes will be negotiated. For an IA where *ird_ord_ia_support* is IT_FALSE, the
4064 IT_LISTEN_SUPPRESS_IRD_ORD bit is ignored. For an IA where *ird_ord_ia_support* is
4065 IT_TRUE but *ird_ord_suppressible* is IT_FALSE, attempting to set
4066 IT_LISTEN_SUPPRESS_IRD_ORD will yield an immediate error.

4067 A backlog for the incoming Connection Request Events is provided by the size of the Simple
4068 Event Dispatcher to which the Events are directed. If a Connection Request arrives while the
4069 Simple Event Dispatcher is full, it is discarded and the Active side of the Connection
4070 establishment attempt will receive an IT_CM_MSG_CONN_NONPEER_REJECT_EVENT
4071 Event, with IT_CN_REJ_TIMEOUT as the reject reason code.

4072 **RETURN VALUE**

4073 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

4074	IT_ERR_INVALID_CONN_QUAL	The Connection Qualifier (<i>conn_qual</i>) was invalid.
4075	IT_ERR_CONN_QUAL_BUSY	The Connection Qualifier was already in use.
4076	IT_ERR_NO_PERMISSION	The Consumer did not have the proper permissions to perform the requested operation.
4077		
4078	IT_ERR_RESOURCES	The requested operation failed due to insufficient resources.
4079		
4080	IT_ERR_INVALID_CONN_EVD	The Connection Simple Event Dispatcher Handle was invalid.
4081		
4082	IT_ERR_INVALID_IA	The Interface Adapter Handle (<i>ia_handle</i>) was invalid.
4083	IT_ERR_INVALID_SPIGOT	An invalid Spigot ID was specified.
4084	IT_ERR_INVALID_FLAGS	The <i>flags</i> value was invalid.
4085	IT_ERR_INVALID_EVD_TYPE	The Event Stream Type for the Event Dispatcher was invalid.
4086		
4087	IT_ERR_IA_CATASTROPHE	The IA has experienced a catastrophic error and is in the disabled state. None of the output parameters from this routine are valid. See <i>it_ia_info_t</i> for a description of the disabled state.
4088		
4089		
4090		

4091 **SEE ALSO**

4092 *it_listen_free()*, *it_listen_query()*, *it_cm_msg_events*

it_listen_free()

4093

4094 NAME

4095 `it_listen_free` – free a Listen Point

4096 SYNOPSIS

4097 `#include <it_api.h>`

4098

4099 `it_status_t it_listen_free(`

4100 `IN it_listen_handle_t listen_handle`

4101 `);`

4102 DESCRIPTION

4103 *listen_handle* Identifies the Listen Point to be destroyed.

4104 Frees a Listen Point associated with a Connection Qualifier. Upon return no more Connection
4105 Requests will be posted for the associated Connection Qualifier. Previously posted un-reaped
4106 Connection Requests, if any, will remain valid on the *connect_evd* and therefore can be used as
4107 input to either *it_ep_accept* or *it_reject*.

4108 Once *it_listen_free* returns, *listen_handle* may no longer be used.

4109 RETURN VALUE

4110 A successful call returns `IT_SUCCESS`. Otherwise, an error code is returned as described below:

4111 `IT_ERR_INVALID_LISTEN` The Listen Point Handle (*listen_handle*) was invalid.

4112 `IT_ERR_IA_CATASTROPHE` The IA has experienced a catastrophic error and is in the
4113 disabled state. None of the output parameters from this routine
4114 are valid. See *it_ia_info_t* for a description of the disabled
4115 state.

4116 SEE ALSO

4117 *it_listen_create()*, *it_listen_query()*

4118

it_listen_query()

4119

4120 NAME

4121 `it_listen_query` – query parameters associated with a Listen Point

4122 SYNOPSIS

```
4123 #include <it_api.h>
4124
4125 it_status_t it_listen_query(
4126     IN    it_listen_handle_t    listen_handle,
4127     IN    it_listen_param_mask_t mask,
4128     OUT   it_listen_param_t     *params
4129 );
4130
4131 typedef enum {
4132     IT_LISTEN_PARAM_ALL           = 0x0001,
4133     IT_LISTEN_PARAM_IA_HANDLE    = 0x0002,
4134     IT_LISTEN_PARAM_SPIGOT_ID    = 0x0004,
4135     IT_LISTEN_PARAM_CONNECT_EVD  = 0x0008,
4136     IT_LISTEN_PARAM_CONN_QUAL   = 0x0010
4137 } it_listen_param_mask_t;
4138
4139 typedef struct {
4140     it_ia_handle_t    ia_handle;    /* IT_LISTEN_PARAM_IA_HANDLE */
4141     size_t            spigot_id;    /* IT_LISTEN_PARAM_SPIGOT_ID */
4142     it_evd_handle_t   connect_evd;  /* IT_LISTEN_PARAM_CONNECT_EVD */
4143     it_conn_qual_t    connect_qual; /* IT_LISTEN_PARAM_CONN_QUAL */
4144 } it_listen_param_t;
```

4145 DESCRIPTION

4146 *listen_handle* Handle associated with the Listen Point being queried.

4147 *mask* Bitwise OR of flags for desired parameters.

4148 *params* Pointer to Consumer-allocated structure whose members are written with
4149 the desired Listen Point parameters and attributes.

4150 *it_listen_query* queries the parameters associated with a Listen Point. On return, each field of
4151 *params* is only valid if the corresponding flag as shown in the Synopsis is set in the *mask*
4152 argument. The *mask* value `IT_LISTEN_PARAM_ALL` causes all fields to be returned.

4153 The definition of each field of *params* follows:

4154 *ia_handle*: Interface Adapter Handle.

4155 *spigot_id* Interface Adapter Spigot identifier.

4156 *connect_evd* The Handle of the Simple Event Dispatcher where incoming Connection
4157 Request Events for this Listen Point are posted.

4158 *connect_qual* The Connection Qualifier associated with the Listen Point.

4159 **RETURN VALUE**

4160 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

4161 IT_ERR_INVALID_LISTEN The Listen Point Handle (*listen_handle*) was invalid.

4162 IT_ERR_INVALID_MASK The mask contained invalid flag values.

4163 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the
4164 disabled state. None of the output parameters from this routine
4165 are valid. See *it_ia_info_t* for a description of the disabled
4166 state.

4167 **SEE ALSO**

4168 *it_listen_free()*, *it_listen_create()*

4169

it_lmr_create()

4170

4171 NAME

4172 `it_lmr_create` – create a Local Memory Region (LMR) and register with an Interface Adapter

4173 SYNOPSIS

```
4174 #include <it_api.h>
4175
4176 it_status_t it_lmr_create(
4177     IN      it_pz_handle_t    pz_handle,
4178     IN      void              *addr,
4179     IN      it_length_t       length,
4180     IN      it_addr_mode_t    addr_mode,
4181     IN      it_mem_priv_t     privs,
4182     IN      it_lmr_flag_t     flags,
4183     IN      uint32_t          shared_id,
4184     OUT     it_lmr_handle_t   *lmr_handle,
4185     IN OUT  it_rmr_context_t  *rmr_context
4186 );
4187
4188 typedef uint32_t it_rmr_context_t;
4189
4190 #ifdef IT_32BIT
4191     typedef uint32_t it_length_t; /* a 32-bit platform */
4192 #else
4193     typedef uint64_t it_length_t; /* a 64-bit platform */
4194 #endif
4195
4196 typedef enum {
4197     IT_LMR_FLAG_NONE          = 0x0001,
4198     IT_LMR_FLAG_SHARED        = 0x0002,
4199     IT_LMR_FLAG_NONCOHERENT   = 0x0004
4200 } it_lmr_flag_t;
```

4201 DESCRIPTION

4202	<i>pz_handle</i>	Protection Zone in which to create LMR.
4203	<i>addr</i>	Starting address of LMR.
4204	<i>length</i>	Length of LMR in bytes.
4205	<i>addr_mode</i>	Addressing mode of LMR to be created.
4206	<i>privs</i>	Bitwise OR of access privilege values for LMR, taken from <i>it_mem_priv_t</i> .
4207	<i>flags</i>	Bitwise OR of modifier flags.
4208	<i>shared_id</i>	Optional identifier for sharing Interface Adapter translation resources.
4209	<i>lmr_handle</i>	Returned Handle for created LMR.
4210	<i>rmr_context</i>	Optionally returned Context allowing remote access to this LMR.

4211 The *it_lmr_create* routine allows an Interface Adapter to access a contiguous Local Memory
4212 Region in a given linear address space known to the Consumer; memory that is to be the Source
4213 or Destination of a DTO must first be registered using this call. The region starts at address *addr*,
4214 also known as the Base Address of the LMR, and extends for *length* bytes. The specified address
4215 range must already be valid in the Consumer's linear address space. The Interface Adapter is
4216 implicitly identified by the *pz_handle* argument. Registering a memory range that does not
4217 correspond to physically backed memory, such as the non-cacheable I/O address space, may
4218 work on some Implementations but not others. Applications that rely on this behavior will not be
4219 portable. The range can refer to memory that is exclusive to the calling process, or is being
4220 shared with other processes. Some Interface Adapters may require a memory region to be locked
4221 in physical memory. Such locking, if required, will be performed by the *it_lmr_create*
4222 implementation and is not the Consumer's responsibility.

4223 If the *addr_mode* argument (*it_addr_mode_t*) selects Absolute Addressing, then DTOs will
4224 access the LMR through absolute addresses in the given linear address space. If the *addr_mode*
4225 argument selects Relative Addressing, then DTOs will access the LMR through offsets relative
4226 to the Base Address of the LMR. Consumers can check the IA attribute
4227 *addr_mode_relative_support* to determine whether Relative Addressing is supported.

4228 The type of access granted is specified by the *privs* argument as the bitwise-inclusive OR of one
4229 or more of the bit values `IT_PRIV_LOCAL_READ`, `IT_PRIV_LOCAL_WRITE`,
4230 `IT_PRIV_REMOTE_READ`, and `IT_PRIV_REMOTE_WRITE`. Unless otherwise noted below,
4231 all bit combinations are allowed. See *it_mem_priv_t* for bit definitions and predefined bit
4232 combinations. It is invalid to set *privs* to 0 (no access privileges), or to include
4233 `IT_PRIV_REMOTE_WRITE` without also including `IT_PRIV_LOCAL_WRITE`. Consumers
4234 are advised to use `IT_PRIV_LOCAL_READ` or `IT_PRIV_LOCAL` (which equals
4235 `IT_PRIV_LOCAL_READ | IT_PRIV_LOCAL_WRITE`) as local access privileges for
4236 maximum portability between transports (see also Extended Description). The access privileges
4237 required for a DTO's Source or Destination buffer are specified on the corresponding DTO
4238 reference page.

4239 It is not possible to grant access privileges to which a process is not already entitled. If the
4240 calling process does not have read or write access privileges to the memory region, then any
4241 attempt to grant those privileges to the Interface Adapter will cause *it_lmr_create* to fail.

4242 The *flags* argument is a bitwise OR of zero or more of the following options. The value
4243 `IT_LMR_FLAG_NONE` or 0 may be used to specify no options.

4244	<code>IT_LMR_FLAG_SHARED</code>	This flag may be used to conserve finite Interface Adapter translation resources by sharing resources between multiple LMRs. The LMRs may have been created in the caller's process or in a different process. If set, then the Implementation will create a new LMR that re-uses resources from a matching LMR, which is defined as an existing LMR that was created using the same value of the <i>shared_id</i> argument, refers to the same physical memory pages as the new LMR, and has the same coherency mode. If any of these conditions are not met, the LMRs do not match. If a matching LMR is not found, then a new LMR is created and <i>shared_id</i> is associated with it. The value of <i>shared_id</i> is only an efficiency aid for the matching process and need not be unique. For example, if unrelated callers supply the same value for <i>shared_id</i> , matches for an LMR will still be found
4245		
4246		
4247		
4248		
4249		
4250		
4251		
4252		
4253		
4254		
4255		
4256		
4257		

4258 if they exist, with no false matches, but the search may take
4259 longer on some Implementations. If IT_LMR_FLAG_SHARED
4260 is not specified, then *shared_id* is ignored.

4261 IT_LMR_FLAG_NONCOHERENT Controls whether an LMR is created in coherent or non-coherent
4262 mode. Coherent mode is the default and is supported by all
4263 Implementations. Non-coherent mode is not supported by all
4264 Implementations, and IT_LMR_FLAG_NONCOHERENT is
4265 silently ignored on such Implementations.

4266 Set IT_LMR_FLAG_NONCOHERENT to create an LMR in non-coherent mode. Non-coherent
4267 mode may yield higher throughput for large DTOs, but may also increase latency for small
4268 DTOs. The downside of requesting non-coherent mode is that the Consumer must synchronize
4269 between local and remote access to the memory region using the *it_lmr_sync_rdma_write* and
4270 *it_lmr_sync_rdma_read* calls.

4271 The coherency mode of an LMR is inherited by any RMR that is linked to it.

4272 An RMR Context allowing remote access to the memory region will be created if the *privs*
4273 argument includes either IT_PRIV_REMOTE_READ or IT_PRIV_REMOTE_WRITE. The
4274 Context will be returned in the location pointed to by *rmr_context* if the input value of
4275 *rmr_context* is not NULL. The returned *rmr_context* is only valid if *privs* includes remote
4276 privileges and the call returns successfully; otherwise, it is undefined. The RMR Context may
4277 also be retrieved using *it_lmr_query*. The *rmr_context* is returned in network byte order, and
4278 may be passed by value to any remote process that wishes to use the Context in DTOs that target
4279 the corresponding LMR. Destroying the LMR using *it_lmr_free* revokes remote access using
4280 this RMR Context. Calling *it_lmr_modify* with remote access enabled results in a new RMR
4281 Context being associated with the LMR and revokes remote access using any previous RMR
4282 Contexts, as long as the new RMR Context is not a re-use of a previously generated RMR
4283 Context.

4284 The newly created LMR Handle is returned in the *lmr_handle* argument. A process may create
4285 multiple LMRs, and the address ranges of different LMRs may overlap.

4286 After a memory range has been registered with the IA using *it_lmr_create*, the Consumer should
4287 not call routines outside of the IT-API that would invalidate any part of the memory referred to
4288 by the LMR or revoke access privileges that were granted to the IA at registration time.
4289 Disallowed operations include but are not limited to unmapping part of the range using *munmap*,
4290 revoking privileges using *mprotect*, unlocking memory using *munlock*, truncating a file for file-
4291 backed regions, etc. Violation of this rule may result in DTO failures, data corruption in the
4292 Consumer's LMR, and/or program termination. These effects may extend to other Consumer
4293 processes if the LMR is in shared memory. However, the Implementation must prevent any
4294 adverse effect on unrelated processes that do not use this memory object.

4295 EXTENDED DESCRIPTION

4296 For the InfiniBand Transport, it is invalid not to include IT_PRIV_LOCAL_READ in the *privs*
4297 argument, or to include IT_PRIV_REMOTE_WRITE without also including
4298 IT_PRIV_LOCAL_WRITE when creating an LMR; such attempts will result in an
4299 IT_ERR_INVALID_PRIVS immediate error. Moreover, if the Consumer wishes to link an
4300 RMR with the IT_PRIV_REMOTE_WRITE access privilege to the LMR, then the LMR's
4301 access privileges must include IT_PRIV_LOCAL_WRITE.

4302 Also for InfiniBand, the Implementation may round the requested *addr* down and/or round the
4303 requested *length* up, and thus allow the Interface Adapter to access memory slightly outside the
4304 specified boundaries, but never beyond the IA pages that include the requested starting and
4305 ending addresses. The actual starting address and the actual length may be queried using
4306 *it_lmr_query*. Note that if the *privs* argument enables remote access, then remote Consumers
4307 may also access memory slightly outside the requested boundaries. If this is undesirable, the
4308 Consumer should not enable remote access in this routine, but should instead create and link an
4309 RMR using *it_rmr_create* and *it_rmr_link*, respectively, which guarantees byte-level registration
4310 granularity.

4311 **BACKWARDS COMPATIBILITY**

4312 *it_lmr_create* called with *addr_mode* set to IT_ADDR_MODE_ABSOLUTE behaves
4313 identically as *it_lmr_create* (v1.0).

4314 See also Appendix C.

4315 **RETURN VALUE**

4316 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

4317	IT_ERR_ACCESS	The Consumer was not allowed to have the requested
4318		memory privileges.
4319	IT_ERR_FAULT	Part or all of the supplied address range was invalid.
4320	IT_ERR_INVALID_ADDR_MODE	The addressing mode (<i>addr_mode</i>) was invalid or
4321		unsupported.
4322	IT_ERR_INVALID_FLAGS	The <i>flags</i> value was invalid.
4323	IT_ERR_INVALID_PRIVS	The requested memory privileges (<i>privs</i>) contained an
4324		invalid flag.
4325	IT_ERR_INVALID_PZ	The Protection Zone Handle (<i>pz_handle</i>) was invalid.
4326	IT_ERR_RESOURCES	The requested operation failed due to insufficient
4327		resources.
4328	IT_ERR_RESOURCE_LMR_LENGTH	The underlying transport could not allocate an LMR of
4329		the requested <i>length</i> at this time.
4330	IT_ERR_IA_CATASTROPHE	The IA has experienced a catastrophic error and is in the
4331		disabled state. None of the output parameters from this
4332		routine are valid. See <i>it_ia_info_t</i> for a description of the
4333		disabled state.

4334 **APPLICATION USAGE**

4335 Memory that is to be the Source or Destination of a DTO must first be registered using
4336 *it_lmr_create*. The Consumer typically copies the returned *lmr_handle* to an *it_lmr_triplet_t*
4337 structure that is used in DTO calls such as *it_post_send*.

4338 If the LMR is created with flags that enable remote access, then the Consumer typically passes
4339 the returned RMR Context to a remote peer using a DTO. The remote peer uses the RMR
4340 Context in RDMA calls such as *it_post_rdma_write* that access memory within the range of the
4341 LMR.

4342 Consumers can enable remote access more selectively over any portion of an LMR by creating
4343 an RMR and linking the RMR to the desired region of the LMR using *it_rmr_link*. This
4344 operation returns an RMR Context.

4345 **SEE ALSO**

4346 *it_lmr_free()*, *it_lmr_query()*, *it_lmr_modify()*, *it_lmr_sync_rdma_read()*,
4347 *it_lmr_sync_rdma_write()*, *it_addr_mode_t*, *it_mem_priv_t*

4348

it_lmr_free()

4349

4350 NAME

4351 `it_lmr_free` – destroy a Local Memory Region

4352 SYNOPSIS

```
4353 #include <it_api.h>
4354
4355 it_status_t it_lmr_free(
4356     IN it_lmr_handle_t lmr_handle
4357 );
```

4358 DESCRIPTION

4359 *lmr_handle* Handle of Local Memory Region to be destroyed.

4360 The *it_lmr_free* routine destroys the Local Memory Region *lmr_handle*. On return, the Handle
4361 *lmr_handle* and associated RMR context (if any) may no longer be used. A Local Memory
4362 Region may not be destroyed if it has an RMR linked to it; an attempt to do so will fail and
4363 neither the LMR, its RMR Context (if any), nor any linked RMR(s) will be affected. *it_lmr_free*
4364 does not invalidate the memory range represented by *lmr_handle*, and the caller may continue to
4365 reference memory in this range for non-transport operations. If the memory range was locked in
4366 physical memory as a side-effect of the corresponding *it_lmr_create* call, then it will be
4367 unlocked immediately if no portion of the range overlaps with the range of other non-freed
4368 LMRs. Otherwise, the unlock operation may be deferred until the overlapping LMRs are
4369 themselves freed. Note that these may include LMRs created by other Consumers if the range is
4370 in shared memory.

4371 LMRs with memory ranges that overlap the range of *lmr_handle* are not affected by its
4372 destruction. Outstanding DTOs, Link, and Unlink operations that use an LMR that has been
4373 destroyed may or may not complete successfully.

4374 RETURN VALUE

4375 A successful call returns `IT_SUCCESS`. Otherwise, an error code is returned as described below:

4376	<code>IT_ERR_INVALID_LMR</code>	The Local Memory Region Handle (<i>lmr_handle</i>) was
4377		invalid.
4378	<code>IT_ERR_LMR_BUSY</code>	The Local Memory Region was still referenced by a
4379		Remote Memory Region.
4380	<code>IT_ERR_IA_CATASTROPHE</code>	The IA has experienced a catastrophic error and is in the
4381		disabled state. None of the output parameters from this
4382		routine are valid. See <i>it_ia_info_t</i> for a description of the
4383		disabled state.

4384 SEE ALSO

4385 *it_lmr_create()*, *it_lmr_query()*, *it_lmr_modify()*

4386

it_lmr_modify()

4387

4388 NAME

4389 `it_lmr_modify` – modify selected attributes of a Local Memory Region

4390 SYNOPSIS

```
4391 #include <it_api.h>
4392
4393 it_status_t it_lmr_modify(
4394     IN          it_lmr_handle_t      lmr_handle,
4395     IN          it_lmr_param_mask_t  mask,
4396     IN  const   it_lmr_param_t      *params
4397 );
```

4398 DESCRIPTION

4399 *lmr_handle* Local Memory Region.

4400 *mask* Bitwise OR of flags for specified parameters.

4401 *params* Structure whose members contain the new parameter values.

4402 The *it_lmr_modify* routine changes selected attributes of the Local Memory Region *lmr_handle*.
4403 Attributes to be modified are specified by flags in *mask*. New values for the attributes are
4404 specified by the corresponding fields in the structure pointed to by *params*. Fields and their
4405 corresponding flag values are shown in *it_lmr_param_t*. Note that attributes represented by
4406 fields of *it_lmr_param_t* that are not shown below cannot be modified. The definition of each
4407 field follows:

4408 *pz* The new Protection Zone Handle for the LMR.

4409 *privs* The new memory access privileges for the LMR. See *it_mem_priv_t* and
4410 *it_lmr_create* for flag definitions and restrictions.

4411 If remote access privileges are specified in *privs* and the routine returns successfully, then a new
4412 RMR Context has been created and associated with the LMR. The LMR's new associated RMR
4413 Context may be retrieved using *it_lmr_query*. The Implementation of *it_lmr_modify* strives to
4414 generate a new RMR Context that is different from any RMR Contexts generated previously. As
4415 long as the new RMR Context does not match a previously generated RMR Context, remote
4416 access using a previously generated RMR Context is revoked. If no remote access privileges are
4417 specified in *privs* and the routine returns successfully, then no RMR Context remains associated
4418 with the LMR, and remote access to the LMR is disabled. If the *lmr_handle* is invalidated due to
4419 an error, any associated RMR Context is invalidated, and any IA access to the LMR is disabled.

4420 A Local Memory Region may not be modified if it is still referenced by bound Remote Memory
4421 Regions; an attempt to do so will fail with an error return, and the LMR will not be modified or
4422 affected.

4423 The Consumer should not modify an LMR whose LMR Handle or RMR Context is used in
4424 outstanding DTOs, RMR Link, or RMR Unlink operations. The Consumer must dequeue the
4425 Completion Events for all such operations prior to modifying the LMR. If this rule is not
4426 followed, the Outstanding Operations may fail and complete with an error status.

4427 If the *lmr_handle* or *mask* parameter is invalid, or the LMR has any RMRs linked to it, then the
4428 operation fails and, if *lmr_handle* was valid, the LMR is not modified or affected and any RMRs
4429 linked to the LMR are also unaffected.

4430 For all other errors, the *lmr_handle* and any associated RMR Context are invalidated and may no
4431 longer be used. Any resources that were associated with the LMR are freed, as if the Consumer
4432 was calling *it_lmr_free* on the *lmr_handle*.

4433 **RETURN VALUE**

4434 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

4435 IT_ERR_ACCESS The Consumer was not allowed to have the requested memory
4436 privileges.

4437 IT_ERR_INVALID_LMR The Local Memory Region Handle (*lmr_handle*) was invalid.

4438 IT_ERR_INVALID_MASK The *mask* contained invalid flag values.

4439 IT_ERR_INVALID_PRIVS The requested memory privileges (*privs*) contained an invalid flag.

4440 IT_ERR_INVALID_PZ The Protection Zone Handle (*pz_handle*) was invalid.

4441 IT_ERR_LMR_BUSY The Local Memory Region was still referenced by a Remote
4442 Memory Region.

4443 IT_ERR_RESOURCES The requested operation failed due to insufficient resources.

4444 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the disabled
4445 state. None of the output parameters from this routine are valid.
4446 See *it_ia_info_t* for a description of the disabled state.

4447 **APPLICATION USAGE**

4448 Although *it_lmr_modify* can be used to change the remote access privileges for an LMR, this is a
4449 much more expensive operation than linking an RMR to an LMR using *it_rmr_link*.

4450 **SEE ALSO**

4451 *it_lmr_create()*, *it_lmr_free()*, *it_lmr_query()*, *it_addr_mode_t*, *it_lmr_param_t*,
4452 *it_lmr_param_mask_t*, *it_mem_priv_t*

4453

it_lmr_query()

4454

4455 NAME

4456 `it_lmr_query` – get attributes of a Local Memory Region

4457 SYNOPSIS

```
4458 #include <it_api.h>
4459
4460 it_status_t it_lmr_query(
4461     IN    it_lmr_handle_t    lmr_handle,
4462     IN    it_lmr_param_mask_t mask,
4463     OUT   it_lmr_param_t     *params
4464 );
4465
4466 typedef enum {
4467     IT_LMR_PARAM_ALL           = 0x000001,
4468     IT_LMR_PARAM_IA           = 0x000002,
4469     IT_LMR_PARAM_PZ           = 0x000004,
4470     IT_LMR_PARAM_ADDR         = 0x000008,
4471     IT_LMR_PARAM_LENGTH       = 0x000010,
4472     IT_LMR_PARAM_MEM_PRIV     = 0x000020,
4473     IT_LMR_PARAM_FLAG         = 0x000040,
4474     IT_LMR_PARAM_SHARED_ID    = 0x000080,
4475     IT_LMR_PARAM_RMR_CONTEXT  = 0x000100,
4476     IT_LMR_PARAM_ACTUAL_ADDR  = 0x000200,
4477     IT_LMR_PARAM_ACTUAL_LENGTH = 0x000400,
4478     IT_LMR_PARAM_ADDR_MODE    = 0x000800
4479 } it_lmr_param_mask_t;
4480
4481 typedef struct {
4482     it_ia_handle_t    ia;           /* IT_LMR_PARAM_IA */
4483     it_pz_handle_t    pz;           /* IT_LMR_PARAM_PZ */
4484     void               *addr;       /* IT_LMR_PARAM_ADDR */
4485     it_length_t        length;      /* IT_LMR_PARAM_LENGTH */
4486     it_mem_priv_t      privs;       /* IT_LMR_PARAM_MEM_PRIV */
4487     it_lmr_flag_t      flags;       /* IT_LMR_PARAM_FLAG */
4488     uint32_t           shared_id;    /* IT_LMR_PARAM_SHARED_ID */
4489     it_rmr_context_t   rmr_context; /* IT_LMR_PARAM_RMR_CONTEXT */
4490     void               *actual_addr; /* IT_LMR_PARAM_ACTUAL_ADDR */
4491     it_length_t        actual_length; /* IT_LMR_PARAM_ACTUAL_LENGTH */
4492     it_addr_mode_t     addr_mode;    /* IT_LMR_PARAM_ADDR_MODE */
4493 } it_lmr_param_t;
```

4494 DESCRIPTION

4495 *lmr_handle* Local Memory Region.

4496 *mask* Bitwise OR of flags for desired parameters.

4497 *params* Structure whose members are written with the desired parameters.

4498 The *it_lmr_query* routine returns the desired attributes of the Local Memory Region *lmr_handle*
4499 in the structure pointed to by *params*. On return, each field of *params* is only valid if the

4500		corresponding flag as shown in the Synopsis is set in the <i>mask</i> argument. The <i>mask</i> value
4501		IT_LMR_PARAM_ALL causes all fields to be returned.
4502		The definition of each field of <i>params</i> follows:
4503	<i>ia</i>	The Interface Adapter Handle specified to create the LMR.
4504	<i>pz</i>	The Protection Zone Handle specified to create the LMR.
4505	<i>addr</i>	The requested starting address or, equivalently, Base Address of the LMR.
4506		To be interpreted as an absolute address, regardless of <i>addr_mode</i> .
4507	<i>length</i>	The requested length in bytes of the LMR.
4508	<i>privs</i>	The memory access privileges of the LMR. For the definition of bits, see
4509		it_mem_priv_t .
4510	<i>flags</i>	The flags specified to create the LMR.
4511	<i>shared_id</i>	The <i>shared_id</i> specified to create the LMR, if <i>flags</i> included
4512		IT_LMR_FLAG_SHARED. Otherwise, undefined.
4513	<i>rmr_context</i>	The RMR Context associated with the LMR, or undefined if <i>privs</i> does not
4514		include remote access. Returned in network byte order.
4515	<i>actual_addr</i>	The actual starting address for which bounds checking is done for data
4516		transfers. To be interpreted as an absolute address, regardless of
4517		<i>addr_mode</i> .
4518	<i>actual_length</i>	The actual length for which bounds checking is done for data transfers.
4519	<i>addr_mode</i>	The addressing mode of the LMR.

4520 EXTENDED DESCRIPTION

4521 Implementations based on iWARP and Implementations on any transport supporting Relative
4522 Addressing will support LMR creation and bounds checking with byte-level granularity.

4523 Implementations based on InfiniBand 1.2 can be expected to support LMR creation and bounds
4524 checking with byte-level granularity.

4525 For Implementations with byte-level granularity for LMRs, the actual starting address and actual
4526 length will equal the requested starting address and requested length, respectively. For LMRs
4527 with Relative Addressing, LMR byte-level granularity ensures that the starting address of the
4528 LMR is unambiguous.

4529 RETURN VALUE

4530 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

4531	IT_ERR_INVALID_LMR	The Local Memory Region Handle (<i>lmr_handle</i>) was invalid.
4532	IT_ERR_INVALID_MASK	The <i>mask</i> contained invalid flag values.
4533	IT_ERR_IA_CATASTROPHE	The IA has experienced a catastrophic error and is in the disabled
4534		state. None of the output parameters from this routine are valid.
4535		See it_ia_info_t for a description of the disabled state.

4536 **SEE ALSO**
4537 *it_lmr_create(), it_lmr_modify(), it_lmr_free(), it_addr_mode_t, it_mem_priv_t*

it_lmr_sync_rdma_read()

4538

4539 NAME

4540 `it_lmr_sync_rdma_read` – make memory changes visible to an incoming RDMA Read operation

4541 SYNOPSIS

```
4542 #include <it_api.h>
4543
4544 it_status_t it_lmr_sync_rdma_read(
4545     IN const it_lmr_triplet_t *local_segments,
4546     IN size_t num_segments
4547 );
```

4548 DESCRIPTION

4549 *local_segments* Array of buffer segments.

4550 *num_segments* Number of segments in the array.

4551 The *it_lmr_sync_rdma_read* routine is needed if and only if an LMR was created in non-
4552 coherent mode using `IT_LMR_FLAG_NONCOHERENT`.

4553 If a Local Memory Region is created in non-coherent mode, then the Consumer must call
4554 *it_lmr_sync_rdma_read* after modifying data in a memory range in this region that will be the
4555 target of an *incoming* RDMA Read operation. *it_lmr_sync_rdma_read* must be called after the
4556 Consumer has modified the memory range but before the RDMA Read operation starts, and the
4557 memory range that will be accessed by the RDMA Read must be supplied by the caller in the
4558 *local_segments* array. After this call returns, the RDMA Read operation may safely see the
4559 modified contents of the memory range. It is permissible to batch synchronizations for multiple
4560 RDMA Read operations in a single call, by passing a *local_segments* array that includes all
4561 modified memory ranges. The *local_segments* entries need not contain the same LMR, and need
4562 not be in the same Protection Zone.

4563 If an RDMA Read operation on an LMR created in non-coherent mode attempts to read from a
4564 memory range that is not properly synchronized using *it_lmr_sync_rdma_read*, the returned
4565 contents are undefined.

4566 If the *local_segments* specified by the Consumer contain both normal LMRs (i.e., those created
4567 with `IT_LMR_FLAG_NONCOHERENT` cleared) and non-coherent LMRs (i.e., those created
4568 with `IT_LMR_FLAG_NONCOHERENT` set), only the non-coherent LMRs will be
4569 synchronized by this routine; the normal LMRs will be unaffected.

4570 RETURN VALUE

4571 This call is a no-op and always returns successfully if the Implementation does not support non-
4572 coherent mode, or if none of the LMRs in *local_segments* were created using the
4573 `IT_LMR_FLAG_NONCOHERENT` flag.

4574 A successful call returns `IT_SUCCESS`. Otherwise, an error code is returned as described below:

4575 `IT_ERR_RANGE` The address range for a local segment fell outside the boundaries
4576 of the corresponding Local Memory Region and the Local
4577 Memory Region was created in non-coherent mode.

4578 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the disabled
4579 state. None of the output parameters from this routine are valid.
4580 See *it_ia_info_t* for a description of the disabled state.

4581 **APPLICATION USAGE**

4582 Determining when an RDMA Read will start and what memory range it will read is the
4583 Consumer's responsibility. One possibility is to have the Consumer that is modifying memory to
4584 call *it_lmr_sync_rdma_read* and then post a Send DTO message that identifies the range in the
4585 body of the Send. The Consumer wishing to do the RDMA Read can receive this message and
4586 thus know when it is safe to initiate the RDMA Read operation.

4587 **SEE ALSO**

4588 *it_lmr_create()*, *it_lmr_sync_rdma_write()*, *it_lmr_triplet_t*

4589

it_lmr_sync_rdma_write()

4590

4591 NAME

4592 it_lmr_sync_rdma_write – make effects of an incoming RDMA Write operation visible to
4593 Consumer

4594 SYNOPSIS

```
4595 #include <it_api.h>  
4596  
4597 it_status_t it_lmr_sync_rdma_write(  
4598     IN const it_lmr_triplet_t *local_segments,  
4599     IN          size_t          num_segments  
4600 );
```

4601 DESCRIPTION

4602 *local_segments* Array of buffer segments.

4603 *num_segments* Number of segments in the array.

4604 The *it_lmr_sync_rdma_write* routine is needed if and only if an LMR was created in non-
4605 coherent mode using IT_LMR_FLAG_NONCOHERENT.

4606 If a Local Memory Region is created in non-coherent mode, then the Consumer must call
4607 *it_lmr_sync_rdma_write* before reading data from a memory range in this region that was the
4608 target of an *incoming* RDMA Write operation. *it_lmr_sync_rdma_write* must be called after the
4609 RDMA Write operation completes, and the memory range that was modified by the RDMA
4610 Write must be supplied by the caller in the *local_segments* array. After this call returns, the
4611 Consumer may safely see the modified contents of the memory range. It is permissible to batch
4612 synchronizations of multiple RDMA Write operations in a single call, by passing a
4613 *local_segments* array that includes all modified memory ranges. The *local_segments* entries need
4614 not contain the same LMR, and need not be in the same Protection Zone.

4615 The Consumer must also use *it_lmr_sync_rdma_write* when performing local writes to a
4616 memory range that was or will be the target of incoming RDMA Writes. After performing the
4617 local write, the Consumer must call *it_lmr_sync_rdma_write* before the RDMA Write is
4618 initiated. Conversely, after an RDMA Write completes, the Consumer must call
4619 *it_lmr_sync_rdma_write* before performing a local write to the same range.

4620 If the Consumer attempts to read from a memory range in an LMR that was created in non-
4621 coherent mode, without properly synchronizing using *it_lmr_sync_rdma_write*, the returned
4622 contents are undefined. If the Consumer attempts to write to a memory range without properly
4623 synchronizing, the contents of the memory range become undefined.

4624 If the *local_segments* specified by the Consumer contain both normal LMRs (i.e., those created
4625 with IT_LMR_FLAG_NONCOHERENT cleared) and non-coherent LMRs (i.e., those created
4626 with IT_LMR_FLAG_NONCOHERENT set), only the non-coherent LMRs will be
4627 synchronized by this routine; the normal LMRs will be unaffected.

4628 RETURN VALUE

4629 This call is a no-op and always returns successfully if the Implementation does not support non-
4630 coherent mode, or if none of the LMRs in *local_segments* were created using the
4631 IT_LMR_FLAG_NONCOHERENT flag.

4632 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

4633	IT_ERR_RANGE	The address range for a local segment fell outside the boundaries of the corresponding Local Memory Region and the Local Memory Region was created in non-coherent mode.
4634		
4635		
4636	IT_ERR_IA_CATASTROPHE	The IA has experienced a catastrophic error and is in the disabled state. None of the output parameters from this routine are valid. See <i>it_ia_info_t</i> for a description of the disabled state.
4637		
4638		

4639 **APPLICATION USAGE**

4640 Determining when an RDMA Write completes and determining which memory range was
 4641 modified is the Consumer's responsibility. One possibility is for the RDMA Write initiator to
 4642 post a Send DTO message after each RDMA Write that identifies the range in the body of the
 4643 Send. The Consumer at the target of the RDMA Write can receive the message and thus know
 4644 when and how to call *it_lmr_sync_rdma_write*.

4645 **SEE ALSO**

4646 *it_lmr_create()*, *it_lmr_sync_rdma_read()*, *it_lmr_triplet_t*

4647

4648

it_make_rdma_addr_absolute()

4649 **NAME**

4650 `it_make_rdma_addr_absolute` – make a platform-independent RDMA address for Absolute
4651 Addressing

4652 **SYNOPSIS**

```
4653 #include <it_api.h>  
4654  
4655 typedef uint64_t it_rdma_addr_t;  
4656  
4657 it_rdma_addr_t it_make_rdma_addr_absolute(  
4658     void *addr  
4659 );
```

4660 **DESCRIPTION**

4661 `addr` Local address.

4662 The `it_make_rdma_addr_absolute` routine takes a local address `addr` that may be the target of a
4663 remote operation, and returns a 64-bit platform-independent representation of that address in
4664 network byte order and suitable for Absolute Addressings, called an RDMA address (see
4665 [it_addr_mode_t](#) for a description of Absolute Addressing). A network peer may use this RDMA
4666 address in RDMA Read and Write operations. `it_make_rdma_addr_absolute` performs no
4667 validity checking on `addr`, so `addr` is not required to lie within a currently registered LMR when
4668 `it_make_rdma_addr_absolute` is called.

4669 **RETURN VALUE**

4670 This function always succeeds and returns a 64-bit RDMA address.

4671 **APPLICATION USAGE**

4672 The returned RDMA address must be communicated to a network peer in order to be used in
4673 RDMA operations. The Consumer is responsible for performing this communication.

4674 Because the RDMA address is in network byte order, a Consumer wishing to perform address
4675 arithmetic must first convert it to host byte order, which may be done using the [it_ntoh64](#)
4676 function. Derived addresses must be converted back to network byte order using [it_hton64](#)
4677 before being used in RDMA operations.

4678 **SEE ALSO**

4679 [it_post_rdma_write\(\)](#), [it_ntoh64](#), [it_hton64](#), [it_make_rdma_addr_relative](#), [it_addr_mode_t](#)

it_post_rdma_read()

4710

4711 NAME

4712 `it_post_rdma_read` – post an RDMA Read DTO to an Endpoint

4713 SYNOPSIS

```
4714 #include <it_api.h>
4715
4716 it_status_t it_post_rdma_read (
4717     IN          it_ep_handle_t      ep_handle,
4718     IN  const   it_lmr_triplet_t    *local_segments,
4719     IN          size_t              num_segments,
4720     IN          it_dto_cookie_t     cookie,
4721     IN          it_dto_flags_t      dto_flags,
4722     IN          it_rdma_addr_t      rdma_addr,
4723     IN          it_rmr_context_t    rmr_context
4724 );
```

4725 APPLICABILITY

4726 `it_post_rdma_read` is applicable only to Endpoints created for the RC service type.

4727 DESCRIPTION

4728	<code>ep_handle</code>	Handle for the Endpoint – the local side of the Connection.
4729	<code>local_segments</code>	Vector of <code>it_lmr_triplet_t</code> data structures that specifies the local buffer where data should be deposited. Can be NULL for a zero-sized message.
4730		
4731	<code>num_segments</code>	Number of <code>it_lmr_triplet_t</code> data structures in <code>local_segments</code> . Can be zero for a zero-sized message.
4732		
4733	<code>cookie</code>	Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the RDMA Read.
4734		
4735	<code>dto_flags</code>	Flags for posted RDMA Read.
4736	<code>rdma_addr</code>	The starting address of the section of the remote buffer from which to read.
4737	<code>rmr_context</code>	The RMR Context corresponding to the remote buffer from which reads will occur.
4738		

4739 `it_post_rdma_read` posts a request to the `ep_handle` Endpoint to transfer data from the section of
4740 the remote buffer (data source) specified by `rmr_context` and starting address `rdma_addr` into the
4741 local buffer (data sink) specified by `local_segments` and `num_segments`, via the reliable
4742 Connection of the `ep_handle` Endpoint. If `num_segments` is non-zero, then the size of the data
4743 transferred is given by the sum of the segment lengths specified by `local_segments`. A zero-sized
4744 message may be transferred. Like all other RDMA Read operations, the maximum number of
4745 `local_segments` is limited by the Endpoint attribute `max_rdma_read_segments`.

4746 For a remote buffer with Absolute Addressing, the starting address `rdma_addr` must be an
4747 absolute address within the remote linear address space, and for a remote buffer with Relative
4748 Addressing, `rdma_addr` must be a byte offset relative to the first byte of the remote buffer. See
4749 `it_addr_mode_t` for details on addressing modes.

4750 The Implementation ensures that an LMR Triplet supports byte alignment for Data Transfer
4751 Operations. If the buffer segments described by *local_segments* overlap, the resulting content of
4752 the local buffer is undefined.

4753 The remote buffer represented by *rmr_context* (data source) must have memory access privileges
4754 including remote read access.

4755 An LMR used in *local_segments* (data sink) must have memory access privileges including
4756 remote write access if the Interface Adapter attribute *rdma_read_requires_remote_write* equals
4757 IT_TRUE, or including local write access otherwise. See also Extended Description and
4758 Application Usage.

4759 The *cookie* (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work
4760 Request. This identifier is completely under Consumer control and opaque to the
4761 Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted
4762 Work Request.

4763 The *dto_flags* value is used as specified in *it_dto_flags_t*.

4764 A successful call returns IT_SUCCESS, which means that the RDMA Read operation was
4765 successfully posted to the transport layer.

4766 The completion of the posted RDMA Read is reported asynchronously to the Consumer
4767 according to the rules defined in *it_dto_flags_t*. An RDMA Read Completion Event is of type
4768 *it_dto_cmpl_event_t*. Any generated RDMA Read Completion Event manifests on the EVD
4769 associated with the Endpoint Send Queue. See *it_ep_rc_create*, *it_dto_status_t*, and
4770 *it_dto_events*. Once a successful Completion Event has been generated for the RDMA Read, the
4771 order of the bytes in the local buffer specified by *num_segments* and *local_segments* corresponds
4772 to the order defined by the section of the remote buffer unless there is local overlap. If there is
4773 local overlap, the content of the local buffer is undefined. Prior to the Completion Event being
4774 generated, the content of the local buffer is Implementation-dependent.

4775 A Consumer shall not modify the local buffer specified by *num_segments* and *local_segments*
4776 until the DTO is completed. When a Consumer does not adhere to this rule, the behavior of the
4777 Implementation and the underlying transport is not defined. A Consumer does get back the
4778 ownership of the array specified by the *local_segments* and *num_segments* arguments (but not
4779 the local buffer identified by this array) when *it_post_rdma_read* returns and is free to use this
4780 array for other calls, to modify it, or to destroy it.

4781 If the reported status of the completion DTO Event corresponding to the posted RDMA Read is
4782 not IT_DTO_SUCCESS, the content of the local buffer is not defined.

4783 The Implementation ensures that the RDMA Read in no way corresponds to any Send or Recv
4784 Data Transfer Operations over the same Connection.

4785 The Implementation ensures that subsequent RDMA Read DTOs posted to the same Endpoint
4786 start and complete in post order. However, the Implementation does not ensure that the RDMA
4787 Read DTOs place their data payloads into their local sink buffers in post order; if the local sink
4788 buffers overlap, their contents will be indeterminate.

4789 In general, Work Requests following an RDMA Read may start execution while the RDMA
4790 Read is in progress (but may not complete before the RDMA Read completes). To ensure
4791 deterministically that an RDMA Read DTO following an RDMA Read DTO starts after the first
4792 RDMA Read completes, specify the IT_BARRIER_FENCE_FLAG on the RDMA Read

4793 following the first RDMA Read. This technique can be used to ensure that the RDMA Read
4794 DTOs place their data payloads into their local sink buffers in post order. It can also be used to
4795 prevent exceeding the IRD of the remote Endpoint.

4796 Any Work Request posted to an Endpoint's Send Queue (thus also an RDMA Read) after a call
4797 to *it_rmr_link* or *it_rmr_unlink* will not begin execution until the Link or Unlink operation has
4798 completed.

4799 The Implementation ensures that all data for a given RDMA Read operation is transferred from
4800 the section of the remote buffer into the local buffer before an RDMA Read completion is
4801 generated with the status of IT_DTO_SUCCESS.

4802 **EXTENDED DESCRIPTION**

4803 For InfiniBand, the IA writing to a local buffer (data sink) in an RDMA Read DTO is considered
4804 a local write access, as indicated by the *it_ia_info_t* attribute *rdma_read_requires_remote_write*
4805 being IT_FALSE; consequently, the local buffer need only have the local write access privilege
4806 set and the Endpoint *ep_handle* need not have the *rdma_write_enable* attribute set to IT_TRUE.

4807 For iWARP, however, the IA writing to a local buffer in an RDMA Read DTO is considered a
4808 remote write access (by an incoming RDMA Read Response message), as indicated by the
4809 *it_ia_info_t* attribute *rdma_read_requires_remote_write* being IT_TRUE; hence, the local buffer
4810 must have the remote write access privilege set and the Endpoint *ep_handle* must have the
4811 *rdma_write_enable* attribute set to IT_TRUE.

4812 **RETURN VALUE**

4813 A successful call returns IT_SUCCESS.

4814 Posting to an Endpoint that is not in the IT_EP_STATE_CONNECTED or IT_EP_STATE_
4815 NONOPERATIONAL state will return the IT_ERR_INVALID_EP_STATE immediate error.

4816 The possible immediate errors for *it_post_rdma_read* are listed below:

4817 4818	IT_ERR_INVALID_DTO_FLAGS	The Data Transfer Operation flags (<i>dto_flags</i>) value was invalid.
4819	IT_ERR_INVALID_EP	The Endpoint Handle (<i>ep_handle</i>) was invalid.
4820 4821	IT_ERR_INVALID_EP_STATE	The Endpoint was not in the proper state for the attempted operation.
4822 4823	IT_ERR_INVALID_EP_TYPE	The attempted operation was invalid for the Service Type of the Endpoint.
4824 4825	IT_ERR_INVALID_NUM_SEGMENTS	The requested number of segments (<i>num_segments</i>) was larger than the Endpoint supports.
4826 4827	IT_ERR_TOO_MANY_POSTS	The operation failed due to an overflow of a Work Queue.
4828 4829 4830 4831	IT_ERR_IA_CATASTROPHE	The IA has experienced a catastrophic error and is in the disabled state. None of the output parameters from this routine are valid. See <i>it_ia_info_t</i> for a description of the disabled state

4832 ASYNCHRONOUS ERRORS

4833 For Work Requests posted to an Endpoint in the `IT_EP_STATE_CONNECTED` state, a
4834 completion status (see [it_dto_status_t](#)) other than `IT_DTO_SUCCESS` will break the Connection
4835 by moving the Endpoint to the `IT_EP_STATE_NONOPERATIONAL` state and deliver an
4836 `IT_CM_MSG_CONN_BROKEN_EVENT` Event to the Connect EVD of *ep_handle*. Once the
4837 Connection is broken, all outstanding and in-progress operations on the Connection will
4838 complete with an error status.

4839 Any posting to an Endpoint that is in the `IT_EP_STATE_NONOPERATIONAL` state will be
4840 flushed with completion status set to `IT_DTO_ERR_FLUSHED`.

4841 For locally or remotely detected errors that can be reported as Completion Errors, see also
4842 [it_dto_status_t](#).

4843 The handling of remotely detected errors is transport- and Implementation-dependent; end-to-
4844 end completions are not supported for certain transports or DTOs.

4845 An RDMA Read operation may result in a remotely detected access violation (IRRQ access
4846 violation for iWARP) at the data source.

4847 The remote Implementation verifies the accessibility of the remote buffer represented by
4848 *rmr_context* based on the buffer's access rights, the buffer's association with the remote
4849 Endpoint that processes the operation, and the incoming RDMA operations allowed on the
4850 remote Endpoint. It declares an access violation if either the remote buffer has insufficient
4851 access rights, if *rmr_context* represents an LMR or Wide RMR whose Protection Zone does not
4852 match the Protection Zone of the remote Endpoint, if *rmr_context* represents a Narrow RMR that
4853 is associated with a different remote Endpoint, or if the targeted remote Endpoint does not have
4854 incoming RDMA Read operations enabled. An access violation at the data source is also
4855 declared if the RDMA Read operation exceeds the bounds of the remote buffer.

4856 An access violation at the data source will surface remotely as an `IT_ASYNC_AFF_`
4857 `EP_L_ACCESS_VIOLATION` Affiliated Asynchronous Error on the IB transport, and as an
4858 `IT_ASYNC_AFF_EP_L_IRRQ_ACCESS_VIOLATION` Affiliated Asynchronous Error on the
4859 iWARP Transport.

4860 An access violation at the data source will surface locally as follows:

4861 For the IB transport, an `IT_DTO_ERR_REMOTE_ACCESS` Completion Error will occur.

4862 For the iWARP Transport, either an `IT_ASYNC_AFF_EP_R_ACCESS_VIOLATION`
4863 Affiliated Asynchronous Error will occur after the DTO has already completed with
4864 `IT_DTO_SUCCESS`, or an `IT_DTO_ERR_REMOTE_ACCESS` Completion Error will occur.

4865 An RDMA Read operation may result in a locally detected access violation at the data sink.

4866 The local Implementation verifies the accessibility of the local buffer represented by
4867 *local_segments* and *num_segments* based on each segment's access rights and association with
4868 the local Endpoint, and the incoming RDMA operations allowed on the local Endpoint. It
4869 declares an access violation if an LMR in a local segment has insufficient access rights or a
4870 Protection Zone that does not match the Protection Zone of the local Endpoint, or if the
4871 *rdma_read_requires_remote_write* IA attribute equals `IT_TRUE` and the local Endpoint has
4872 incoming RDMA Write operations disabled. An access violation is also declared if the RDMA
4873 Read operation exceeds the bounds of an LMR in a local segment.

4874 An access violation at the data sink will surface locally as follows:
4875 For the IB transport, an IT_DTO_ERR_LOCAL_PROTECTION Completion Error will occur.
4876 For the iWARP Transport, either an IT_DTO_ERR_LOCAL_PROTECTION Completion Error
4877 will occur or an IT_ASYNC_AFF_EP_L_ACCESS_VIOLATION Affiliated Asynchronous
4878 Error will surface after the DTO has already completed with IT_DTO_SUCCESS.
4879 Despite using the RC service, an RDMA Read DTO may fail to successfully deliver the contents
4880 of the section of the remote buffer into the local buffer. This may result in a broken Connection
4881 or lead to data corruption in the local buffer, which is detected locally with high probability. In
4882 case of local data corruption, either an IT_ASYNC_AFF_EP_L_LLDP_ERROR Affiliated
4883 Asynchronous Error will surface after the DTO has already completed with
4884 IT_DTO_SUCCESS, or an IT_DTO_ERR_TRANSPORT Completion Error will occur.
4885 If a type of remotely detected error can manifest locally in different ways – i.e., as an Affiliated
4886 Asynchronous Error or as a Completion Error – transport-independent programming requires
4887 Consumers to be prepared to deal with either.

4888 **APPLICATION USAGE**

4889 This function is used after a Connection has been established to transfer data from a Consumer-
4890 specified section of a remote buffer to a Consumer-specified local buffer.
4891 For writing transport-independent code, the Consumer should specify the remote and local write
4892 access privileges of an RDMA Read DTO's data sink and the *rdma_write_enable* attribute of the
4893 local Endpoint according to the IA attribute *rdma_read_requires_remote_write* obtained via
4894 *it_ia_query*. The Consumer may also enable both remote write and local write access for the data
4895 sink, regardless of the *rdma_read_requires_remote_write* attribute.
4896 The Consumer should use unique identifiers for *cookie* if they desire to identify each DTO. If the
4897 Consumer does not require a unique DTO identifier, the value of zero or NULL can be used.
4898 For best RDMA Read operation performance, the Consumer should align each buffer segment of
4899 *local_segments* to the *dto_alignment_hint* in the IA attributes obtained via *it_ia_query*.

4900 **SEE ALSO**

4901 *it_post_send()*, *it_post_sendto()*, *it_post_rcv()*, *it_post_rcvfrom()*, *it_post_rdma_write()*,
4902 *it_ep_rc_create()*, *it_addr_mode_t*, *it_ia_query()*, *it_dto_cookie_t*, *it_dto_events*, *it_dto_flags_t*,
4903 *it_dto_status_t*, *it_ia_info_t*, *it_lmr_triplet_t*

it_post_rdma_read_to_rmr()

4904

4905 NAME

4906 `it_post_rdma_read_to_rmr` – post an RDMA Read DTO to an Endpoint

4907 SYNOPSIS

```
4908 #include <it_api.h>
4909
4910 it_status_t it_post_rdma_read_to_rmr (
4911     IN          it_ep_handle_t      ep_handle,
4912     IN  const   it_rmr_triplet_t    *local_segments,
4913     IN          size_t              num_segments,
4914     IN          it_dto_cookie_t     cookie,
4915     IN          it_dto_flags_t      dto_flags,
4916     IN          it_rdma_addr_t      rdma_addr,
4917     IN          it_rmr_context_t    rmr_context
4918 );
```

4919 APPLICABILITY

4920 `it_post_rdma_read_to_rmr` is applicable only to Endpoints created for the RC service type and is
4921 supported only if the Interface Adapter attribute `rdma_read_local_extensions` (see [it_ia_info_t](#))
4922 is `IT_TRUE`.

4923 DESCRIPTION

4924	<code>ep_handle</code>	Handle for the Endpoint – the local side of the Connection.
4925	<code>local_segments</code>	Vector of <code>it_rmr_triplet_t</code> data structures that specifies the local buffer where data should be deposited. Can be NULL for a zero-sized message.
4926		
4927	<code>num_segments</code>	Number of <code>it_rmr_triplet_t</code> data structures in <code>local_segments</code> . Can be zero for a zero-sized message.
4928		
4929	<code>cookie</code>	Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the RDMA Read.
4930		
4931	<code>dto_flags</code>	Flags for posted RDMA Read.
4932	<code>rdma_addr</code>	The starting address of the section of the remote buffer from which to read.
4933	<code>rmr_context</code>	The RMR Context corresponding to the remote buffer from which reads will occur.
4934		

4935 `it_post_rdma_read_to_rmr` posts a request to the `ep_handle` Endpoint to transfer data from the
4936 section of the remote buffer (data source) specified by `rmr_context` and `rdma_addr` into the local
4937 buffer (data sink) specified by `local_segments` and `num_segments`, via the reliable Connection of
4938 the `ep_handle` Endpoint. If `num_segments` is non-zero, then the size of the data transferred is
4939 given by the sum of the segment lengths specified by `local_segments`. A zero-sized message may
4940 be transferred. Like all other RDMA Read operations, the maximum number of `local_segments`
4941 is limited by the Endpoint attribute `max_rdma_read_segments`.

4942 For a remote buffer with Absolute Addressing, the starting address `rdma_addr` must be an
4943 absolute address within the remote linear address space, and for a remote buffer with Relative

4944 Addressing, *rdma_addr* must be a byte offset relative to the first byte of the remote buffer. See
4945 [it_addr_mode_t](#) for details on addressing modes.

4946 The Implementation ensures that an RMR Triplet supports byte alignment for Data Transfer
4947 Operations. If the buffer segments described by *local_segments* overlap, the resulting content of
4948 the local buffer is undefined.

4949 The remote buffer represented by *rmr_context* (data source) must have memory access privileges
4950 including remote read access.

4951 An RMR used in *local_segments* (data sink) must have memory access privileges including
4952 remote write access.

4953 The *cookie* ([it_dto_cookie_t](#)) allows the Consumer to associate an identifier with each Work
4954 Request. This identifier is completely under Consumer control and opaque to the
4955 Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted
4956 Work Request.

4957 The *dto_flags* value is used as specified in [it_dto_flags_t](#).

4958 A successful call returns IT_SUCCESS, which means that the RDMA Read operation was
4959 successfully posted to the transport layer.

4960 The completion of the posted RDMA Read is reported asynchronously to the Consumer
4961 according to the rules defined in [it_dto_flags_t](#). An RDMA Read Completion Event is of type
4962 [it_dto_cmpl_event_t](#). Any generated RDMA Read Completion Event manifests on the EVD
4963 associated with the Endpoint Send Queue. See [it_ep_rc_create](#), [it_dto_status_t](#), and
4964 [it_dto_events](#). Once a successful Completion Event has been generated for the RDMA Read, the
4965 order of the bytes in the local buffer specified by *num_segments* and *local_segments* corresponds
4966 to the order defined by the section of the remote buffer unless there is local overlap. If there is
4967 local overlap, the content of the local buffer is undefined. Prior to the Completion Event being
4968 generated, the content of the local buffer is Implementation-dependent.

4969 A Consumer shall not modify the local buffer specified by *num_segments* and *local_segments*
4970 until the DTO is completed. When a Consumer does not adhere to this rule, the behavior of the
4971 Implementation and the underlying transport is not defined. A Consumer does get back the
4972 ownership of the array specified by the *local_segments* and *num_segments* arguments (but not
4973 the local buffer identified by this array) when *it_post_rdma_read_to_rmr* returns and is free to
4974 use this array for other calls, to modify it, or to destroy it.

4975 If the reported status of the completion DTO Event corresponding to the posted RDMA Read is
4976 not IT_DTO_SUCCESS, the content of the local buffer is not defined.

4977 The Implementation ensures that the RDMA Read in no way corresponds to any Send or Recv
4978 Data Transfer Operations over the same Connection.

4979 The Implementation ensures that subsequent RDMA Read DTOs posted to the same Endpoint
4980 start and complete in post order. However, the Implementation does not ensure that the RDMA
4981 Read DTOs place their data payloads into their local sink buffers in post order; if the local sink
4982 buffers overlap, their contents will be indeterminate.

4983 In general, Work Requests following an RDMA Read may start execution while the RDMA
4984 Read is in progress (but may not complete before the RDMA Read completes). To ensure
4985 deterministically that an RDMA Read DTO following an RDMA Read DTO starts after the first
4986 RDMA Read completes, specify the IT_BARRIER_FENCE_FLAG on the RDMA Read

4987 following the first RDMA Read. This technique can be used to ensure that the RDMA Read
4988 DTOs place their data payloads into their local sink buffers in post order. It can also be used to
4989 prevent exceeding the IRD of the remote Endpoint.

4990 Any Work Request posted to an Endpoint's Send Queue (thus also an RDMA Read) after a call
4991 to *it_rmr_link* or *it_rmr_unlink* will not begin execution until the Link or Unlink operation has
4992 completed.

4993 The Implementation ensures that all data for a given RDMA Read operation is transferred from
4994 the section of the remote buffer into the local buffer before an RDMA Read completion is
4995 generated with the status of IT_DTO_SUCCESS.

4996 **EXTENDED DESCRIPTION**

4997 This call is supported only by iWARP. It allows using local RMRs as data sinks.

4998 For the iWARP Transport, the IA writing to a local buffer (data sink) in an RDMA Read DTO is
4999 considered a remote write access (by an incoming RDMA Read Response message), as indicated
5000 by the *it_ia_info_t* attribute *rdma_read_requires_remote_write* being IT_TRUE; hence, the local
5001 buffer must have the remote write access privilege set and the Endpoint *ep_handle* must have the
5002 *rdma_write_enable* attribute set to IT_TRUE.

5003 **RETURN VALUE**

5004 A successful call returns IT_SUCCESS.

5005 Posting to an Endpoint that is not in the IT_EP_STATE_CONNECTED or IT_EP_STATE_
5006 NONOPERATIONAL state will return the IT_ERR_INVALID_EP_STATE immediate error.

5007 The possible immediate errors for *it_post_rdma_read_to_rmr* are listed below:

5008 IT_ERR_INVALID_DTO_FLAGS The Data Transfer Operation flags (*dto_flags*) value was
5009 invalid.

5010 IT_ERR_INVALID_EP The Endpoint Handle (*ep_handle*) was invalid.

5011 IT_ERR_INVALID_EP_STATE The Endpoint was not in the proper state for the
5012 attempted operation.

5013 IT_ERR_INVALID_EP_TYPE The attempted operation was invalid for the Service
5014 Type of the Endpoint.

5015 IT_ERR_INVALID_NUM_SEGMENTS The requested number of segments (*num_segments*) was
5016 larger than the Endpoint supports.

5017 IT_ERR_OP_NOT_SUPPORTED The operation failed as it is not supported on the IA.

5018 IT_ERR_TOO_MANY_POSTS The operation failed due to an overflow of a work queue.

5019 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the
5020 disabled state. None of the output parameters from this
5021 routine are valid. See *it_ia_info_t* for a description of the
5022 disabled state

5023 **ASYNCHRONOUS ERRORS**

5024 For Work Requests posted to an Endpoint in the IT_EP_STATE_CONNECTED state, a
5025 completion status (see *it_dto_status_t*) other than IT_DTO_SUCCESS will break the Connection

5026 by moving the Endpoint to the `IT_EP_STATE_NONOPERATIONAL` state and deliver an
5027 `IT_CM_MSG_CONN_BROKEN_EVENT` Event to the Connect EVD of *ep_handle*. Once the
5028 Connection is broken, all outstanding and in-progress operations on the Connection will
5029 complete with an error status.

5030 Any posting to an Endpoint that is in the `IT_EP_STATE_NONOPERATIONAL` state will be
5031 flushed with completion status set to `IT_DTO_ERR_FLUSHED`.

5032 For locally or remotely detected errors that can be reported as Completion Errors, see also
5033 [*it_dto_status_t*](#).

5034 The handling of remotely detected errors is transport- and Implementation-dependent; end-to-
5035 end completions are not supported for certain transports or DTOs.

5036 An RDMA Read operation may result in a remotely detected access violation (IRRQ access
5037 violation for iWARP) at the data source.

5038 The remote Implementation verifies the accessibility of the remote buffer represented by
5039 *rnr_context* based on the buffer's access rights, the buffer's association with the remote
5040 Endpoint that processes the operation, and the incoming RDMA operations allowed on the
5041 remote Endpoint. It declares an access violation if either the remote buffer has insufficient
5042 access rights, if *rnr_context* represents an LMR or Wide RMR whose Protection Zone does not
5043 match the Protection Zone of the remote Endpoint, if *rnr_context* represents a Narrow RMR that
5044 is associated with a different remote Endpoint, or if the targeted remote Endpoint does not have
5045 incoming RDMA Read operations enabled. An access violation at the data source is also
5046 declared if the RDMA Read operation exceeds the bounds of the remote buffer.

5047 An access violation at the data source will surface remotely as an `IT_ASYNC_AFF_EP_L_`
5048 `ACCESS_VIOLATION` Affiliated Asynchronous Error on the IB transport, and as an
5049 `IT_ASYNC_AFF_EP_L_IRRQ_ACCESS_VIOLATION` Affiliated Asynchronous Error on the
5050 iWARP Transport.

5051 An access violation at the data source will surface locally as follows:

5052 For the IB transport, an `IT_DTO_ERR_REMOTE_ACCESS` Completion Error will occur.

5053 For the iWARP Transport, either an `IT_ASYNC_AFF_EP_R_ACCESS_VIOLATION`
5054 Affiliated Asynchronous Error will surface after the DTO has already completed with
5055 `IT_DTO_SUCCESS`, or an `IT_DTO_ERR_REMOTE_ACCESS` Completion Error will occur.

5056 An RDMA Read operation may result in a locally detected access violation at the data sink.

5057 The local Implementation verifies the accessibility of the local buffer represented by
5058 *local_segments* and *num_segments* based on each segment's access rights and association with
5059 the local Endpoint and the incoming RDMA operations allowed on the local Endpoint. It
5060 declares an access violation if an RMR in a local segment has insufficient access rights or a
5061 Protection Zone that does not match the Protection Zone of the local Endpoint, if an RMR in a
5062 local segment is a Narrow RMR that is associated with a different local Endpoint, or if the
5063 *rdma_read_requires_remote_write* IA attribute equals `IT_TRUE` and the local Endpoint has
5064 incoming RDMA Write operations disabled. An access violation is also declared if the RDMA
5065 Read operation exceeds the bounds of an RMR in a local segment.

5066 An access violation at the data sink will surface locally as follows:

5067 For the IB transport, an `IT_DTO_ERR_LOCAL_PROTECTION` Completion Error will occur.

5068 For the iWARP Transport, either an `IT_DTO_ERR_LOCAL_PROTECTION` Completion Error
5069 will occur or an `IT_ASYNC_AFF_EP_L_ACCESS_VIOLATION` Affiliated Asynchronous
5070 Error will surface after the DTO has already completed with `IT_DTO_SUCCESS`.

5071 Despite using the RC service, an RDMA Read DTO may fail to successfully deliver the contents
5072 of the section of the remote buffer into the local buffer. This may result in a broken Connection
5073 or lead to data corruption in the local buffer, which is detected locally with high probability. In
5074 case of local data corruption, either an `IT_ASYNC_AFF_EP_L_LLQ_ERROR` Affiliated
5075 Asynchronous Error will surface after the DTO has already completed with
5076 `IT_DTO_SUCCESS`, or an `IT_DTO_ERR_TRANSPORT` Completion Error will occur.

5077 If a type of remotely detected errors can manifest locally in different ways – i.e., as an Affiliated
5078 Asynchronous Error or as a Completion Error – transport-independent programming requires
5079 Consumers to be prepared to deal with either.

5080 APPLICATION USAGE

5081 This function is used after a Connection has been established to transfer data from a Consumer-
5082 specified section of a remote buffer to a Consumer-specified local buffer.

5083 For writing transport-independent code, the Consumer should specify the remote write access
5084 privilege of an RDMA Read DTO's data sink and the `rdma_write_enable` attribute of the local
5085 Endpoint according to `rdma_read_requires_remote_write` in the IA attributes obtained via
5086 [it_ia_query](#). The Consumer may also enable both remote write and local write access for the data
5087 sink, regardless of the `rdma_read_requires_remote_write` attribute.

5088 The Consumer should use unique identifiers for `cookie` if they desire to identify each DTO. If the
5089 Consumer does not require a unique DTO identifier, the value of zero or NULL can be used.

5090 For best RDMA Read operation performance, the Consumer should align each buffer segment of
5091 `local_segments` to the `dto_alignment_hint` in the IA attributes obtained via [it_ia_query](#).

5092 SEE ALSO

5093 [it_post_send\(\)](#), [it_post_sendto\(\)](#), [it_post_recv\(\)](#), [it_post_recvfrom\(\)](#), [it_post_rdma_read\(\)](#),
5094 [it_post_rdma_write\(\)](#), [it_rmr_link\(\)](#), [it_rmr_unlink\(\)](#), [it_ep_rc_create\(\)](#), [it_ia_query\(\)](#),
5095 [it_addr_mode_t](#), [it_dto_cookie_t](#), [it_dto_events](#), [it_dto_flags_t](#), [it_dto_status_t](#), [it_ia_info_t](#),
5096 [it_rmr_triplet_t](#)

5097

5098

it_post_rdma_write()

5099 **NAME**

5100 `it_post_rdma_write` – post an RDMA Write DTO to an Endpoint

5101 **SYNOPSIS**

```
5102 #include <it_api.h>
5103
5104 it_status_t it_post_rdma_write (
5105     IN          it_ep_handle_t      ep_handle,
5106     IN  const   it_lmr_triplet_t    *local_segments,
5107     IN          size_t              num_segments,
5108     IN          it_dto_cookie_t     cookie,
5109     IN          it_dto_flags_t      dto_flags,
5110     IN          it_rdma_addr_t      rdma_addr,
5111     IN          it_rmr_context_t     rmr_context
5112 );
```

5113 **APPLICABILITY**

5114 `it_post_rdma_write` is applicable only to Endpoints created for the RC service type.

5115 **DESCRIPTION**

5116	<code>ep_handle</code>	Handle for the Endpoint – the local side of the Connection.
5117	<code>local_segments</code>	Vector of <code>it_lmr_triplet_t</code> data structures that specifies the local buffer that contains data to be transferred. Can be NULL for a zero-sized message.
5118		
5119	<code>num_segments</code>	Number of <code>it_lmr_triplet_t</code> data structures in <code>local_segments</code> . Can be zero for a zero-sized message.
5120		
5121	<code>cookie</code>	Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the RDMA Write.
5122		
5123	<code>dto_flags</code>	Flags for posted RDMA Write.
5124	<code>rdma_addr</code>	The starting address of the section of the remote buffer to which to write.
5125	<code>rmr_context</code>	The RMR Context corresponding to the remote buffer to which writes will occur.
5126		

5127 `it_post_rdma_write` posts a request to the `ep_handle` Endpoint to transfer all the data from the local buffer (data source) specified by `local_segments` and `num_segments` into the section of the remote buffer (data sink) specified by `rmr_context` and `rdma_addr`, via the reliable Connection of the `ep_handle` Endpoint. If `num_segments` is non-zero, then the size of the data transferred is given by the sum of the segment lengths specified by `local_segments`. A zero-sized message may be transferred.

5133 For a remote buffer with Absolute Addressing, the starting address `rdma_addr` must be an absolute address within the remote linear address space, and for a remote buffer with Relative Addressing, `rdma_addr` must be a byte offset relative to the first byte of the remote buffer. See [it_addr_mode_t](#) for details on addressing modes.

5137 The Implementation ensures that an LMR Triplet supports byte alignment for Data Transfer
5138 Operations. The buffer segments described by *local_segments* can overlap; it is safe to use
5139 overlapping buffer segments as a data source.

5140 An LMR used in *local_segments* (data source) must have memory access privileges including
5141 local read access.

5142 The remote buffer represented by *rmr_context* (data sink) must have memory access privileges
5143 including remote write access.

5144 The *cookie* (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work
5145 Request. This identifier is completely under Consumer control and opaque to the
5146 Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted
5147 Work Request.

5148 The *dto_flags* value is used as specified in *it_dto_flags_t*.

5149 A successful call returns IT_SUCCESS, which means that the RDMA Write operation was
5150 successfully posted to the transport layer.

5151 The completion of the posted RDMA Write is reported asynchronously to the Consumer
5152 according to the rules defined in *it_dto_flags_t*. An RDMA Write Completion Event is of type
5153 *it_dto_cmpl_event_t*. Any generated RDMA Write Completion Event manifests on the EVD
5154 associated with the Endpoint Send Queue. See *it_ep_rc_create*, *it_dto_status_t*, and
5155 *it_dto_events*. The Completion Event for the *it_post_rdma_write* call indicates to the Consumer
5156 that the local buffer is under Consumer control again and that the contents of the local buffer
5157 will be transported reliably, but does not guarantee that these contents have been successfully
5158 delivered into the memory of the remote Consumer; wherever necessary, this restriction is made
5159 explicit by designating a Completion as local or by stating that operations complete locally. See
5160 Section 5.4 of [DDP-IETF] for background information. However, once the contents of the local
5161 buffer reach the remote Consumer memory, the order of the bytes in the remote memory
5162 corresponds to the order defined by the *local_segments*.

5163 A Consumer should not modify the local buffer specified by *num_segments* and *local_segments*
5164 until the DTO is locally completed. When a Consumer does not adhere to this rule, the behavior
5165 of the Implementation and the underlying transport is not defined. A Consumer does get back the
5166 ownership of the array specified by the *local_segments* and *num_segments* arguments (but not
5167 the local buffer identified by this array) when *it_post_rdma_write* returns and is free to use this
5168 array for other calls, to modify it, or to destroy it.

5169 The Implementation ensures that the RDMA Write in no way corresponds to any Receive Data
5170 Transfer Operations over the same Connection.

5171 The Implementation ensures that subsequent RDMA Write DTOs posted to the same Endpoint
5172 start and complete locally in post order. However, the Implementation does not ensure that the
5173 RDMA Write DTOs place their data payloads into their remote buffers in post order; if the
5174 targeted sections of the remote buffers overlap, their contents will be indeterminate.

5175 The Implementation ensures that each RDMA Write DTO posted to an Endpoint prior to a Send
5176 DTO posted to the same Endpoint has its complete data payload delivered to the remote memory
5177 prior to the completion of the Receive DTO at the remote side matching that Send. However, the
5178 Implementation does not ensure that the RDMA Write DTO places its entire data payload into
5179 its remote buffer before the Receive DTO places the data payload of the incoming Send into its

5180 receive buffer; if the targeted sections of the remote buffers overlap, their contents will be
5181 indeterminate.

5182 In general, Work Requests following an RDMA Read may start execution while the RDMA
5183 Read is in progress (but may not complete before the RDMA Read completes). To ensure
5184 deterministically that an RDMA Write DTO following an RDMA Read DTO starts after the
5185 RDMA Read completes, specify the `IT_BARRIER_FENCE_FLAG` on the RDMA Write DTO.

5186 Any Work Request posted to an Endpoint's Send Queue (thus also an RDMA Write) after a call
5187 to *it_rmr_link* or *it_rmr_unlink* will not begin execution until the Link or Unlink operation has
5188 completed.

5189 **EXTENDED DESCRIPTION**

5190 See *it_lmr_sync_rdma_write* for a discussion of ramifications of RDMA Write use on non-
5191 coherent systems.

5192 On the InfiniBand Transport, the Implementation ensures that subsequent RDMA Write DTOs
5193 or an RDMA Write DTO followed by a Send DTO posted to the same Endpoint cause data
5194 payloads to be placed in remote memory in post order; relying on such placement ordering
5195 represents a transport-dependent programming practice.

5196 **RETURN VALUE**

5197 A successful call returns `IT_SUCCESS`.

5198 Posting to an Endpoint that is not in the `IT_EP_STATE_CONNECTED` or `IT_EP_STATE_`
5199 `NONOPERATIONAL` state will return the `IT_ERR_INVALID_EP_STATE` immediate error.

5200 The possible immediate errors for *it_post_rdma_write* are listed below:

5201	<code>IT_ERR_INVALID.DTO_FLAGS</code>	The Data Transfer Operation flags (<i>dto_flags</i>) value was 5202 invalid.
5203	<code>IT_ERR_INVALID_EP</code>	The Endpoint Handle (<i>ep_handle</i>) was invalid.
5204	<code>IT_ERR_INVALID_EP_STATE</code>	The Endpoint was not in the proper state for the attempted 5205 operation.
5206	<code>IT_ERR_INVALID_EP_TYPE</code>	The attempted operation was invalid for the Service Type 5207 of the Endpoint.
5208	<code>IT_ERR_INVALID_NUM_SEGMENTS</code>	The requested number of segments (<i>num_segments</i>) was 5209 larger than the Endpoint supports.
5210	<code>IT_ERR_TOO_MANY_POSTS</code>	The operation failed due to an overflow of a Work Queue.
5211	<code>IT_ERR_IA_CATASTROPHE</code>	The IA has experienced a catastrophic error and is in the 5212 disabled state. None of the output parameters from this 5213 routine are valid. See <i>it_ia_info_t</i> for a description of the 5214 disabled state.

5215 **ASYNCHRONOUS ERRORS**

5216 For Work Requests posted to an Endpoint in the `IT_EP_STATE_CONNECTED` state, a
5217 completion status (see *it_dto_status_t*) other than `IT.DTO_SUCCESS` will break the Connection
5218 by moving the Endpoint to the `IT_EP_STATE_NONOPERATIONAL` state and deliver an
5219 `IT_CM_MSG_CONN_BROKEN_EVENT` Event to the Connect EVD of *ep_handle*. Once the

5220 Connection is broken, all outstanding and in-progress operations on the Connection will
5221 complete with an error status.

5222 Any posting to an Endpoint that is in the `IT_EP_STATE_NONOPERATIONAL` state will be
5223 flushed with completion status set to `IT_DTO_ERR_FLUSHED`.

5224 For locally or remotely detected errors that can be reported as Completion Errors, see also
5225 [*it_dto_status_t*](#).

5226 The handling of remotely detected errors is transport- and Implementation-dependent; end-to-
5227 end completions are not supported for certain transports or DTOs.

5228 An RDMA Write operation may result in a remotely detected access violation, also referred to as
5229 a protection error by some transports.

5230 The remote Implementation verifies the accessibility of the remote buffer represented by
5231 *rmr_context* based on the buffer's access rights, the buffer's association with the remote
5232 Endpoint that processes the operation, and the RDMA operations allowed on the remote
5233 Endpoint. It declares an access violation if either the remote buffer has insufficient access rights,
5234 if *rmr_context* represents an LMR or Wide RMR whose Protection Zone does not match the
5235 Protection Zone of the remote Endpoint, if *rmr_context* represents a Narrow RMR that is
5236 associated with a different remote Endpoint, or if the targeted remote Endpoint has incoming
5237 RDMA Write operations disabled. An access violation is also declared if the RDMA Write
5238 operation exceeds the bounds of the remote buffer.

5239 An access violation due to an RDMA Write operation will surface remotely as an
5240 `IT_ASYNC_AFF_EP_L_ACCESS_VIOLATION` Affiliated Asynchronous Error on both the IB
5241 and iWARP Transports.

5242 An access violation due to an RDMA Write operation will surface locally as follows:

5243 For the IB transport, an `IT_DTO_ERR_REMOTE_ACCESS` Completion Error will occur.

5244 For the iWARP Transport, either an `IT_ASYNC_AFF_EP_R_ACCESS_VIOLATION`
5245 Affiliated Asynchronous Error will surface after the DTO has already completed with
5246 `IT_DTO_SUCCESS` or, if the Implementation is able to convert a Terminate message to a
5247 Completion Error, an `IT_DTO_ERR_REMOTE_ACCESS` Completion Error will occur.

5248 Despite using the RC service, an RDMA Write operation may fail to successfully deliver the
5249 contents of the local buffer into the remote buffer. This may result in data loss or data corruption
5250 in the remote buffer, which is detected by the remote Implementation with high probability.

5251 For the IB transport, a data loss or data corruption in the remote buffer causes the RDMA Write
5252 DTO to complete with an `IT_DTO_ERR_TRANSPORT` Completion Error.

5253 For the iWARP Transport, a data loss or data corruption in the remote buffer will surface
5254 remotely as an `IT_ASYNC_AFF_EP_L_LLQ_ERROR` Affiliated Asynchronous Error, and
5255 locally as an `IT_ASYNC_AFF_EP_R_ERROR` Affiliated Asynchronous Error after the RDMA
5256 Write DTO has already completed with `IT_DTO_SUCCESS`.

5257 If a type of remotely detected error can manifest locally in different ways – i.e., as an Affiliated
5258 Asynchronous Error or as a Completion Error – transport-independent programming requires
5259 Consumers to be prepared to deal with either.

5260 **APPLICATION USAGE**

5261 This function is used after a Connection has been established to transfer data from a Consumer-
5262 specified local buffer to a section of a remote buffer on the other side of the Connection.

5263 The Consumer should use unique identifiers for *cookie* if they desire to identify each DTO. If the
5264 Consumer does not require a unique DTO identifier, the value of zero or NULL can be used.

5265 For best RDMA Write operation performance, the Consumer should align each buffer segment
5266 of *local_segments* to the *dto_alignment_hint* in the IA attributes obtained via *it_ia_query*.

5267 There are a variety of ways to guarantee the delivery of a local buffer via RDMA Write into the
5268 memory of the remote Consumer. One way would be for the Consumer to send a message over
5269 the Connection after the RDMA Write had completed, and then wait for the remote Consumer to
5270 reply to that message. When the remote Consumer reaps the Receive Completion for the Send
5271 from the local Consumer, the payload of the RDMA Write must have been delivered into the
5272 remote memory.

5273 **SEE ALSO**

5274 *it_post_send()*, *it_post_sendto()*, *it_post_rcv()*, *it_post_rcvfrom()*, *it_post_rdma_read()*,
5275 *it_rmr_link()*, *it_rmr_unlink()*, *it_ep_rc_create()*, *it_ia_query()*, *it_addr_mode_t*,
5276 *it_dto_cookie_t*, *it_dto_events*, *it_dto_status_t*, *it_dto_flags_t*, *it_ia_info_t*, *it_lmr_triplet_t*

it_post_recv()

5277

5278 NAME

5279

it_post_recv – post a Receive DTO to an Endpoint or Shared Receive Queue

5280 SYNOPSIS

5281

```
#include <it_api.h>
```

5282

```
it_status_t it_post_recv(  
    IN          it_handle_t          handle,  
    IN const    it_lmr_triplet_t     *local_segments,  
    IN          size_t               num_segments,  
    IN          it_dto_cookie_t      cookie,  
    IN          it_dto_flags_t       dto_flags  
);
```

5283

5284

5285

5286

5287

5288

5289

5290 APPLICABILITY

5291

it_post_recv is applicable only to Endpoints created for the RC service type or to Shared Receive Queues.

5292

5293 DESCRIPTION

5294

handle Handle for the Endpoint or S-RQ.

5295

local_segments Vector of *it_lmr_triplet_t* data structures that specifies the local buffer to contain the data to be received. Can be NULL for a zero-sized message.

5296

5297

num_segments Number of *it_lmr_triplet_t* data structures in *local_segments*. Can be zero for a zero-sized message.

5298

5299

cookie Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the Receive.

5300

5301

dto_flags Flags for posted Receive.

5302

it_post_recv posts a request to the IT Object referenced by *handle*, which may refer to an Endpoint or Shared Receive Queue, to deposit all incoming data corresponding to a single Send DTO into the local Receive buffer (data sink) specified by *local_segments* and *num_segments*. If an Endpoint has an associated S-RQ and the Consumer passes the Endpoint handle rather than the S-RQ handle as the *handle* argument to *it_post_recv*, an IT_ERR_INVALID_SRQ error shall be returned. If *num_segments* is non-zero, then the size of the Receive buffer is given by the sum of the segment lengths specified by *local_segments*. The Receive buffer may have a size of zero.

5303

5304

5305

5306

5307

5308

5309

5310

An LMR used in *local_segments* (data sink) must have memory access privileges including local write access.

5311

5312

The Implementation ensures that an LMR Triplet supports byte alignment for Data Transfer Operations. The Implementation allows the buffer segments described by the *local_segments* vector to overlap but the resulting Receive behavior is undefined.

5313

5314

5315

The *cookie* (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work Request. This identifier is completely under Consumer control and opaque to the

5316

5317 Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted
5318 Work Request.

5319 The *dto_flags* value is used as specified in *it_dto_flags_t*.

5320 A successful call returns IT_SUCCESS, which means that the Receive operation was
5321 successfully posted to the transport layer.

5322 If an Endpoint has an associated S-RQ and an incoming Send arrives for that Endpoint, then the
5323 Implementation behaves as if a Receive Work Request is dequeued from the S-RQ and enqueued
5324 onto the Endpoint's local RQ. Receive Work Requests are not necessarily dequeued from the S-
5325 RQ in the order in which Send messages arrive. Depending on Implementation, an out-of-order
5326 arrival of a Send message may cause more than one Receive WR to be dequeued from the S-RQ.
5327 An Interface Adapter may support the Endpoint Soft High Watermark and/or Endpoint Hard
5328 High Watermark mechanisms (see *it_ep_attributes_t* and *it_ia_info_t*), allowing Consumers to
5329 bound the effects of out-of-order arrivals.

5330 The completion of the posted Receive is reported asynchronously to the Consumer according to
5331 the rules defined in *it_dto_flags_t*. A Receive Completion Event is of type *it_dto_cmpl_event_t*.
5332 Exactly one Receive Completion Event is always generated and manifests on the EVD
5333 associated with the Endpoint to which the corresponding incoming Send operation was directed
5334 (see *it_ep_rc_create*).

5335 If the reported *dto_status* of the Receive Completion Event (see *it_dto_cmpl_event_t*) is
5336 IT_DTO_SUCCESS (see *it_dto_status_t*), then all data from an incoming Send message has
5337 been transferred into the Receive buffer and the size of the received data can be retrieved
5338 through the *transferred_length* member of the Event. Reception of a zero-sized message is
5339 supported and will consume a posted Receive. Once a successful Completion Event has been
5340 generated for the Receive, the order of the bytes in the local buffer specified by *local_segments*
5341 and *num_segments* corresponds to the order defined by the *local_segments* of the corresponding
5342 Send operation unless there is overlap among the segments of the local Receive buffer. If there
5343 is such an overlap, the content of the local Receive buffer after the Completion Event has been
5344 generated is undefined. Prior to the Completion Event being generated, the content of the local
5345 buffer is Implementation-dependent.

5346 A Consumer should not modify the local buffer specified by *num_segments* and *local_segments*
5347 until the DTO is completed. When a Consumer does not adhere to this rule, the behavior of the
5348 Implementation and the underlying transport is not defined. A Consumer does get back the
5349 ownership of the array specified by the *local_segments* and *num_segments* arguments (but not
5350 the local buffer identified by this array) when *it_post_rcv* returns and is free to use this array for
5351 other calls, to modify it, or to destroy it.

5352 The Implementation ensures that each Receive corresponds to one and only one remote Send and
5353 in no way corresponds to any RDMA Read or RDMA Write Data Transfer Operations over the
5354 same Connection.

5355 The Implementation ensures that an RDMA Write DTO from a remote Endpoint preceding a
5356 Send DTO from the same remote Endpoint has fully delivered its payload prior to the
5357 completion of the Receive DTO corresponding to the Send DTO. However, the Implementation
5358 does not ensure that the RDMA Write DTO places its entire data payload into its Destination
5359 buffer before the Receive DTO places the data payload of the incoming Send into its Receive
5360 buffer; if these two buffers overlap, their contents will be indeterminate.

5361 The Implementation ensures that Receive DTOs matching subsequent Send DTOs from the same
5362 remote Endpoint complete in the post order of these Send DTOs. However, the Implementation
5363 does not ensure that the matching Receive DTOs place the data payloads into their Receive
5364 buffers in the post order of the Send DTOs; if the Receive buffers overlap, their contents will be
5365 indeterminate. If a collection of Endpoints is using an S-RQ, the Receive Completion Events
5366 will be generated for a particular Endpoint in the same order that the corresponding Sends were
5367 done on the associated remote Endpoint.

5368 For Receives that are posted to an Endpoint, the Implementation ensures that all Receives start
5369 and complete in the order posted. The same is not true for Receives that are posted to an S-RQ;
5370 such Receive Work Requests do not necessarily complete in the order posted, nor in the order in
5371 which Send messages arrive, nor in the order in which the Receive Work Requests are dequeued
5372 from the S-RQ.

5373 There is no order relationship between completions of Receive DTOs and any other Work
5374 Requests posted to an Endpoint.

5375 **RETURN VALUE**

5376 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

5377 IT_ERR_INVALID_DTO_FLAGS The Data Transfer Operation flags (*dto_flags*) value
5378 was invalid.

5379 IT_ERR_INVALID_EP The input handle (*handle*) was invalid.

5380 IT_ERR_INVALID_EP_TYPE The attempted operation was invalid for the Service
5381 Type of the Endpoint.

5382 IT_ERR_INVALID_NUM_SEGMENTS The requested number of segments (*num_segments*)
5383 was larger than the Endpoint or S-RQ supports.

5384 IT_ERR_INVALID_SRQ The S-RQ handle was invalid.

5385 IT_ERR_TOO_MANY_POSTS The operation failed due to an overflow of a Work
5386 Queue.

5387 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in
5388 the disabled state. None of the output parameters from
5389 this routine are valid. See [it_ia_info_t](#) for a description
5390 of the disabled state.

5391 **ASYNCHRONOUS ERRORS**

5392 A completion status (see [it_dto_status_t](#)) other than IT_DTO_SUCCESS will break the
5393 Connection associated with the Endpoint that received the corresponding Send message by
5394 moving that Endpoint to the IT_EP_STATE_NONOPERATIONAL state and delivering an
5395 IT_CM_MSG_CONN_BROKEN_EVENT Event to the Connect EVD of that Endpoint. Once
5396 that Connection is broken, all outstanding and in-progress operations on that Connection will
5397 complete with an error status. If the reported completion status is not IT_DTO_SUCCESS, the
5398 content of the local buffer is not defined.

5399 Any posting to an Endpoint that is in the IT_EP_STATE_NONOPERATIONAL state will be
5400 flushed with completion status set to IT_DTO_ERR_FLUSHED.

5401 For errors that can be reported as Completion Errors, see also [it_dto_status_t](#).

5402 If no Receive buffers are posted in time, then an incoming Send will fail to deliver its data. In
5403 order to avoid this error, the Consumer should post Receive resources prior to the remote
5404 Consumer posting a Send. The lack of a Receive buffer is handled as follows:

5405 For IB and iWARP, an `IT_ASYNC_AFF_EP_L_RECV_ACCESS_VIOLATION` Affiliated
5406 Asynchronous Error is generated.

5407 For the IB transport, the Send DTO that was posted remotely completes with an
5408 `IT.DTO.ERR.REMOTE_RESPONDER` error status.

5409 For the iWARP Transport, an `IT_ASYNC_AFF_EP_R_RECV_ACCESS_VIOLATION`
5410 Affiliated Asynchronous Error will surface remotely after the Send DTO posted remotely has
5411 already completed with `IT.DTO.SUCCESS`.

5412 If the Receive buffer is not sufficient in size for the data payload of the incoming Send message,
5413 a length error will occur. In order to avoid length errors, the Consumer should post a buffer large
5414 enough for the incoming Send data. Length errors are handled as follows:

5415 For both the IB and iWARP Transports, the Receive DTO completes with an
5416 `IT.DTO.ERR.LOCAL_LENGTH` error status.

5417 For the IB transport, the Send DTO that was posted remotely completes with an
5418 `IT.DTO.ERR.REMOTE_RESPONDER` error status.

5419 For the iWARP Transport, an `IT_ASYNC_AFF_EP_R_RECV_LENGTH_ERROR` Affiliated
5420 Asynchronous Error will surface remotely after the Send DTO posted remotely has already
5421 completed with `IT.DTO.SUCCESS`.

5422 Despite using the RC service, an incoming Send message may fail to successfully deliver its
5423 data; any data loss or data corruption in the Receive buffer is detected with high probability.

5424 For the IB transport, a data loss or data corruption in the Receive buffer causes the Send DTO
5425 posted remotely and the Receive DTO matching the Send DTO to complete with an
5426 `IT.DTO.ERR.TRANSPORT` Completion Error.

5427 For the iWARP Transport, a data loss or data corruption in the Receive buffer causes the
5428 Receive DTO to complete with an `IT.DTO.ERR.TRANSPORT` Completion Error and an
5429 `IT_ASYNC_AFF_EP_R_ERROR` Affiliated Asynchronous Error to surface remotely after the
5430 Send DTO posted remotely has already completed with `IT.DTO.SUCCESS`.

5431 **APPLICATION USAGE**

5432 This function is used to post a Receive DTO to an Endpoint or Shared Receive Queue,
5433 requesting the transfer of data into a Consumer-specified local buffer from a buffer specified by
5434 the corresponding Send operation on the other side of the Connection.

5435 A Receive DTO can be posted to an Endpoint in any RC Endpoint state (see *it_ep_state_t*) other
5436 than `IT_EP_STATE_NONOPERATIONAL` – i.e., even before Connection establishment – in
5437 order to prepare the Endpoint for reception.

5438 The Consumer should use unique identifiers for *cookie* if they desire to identify each DTO. If the
5439 Consumer does not require a unique DTO identifier, the value of zero or NULL can be used.

5440 For best Receive operation performance, the Consumer should align each buffer segment of
5441 *local_segments* to the *dto_alignment_hint* in the IA attributes obtained via *it_ia_query*.

5442 **SEE ALSO**

5443 *it_post_send(), it_post_sendto(), it_post_rcvfrom(), it_post_rdma_read(), it_post_rdma_write(),*
5444 *it_dto_status_t, it_dto_events, it_dto_flags_t, it_ep_rc_create(), it_lmr_triplet_t, it_ia_query(),*
5445 *it_dto_cookie_t, it_ia_info_t, it_ep_attributes_t, it_ep_state_t*

5446

it_post_recvfrom()

5447

5448 NAME

5449 it_post_recvfrom – post a Receive DTO to a datagram Endpoint

5450 SYNOPSIS

```
5451 #include <it_api.h>
5452
5453 it_status_t it_post_recvfrom(
5454     IN          it_ep_handle_t      ep_handle,
5455     IN const    it_lmr_triplet_t    *local_segments,
5456     IN          size_t              num_segments,
5457     IN          it_dto_cookie_t     cookie,
5458     IN          it_dto_flags_t      dto_flags
5459 );
```

5460 APPLICABILITY

5461 *it_post_recvfrom* is applicable only to Endpoints created for the UD service type.

5462 DESCRIPTION

5463 *ep_handle* Handle for the local datagram Endpoint.

5464 *local_segments* Vector of *it_lmr_triplet_t* data structures that specifies the local buffer that
5465 will contain the received data. Local buffer must be at least 40 bytes.

5466 *num_segments* Number of *it_lmr_triplet_t* data structures in *local_segments*. Must be at
5467 least one.

5468 *cookie* Consumer-provided cookie that is returned to the Consumer in the
5469 Completion Event corresponding to the Receive.

5470 *dto_flags* Flags for posted Receive.

5471 *it_post_recvfrom* posts a request to the *ep_handle* datagram Endpoint to deposit all incoming
5472 data corresponding to a single Send from a remote datagram Endpoint into the local Receive
5473 buffer (data sink) specified by *local_segments* and *num_segments*.

5474 The size of the Receive buffer is given by the sum of the segment lengths specified by
5475 *local_segments*. The first 40 bytes of the Consumer's local buffer are reserved for
5476 Implementation use. For a zero-sized message, the minimum size for the local buffer in
5477 *local_segments* is 40 bytes. To accommodate a larger message, the Consumer should provide a
5478 local buffer in *local_segments* at least 40 bytes bigger than their expected incoming message
5479 size. See *it_dto_events* for more details.

5480 An LMR used in *local_segments* (data sink) must have memory access privileges including local
5481 write access.

5482 The Implementation ensures that an LMR Triplet supports byte alignment for Data Transfer
5483 Operations. The Implementation allows the buffer segments described by the *local_segments*
5484 vector to overlap but the resulting Receive behavior is undefined.

5485 The *cookie* (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work
5486 Request. This identifier is completely under Consumer control and opaque to the

5487 Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted
5488 Work Request.

5489 The *dto_flags* value is used as specified in *it_dto_flags_t*.

5490 A successful call returns IT_SUCCESS, which means that the Receive operation was
5491 successfully posted to the transport layer.

5492 The completion of the posted Receive is reported asynchronously to the Consumer according to
5493 the rules defined in *it_dto_flags_t*. Exactly one Receive Completion Event of type
5494 *it_all_dto_cmpl_event_t* is always generated and manifests on the EVD associated with the
5495 Endpoint Receive Queue (see *it_ep_ud_create*).

5496 If the reported *dto_status* of the Receive Completion Event (see *it_all_dto_cmpl_event_t*) is
5497 IT_DTO_SUCCESS (see *it_dto_status_t*), then the size of the received data can be retrieved
5498 through the *transferred_length* member of the Event. Once a successful Completion Event has
5499 been generated for the Receive, the order of the bytes in the local buffer specified by
5500 *local_segments* and *num_segments* corresponds to the order defined by the *local_segments* of the
5501 corresponding Send operation unless there is overlap among the segments of the local Receive
5502 buffer. If there is such an overlap, the content of the local buffer after the Completion Event has
5503 been generated is undefined. Prior to the Completion Event being generated, the content of the
5504 local buffer is Implementation-dependent. A successful Completion Event indicates that the data
5505 has been delivered uncorrupted into the local buffer.

5506 If the reported completion status is not IT_DTO_SUCCESS, the content of the local buffer is not
5507 defined.

5508 A Consumer should not modify the local buffer specified by *num_segments* and *local_segments*
5509 until the DTO is completed. When a Consumer does not adhere to this rule, the behavior of the
5510 Implementation and the underlying transport is not defined. A Consumer does get back the
5511 ownership of the array specified by the *local_segments* and *num_segments* arguments (but not
5512 the local buffer identified by this array) when *it_post_recvfrom* returns and is free to use this
5513 array for other calls, to modify it, or to destroy it.

5514 The Implementation ensures that each Receive corresponds to one and only one remote Send.

5515 There is no relationship guaranteed on the order of Receive completions and the order of the
5516 posting of the corresponding Sends at remote datagram Endpoints.

5517 The Implementation ensures that all Receives start and complete in the order posted.

5518 The Implementation ensures that all data from a given Send operation is transferred from the
5519 remote buffer and into the local buffer before a Receive completion is generated with
5520 IT_DTO_SUCCESS.

5521 **RETURN VALUE**

5522 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

5523 5524	IT_ERR_INVALID_DTO_FLAGS	The Data Transfer Operation flags (<i>dto_flags</i>) value was invalid.
5525	IT_ERR_INVALID_EP	The Endpoint Handle (<i>ep_handle</i>) was invalid.
5526 5527	IT_ERR_INVALID_EP_TYPE	The attempted operation was invalid for the Service Type of the Endpoint.

5528	IT_ERR_INVALID_NUM_SEGMENTS	The requested number of segments
5529		(<i>num_segments</i>) was larger than the Endpoint
5530		supports.
5531	IT_ERR_TOO_MANY_POSTS	The operation failed due to an overflow of a Work
5532		Queue.
5533	IT_ERR_IA_CATASTROPHE	The IA has experienced a catastrophic error and is
5534		in the disabled state. None of the output
5535		parameters from this routine are valid. See
5536		<i>it_ia_info_t</i> for a description of the disabled state.

5537 **ASYNCHRONOUS ERRORS**

5538 Any posting to an Endpoint that is in the `IT_EP_STATE_UD_NONOPERATIONAL` state will
5539 be flushed with completion status (*it_dto_status_t*) set to `IT_DTO_ERR_FLUSHED`.

5540 For errors that can be reported as Completion Errors, see also *it_dto_status_t*.

5541 If no Receive buffers are posted in time, then an incoming Send will fail to deliver its data. In
5542 order to avoid this error, the Consumer should post Receive resources prior to the remote
5543 Consumer posting a Send. The lack of a Receive buffer is handled as follows:

5544 For the IB transport, an `IT_ASYNC_AFF_EP_L_RECV_ACCESS_VIOLATION` Affiliated
5545 Asynchronous Error is generated.

5546 If the size of an incoming message is larger than the size of the local buffer or larger than the
5547 MTU of the local Spigot, a length error will occur. In order to avoid length errors, the Consumer
5548 should post a buffer large enough for the incoming Send data. Length errors are handled as
5549 follows:

5550 For the IB transport, the Receive DTO completes with an `IT_DTO_ERR_LOCAL_LENGTH`
5551 error status.

5552 **APPLICATION USAGE**

5553 This function is used to transfer data into a Consumer-specified local buffer from a buffer
5554 specified by the corresponding Send operation at the remote Endpoint.

5555 The Consumer should use unique identifiers for *cookie* if they desire to identify each DTO. If the
5556 Consumer does not require a unique DTO identifier, the value of zero or NULL can be used.

5557 For best Receive operation performance, the Consumer should align each buffer segment of
5558 *local_segments* to the *dto_alignment_hint* in the IA attributes obtained via *it_ia_query*.

5559 **SEE ALSO**

5560 *it_post_send()*, *it_post_sendto()*, *it_post_recv()*, *it_post_rdma_read()*, *it_post_rdma_write()*,
5561 *it_dto_status_t*, *it_dto_events*, *it_dto_flags_t*, *it_ep_ud_create()*, *it_lmr_triplet_t*, *it_ia_query()*,
5562 *it_dto_cookie_t*, *it_ia_info_t*

it_post_send()

5563

5564 NAME

5565

it_post_send – post a Send DTO to a connected Endpoint

5566 SYNOPSIS

5567

```
#include <it_api.h>
```

5568

```
it_status_t it_post_send(  
    IN          it_ep_handle_t      ep_handle,  
    IN const    it_lmr_triplet_t    *local_segments,  
    IN          size_t              num_segments,  
    IN          it_dto_cookie_t     cookie,  
    IN          it_dto_flags_t     dto_flags  
);
```

5569

5570

5571

5572

5573

5574

5575

5576 APPLICABILITY

5577

it_post_send is applicable only to Endpoints created for the RC service type.

5578 DESCRIPTION

5579

ep_handle Handle for the Endpoint – the local side of the Connection.

5580

local_segments Vector of *it_lmr_triplet_t* data structures that specifies the local buffer that contains data to be transferred. Can be NULL for a zero-sized message.

5581

5582

num_segments Number of *it_lmr_triplet_t* data structures in *local_segments*. Can be zero for a zero-sized message.

5583

5584

cookie Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the Send.

5585

5586

dto_flags Flags for posted Send.

5587

it_post_send posts a request to the *ep_handle* Endpoint to transfer over the reliable Connection of that Endpoint all the data from the local buffer (data source) specified by *local_segments* and *num_segments* into a remote buffer specified by a single corresponding Receive on the other side of the Connection. If *num_segments* is non-zero, then the size of the data transferred is given by the sum of the segment lengths specified by *local_segments*. A zero-sized message may be transferred over the Connection and will consume a buffer specified by the corresponding Receive.

5588

5589

5590

5591

5592

5593

5594

The Implementation ensures that an LMR Triplet supports byte alignment for Data Transfer Operations. The buffer segments described by *local_segments* can overlap; it is safe to use overlapping buffer segments as a data source.

5595

5596

5597

An LMR used in *local_segments* (data source) must have memory access privileges including local read access.

5598

5599

The *cookie* (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work Request. This identifier is completely under Consumer control and opaque to the Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted Work Request.

5600

5601

5602

5603 The *dto_flags* value is used as specified in *it_dto_flags_t*.

5604 A successful call returns IT_SUCCESS, which means that the Send operation was successfully
5605 posted to the transport layer.

5606 The completion of the posted Send is reported asynchronously to the Consumer according to the
5607 rules defined in *it_dto_flags_t*. A Send Completion Event is of type *it_dto_cmpl_event_t*. Any
5608 generated Send Completion Event manifests on the EVD associated with the Endpoint Send
5609 Queue. See *it_ep_rc_create*, *it_dto_status_t*, and *it_dto_events*. The Completion Event for the
5610 *it_post_send* call indicates to the Consumer that the local buffer is under Consumer control again
5611 and that the contents of the local buffer will be transported reliably, but does not guarantee that
5612 these contents have been successfully received by the remote Consumer; wherever necessary,
5613 this restriction is made explicit by designating a Completion as local or by stating that operations
5614 complete locally. See Section 5.4 of [DDP-IETF] for background information. The contents of
5615 the local buffer are only guaranteed to have reached the remote Consumer's memory when the
5616 remote Consumer reaps a successful completion for the Receive operation that matches the Send
5617 initiated by the *it_post_send* call. However, once the contents of the local buffer reach the
5618 remote Consumer memory, the order of the bytes in the remote buffer specified by the Receive
5619 operation at the remote Endpoint corresponds to the order defined by the Send side
5620 *local_segments*, subject to overlap constraints. See *it_post_recv* for details on overlap
5621 constraints.

5622 A Consumer should not modify the local buffer specified by *num_segments* and *local_segments*
5623 until the DTO is locally completed. When a Consumer does not adhere to this rule, the content
5624 of the local buffer at the receiving side is undefined after the matching Receive operation
5625 completes. A Consumer does get back the ownership of the *num_segments* and *local_segments*
5626 arguments (but not the local buffer identified by them) when *it_post_send* returns and is free to
5627 use the *num_segments* and *local_segments* arguments for other calls, to modify them, or to
5628 destroy them. A Consumer does get back the ownership of the array specified by the
5629 *local_segments* and *num_segments* arguments (but not the local buffer identified by this array)
5630 when *it_post_send* returns and is free to use this array for other calls, to modify it, or to destroy
5631 it.

5632 The Implementation ensures that each Send corresponds to one and only one remote Receive and
5633 in no way corresponds to any locally or remotely posted RDMA Read or RDMA Write Data
5634 Transfer Operations over the same Connection.

5635 The Implementation ensures that each RDMA Write DTO posted to an Endpoint prior to a Send
5636 DTO posted to the same Endpoint has its complete data payload delivered to the remote memory
5637 prior to the completion of the Receive DTO at the remote side matching the Send DTO.
5638 However, the Implementation does not ensure that the RDMA Write DTO places its entire data
5639 payload into its remote buffer before the Receive DTO places the data payload of the incoming
5640 Send into its Receive buffer; if the remote buffers overlap, their contents will be indeterminate.

5641 The Implementation ensures that subsequent Send DTOs posted to the same Endpoint start and
5642 complete locally in post order and that the Receive DTOs at the remote side matching the Send
5643 DTOs complete in the same order. However, the Implementation does not ensure that the
5644 matching Receive DTOs place the data payloads into their Receive buffers in the post order of
5645 the Send DTOs; if the Receive buffers overlap, their contents will be indeterminate.

5646 In general, Work Requests following an RDMA Read may start execution while the RDMA
5647 Read is in progress (but may not complete before the RDMA Read completes). To ensure

5648 deterministically that a Send DTO following an RDMA Read DTO starts after the RDMA Read
5649 completes, specify the `IT_BARRIER_FENCE_FLAG` on the Send DTO.

5650 Any Work Request posted to the Send Queue of an Endpoint (thus also a Send) after a call to
5651 [it_rmr_link](#) or [it_rmr_unlink](#) will not begin execution until the Link or Unlink operation has
5652 completed.

5653 **EXTENDED DESCRIPTION**

5654 On the InfiniBand Transport, the Implementation ensures that an RDMA Write DTO followed
5655 by a Send DTO posted to the same Endpoint cause data payloads to be placed in remote memory
5656 in post order; relying on such placement ordering represents a transport-dependent programming
5657 practice.

5658 **RETURN VALUE**

5659 A successful call returns `IT_SUCCESS`.

5660 Posting to an Endpoint that is not in the `IT_EP_STATE_CONNECTED` or `IT_EP_STATE_`
5661 `NONOPERATIONAL` state will return the `IT_ERR_INVALID_EP_STATE` immediate error.

5662 The possible immediate errors for `it_post_send` are listed below:

5663 `IT_ERR_INVALID.DTO_FLAGS` The Data Transfer Operation flags (`dto_flags`) value was
5664 invalid.

5665 `IT_ERR_INVALID_EP` The Endpoint Handle (`ep_handle`) was invalid.

5666 `IT_ERR_INVALID_EP_STATE` The Endpoint was not in the proper state for the attempted
5667 operation.

5668 `IT_ERR_INVALID_EP_TYPE` The attempted operation was invalid for the Service Type
5669 of the Endpoint.

5670 `IT_ERR_INVALID_NUM_SEGMENTS` The requested number of segments (`num_segments`) was
5671 larger than the Endpoint supports.

5672 `IT_ERR_TOO_MANY_POSTS` The operation failed due to an overflow of a Work Queue.

5673 `IT_ERR_IA_CATASTROPHE` The IA has experienced a catastrophic error and is in the
5674 disabled state. None of the output parameters from this
5675 routine are valid. See [it_ia_info_t](#) for a description of the
5676 disabled state.

5677 **ASYNCHRONOUS ERRORS**

5678 For Work Requests posted to an Endpoint in the `IT_EP_STATE_CONNECTED` state, a
5679 completion status (see [it_dto_status_t](#)) other than `IT.DTO_SUCCESS` will break the Connection
5680 by moving the Endpoint to the `IT_EP_STATE_NONOPERATIONAL` state and deliver an
5681 `IT_CM_MSG_CONN_BROKEN_EVENT` Event to the Connect EVD of `ep_handle`. Once the
5682 Connection is broken, all outstanding and in-progress operations on the Connection will
5683 complete with an error status.

5684 Any posting to an Endpoint that is in the `IT_EP_STATE_NONOPERATIONAL` state will be
5685 flushed with completion status set to `IT.DTO_ERR_FLUSHED`.

5686 For locally or remotely detected errors that can be reported as Completion Errors, see also
5687 [it_dto_status_t](#).

5688 The handling of remotely detected errors is transport- and Implementation-dependent; end-to-
5689 end completions are not supported for certain transports or DTOs.

5690 If no Receive buffers are posted in time at the remote end, then a Send will fail to successfully
5691 deliver the contents of the local buffer. In order to avoid this error, the remote Consumer should
5692 post Receive resources prior to the local Consumer posting the Send. The lack of a remote
5693 Receive buffer is handled as follows:

5694 For IB and iWARP, an IT_ASYNC_AFF_EP_L_RECV_ACCESS_VIOLATION Affiliated
5695 Asynchronous Error will surface remotely.

5696 For the IB transport, the Send DTO completes with an IT_DTO_ERR_REMOTE_RESPONDER
5697 error status.

5698 For the iWARP Transport, an IT_ASYNC_AFF_EP_R_RECV_ACCESS_VIOLATION
5699 Affiliated Asynchronous Error will surface after the Send DTO has already completed with
5700 IT_DTO_SUCCESS.

5701 If the Receive buffer of the Receive DTO at the remote side matching the Send DTO is not
5702 sufficient in size for the data payload of the Send DTO, a length error will occur. In order to
5703 avoid length errors, the remote Consumer should post a buffer large enough for the incoming
5704 Send data. Remotely detected length errors are handled as follows:

5705 For both the IB and iWARP Transports, the Receive DTO at the remote side completes with an
5706 IT_DTO_ERR_LOCAL_LENGTH error status.

5707 For the IB transport, the Send DTO completes with an IT_DTO_ERR_REMOTE_RESPONDER
5708 error status.

5709 For the iWARP Transport, an IT_ASYNC_AFF_EP_R_RECV_LENGTH_ERROR Affiliated
5710 Asynchronous Error will surface after the Send DTO has already completed with
5711 IT_DTO_SUCCESS.

5712 Despite using the RC service, a Send operation may fail to successfully deliver the contents of
5713 the local buffer. This may result in data loss or data corruption in the remote buffer, which is
5714 detected by the remote Implementation with high probability.

5715 For the IB transport, a data loss or data corruption in the remote buffer causes the Send DTO and
5716 possibly the Receive DTO matching the Send DTO to complete with an
5717 IT_DTO_ERR_TRANSPORT Completion Error.

5718 For the iWARP Transport, a data loss or data corruption in the remote buffer causes the Receive
5719 DTO matching the Send DTO to complete with an IT_DTO_ERR_TRANSPORT Completion
5720 Error, and locally as an IT_ASYNC_AFF_EP_R_ERROR Affiliated Asynchronous Error after
5721 the Send DTO has already completed with IT_DTO_SUCCESS.

5722 If a type of remotely detected error can manifest locally in different ways – i.e., as an Affiliated
5723 Asynchronous Error or as a Completion Error – transport-independent programming requires
5724 Consumers to be prepared to deal with either.

5725 **APPLICATION USAGE**

5726 This function is used after a Connection has been established to transfer data from a Consumer-
5727 specified local buffer to a buffer specified by the corresponding Receive operation on the other
5728 side of the Connection.

5729 The Consumer should use unique identifiers for *cookie* if they desire to identify each DTO. If the
5730 Consumer does not require a unique DTO identifier, the value of zero or NULL can be used.

5731 For best Send operation performance, the Consumer should align each buffer segment of
5732 *local_segments* to the *dto_alignment_hint* in the IA attributes obtained via *it_ia_query*.

5733 **SEE ALSO**

5734 *it_post_sendto()*, *it_post_rcv()*, *it_post_rcvfrom()*, *it_post_rdma_read()*, *it_post_rdma_write()*,
5735 *it_dto_status_t*, *it_dto_events*, *it_dto_flags_t*, *it_ep_rc_create()*, *it_lmr_triplet_t*, *it_ia_query()*,
5736 *it_dto_cookie_t*, *it_ia_info_t*

it_post_sendto()

5737

5738 NAME

5739

it_post_sendto – post a Send DTO to a datagram Endpoint

5740 SYNOPSIS

5741

```
#include <it_api.h>
```

5742

5743

```
it_status_t it_post_sendto(
```

5744

```
    IN          it_ep_handle_t          ep_handle,
```

5745

```
    IN const    it_lmr_triplet_t        *local_segments,
```

5746

```
    IN          size_t                  num_segments,
```

5747

```
    IN          it_dto_cookie_t         cookie,
```

5748

```
    IN          it_dto_flags_t          dto_flags,
```

5749

```
    IN const    it_dg_remote_ep_addr_t *remote_ep_addr
```

5750

```
);
```

5751 APPLICABILITY

5752

it_post_sendto is applicable only to Endpoints created for the UD service type.

5753 DESCRIPTION

5754

ep_handle Handle for the local datagram Endpoint.

5755

local_segments Vector of *it_lmr_triplet_t* that specifies the local buffer that contains data to be transferred. Can be NULL for a zero-sized message.

5756

5757

num_segments Number of *it_lmr_triplet_t* data structures in *local_segments*. Can be zero for a zero-sized message.

5758

5759

cookie Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the Send.

5760

5761

dto_flags Flags for posted Send.

5762

remote_ep_addr Remote datagram Endpoint address.

5763

it_post_sendto posts a request to the *ep_handle* datagram Endpoint to transfer all the data from the local buffer (data source) specified by *local_segments segments* and *num_segments* into a remote buffer specified by a single corresponding Receive at the remote datagram Endpoint identified by *remote_ep_addr*. If *num_segments* is non-zero, then the size of the data transferred is given by the sum of the segment lengths specified by *local_segments*. No guarantee of delivery is provided.

5764

5765

5766

5767

5768

5769

The Implementation ensures that an LMR Triplet supports byte alignment for Data Transfer Operations. The buffer segments described by *local_segments* can overlap; it is safe to use overlapping buffer segments as a data source.

5770

5771

5772

An LMR used in *local_segments* (data source) must have memory access privileges including local read access.

5773

5774

The *cookie* (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work Request. This identifier is completely under Consumer control and opaque to the

5775

5776 Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted
5777 Work Request.

5778 The *dto_flags* value is used as specified in *it_dto_flags_t*.

5779 *remote_ep_addr* specifies the Destination for the *it_post_sendto* operation. See
5780 *it_dg_remote_ep_addr_t* for details on the format of this data structure.

5781 A successful call returns IT_SUCCESS, which means that the Send operation was successfully
5782 posted to the transport layer.

5783 The completion of the posted Send is reported asynchronously to the Consumer according to the
5784 rules defined in *it_dto_flags_t*. A Send Completion Event is of type *it_all_dto_cmpl_event_t*.
5785 Any generated Send Completion Event manifests on the EVD associated with the Endpoint Send
5786 Queue. See *it_ep_ud_create*, *it_dto_status_t* and *it_dto_events*.

5787 Once a successful Completion Event has been generated at the receiver, the order of the bytes in
5788 the remote buffer specified by the Receive operation at the remote Endpoint corresponds to the
5789 order defined by the Send side *local_segments* subject to overlap constraints. See
5790 *it_post_recvfrom* for details on overlap constraints.

5791 A Consumer should not modify the local buffer specified by *num_segments* and *local_segments*
5792 until the DTO is completed. When a Consumer does not adhere to this rule, the behavior of the
5793 Implementation and the underlying transport is not defined. A Consumer does get back the
5794 ownership of the array specified by the *local_segments* and *num_segments* arguments (but not
5795 the local buffer identified by this array) when *it_post_sendto* returns and is free to use this array
5796 for other calls, to modify it, or to destroy it.

5797 The Implementation ensures that all Sends start and complete in the order posted.

5798 The Implementation makes no delivery order guarantees for Unreliable Datagrams.

5799 There is no delivery order or completion order between Receive Data Transfer Operations on
5800 different Destinations that correspond to the Sends posted in order to the same Unreliable
5801 Datagram Endpoint.

5802 When *it_post_sendto* completes with IT_DTO_SUCCESS or IT_DTO_ERR_LOCAL_EP there
5803 is no guarantee that the DTO has reached the remote Endpoint.

5804 **RETURN VALUE**

5805 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

5806 5807 5808	IT_ERR_INVALID_AH	The Address Handle within <i>remote_ep_addr</i> was invalid or the does not match the <i>spigot_id</i> of the Endpoint.
5809 5810	IT_ERR_INVALID_DTO_FLAGS	The Data Transfer Operation flags (<i>dto_flags</i>) value was invalid.
5811	IT_ERR_INVALID_EP	The Endpoint Handle (<i>ep_handle</i>) was invalid.
5812 5813	IT_ERR_INVALID_EP_TYPE	The attempted operation was invalid for the Service Type of the Endpoint.
5814 5815	IT_ERR_INVALID_NUM_SEGMENTS	The requested number of segments (<i>num_segments</i>) was larger than the Endpoint supports.

5816 IT_ERR_TOO_MANY_POSTS The operation failed due to an overflow of a Work
5817 Queue.

5818 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in
5819 the disabled state. None of the output parameters
5820 from this routine are valid. See *it_ia_info_t* for a
5821 description of the disabled state.

5822 **ASYNCHRONOUS ERRORS**

5823 Any posting to an Endpoint that is in the IT_EP_STATE_UD_NONOPERATIONAL state will
5824 be flushed with completion status (*it_dto_status_t*) set to IT_DTO_ERR_FLUSHED.

5825 For errors that can be reported as Completion Errors, see also *it_dto_status_t*.

5826 If no Receive buffers are posted in time at the remote end, then a Send will fail to successfully
5827 deliver the contents of the local buffer. In order to avoid this error, the remote Consumer should
5828 post Receive resources prior to the local Consumer posting the Send. The lack of a remote
5829 Receive buffer is handled as follows:

5830 For the IB transport, an IT_ASYNC_AFF_EP_L_RECV_ACCESS_VIOLATION Affiliated
5831 Asynchronous Error will surface remotely.

5832 If the Receive buffer of the Receive DTO at the remote side matching the Send DTO is not
5833 sufficient in size for the data payload of the Send DTO, a length error will occur. In order to
5834 avoid length errors, the remote Consumer should post a buffer large enough for the incoming
5835 Send data. Remotely detected length errors are handled as follows:

5836 For the IB transport, the Receive DTO at the remote side completes with an
5837 IT_DTO_ERR_LOCAL_LENGTH error status.

5838 A completion status IT_DTO_ERR_LOCAL_EP may be returned in case of an Address Handle
5839 inconsistency (see Application Usage).

5840 **APPLICATION USAGE**

5841 This function is used to transfer data from a Consumer-specified local buffer to a buffer
5842 specified by the corresponding Receive operation at a remote datagram Endpoint.

5843 The Consumer should use unique identifiers for *cookie* if they desire to identify each DTO. If the
5844 Consumer does not require a unique DTO identifier, the value of zero or NULL can be used.

5845 For best Send operation performance, the Consumer should align each buffer segment of
5846 *local_segments* to the *dto_alignment_hint* in the IA attributes obtained via *it_ia_query*.

5847 An Address Handle corresponds to a specific Spigot on an IA. Attempting to *it_post_sendto* on
5848 an Endpoint using an Address Handle that does not correspond to the Spigot associated with the
5849 Endpoint must be avoided by the Consumer. If the Consumer persists in this practice, they must
5850 write error handling code to deal with three possible error cases: One – the *it_post_sendto* call
5851 will return the IT_ERR_INVALID_AH error immediately. Or two – the DTO will complete in
5852 error with the *it_dto_status* set to IT_DTO_ERR_LOCAL_EP. Or three – there will be no
5853 indication of error. The three possible cases represent the allowable implementations of the
5854 underlying technology.

5855 **SEE ALSO**

5856 *it_post_send(), it_post_recv(), it_post_recvfrom(), it_post_rdma_read(), it_post_rdma_write(),*
5857 *it_dto_status_t, it_dto_events, it_dg_remote_ep_addr_t, it_dto_flags_t, it_ep_ud_create(),*
5858 *it_lmr_triplet_t, it_ia_query(), it_dto_cookie_t, it_ia_info_t*

5859

it_pz_create()

5860

5861 NAME

5862 `it_pz_create` – create a new Protection Zone

5863 SYNOPSIS

```
5864 #include <it_api.h>
5865
5866 it_status_t it_pz_create(
5867     IN  it_ia_handle_t  ia_handle,
5868     OUT it_pz_handle_t  *pz_handle
5869 );
```

5870 DESCRIPTION

5871 *ia_handle* Interface Adapter on which the Protection Zone will be created.

5872 *pz_handle* Handle of new Protection Zone

5873 The *it_pz_create* routine creates a new Protection Zone that may be used to create Local
5874 Memory Regions, Remote Memory Regions, transport Endpoints, or Address Handles on the
5875 Interface Adapter identified by *ia_handle*. The Protection Zone is returned in *pz_handle*.

5876 RETURN VALUE

5877 A successful call returns `IT_SUCCESS`. Otherwise, returns an error code as described below:

5878 `IT_ERR_RESOURCES` The requested operation failed due to insufficient resources.

5879 `IT_ERR_INVALID_IA` The Interface Adapter Handle (*ia_handle*) was invalid.

5880 `IT_ERR_IA_CATASTROPHE` The IA has experienced a catastrophic error and is in the disabled
5881 state. None of the output parameters from this routine are valid.
5882 See *it_ia_info_t* for a description of the disabled state.

5883 APPLICATION USAGE

5884 An LMR, RMR, Endpoint, or Address Handle cannot be created without supplying a Protection
5885 Zone. An LMR, RMR, or Address Handle may only be used in concert with an Endpoint having
5886 the same Protection Zone. In DTO, Link, and Unlink operations, the Protection Zone of the local
5887 Endpoint and the LMR must match, or the operation will fail. In RDMA operations, the
5888 Protection Zone of the RMR associated with the RMR Context must match that of the remote
5889 Endpoint. In datagram DTO operations, the Protection Zone of the local Address Handle
5890 identifying the Destination must match that of the local Endpoint. In Link and Unlink operations,
5891 the Protection Zone of the LMR and RMR must match.

5892 SEE ALSO

5893 [*it_pz_free\(\)*](#), [*it_pz_query\(\)*](#)

it_pz_free()

5894

5895 NAME

5896 `it_pz_free` – destroy a Protection Zone

5897 SYNOPSIS

```
5898 #include <it_api.h>
5899
5900 it_status_t it_pz_free(
5901     IN it_pz_handle_t pz_handle
5902 );
```

5903 DESCRIPTION

5904 *pz_handle* Handle of Protection Zone to be destroyed.

5905 The *it_pz_free* routine destroys the Protection Zone *pz_handle*. On successful return, the
5906 *pz_handle* may no longer be used. An attempt to free a Protection Zone that is still referenced by
5907 undestroyed Endpoints, Local Memory Regions, Remote Memory Regions, or Address Handles
5908 will fail with IT_ERR_PZ_BUSY, and the Protection Zone will be unaffected.

5909 RETURN VALUE

5910 A successful call returns IT_SUCCESS. Otherwise, returns an error code as described below:

5911 IT_ERR_INVALID_PZ The Protection Zone Handle (*pz_handle*) was invalid.

5912 IT_ERR_PZ_BUSY The Protection Zone was still in use.

5913 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the disabled
5914 state. None of the output parameters from this routine are valid.
5915 See *it_ia_info_t* for a description of the disabled state.

5916 SEE ALSO

5917 *it_pz_create()*, *it_pz_query()*

it_pz_query()

5918

5919 NAME

5920 `it_pz_query` – get attributes of a Protection Zone

5921 SYNOPSIS

```
5922 #include <it_api.h>
5923
5924 it_status_t it_pz_query(
5925     IN    it_pz_handle_t    pz_handle,
5926     IN    it_pz_param_mask_t mask,
5927     OUT   it_pz_param_t     *params
5928 );
5929
5930 typedef enum {
5931     IT_PZ_PARAM_ALL = 0x01,
5932     IT_PZ_PARAM_IA  = 0x02
5933 } it_pz_param_mask_t;
5934
5935 typedef struct {
5936     it_ia_handle_t ia; /* IT_PZ_PARAM_IA */
5937 } it_pz_param_t;
```

5938 DESCRIPTION

5939 *pz_handle* Protection Zone.

5940 *mask* Bitwise OR of flags for desired parameters.

5941 *params* Structure whose members are written with the desired parameters.

5942 The *it_pz_query* routine returns the desired parameters of the Protection Zone *pz_handle* in the structure pointed to by *params*. On return, each field of *params* is only valid if the corresponding flag as shown in the Synopsis is set in the mask argument. The mask value `IT_PZ_PARAM_ALL` causes all fields to be returned.

5946 The definition of each field of *params* follows:

5947 *ia* The Interface Adapter Handle specified to create the Protection Zone.

5948 RETURN VALUE

5949 A successful call returns `IT_SUCCESS`. Otherwise, returns an error code as described below:

5950 `IT_ERR_INVALID_PZ` The Protection Zone Handle (*pz_handle*) was invalid.

5951 `IT_ERR_INVALID_MASK` The *mask* contained invalid flag values.

5952 `IT_ERR_IA_CATASTROPHE` The IA has experienced a catastrophic error and is in the disabled state. None of the output parameters from this routine are valid. See *it_ia_info_t* for a description of the disabled state.

5955 SEE ALSO

5956 *it_pz_create()*, *it_pz_free()*

it_reject()

5957

5958 NAME

5959

it_reject – reject an incoming Connection Request or Connection Reply

5960 SYNOPSIS

5961

```
#include <it_api.h>
```

5962

```
it_status_t it_reject(  
    IN          it_cn_est_identifier_t  cn_est_id,  
    IN const unsigned char             *private_data,  
    IN          size_t                 private_data_length  
);
```

5968

5969

```
typedef uint64_t it_cn_est_identifier_t;
```

5970 APPLICABILITY

5971

it_reject is applicable only to the RC service type.

5972 DESCRIPTION

5973

cn_est_id Connection Establishment Identifier associated with the Connection Request to be rejected. Calling *it_reject* destroys the identifier. See [it_ep_accept](#) for a definition of this data type.

5974

5975

5976

private_data Opaque Private Data to be sent in the IT_CM_MSG_CONN_PEER_REJECT_EVENT Event delivered to the Remote Consumer. If the IA does not support Private Data, *private_data_length* must be zero.

5977

5978

5979

private_data_length Length of *private_data*. This field must be 0 if the IA does not support Private Data.

5980

5981

it_reject rejects an incoming Connection Request or Connection Reply. The Remote Endpoint will receive an IT_CM_MSG_CONN_PEER_REJECT_EVENT Event on its IT_CM_MSG_EVENT_STREAM Simple Event Dispatcher, and that Endpoint will transition into the IT_EP_STATE_NONOPERATIONAL state.

5982

5983

5984

5985

For two-way Connection establishment, *it_reject* can only be called on the Passive side in response to the IT_CM_REQ_CONN_REQUEST_EVENT Event.

5986

5987

For three-way Connection establishment, *it_reject* can be called on the Passive side in response to the IT_CM_REQ_CONN_REQUEST_EVENT, or on the Active side in response to the IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT. If *it_reject* is called on the active side, the local Endpoint associated with the Connection establishment transitions to the IT_EP_STATE_NONOPERATIONAL state. See the [it_ep_state_t](#) reference page for a description of this Endpoint state.

5988

5989

5990

5991

5992

5993

Once the Endpoint is in the IT_EP_STATE_NONOPERATIONAL state, any pending Data Transfer Operations or Link or Unlink operations on the Endpoint will be flushed and will generate Completion Events with a Status of IT_DTO_ERR_FLUSHED.

5994

5995

5996

The Connection Establishment Identifier, *cn_est_id*, is freed by *it_reject*.

5997 **RETURN VALUE**

5998 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

5999 IT_ERR_INVALID_CN_EST_ID The Connection Establishment Identifier (*cn_est_id*)
6000 was invalid.

6001 IT_ERR_PDATA_NOT_SUPPORTED Private Data was supplied by the Consumer but this
6002 Interface Adapter does not support Private Data. See
6003 [it_ia_query](#) for the IAs capabilities to support Private
6004 Data.

6005 IT_ERR_INVALID_PDATA_LENGTH The Interface Adapter supports Private Data, but the
6006 length specified exceeded the Interface Adapter's
6007 capabilities.

6008 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in
6009 the disabled state. None of the output parameters from
6010 this routine are valid. See [it_ia_info_t](#) for a description
6011 of the disabled state.

6012 **APPLICATION USAGE**

6013 1. The Consumer is responsible for coordinating the use of functions that free a Connection
6014 Establishment Identifier (*cn_est_id*) such as [it_ep_accept](#), [it_reject](#), [it_ep_disconnect](#), and
6015 [it_handoff](#). The behavior of functions that are passed in an invalid Connection
6016 Establishment Identifier is indeterminate.

6017 2. Calling [it_reject](#) does not necessarily mean that the remote end will receive a peer reject
6018 Event; they might receive a non-peer reject Event (e.g., because the REJ message got lost
6019 on IB and the connection establishment attempt timed out). If the remote does get a peer
6020 reject Event, the Private Data contained within that Event will be exactly what was
6021 furnished to [it_reject](#).

6022 **SEE ALSO**

6023 [it_ep_accept\(\)](#), [it_ep_connect\(\)](#), [it_cm_req_events](#), [it_cm_msg_events](#), [it_ep_state_t](#),
6024 [it_handoff\(\)](#), [it_ia_query\(\)](#)

6025

it_rmr_create()

6026

6027 NAME

6028 `it_rmr_create` – create a Remote Memory Region (RMR)

6029 SYNOPSIS

```
6030 #include <it_api.h>
6031
6032 it_status_t it_rmr_create(
6033     IN  it_pz_handle_t    pz_handle,
6034     IN  it_rmr_type_t     rmr_type,
6035     OUT it_rmr_handle_t  *rmr_handle
6036 );
```

6037 APPLICABILITY

6038 Remote Memory Regions can be used only for the RC service type.

6039 DESCRIPTION

6040 *pz_handle* Protection Zone in which the Remote Memory Region will be created.

6041 *rmr_type* Desired type for the Remote Memory Region. RMR types are defined in
6042 [it_rmr_type_t](#).

6043 *rmr_handle* Handle of new Remote Memory Region.

6044 The *it_rmr_create* routine creates a Remote Memory Region within a Protection Zone identified
6045 by the *pz_handle* argument and associated with the Interface Adapter implicitly identified by
6046 *pz_handle*. RMRs can be accessed only remotely (for setting access privileges, see [it_rmr_link](#)).
6047 The restriction to remote access does not preclude a special case where a “local RMR” is used
6048 locally in a DTO – see the Extended Description for details.

6049 The desired RMR type is specified through the *rmr_type* argument. The allowable RMR types
6050 are Implementation-dependent (see [it_rmr_type_t](#) for details).

6051 If the Consumer specifies an *rmr_type* of `IT_RMR_TYPE_DEFAULT`, the Implementation will
6052 choose a type of RMR that is supported by the underlying IA, which may be either a Narrow or a
6053 Wide RMR. Requesting an RMR type that is not supported by the Implementation will cause
6054 *it_rmr_create* to fail.

6055 An RMR cannot be used as a target for DTOs until it has been linked with [it_rmr_link](#).

6056 An unlinked Wide or Narrow RMR may be used as a target for Link operations via all Endpoints
6057 in the PZ of the RMR.

6058 A linked Wide RMR may be used as a target for DTOs and Link/Unlink operations via all
6059 Endpoints in the PZ of the RMR. A linked Narrow RMR may be used as a target for DTOs and
6060 Unlink operations only via the Endpoint through which it was linked.

6061 EXTENDED DESCRIPTION

6062 InfiniBand Implementations at least support Wide RMRs and, if Verb Extensions (BMM) are
6063 provided, also Narrow RMRs; iWARP Implementations at least support Narrow RMRs.

6064 For iWARP only, RMRs can also be used locally, namely as data sinks in an
6065 [it_post_rdma_read_to_rmr](#) call; see the Extended Description of [it_post_rdma_read_to_rmr](#) for
6066 why iWARP nevertheless considers the writing to a local data sink of an RDMA Read DTO a
6067 remote write access.

6068 **BACKWARDS COMPATIBILITY**

6069 [it_rmr_create](#) (v2.0) called with *rmr_type* set to IT_RMR_TYPE_WIDE behaves identically as
6070 [it_rmr_create](#) (v1.0), which creates Wide RMRs only.

6071 [it_rmr_create](#) (v2.0) called with *rmr_type* set to IT_RMR_TYPE_DEFAULT selects the
6072 appropriate RMR type for the given Implementation.

6073 See also Appendix C.

6074 **RETURN VALUE**

6075 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

6076 IT_ERR_INVALID_PZ The Protection Zone Handle (*pz_handle*) was invalid.

6077 IT_ERR_INVALID_RMR_TYPE The RMR type (*rmr_type*) was invalid or not supported
6078 by the Implementation.

6079 IT_ERR_RESOURCES The requested operation failed due to insufficient
6080 resources.

6081 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the
6082 disabled state. None of the output parameters from this
6083 routine are valid. See [it_ia_info_t](#) for a description of the
6084 disabled state.

6085 **APPLICATION USAGE**

6086 The returned RMR must be linked to a Local Memory Region by means of [it_rmr_link](#) before it
6087 can be used in Data Transfer Operations. Remote use of the linked RMR also requires exposing
6088 the RMR to a remote Consumer through an associated RMR Context.

6089 Creating an RMR is a relatively expensive operation. Once created, however, an RMR may be
6090 linked repeatedly to different LMR address ranges using the more efficient [it_rmr_link](#) call, as
6091 long as a linked RMR is first unlinked through [it_rmr_unlink](#). Linking an RMR is much more
6092 efficient than changing remote access privileges using [it_lmr_modify](#).

6093 **SEE ALSO**

6094 [it_rmr_link\(\)](#), [it_rmr_free\(\)](#), [it_rmr_query\(\)](#)

it_rmr_free()

6095

6096 NAME

6097 `it_rmr_free` – destroy a Remote Memory Region

6098 SYNOPSIS

```
6099 #include <it_api.h>
6100
6101 it_status_t it_rmr_free(
6102     IN it_rmr_handle_t rmr_handle
6103 );
```

6104 APPLICABILITY

6105 Remote Memory Regions can be used only for the RC service type. See [it_rmr_create](#).

6106 DESCRIPTION

6107 `rmr_handle` Handle of Remote Memory Region to be destroyed.

6108 The `it_rmr_free` routine destroys the Remote Memory Region `rmr_handle`. If the RMR is
6109 currently linked to an LMR, then the RMR linking is also destroyed. On return, the `rmr_handle`
6110 may no longer be used, and the associated RMR Context may no longer be used. RMRs with
6111 memory ranges that overlap the range of `rmr_handle` are not affected by its destruction.

6112 Outstanding remote DTOs that use the RMR Context of this RMR may either complete
6113 successfully or fail with an access violation error.

6114 RETURN VALUE

6115 A successful call returns `IT_SUCCESS`. Otherwise, an error code is returned as described below:

6116 `IT_ERR_INVALID_RMR` The Remote Memory Region Handle (`rmr_handle`) was invalid.

6117 `IT_ERR_IA_CATASTROPHE` The IA has experienced a catastrophic error and is in the disabled
6118 state. None of the output parameters from this routine are valid.
6119 See [it_ia_info_t](#) for a description of the disabled state.

6120 APPLICATION USAGE

6121 Since the number of possible RMR Context values is finite, the Implementation will eventually
6122 re-use previously freed values in a new linking. If a DTO using an RMR Context is posted after
6123 that Context is freed, it is theoretically possible for the Context to be re-used before the DTO
6124 completes, and for the DTO to complete under the new linking for the Context, resulting in data
6125 corruption. To avoid this, the Consumer should not free an RMR which may be the target of
6126 outstanding DTOs. This may require coordination between local and remote Consumers, and
6127 such coordination is the Consumer's responsibility.

6128 SEE ALSO

6129 [it_rmr_create\(\)](#), [it_rmr_query\(\)](#)

6130

it_rmr_link()

6131
6132 **NAME**
6133 `it_rmr_link` – post operation to Link a Remote Memory Region to a memory range

6134 **SYNOPSIS**
6135 `#include <it_api.h>`
6136
6137 `it_status_t it_rmr_link(
6138 IN it_rmr_handle_t rmr_handle,
6139 IN it_lmr_handle_t lmr_handle,
6140 IN void *addr,
6141 IN it_length_t length,
6142 IN it_addr_mode_t addr_mode,
6143 IN it_mem_priv_t privs,
6144 IN it_ep_handle_t ep_handle,
6145 IN it_dto_cookie_t cookie,
6146 IN it_dto_flags_t dto_flags,
6147 OUT it_rmr_context_t *rmr_context
6148);`

6149 **APPLICABILITY**
6150 `it_rmr_link` is applicable only to Endpoints created for the RC service type.

6151 DESCRIPTION

6152	<i>rmr_handle</i>	Handle of RMR that will be linked.
6153	<i>lmr_handle</i>	LMR to which RMR will be linked.
6154	<i>addr</i>	Starting address of RMR to be linked.
6155	<i>length</i>	Length in bytes of RMR to be linked. Must not be 0.
6156	<i>addr_mode</i>	Addressing mode of RMR to be linked.
6157	<i>privs</i>	Bitwise OR of remote access privilege flags for linked RMR, taken from it_mem_priv_t .
6159	<i>ep_handle</i>	Endpoint on which to post the Link operation.
6160	<i>cookie</i>	Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the RMR Link operation.
6161		
6162	<i>dto_flags</i>	Bitwise OR of options for operation handling.
6163	<i>rmr_context</i>	Returned Context allowing remote access to the linked RMR.

6164 The `it_rmr_link` routine posts to Endpoint `ep_handle` an operation to Link the Remote Memory
6165 Region `rmr_handle` to the segment of an LMR specified by the `lmr_handle`, `addr`, and `length`
6166 arguments. It returns a new `rmr_context` value in network byte order that can be transferred by
6167 the local Consumer to a remote Consumer to be used for an RDMA operation.

6168 The arguments *addr* and *length* provide the starting address, also known as the Base Address,
6169 and the length in bytes, respectively, of the region to be linked. The starting address *addr* must
6170 be in the same linear address space as used by the underlying LMR given by *lmr_handle*. The
6171 specified address range must fall within the address range of the underlying LMR.

6172 If the *addr_mode* argument (*it_addr_mode_t*) selects Absolute Addressing, then DTOs will
6173 access the RMR through absolute addresses in the given linear address space. If the *addr_mode*
6174 argument selects Relative Addressing, then DTOs will access the RMR through offsets relative
6175 to the Base Address of the RMR, which is within the address range of the underlying LMR.
6176 Consumers can check the IA attribute *addr_mode_relative_support* to determine whether
6177 Relative Addressing is supported.

6178 An RMR can be linked only if the Endpoint *ep_handle* is in IT_EP_STATE_CONNECTED
6179 state.

6180 While any RMR is associated with a Protection Zone, successfully linking a Narrow RMR
6181 causes the RMR to be also associated with the Endpoint *ep_handle* to which the RMR Link
6182 operation is posted (see *it_rmr_type_t*).

6183 A Narrow RMR can be linked if it is not currently in linked state, while a Wide RMR can be
6184 (re)linked regardless of whether it is currently linked.

6185 The RMR handle must be valid. The LMR handle must be valid. For any RMR type, the
6186 Protection Zones of the RMR, underlying LMR, and Endpoint must match. For any RMR type
6187 and regardless of the *addr_mode* argument, the underlying LMR must use Absolute Addressing.
6188 An RMR of type IT_RMR_TYPE_NARROW must be in unlinked state to be linkable.

6189 Remote access to an RMR is enforced with byte-level granularity.

6190 RMRs can be accessed only remotely, and any local access privileges specified in the *privs*
6191 argument are ignored. The type of remote access to be allowed is specified by the *privs*
6192 argument as a bitwise-inclusive OR of one or more of the bit values IT_PRIV_
6193 REMOTE_READ and IT_PRIV_REMOTE_WRITE. Either or both of IT_PRIV_
6194 REMOTE_READ and IT_PRIV_REMOTE_WRITE must be included; see *it_mem_priv_t* for
6195 bit definitions and predefined bit combinations.

6196 It is legal to request remote access rights that exceed the remote access rights of the underlying
6197 LMR.

6198 The *cookie* (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work
6199 Request. This identifier is completely under Consumer control and opaque to the
6200 Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted
6201 Work Request.

6202 Request handling is specified by the *dto_flags* argument and is the bitwise OR of zero or more of
6203 the following flags:

6204 IT_COMPLETION_FLAG
6205 IT_NOTIFY_FLAG
6206 IT_BARRIER_FENCE_FLAG

6207 For the definition of these flags, see *it_dto_flags_t*. In addition, *it_rmr_link* automatically fences
6208 all DTO, Link, and Unlink operations subsequently submitted on the Endpoint *ep_handle* such
6209 that none of these operations starts until the currently posted Link operation has completed.

6210 The completion of the posted Link operation is reported asynchronously to the Consumer
6211 according to the rules defined in *it_dto_flags_t*. An RMR Link Completion Event is of type
6212 *it_dto_cmpl_event_t*. Any RMR Link Completion Event generated manifests on the EVD
6213 associated with the Endpoint Send Queue. The Event type is IT_RMR_LINK_CMPL_EVENT.

6214 The value of *rmr_context* is immediately available when *it_rmr_link* returns, but it may not be
6215 used by a remote host for an RDMA operation until the Link Completion Event occurs.
6216 Violation of this rule may result in an error and a broken Connection for the reliable Connection
6217 Endpoint on which the RDMA operation is posted. See Application Usage for more details.

6218 After a successful Link Completion Event, any previous linking for the RMR is invalidated.
6219 Successfully linking an RMR results in a new RMR Context being associated with the RMR and
6220 revokes remote access using any previous RMR Contexts, as long as the new RMR Context is
6221 not a re-use of a previously generated RMR Context. The revocation of remote access may affect
6222 operations that are outstanding when *it_rmr_link* is called. Consumers should make sure that
6223 such operations complete before calling *it_rmr_link*.

6224 The new linking remains valid until the next Link or Unlink operation completes successfully, or
6225 until the RMR is destroyed. A Link operation will never be partially successful over a subset of
6226 the requested memory range; it either succeeds completely or fails without invalidating any
6227 portion of the previous linking.

6228 If *it_rmr_link* returns successfully, but the Link Completion Event status indicates failure, then
6229 any previous linking and RMR Context remains valid.

6230 Any Work Request posted to an Endpoint's Send Queue after a call to *it_rmr_link* will not begin
6231 execution until the Link operation has completed.

6232 **EXTENDED DESCRIPTION**

6233 For the InfiniBand Transport, specifying the IT_PRIV_REMOTE_WRITE access privilege for
6234 the RMR requires that the underlying LMR be enabled for local write access.

6235 For the InfiniBand Transport, linking a Wide RMR with Relative Addressing is typically not
6236 supported (even if the IA attribute *addr_mode_relative_support* indicates support for Relative
6237 Addressing); in this case, an IT_ERR_INVALID_ADDR_MODE immediate error is returned.

6238 **BACKWARDS COMPATIBILITY**

6239 *it_rmr_link* called for a Wide RMR and with *addr_mode* set to IT_ADDR_MODE_ABSOLUTE
6240 behaves identically as *it_rmr_bind* (v1.0).

6241 See also Appendix C.

6242 **RETURN VALUE**

6243 A successful call returns IT_SUCCESS. Otherwise, an immediate error is returned and, if the
6244 RMR was previously linked, the previous linking and RMR Context remains valid. It is possible
6245 for *it_rmr_link* to return success but for the Completion Event to indicate failure.

6246 Posting to an Endpoint that is not in the IT_EP_STATE_CONNECTED or
6247 IT_EP_STATE_NONOPERATIONAL state will return the IT_ERR_INVALID_EP_STATE
6248 immediate error.

6249 The possible immediate errors for *it_rmr_link* are listed below.

6250	IT_ERR_ADDRESS	The address (<i>addr</i>) fell outside the boundaries specified by the Local Memory Region.
6251		
6252	IT_ERR_INVALID_ADDR_MODE	The addressing mode (<i>addr_mode</i>) was invalid, unsupported, or unsupported for the RMR type of the given RMR.
6253		
6254		
6255	IT_ERR_INVALID_DTO_FLAGS	The Data Transfer Operation flags (<i>dto_flags</i>) value was invalid.
6256		
6257	IT_ERR_INVALID_EP	The Endpoint Handle (<i>ep_handle</i>) was invalid.
6258	IT_ERR_INVALID_EP_STATE	The Endpoint was not in the proper state for the attempted operation.
6259		
6260	IT_ERR_INVALID_EP_TYPE	The attempted operation was invalid for the Service Type of the Endpoint.
6261		
6262	IT_ERR_INVALID_LENGTH	The value of <i>length</i> fell outside the boundaries of the Local Memory Region or the value of <i>length</i> was 0.
6263		
6264	IT_ERR_INVALID_LMR	The Local Memory Region Handle (<i>lmr_handle</i>) was invalid.
6265		
6266	IT_ERR_INVALID_PRIVS	The requested memory privileges (<i>privs</i>) contained an invalid flag.
6267		
6268	IT_ERR_INVALID_RMR	The Remote Memory Region Handle (<i>rmr_handle</i>) was invalid.
6269		
6270	IT_ERR_RESOURCES	The requested operation failed due to insufficient resources.
6271	IT_ERR_TOO_MANY_POSTS	The operation failed due to an overflow of a Work Queue.
6272	IT_ERR_IA_CATASTROPHE	The IA has experienced a catastrophic error and is in the disabled state. None of the output parameters from this routine are valid. See it_ia_info_t for a description of the disabled state.
6273		
6274		
6275		

6276 **ASYNCHRONOUS ERRORS**

6277 For Work Requests posted to an Endpoint in the IT_EP_STATE_CONNECTED state, a completion status (see [it_dto_status_t](#)) other than IT_DTO_SUCCESS will break the Connection by moving the Endpoint to the IT_EP_STATE_NONOPERATIONAL state and deliver an IT_CM_MSG_CONN_BROKEN_EVENT Event to the Connect EVD of *ep_handle*. Once the Connection is broken, all outstanding and in-progress operations on the Connection will complete with an error status.

6283 Any posting to an Endpoint that is in the IT_EP_STATE_NONOPERATIONAL state will be flushed with completion status set to IT_DTO_ERR_FLUSHED.

6285 If the RMR address range specified with *addr* and *length* exceeds the address range of the underlying LMR, a Completion Error is generated with completion status set to IT_RMR_OPERATION_FAILED.

6288 An `IT_RMR_OPERATION_FAILED` completion status also results in case of an invalid RMR
6289 handle or invalid LMR handle, unless the Implementation handles these errors as immediate
6290 errors.

6291 An `IT_RMR_OPERATION_FAILED` completion status also results if the Protection Zones of
6292 the RMR, underlying LMR, and Endpoint do not match, if the underlying LMR does not use
6293 Absolute Addressing, or if the RMR is a Narrow RMR and not in unlinked state.

6294 For InfiniBand, it is invalid to request remote write access if the memory access privileges of the
6295 underlying LMR do not include local write access. Such an attempt also causes an
6296 `IT_RMR_OPERATION_FAILED` completion status.

6297 Linking an RMR to an LMR that uses Relative Addressing is invalid and results in a Completion
6298 Error with the `IT_DTO_ERR_LOCAL_MM_OPERATION` completion status.

6299 For the InfiniBand Transport, linking a Narrow RMR with a length of zero is invalid and results
6300 in a Completion Error with the `IT_DTO_ERR_LOCAL_MM_OPERATION` completion status.

6301 **APPLICATION USAGE**

6302 The `it_rmr_link` operation is lightweight compared to creating an RMR or an LMR. An
6303 application concerned with efficiency would typically create one or more RMRs at initialization
6304 time that could be linked multiple times to enable remote access to different peers as needed.

6305 The Consumer should use unique identifiers for `cookie` if they desire to identify each DTO. If the
6306 Consumer does not require a unique DTO identifier, the value of zero or NULL can be used.

6307 The local Consumer has several options for ensuring that the remote Consumer does not use
6308 `rmr_context` before the Link Completion Event occurs. One is to wait for the Completion Event
6309 on the Send EVD of the specified Endpoint `ep_handle` before sending the `rmr_context` to a peer.
6310 Another option is to send the `rmr_context` to a peer by posting a DTO to the same Endpoint
6311 `ep_handle` that was used to Link the RMR. The barrier-fencing behavior of `it_rmr_link` ensures
6312 that the DTO does not start until the Link Completion Event has occurred. If the Link fails with
6313 a Completion Error, the Connection will be broken and the DTO flushed, so the `rmr_context` will
6314 not be sent.

6315 For reasons already described, the completion of an RMR Link operation represents an
6316 important change that Consumers may need to monitor. One way to do this is to set
6317 `IT_COMPLETION_FLAG` in `dto_flags`, which will generate a Completion Event to indicate
6318 when the RMR has been linked. Consumers who do not set `IT_COMPLETION_FLAG` must
6319 rely on ordering semantics to infer when the RMR has been successfully linked. For example, if
6320 a subsequent DTO posted to the Send Queue of the same EP completes successfully, then the
6321 Link operation has completed, because DTOs posted to the Send Queue of an EP must wait for a
6322 Link operation to complete before processing. If the Link operation fails, a Link Completion
6323 Event is generated regardless of the use of `IT_COMPLETION_FLAG`.

6324 On the InfiniBand Transport, it is possible to prevent further accesses to a (linked) RMR by
6325 linking it with `privs` set to 0, but it is recommended and preferable to use `it_rmr_unlink` instead.

6326 **FUTURE DIRECTIONS**

6327 Calling `it_rmr_link` on a Wide RMR that is already in the linked state is currently allowed, while
6328 the same operation is not allowed for a linked Narrow RMR. A future version of the IT-API may
6329 require the Consumer to unlink any linked RMR using `it_rmr_unlink` before linking it, regardless
6330 of the transport.

6331 **SEE ALSO**
6332 *it_lmr_create(), it_rmr_create(), it_rmr_unlink(), it_dto_flags_t, it_dto_events, it_addr_mode_t,*
6333 *it_mem_priv_t*

it_rmr_query()

6334

6335 NAME

6336

it_rmr_query – get attributes of a Remote Memory Region

6337 SYNOPSIS

6338

```
#include <it_api.h>
```

6339

```
it_status_t it_rmr_query(  
    IN    it_rmr_handle_t    rmr_handle,  
    IN    it_rmr_param_mask_t mask,  
    OUT   it_rmr_param_t     *params  
);
```

6344

6345

```
typedef enum {  
    IT_RMR_PARAM_ALL           = 0x000001,  
    IT_RMR_PARAM_IA           = 0x000002,  
    IT_RMR_PARAM_PZ           = 0x000004,  
    IT_RMR_PARAM_LINKED       = 0x000008,  
    IT_RMR_PARAM_LMR          = 0x000010,  
    IT_RMR_PARAM_ADDR          = 0x000020,  
    IT_RMR_PARAM_LENGTH        = 0x000040,  
    IT_RMR_PARAM_MEM_PRIV      = 0x000080,  
    IT_RMR_PARAM_RMR_CONTEXT    = 0x000100,  
    IT_RMR_PARAM_TYPE          = 0x000200,  
    IT_RMR_PARAM_ADDR_MODE     = 0x000400  
} it_rmr_param_mask_t;
```

6358

6359

```
typedef struct {  
    it_ia_handle_t    ia;           /* IT_RMR_PARAM_IA */  
    it_pz_handle_t    pz;           /* IT_RMR_PARAM_PZ */  
    it_boolean_t      linked;       /* IT_RMR_PARAM_LINKED */  
    it_lmr_handle_t    lmr;         /* IT_RMR_PARAM_LMR */  
    void *             addr;         /* IT_RMR_PARAM_ADDR */  
    it_length_t        length;      /* IT_RMR_PARAM_LENGTH */  
    it_mem_priv_t      privs;        /* IT_RMR_PARAM_MEM_PRIV */  
    it_rmr_context_t   rmr_context; /* IT_RMR_PARAM_RMR_CONTEXT */  
    it_rmr_type_t      type;         /* IT_RMR_PARAM_TYPE */  
    it_addr_mode_t     addr_mode;    /* IT_RMR_PARAM_ADDR_MODE */  
} it_rmr_param_t;
```

6372

APPLICABILITY

6373

Remote Memory Regions can be used only for the RC service type. See [it_rmr_create](#).

6374 DESCRIPTION

6375

rmr_handle Remote Memory Region.

6376

mask Bitwise OR of flags for desired parameters.

6377

params Structure whose members are written with the desired parameters.

6378

The *it_rmr_query* routine returns the desired attributes of the Remote Memory Region

6379

rmr_handle in the structure pointed to by *params*. The *mask* argument specifies which fields of

6380 *params* are returned, and the values returned in other fields are undefined. See the Synopsis for
6381 the correspondence between *mask* values and fields. The *mask* value `IT_RMR_PARAM_ALL`
6382 causes all fields to be returned.

6383 The definition of each field of *params* follows, some of which depend on whether the RMR is
6384 currently linked to an LMR as a result of using *it_rmr_link*:

6385	<i>ia</i>	The Interface Adapter Handle specified to create the RMR.
6386	<i>pz</i>	The Protection Zone Handle specified to create the RMR.
6387	<i>linked</i>	<code>IT_TRUE</code> if the RMR is currently linked; otherwise, <code>IT_FALSE</code> .
6388	<i>lmr</i>	The Local Memory Region to which the RMR is currently linked. Undefined if 6389 RMR is not linked.
6390	<i>addr</i>	The currently linked starting address or, equivalently, Base Address of the 6391 RMR. To be interpreted as an absolute address, regardless of <i>addr_mode</i> . 6392 Undefined if RMR is not linked.
6393	<i>length</i>	The currently linked length in bytes of the RMR. Undefined if RMR is not 6394 linked.
6395	<i>privs</i>	The currently linked memory access privileges of the RMR. Undefined if RMR 6396 is not linked. For the definition of bits, see <i>it_mem_priv_t</i> .
6397	<i>rmr_context</i>	The currently linked RMR Context associated with the RMR. Undefined if 6398 RMR is not linked. Returned in network byte order.
6399	<i>type</i>	The type of the RMR, either <code>IT_RMR_TYPE_NARROW</code> or 6400 <code>IT_RMR_TYPE_WIDE</code> .
6401	<i>addr_mode</i>	The addressing mode of the RMR. Undefined if RMR is not linked.

6402 If the Consumer calls *it_rmr_query* after posting an RMR Link or RMR Unlink operation, and
6403 before dequeuing the Completion Event of such an operation, then the returned *linked*, *lmr*,
6404 *addr*, *length*, *privs*, *rmr_context*, and *addr_mode* fields may represent the RMR state as it was
6405 prior to posting, or a new RMR state. The Consumer should not rely on the value of these fields
6406 during this time.

6407 **RETURN VALUE**

6408 A successful call returns `IT_SUCCESS`. Otherwise, an error code is returned as described below:

6409	<code>IT_ERR_INVALID_MASK</code>	The <i>mask</i> contained invalid flag values.
6410	<code>IT_ERR_INVALID_RMR</code>	The Remote Memory Region Handle (<i>rmr_handle</i>) was invalid.
6411	<code>IT_ERR_IA_CATASTROPHE</code>	The IA has experienced a catastrophic error and is in the disabled 6412 state. None of the output parameters from this routine are valid. 6413 See <i>it_ia_info_t</i> for a description of the disabled state.

6414 **SEE ALSO**

6415 *it_rmr_create()*, *it_rmr_free()*, *it_rmr_link()*, *it_rmr_context_t*, *it_addr_mode_t*, *it_mem_priv_t*,
6416 *it_rmr_type_t*

it_rmr_unlink()

6417

6418 NAME

6419 `it_rmr_unlink` – post operation to Unlink a Remote Memory Region from its memory range

6420 SYNOPSIS

```
6421 #include <it_api.h>
6422
6423 it_status_t it_rmr_unlink(
6424     IN it_rmr_handle_t rmr_handle,
6425     IN it_ep_handle_t ep_handle,
6426     IN it_dto_cookie_t cookie,
6427     IN it_dto_flags_t dto_flags
6428 );
```

6429 APPLICABILITY

6430 `it_rmr_unlink` is applicable only to Endpoints created for the RC service type.

6431 DESCRIPTION

6432 *rmr_handle* Handle of RMR that will be unlinked.

6433 *ep_handle* Endpoint on which to post the operation.

6434 *cookie* Consumer-provided cookie that is returned to the Consumer in the
6435 Completion Event corresponding to the operation.

6436 *dto_flags* Bitwise OR of options for operation handling.

6437 The `it_rmr_unlink` routine posts to Endpoint *ep_handle* an Unlink operation to Unlink the
6438 Remote Memory Region specified by *rmr_handle*.

6439 An RMR can be unlinked only if the Endpoint *ep_handle* is in `IT_EP_STATE_CONNECTED`
6440 state.

6441 For any RMR type, the Protection Zones of the RMR and Endpoint must match. For a linked
6442 Narrow RMR, the Endpoint *ep_handle* to which the Unlink operation is posted must match the
6443 Endpoint through which the RMR was linked (see [it_rmr_type_t](#)).

6444 An RMR can be unlinked regardless of whether it is currently linked or unlinked.

6445 The *cookie* ([it_dto_cookie_t](#)) allows the Consumer to associate an identifier with each Work
6446 Request. This identifier is completely under Consumer control and opaque to the
6447 Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted
6448 Work Request.

6449 Request handling is specified by the *dto_flags* argument as the bitwise OR of zero or more of the
6450 following flags:

6451 `IT_COMPLETION_FLAG`

6452 `IT_NOTIFY_FLAG`

6453 `IT_BARRIER_FENCE_FLAG`

6454 For the definition of these flags, see [it_dto_flags_t](#).

6455 The completion of the posted Unlink operation is reported asynchronously to the Consumer
6456 according to the rules defined in *it_dto_flags_t*. An RMR Unlink Completion Event is of type
6457 *it_dto_cmpl_event_t*. Any generated RMR Unlink Completion Event manifests on the EVD
6458 associated with the Endpoint Send Queue. The Event type is IT_RMR_LINK_CMPL_EVENT.

6459 After a successful Unlink Completion Event, any previous linking for the RMR is invalidated,
6460 and the RMR Context for the RMR is no longer defined. Any RDMA operation that uses the
6461 previous RMR Context will fail with a protection violation; beware that this may include
6462 operations that are outstanding when *it_rmr_unlink* is called. The Consumer must ensure that
6463 such operations have completed prior to calling *it_rmr_unlink* if successful completions are
6464 desired. An Unlink operation will never be partially successful over a subset of the requested
6465 memory range; it either succeeds completely or fails without invalidating any portion of the
6466 previous linking.

6467 If *it_rmr_unlink* returns successfully but the Completion Event status indicates failure, then any
6468 previous linking and RMR Context remains valid.

6469 Any Work Request posted to an Endpoint's Send Queue after a call to *it_rmr_unlink* will not
6470 begin execution until the Unlink operation has completed.

6471 RETURN VALUE

6472 A successful call returns IT_SUCCESS. Otherwise, an immediate error is returned and, if the
6473 RMR was previously linked the previous linking of the RMR remains valid. It is possible for
6474 *it_rmr_unlink* to return success but for the Completion Event to indicate failure.

6475 Posting to an Endpoint that is not in the IT_EP_STATE_CONNECTED or
6476 IT_EP_STATE_NONOPERATIONAL state will return the IT_ERR_INVALID_EP_STATE
6477 immediate error.

6478 The possible immediate errors for *it_rmr_unlink* are listed below:

6479	IT_ERR_INVALID_DTO_FLAGS	The Data Transfer Operation flags (<i>dto_flags</i>) value was
6480		invalid.
6481	IT_ERR_INVALID_EP	The Endpoint Handle (<i>ep_handle</i>) was invalid.
6482	IT_ERR_INVALID_EP_STATE	The Endpoint was not in the proper state for the attempted
6483		operation.
6484	IT_ERR_INVALID_EP_TYPE	The attempted operation was invalid for the Service Type of
6485		the Endpoint.
6486	IT_ERR_INVALID_RMR	The Remote Memory Region Handle (<i>rmr_handle</i>) was
6487		invalid.
6488	IT_ERR_TOO_MANY_POSTS	The operation failed due to an overflow of a Work Queue.
6489	IT_ERR_IA_CATASTROPHE	The IA has experienced a catastrophic error and is in the
6490		disabled state. None of the output parameters from this routine
6491		are valid. See <i>it_ia_info_t</i> for a description of the disabled
6492		state.

6493 **ASYNCHRONOUS ERRORS**

6494 For Work Requests posted to an Endpoint in the `IT_EP_STATE_CONNECTED` state, a
6495 completion status (see *it_dto_status_t*) other than `IT_DTO_SUCCESS` will break the Connection
6496 by moving the Endpoint to the `IT_EP_STATE_NONOPERATIONAL` state and deliver an
6497 `IT_CM_MSG_CONN_BROKEN_EVENT` Event to the Connect EVD of *ep_handle*. Once the
6498 Connection is broken, all outstanding and in-progress operations on the Connection will
6499 complete with an error status.

6500 Any posting to an Endpoint that is in the `IT_EP_STATE_NONOPERATIONAL` state will be
6501 flushed with completion status set to `IT_DTO_ERR_FLUSHED`.

6502 An `IT_RMR_OPERATION_FAILED` completion status results in case of an invalid RMR
6503 handle, unless the Implementation handles this error as an immediate error.

6504 An `IT_RMR_OPERATION_FAILED` completion status also results if the Protection Zones of
6505 the RMR and the Endpoint do not match or, for a linked Narrow RMR, if the Endpoint to which
6506 the Unlink operation is posted does not match the Endpoint through which the RMR was linked.

6507 **APPLICATION USAGE**

6508 *it_rmr_unlink* may be used to revoke remote Consumer access to an RMR that was previously
6509 granted. In addition, the Consumer must Unlink all RMRs that refer to an LMR in order to
6510 destroy or modify the LMR. Note that the RMR is not considered unlinked until a successful
6511 Completion Event is generated; thus, the Consumer should dequeue the Completion Event
6512 before calling *it_lmr_free*.

6513 A difficulty can arise if the Endpoint that the Consumer was using to Link the RMR has become
6514 disconnected, because an Unlink operation can only be posted to a connected Endpoint. The
6515 preferred solution is to destroy the RMR by calling *it_rmr_free*. In case of a Wide RMR and for
6516 the InfiniBand Transport only, an alternative is for the Consumer to create a special pair of
6517 Endpoints within the same Protection Zone as the Wide RMR and to connect these Endpoints in
6518 loopback mode to each other; the Wide RMR can then be Unlinked through one of the special
6519 Endpoints.

6520 For reasons already described, the completion of an RMR Unlink operation represents an
6521 important change that Consumers may need to monitor. One way to do this is to set
6522 `IT_COMPLETION_FLAG` in *dto_flags*, which will generate a Completion Event to indicate
6523 when the RMR has been unlinked. Consumers who do not set `IT_COMPLETION_FLAG` must
6524 rely on ordering semantics to infer when the RMR has been successfully unlinked. For example,
6525 if a subsequent DTO posted to the Send Queue of the same EP completes successfully, then the
6526 Link operation has completed, because DTOs posted to the Send Queue of an EP must wait for a
6527 Link operation to complete before processing. If the Unlink operation fails, a Completion Event
6528 is generated regardless of the use of `IT_COMPLETION_FLAG`.

6529 **SEE ALSO**

6530 *it_rmr_create()*, *it_rmr_link()*, *it_dto_flags_t*

it_set_consumer_context()

6531

6532 NAME

6533 `it_set_consumer_context` – associate a Consumer Context with an IT Object Handle

6534 SYNOPSIS

```
6535 #include <it_api.h>
6536
6537 it_status_t it_set_consumer_context(
6538     IN it_handle_t handle,
6539     IN it_context_t context
6540 );
```

6541 DESCRIPTION

6542 *handle* Handle for the IT-API object to be associated with the Consumer Context.

6543 *context* The Consumer Context to be associated with the object Handle.

6544 *it_set_consumer_context* associates a Consumer Context with the specified *handle*. See
6545 [it_handle_t](#) for a description of the valid Handle types.

6546 Only a single Consumer Context is provided for any IT Object Handle. If there is a previous
6547 Consumer Context associated with the specified Handle, the new Context replaces the old one.
6548 The value of Context is opaque to the Implementation. The Consumer can disassociate the
6549 existing Context by providing a NULL value for the Context. The Implementation makes no
6550 attempt to synchronize access to the Context.

6551 RETURN VALUE

6552 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

6553 IT_ERR_INVALID_HANDLE The *handle* was invalid.

6554 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the disabled
6555 state. None of the output parameters from this routine are valid.
6556 See [it_ia_info_t](#) for a description of the disabled state.

6557 EXAMPLES

6558 The following code example demonstrates the use of a cast in the call to
6559 *it_set_consumer_context*. The *lmr* object is cast to the generic *it_handle_t* type for the call.

```
6560 it_lmr_handle_t lmr;
6561 it_context_t cxt = 1234;
6562 it_set_consumer_context( (it_handle_t) lmr, cxt);
```

6563 SEE ALSO

6564 [it_get_consumer_context\(\)](#), [it_context_t](#), [it_handle_t](#)

it_socket_convert()

6565
6566 **NAME**
6567 `it_socket_convert` – convert a connected socket into a connected IT-API Endpoint

6568 SYNOPSIS

```
6569 #include <it_api.h>
6570
6571 it_status_t it_socket_convert(
6572     IN      int      sd,
6573     IN      it_ep_handle_t ep_handle,
6574     IN      it_sc_flags_t flags,
6575     IN const unsigned char* msg,
6576     IN      size_t   len
6577 );
6578
6579 typedef enum {
6580     IT_SC_DEFAULT      = 0x0000,
6581     IT_SC_NO_REQ_REP   = 0x0001,
6582 } it_sc_flags_t;
```

6583 APPLICABILITY

6584 `it_socket_convert` is supported only if the Interface Adapter attribute `socket_conversion_support`
6585 (see `it_ia_info_t`) is `IT_TRUE`.

6586 DESCRIPTION

6587 `sd` Socket descriptor.

6588 `ep_handle` Handle for an instance of the local Endpoint.

6589 `flags` Flags for the Conversion attempt.

6590

Features	Name	Bit Value	Description
Default	IT_SC_DEFAULT	0x0000	MPA startup enabled
MPA startup	IT_SC_NO_REQ_REP	0x0001	Suppress MPA startup processing by IT-API implementation

6591

6592 `msg` Last ULP streaming mode message for delivery to the remote responding
6593 Consumer. Must be NULL when the Consumer is in the Conversion
6594 Responder role.

6595 `len` Length of `msg`. This field must be zero when the Consumer is in the
6596 Conversion Responder role.

6597 The `it_socket_convert` routine converts a connected TCP/IP socket (`sd`) into a connected IT-API
6598 Endpoint (`ep_handle`).

6599 Calling *it_socket_convert* with an *sd* parameter not corresponding to a TCP connection or with
6600 an *ep_handle* corresponding to an Endpoint created on an IA where *socket_conversion_support*
6601 is *IT_FALSE* shall yield the return code *IT_ERR_OP_NOT_SUPPORTED*.

6602 This routine is for use by the Consumer both when initiating and when responding to a
6603 Conversion attempt.

6604 The Consumer must provide a connected socket, *sd*, to the *it_socket_convert* routine. Prior to
6605 calling the *it_socket_convert* routine, the Consumer must quiesce all traffic in both directions on
6606 the connection. How the Consumer quiesces the connection is outside the scope of this
6607 specification. Failure to quiesce the connection prior to calling this routine may result in
6608 breaking the TCP/IP connection.

6609 The connection parameter, *sd*, is a socket descriptor as returned by the *socket()* API routine.

6610 After this call returns with *IT_SUCCESS* and before the Conversion process completes, the *sd*
6611 must not be used by the Consumer. Use of the *sd* prior to the Conversion process completing
6612 may abort the Conversion process and cause the generation of a Completion Error. After the
6613 Conversion process has completed, the Consumer can only legally use the *sd* in two other system
6614 calls: they can pass it to the *setsockopt(2)* system call to modify only the *SO_KEEPALIVE*
6615 option, or they pass it to *close(2)* to close it. Support for doing anything else with the *sd* is
6616 Implementation-specific. Support for modifying the *SO_KEEPALIVE* option on a converted
6617 socket is optional; if the Implementation doesn't support this, *setsockopt(2)* will return an error
6618 when the Consumer attempts to do so, but no undefined behavior will result. If the Consumer
6619 attempts to pass *sd* to any other system call besides the two previous specified, or attempts to use
6620 *setsockopt(2)* to modify any option other than *SO_KEEPALIVE*, undefined behavior up to and
6621 including unreported data corruption may result. Calling *close(2)* on the *sd* has no effect upon
6622 the Endpoint.

6623 The Consumer must provide an Endpoint, *ep_handle*, of the Reliable Connected service type to
6624 the *it_socket_convert* routine. The Endpoint must be in the *IT_EP_STATE_UNCONNECTED*
6625 state when input to the routine.

6626 The *flags* parameter for this routine allows the Consumer to control use of underlying features of
6627 the iWARP Transport. Unless stated otherwise, the bitwise OR of any combination of *flags*
6628 values is allowed.

6629 The *flags* value *IT_SC_DEFAULT* causes the underlying implementation to enable MPA
6630 startup.

6631 Setting the bit *IT_SC_NO_REQ_REP* in *flags* allows the Consumer to specify that the
6632 underlying Implementation not use the IETF MPA startup. See [\[MPA-RDMAC\]](#) for more
6633 details.

6634 To initiate the Conversion process, the Consumer must provide a Last ULP Streaming-Mode
6635 Message buffer *msg* of length *len* greater than zero to the *it_socket_convert* routine. The Last
6636 ULP Streaming-Mode Message will be conveyed to the remote Consumer via the underlying
6637 connection. The Consumer is required to supply a Last ULP Streaming-Mode Message to initiate
6638 the Conversion process regardless of whether or not the IETF MPA Startup is used.

6639 Prior to initiating the Conversion process by calling *it_socket_convert* with a non-NULL *msg*
6640 buffer, the Consumer should post at least one Receive DTO to the Endpoint *ep_handle* passed to
6641 the *it_socket_convert* routine. Failure to do so may result in the Endpoint failing to reach the
6642 *IT_EP_STATE_CONNECTED* state.

6643 To respond to an incoming Conversion attempt by a remote Consumer, the Consumer must
6644 specify the *msg* parameter as NULL and specify the length, *len*, parameter as zero. The
6645 Consumer should not call the *it_socket_convert* routine in responding mode prior to consuming
6646 the Last ULP Streaming-Mode Message sent by the remote initiating Consumer (e.g., the local
6647 Consumer should *read()* from the *sd* to consume the remote Consumer's Last ULP Streaming-
6648 Mode Message). The Consumer should receive the peer's Last ULP Streaming-Mode Message
6649 completely so that the *sd* is quiesced prior to calling *it_socket_convert*. How the Consumer
6650 determines the size of the peer's Last ULP Streaming-Mode Message on the responding side is
6651 outside of the scope of this specification. Failure to consume the peer's Last ULP Streaming-
6652 Mode Message completely and thereby to quiesce *sd* before calling *it_socket_convert* may result
6653 in the TCP/IP connection breaking. The Consumer also must not attempt to send any further
6654 streaming mode messages on the *sd* prior to calling *it_socket_convert* as the Conversion
6655 Responder.

6656 How a ULP chooses which side of the connection initiates Conversion and which side responds
6657 to Conversion is outside of the scope of this specification.

6658 In order to safely post RDMA Read operations to an Endpoint, the Consumer must ensure that
6659 the Endpoint at the other end of the converted Connection has compatible *rdma_read_ord*
6660 (ORD) and *rdma_read_ird* (IRD) attributes. See *it_ep_attributes_t* for a description of these
6661 attributes. See Chapter 5 for more details on use of IRD and ORD.

6662 The *it_socket_convert* routine returns control to the Consumer prior to completion of the
6663 Conversion operation. The completion of the Conversion operation is reported asynchronously
6664 via the *connect_sevd_handle* parameter furnished to the *it_ep_rc_create* routine on Endpoint
6665 creation. Successful completion of the Conversion process is indicated by an
6666 IT_CM_MSG_CONN_ESTABLISHED_EVENT queued on the IT_CM_MSG_EVENT_
6667 STREAM SEVD represented by *connect_sevd_handle*. The Consumer may distinguish the
6668 Event corresponding to their Conversion attempt by comparing the *it_ep_handle_t* returned in
6669 the event with that of the Endpoint, *ep_handle*, passed into the *it_socket_convert* routine.

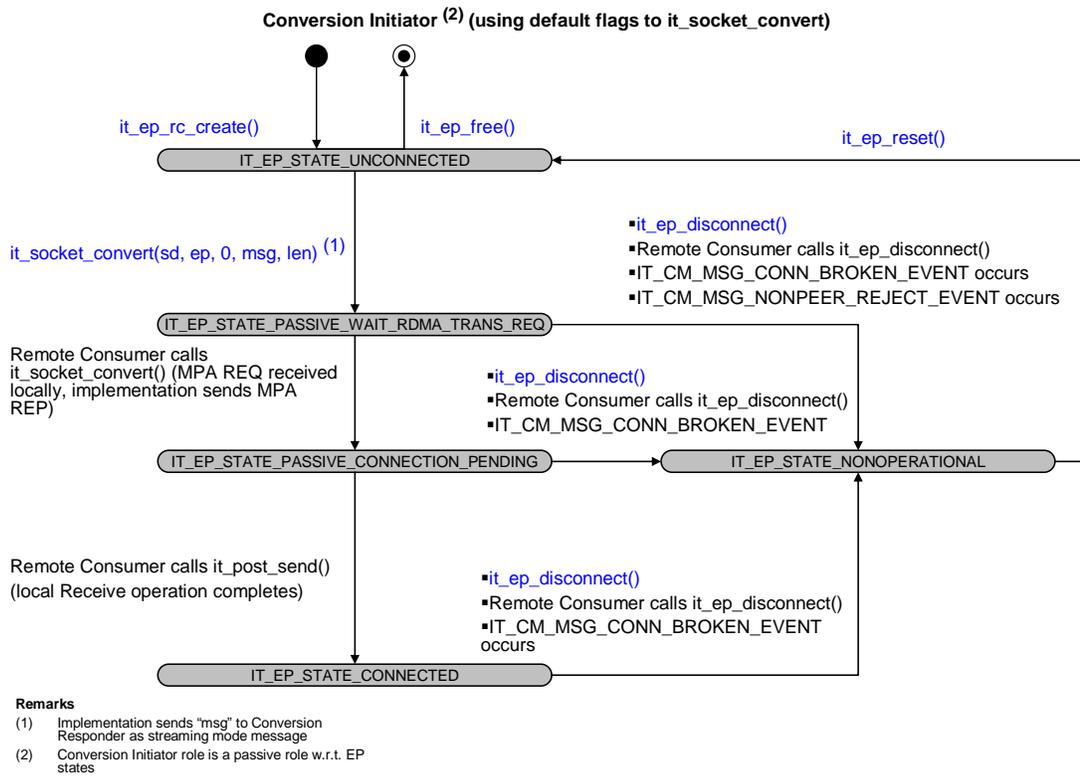
6670 For a complete definition of Endpoint state and a more complete description of the state
6671 transitions, see the Extended Description below. If for any reason an Endpoint Connection fails
6672 to be established the Endpoint will transition into the IT_EP_STATE_NONOPERATIONAL
6673 state and any Receive DTO operations that were successfully posted to the Endpoint will be
6674 completed with an IT_DTO_ERR_FLUSHED status.

6675 The Conversion process may not be completed for the Conversion Initiating Consumer unless
6676 the Conversion Responding Consumer posts a Send message via *it_post_send* to the remote
6677 Endpoint that is connected to, *ep_handle*. The Consumer may abort a Conversion attempt in
6678 progress by making use of the *it_ep_disconnect* routine or the *it_ep_free* routine.

6679 **EXTENDED DESCRIPTION**

6680 The Endpoint state transitions due to the Conversion process depend on the use of the *flags*
6681 parameter.

6682 Setting *flags* to the value IT_SC_DEFAULT yields the following state diagrams for the
6683 initiating and responding sides:

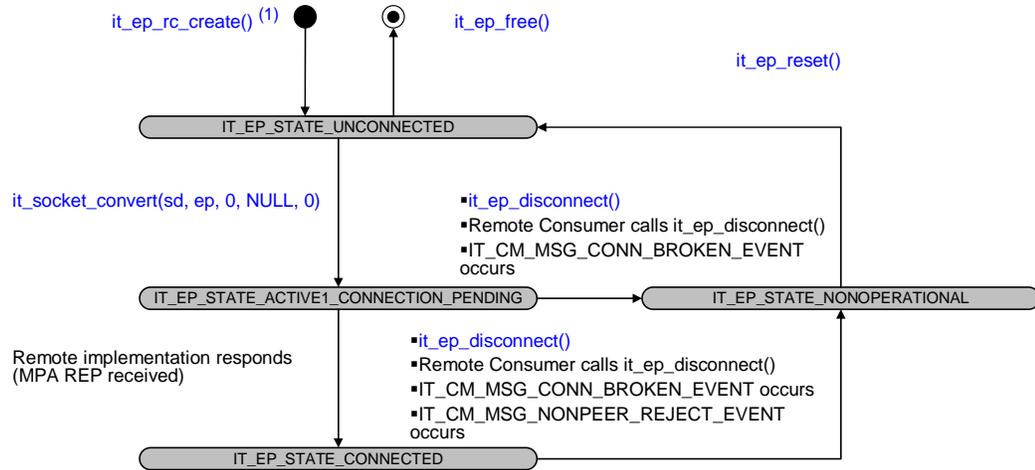


6684

6685

Figure 4: Conversion Initiator with *flags* set to IT_SC_DEFAULT

Conversion Responder ⁽²⁾ (using default flags to `it_socket_convert`)



Remarks

- (1) Consumer enters this state machine after consuming ULP streaming mode message from Conversion Initiator (i.e. the "msg" passed into `it_socket_convert()` by the Conversion Initiator)
- (2) Conversion Responder role is the active role w.r.t. EP states

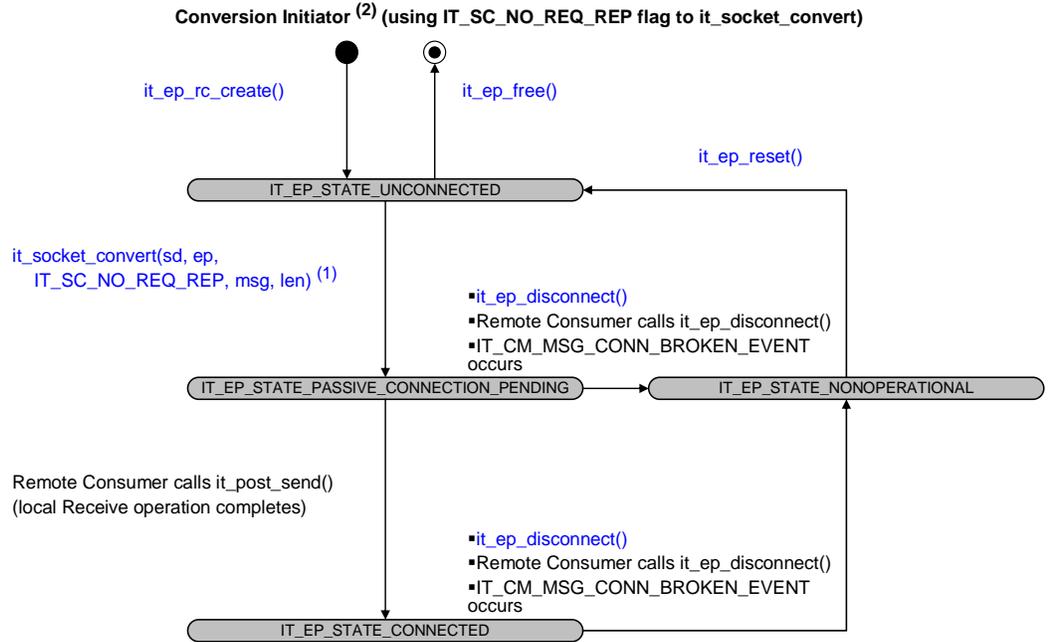
6686

6687

Figure 5: Conversion Responder with *flags* set to `IT_SC_DEFAULT`

6688
6689

Setting the `IT_SC_NO_REQ_REP` bit in `flags` yields the following state diagrams for the initiating and responding sides:



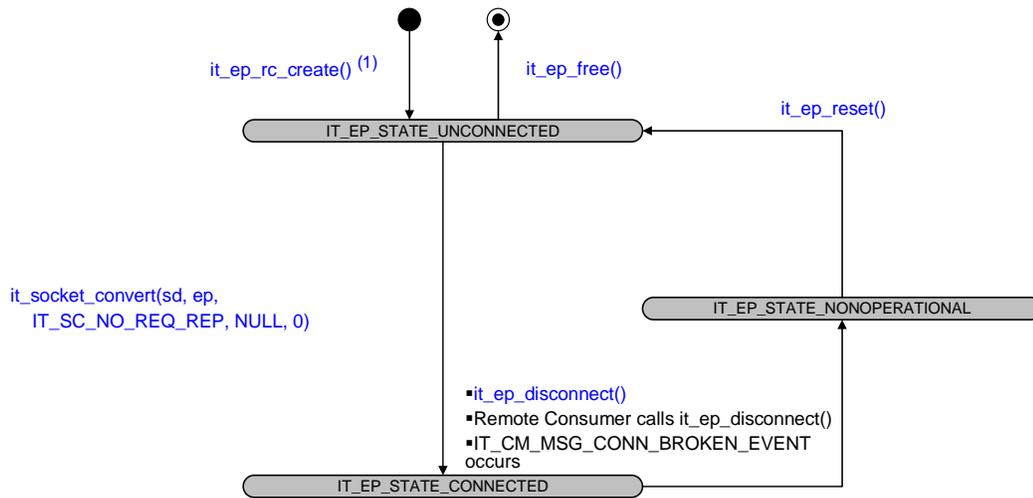
Remarks

- (1) Implementation sends "msg" to Conversion Responder as streaming mode message
- (2) Conversion Initiator role is a passive role w.r.t. EP states

6690
6691

Figure 6: Conversion Initiator with the `IT_SC_NO_REQ_REP` bit set in `flags`

Conversion Responder ⁽²⁾ (using IT_SC_NO_REQ_REP flag to it_socket_convert)



Remarks

- (1) Consumer enters this state machine after consuming ULP streaming mode message from Conversion Initiator (i.e. the "msg" passed into it_socket_convert() by the Conversion Initiator)
- (2) Conversion Responder role is the active role w.r.t. EP states

6692

6693

Figure 7: Conversion Responder with the IT_SC_NO_REQ_REP bit set in flags

6694

6695

6696

6697

6698

6699

The Description section refers to exceptions to the Connection establishment process. Provision appears in the IT-API to support a slightly modified Connection establishment procedure. If the *it_ia_info_t* attribute, *it_extended_iwarp_qp_states*, is reported as IT_TRUE, then the requirement on the Conversion Initiator to post a Receive DTO to the EP before initiating Conversion is relaxed. In this case, any IT-API DTO from the Conversion Responder side is sufficient to generate a Connection event on the Conversion Initiator side.

6700

6701

The *it_socket_convert* call has no provision to convey Private Data. The Consumer must implement their own ULP to exchange data prior or after Socket Conversion.

6702

6703

6704

6705

No explicit provision to reject a Conversion attempt is provided in the IT-API. The Consumer may use *it_ep_disconnect* to abort an ongoing Conversion attempt on the Conversion Initiating side or may close the LLP (*sd*) on the Conversion Responding side instead of calling *it_socket_convert*).

6706

RETURN VALUE

6707

A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

6708

IT_ERR_OP_NOT_SUPPORTED

The operation failed as it is not supported on the IA.

6709

IT_ERR_RESOURCES

The operation failed due to resource limitations.

6710

IT_ERR_INVALID_SD

The socket descriptor *sd* was invalid.

6711	IT_ERR_INVALID_SD_STATE	The socket descriptor <i>sd</i> is not in the proper state to be converted.
6712		
6713	IT_ERR_INVALID_EP	The <i>ep_handle</i> was invalid.
6714	IT_ERR_INVALID_EP_STATE	The Endpoint is not in the proper state to be connected.
6715	IT_ERR_INVALID_EP_TYPE	The Endpoint Service Type does not support this operation.
6716		
6717	IT_ERR_INVALID_FLAGS	The <i>flags</i> values was invalid.
6718	IT_ERR_IA_CATASTROPHE	The IA has experienced a catastrophic error and is in the disabled state. None of the output parameters from this routine are valid. See <i>it_ia_info_t</i> for a description of the disabled state.
6719		
6720		
6721		

6722 ASYNCHRONOUS ERRORS

6723 If the Conversion Initiator uses a Non-permissive AV-RNIC/IETF and the Conversion
6724 Responder uses an RDMAC AV-RNIC, then an IT_CM_MSG_NONPEER_REJECT_EVENT
6725 with the IT_CN_REJ_BAD_CONN_PARMS reject reason code indicating incompatible
6726 protocol versions will be generated at both Conversion Initiator and Conversion Responder. See
6727 [\[INTEROP-IETF\]](#).

6728 Conversely, if the Conversion Initiator uses an RDMAC AV-RNIC and the Conversion
6729 Responder uses a Non-permissive AV-RNIC/IETF, then an IT_CM_MSG_CONN_BROKEN_
6730 EVENT will be generated at the Conversion Initiator and an IT_CM_MSG_NONPEER_
6731 REJECT_EVENT with an IT_CN_REJ_BAD_CONN_PARMS reject reason code indicating
6732 incompatible protocol versions will be generated at the Conversion Responder. See [\[INTEROP-
6733 IETF\]](#).

6734 APPLICATION USAGE

6735 The *it_socket_convert* call may be used for implementing iSER and SDP ULPs.

6736 It is the Consumer's responsibility to monitor forward progress of the Conversion process. The
6737 *it_socket_convert* call will never time out waiting for a reply. The *it_socket_convert* call may
6738 abort if the LLP closes due to an error. Consumers are advised not to perform any operations on
6739 *sd* while the Conversion of *sd* is in progress. Consumers may *close(2)* the *sd* after Conversion,
6740 which indicates only that resources associated with *sd* are no longer needed and has no effect on
6741 the Endpoint provided to *it_socket_convert*.

6742 The iWARP transport currently does not provide a standardized, Endpoint-based interface for
6743 manipulating LLP options after Socket Conversion, such as an iWARP equivalent to the
6744 *setsockopt(2)* system call. Some iWARP Interface Adapters may allow setting a limited set of
6745 LLP options (such as the TCP SO_KEEPALIVE socket option) after a completed Conversion by
6746 *accepting setsockopt(2)* on the *sd* that was provided to *it_socket_convert*. Consumers having a
6747 need to set LLP options are therefore advised to either use *setsockopt(2)* prior to Conversion or
6748 to be prepared for handling failures of *setsockopt(2)* calls if invoked after a completed
6749 Conversion.

6750 SEE ALSO

6751 [it_ep_rc_create\(\)](#), [it_ep_modify\(\)](#), [it_ep_free\(\)](#), [it_ep_disconnect\(\)](#), [it_ia_info_t](#), [it_post_send\(\)](#),
6752 [it_post_recv\(\)](#), [it_post_rdma_write\(\)](#), [it_post_rdma_read\(\)](#)

it_srq_create()

6753
6754 **NAME**
6755 `it_srq_create` – create a Shared Receive Queue (S-RQ) on an Interface Adapter

6756 **SYNOPSIS**
6757

```
#include <it_api.h>
```


6758
6759

```
it_status_t it_srq_create(  
6760     IN  it_pz_handle_t    pz_handle,  
6761     IN  size_t            max_recv_segments,  
6762     IN  size_t            max_recv_dtos,  
6763     OUT it_srq_handle_t  *srq_handle  
6764 );
```

6765 **APPLICABILITY**
6766 `it_srq_create` is applicable only to Endpoints created for the RC service type and is supported
6767 only if the Interface Adapter attribute `srq_support` (see `it_ia_info_t`) is `IT_TRUE`. Endpoints of
6768 the UD service type cannot use an S-RQ.

6769 DESCRIPTION

6770 `pz_handle` Protection Zone in which to create the Shared Receive Queue.

6771 `max_recv_segments` Maximum number of data segments for a local buffer that the Consumer
6772 may specify for a posted Receive DTO.

6773 `max_recv_dtos` Maximum number of outstanding Receive DTOs that the Consumer can
6774 have outstanding on the S-RQ.

6775 `srq_handle` Returned Handle for the Shared Receive Queue.

6776 `it_srq_create` creates a Shared Receive Queue on the Interface Adapter implicitly identified by
6777 the input `pz_handle`. The `srq_handle` output is only valid if the call to `it_srq_create` returns
6778 `IT_SUCCESS`. An S-RQ is an IT Object to which Receive DTOs can be posted using the
6779 `it_post_recv` routine. By default when a Reliable Connection Endpoint is created (via the
6780 `it_ep_rc_create` routine) it has its own private Receive Queue, but the Consumer can override
6781 the default and instead specify that the Endpoint should use a Shared Receive Queue that the
6782 Consumer created using the `it_srq_create` routine.

6783 The Protection Zone `pz_handle` allows the Consumer to control what local memory an incoming
6784 Send operation can access. An incoming Send DTO that is paired with a Receive DTO posted to
6785 an S-RQ can only access memory that is in the same Protection Zone as the S-RQ. In contrast,
6786 an incoming RDMA Read or RDMA Write DTO targeted at an Endpoint that is using an S-RQ
6787 can only access memory that is in the same Protection Zone as the Endpoint. The Protection
6788 Zone of the Endpoint and the Protection Zone of the S-RQ that the Endpoint is using are allowed
6789 to be different. See the Application Usage section below for an example of how this could be
6790 used.

6791 When an Endpoint is using an S-RQ and an incoming Send operation targets that Endpoint, the
6792 Implementation shall attempt to dequeue a Receive DTO from the S-RQ and provide it to the
6793 targeted Endpoint for further processing. The semantics associated with the processing of the

6794 Receive operation are described in the *it_post_recv* routine. If an incoming Send must be
6795 processed and no Receive DTO is available to be dequeued in the S-RQ, or if an error occurs
6796 while processing the Receive DTO (e.g., the Send DTO it gets paired with contains more data
6797 than the Receive DTO can handle) the Connection of the targeted Endpoint shall be broken and
6798 the Endpoint moved to the IT_EP_STATE_NONOPERATIONAL state, but other Endpoints
6799 that are using the same S-RQ shall not be affected. Endpoints that use a common S-RQ maintain
6800 the same isolation from errors on other Endpoints that they would have if they weren't using an
6801 S-RQ.

6802 When an S-RQ is first created, the S-RQ Low Watermark mechanism is disabled. It can only be
6803 enabled by calling *it_srq_modify* to explicitly enable it.

6804 If the call to *it_srq_create* returns IT_SUCCESS the total number of segments allowed in a
6805 Receive operation posted to the S-RQ will be greater than or equal to the *max_recv_segments*
6806 that were requested. To find out exactly how many segments will be allowed, use *it_srq_query*.
6807 Similarly, if the call to *it_srq_create* returns IT_SUCCESS the total number of Receive DTOs
6808 that can be outstanding for the S-RQ will be greater than or equal to the *max_recv_dtos* that
6809 were requested. To find out exactly how many Receive operations can safely be outstanding, use
6810 *it_srq_query*. (If the Consumer attempts to have more Receive DTOs outstanding on an S-RQ
6811 than is given by *it_srq_query*, the Implementation may refuse to allow more Receive DTOs to be
6812 posted; see *it_post_recv* for details.)

6813 RETURN VALUE

6814 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

6815 IT_ERR_INVALID_PZ The Protection Zone Handle (*pz_handle*) was invalid.

6816 IT_ERR_SRQ_NOT_SUPPORTED The IA associated with *pz_handle* does not support
6817 Shared Receive Queues.

6818 IT_ERR_RESOURCES The requested operation failed due to insufficient
6819 resources.

6820 IT_ERR_RESOURCE_RECV.DTO The underlying transport could not allocated the
6821 requested *max_recv_dtos* resources at this time.

6822 IT_ERR_RESOURCE_RSEG The underlying transport could not allocated the
6823 requested *max_recv_segments* resources at this time.

6824 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the
6825 disabled state. None of the output parameters from this
6826 routine are valid. See *it_ia_info_t* for a description of the
6827 disabled state.

6828 APPLICATION USAGE

6829 If a Consumer wishes to use an S-RQ they must first create the S-RQ by calling *it_srq_create*.
6830 They can then furnish the handle for the S-RQ as one of the attributes of the Endpoint when
6831 creating the Endpoint using the *it_ep_rc_create* routine.

6832 Before creating a connection with an Endpoint that is using an S-RQ the Consumer will typically
6833 ensure that the Endpoint that it is connecting will be able to process at least one incoming Send
6834 operation by using the *it_post_recv* routine to post a Receive DTO to the S-RQ.

6835 One example of a situation where it could be useful to allow the Protection Zone associated with
6836 an S-RQ and the Protection Zone associated with an Endpoint that is using the S-RQ to be
6837 different is when multiple clients are performing RDMA Write operations to the same server and
6838 the server wants to prevent an RDMA Write operation from one client from accessing the
6839 RDMA target buffer associated with a different client. Assume that we have two clients C1 and
6840 C2, and that the server processes incoming DTOs from C1 through Endpoint E1, and similarly
6841 processes incoming DTOs from C2 through Endpoint E2. The server can then prevent C1 from
6842 accessing C2's RDMA target buffer (and *vice versa*) by placing the RDMA target buffer for C1
6843 in Protection Zone P1 and assigning P1 to E1, and by placing the RDMA target buffer for C2 in
6844 Protection Zone P2 and assigning P2 to E2. The server also needs to receive incoming Send
6845 operations from both C1 and C2, and may therefore decide to have both E1 and E2 share a single
6846 S-RQ. Since an S-RQ and all Receive buffers used on the S-RQ must have the same Protection
6847 Zone, the PZ associated with the S-RQ will be one of P1 or P2 and thus be different from the PZ
6848 of either E2 or E1, respectively. The PZ associated with the S-RQ could also be a third PZ that is
6849 different from P1 and P2.

6850 **SEE ALSO**

6851 *it_srq_free(), it_srq_query(), it_srq_modify(), it_post_rcv(), it_ep_rc_create()*

6852

it_srq_free()

6853

6854 NAME

6855 `it_srq_free` – destroy a Shared Receive Queue

6856 SYNOPSIS

```
6857 #include <it_api.h>
6858
6859 it_status_t it_srq_free(
6860     IN it_srq_handle_t srq_handle
6861 );
```

6862 APPLICABILITY

6863 `it_srq_free` is applicable only to Endpoints created for the RC service type and is supported only
6864 if the Interface Adapter attribute `srq_support` (see [it_ia_info_t](#)) is IT_TRUE. Endpoints of the
6865 UD service type cannot use an S-RQ.

6866 DESCRIPTION

6867 `srq_handle` Handle of the Shared Receive Queue to be destroyed.

6868 The `it_srq_free` routine destroys the Shared Receive Queue `srq_handle`. On return, the Handle
6869 `srq_handle` may no longer be used. A Shared Receive Queue may not be destroyed if it is still
6870 being used by an Endpoint; an attempt to do so will fail and the S-RQ will not be affected.

6871 If this routine returns success, for each Receive DTO that was outstanding on the S-RQ the
6872 Consumer is guaranteed that either a Completion Event has been enqueued or that no
6873 Completion Event will be enqueued. In addition, the Implementation shall release ownership of
6874 the local buffers associated with all Receive DTOs that were outstanding on the S-RQ back to
6875 the Consumer.

6876 RETURN VALUE

6877 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

6878 IT_ERR_INVALID_SRQ The Shared Receive Queue Handle (`srq_handle`) was
6879 invalid.

6880 IT_ERR_SRQ_BUSY The Shared Receive Queue was still referenced by an
6881 Endpoint.

6882 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the
6883 disabled state. None of the output parameters from this
6884 routine are valid. See [it_ia_info_t](#) for a description of the
6885 disabled state.

6886 SEE ALSO

6887 [it_srq_create](#), [it_srq_query\(\)](#), [it_srq_modify\(\)](#)

6888

it_srq_modify()

6889

6890 NAME

6891 it_srq_modify – modify selected attributes of a Shared Receive Queue

6892 SYNOPSIS

```
6893 #include <it_api.h>
6894
6895 it_status_t it_srq_modify(
6896     IN          it_srq_handle_t      srq_handle,
6897     IN          it_srq_param_mask_t  mask,
6898     IN const    it_srq_param_t      *params
6899 );
```

6900 APPLICABILITY

6901 *it_srq_modify* is applicable only to Endpoints created for the RC service type and is supported
6902 only if the Interface Adapter attribute *srq_support* (see *it_ia_info_t*) is IT_TRUE. Endpoints of
6903 the UD service type cannot use an S-RQ.

6904 DESCRIPTION

6905 *srq_handle* The Shared Receive Queue whose attributes are to be modified.

6906 *mask* Bitwise OR of flags for specified parameters.

6907 *params* Structure whose members contain the new parameter values.

6908 The *it_srq_modify* routine changes selected attributes of the Shared Receive Queue *srq_handle*.
6909 Attributes to be modified are specified by flags in *mask*. New values for the attributes are
6910 specified by the corresponding fields in the structure pointed to by *params*. Fields and their
6911 corresponding flag values are shown in *it_srq_param_t*. Note that attributes represented by
6912 fields of *it_srq_param_t* that are not shown below cannot be modified. The definition of each
6913 field follows:

6914 *max_recv_dtos* The maximum number of Receive DTOs that the Consumer can safely have
6915 outstanding on the S-RQ. This can only be modified if the IA associated
6916 with the S-RQ supports dynamically resizing the S-RQ. See *it_ia_info_t*
6917 for details of how to determine this. If the Consumer attempts to modify this to
6918 a value that is less than the total number of Receive DTOs that are currently
6919 enqueued within the S-RQ, an error shall be returned.

6920 *low_watermark* The S-RQ Low Watermark threshold. If the Consumer attempts to modify
6921 this to a value of zero, an error shall be returned. If the Consumer attempts
6922 to modify this to a value that is greater than *max_recv_dtos*, an error shall
6923 be returned. If the Consumer modifies this to any non-zero value (including
6924 the existing non-zero value if the S-RQ Low Watermark mechanism is
6925 already armed) and *it_srq_modify* returns success, the S-RQ Low
6926 Watermark mechanism shall be armed/rearmed. Once the mechanism is
6927 armed, it shall remain armed until either the S-RQ is destroyed by a call to
6928 *it_srq_free*, or the total number of Receive Work Requests outstanding on
6929 the S-RQ falls below the threshold. If the mechanism is disarmed because
6930 the total number of Receive Work Requests outstanding on the S-RQ falls

6931 below this threshold, an Affiliated Asynchronous Event shall be enqueued
6932 on the Affiliated Asynchronous Event Stream for the IA that the S-RQ is
6933 associated with; see *it_affiliated_event_t* for details. Once the S-RQ Low
6934 Watermark mechanism is disarmed, it can only be rearmed by calling
6935 *it_srq_modify* to establish a new *low_watermark* value.

6936 Modifying the maximum number of Receive DTOs that the S-RQ can safely have outstanding
6937 while there are Endpoints actively using the S-RQ can have an adverse impact upon
6938 performance, and can potentially cause the Connections on Endpoints that are using the S-RQ to
6939 break. The Consumer is therefore advised to only modify this attribute of the S-RQ when data
6940 transfer activity on the Endpoints associated with the S-RQ has been quiesced to minimize the
6941 risk of Connections breaking.

6942 The Consumer is allowed to call *it_srq_modify* to modify the S-RQ Low Watermark for an S-
6943 RQ whose S-RQ Low Watermark mechanism is already armed. The result of doing so is that the
6944 mechanism shall remain armed, but the new S-RQ Low Watermark value shall replace the old
6945 one, subject to the normal bounds checking of the supplied S-RQ Low Watermark value that is
6946 done by the *it_srq_modify* routine.

6947 *it_srq_modify* shall succeed in modifying all the requested attributes atomically; if the attempt to
6948 modify any of the requested attributes generates an error, none of the other attributes supplied to
6949 the call will be applied.

6950 **RETURN VALUE**

6951 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

6952	IT_ERR_INVALID_SRQ	The Shared Receive Queue Handle <i>srq_handle</i> was
6953		invalid.
6954	IT_ERR_INVALID_MASK	The <i>mask</i> contained invalid flag values.
6955	IT_ERR_RESOURCES	The requested operation failed due to insufficient
6956		resources.
6957	IT_ERR_INVALID_SRQ_SIZE	An attempt was made to set the total number of entries in
6958		the S-RQ (<i>max_rcv_dtos</i>) to a value that is less than the
6959		total number of Receive DTOs currently enqueued in the
6960		S-RQ.
6961	IT_ERR_SRQ_LOW_WATERMARK	An attempt was made to either set the S-RQ Low
6962		Watermark (<i>low_watermark</i>) to a value greater than
6963		<i>max_rcv_dtos</i> , or to set the S-RQ Low Watermark to a
6964		value of zero.
6965	IT_ERR_IA_CATASTROPHE	The IA has experienced a catastrophic error and is in the
6966		disabled state. None of the output parameters from this
6967		routine are valid. See <i>it_ia_info_t</i> for a description of the
6968		disabled state.

6969 **SEE ALSO**

6970 *it_srq_create()*, *it_srq_free()*, *it_srq_query()*

6971

it_srq_query()

6972

6973 NAME

6974 `it_srq_query` – get attributes of a Shared Receive Queue

6975 SYNOPSIS

```
6976 #include <it_api.h>
6977
6978 it_status_t it_srq_query(
6979     IN    it_srq_handle_t    srq_handle,
6980     IN    it_srq_param_mask_t mask,
6981     OUT   it_srq_param_t     *params
6982 );
6983
6984 typedef enum {
6985     IT_SRQ_PARAM_ALL           = 0x000001,
6986     IT_SRQ_PARAM_IA           = 0x000002,
6987     IT_SRQ_PARAM_PZ           = 0x000004,
6988     IT_SRQ_PARAM_MAX_RECV_DTO = 0x000008,
6989     IT_SRQ_PARAM_MAX_RECV_SEG = 0x000010,
6990     IT_SRQ_PARAM_LOW_WATERMARK = 0x000020
6991 } it_srq_param_mask_t;
6992
6993 typedef struct {
6994     it_ia_handle_t ia;           /* IT_SRQ_PARAM_IA */
6995     it_pz_handle_t pz;           /* IT_SRQ_PARAM_PZ */
6996     size_t         max_recv_dtos; /* IT_SRQ_PARAM_MAX_RECV_DTO */
6997     size_t         max_recv_segs; /* IT_SRQ_PARAM_MAX_RECV_SEG */
6998     size_t         low_watermark; /* IT_SRQ_PARAM_LOW_WATERMARK */
6999 } it_srq_param_t;
```

7000 APPLICABILITY

7001 `it_srq_query` is applicable only to Endpoints created for the RC service type and is supported
7002 only if the Interface Adapter attribute `srq_support` (see [it_ia_info_t](#)) is `IT_TRUE`. Endpoints of
7003 the UD service type cannot use an S-RQ.

7004 DESCRIPTION

7005 `srq_handle` The Shared Receive Queue whose attributes are being queried.

7006 `mask` Bitwise OR of flags for desired parameters.

7007 `params` Structure whose members are written with the desired parameters.

7008 The `it_srq_query` routine returns the desired attributes of the Shared Receive Queue `srq_handle`
7009 in the structure pointed to by `params`. On return, each field of `params` is only valid if the
7010 corresponding flag as shown in the Synopsis is set in the `mask` argument. The `mask` value
7011 `IT_SRQ_PARAM_ALL` causes all fields to be returned.

7012 The definition of each field of `params` follows:

7013 `ia` The Interface Adapter Handle associated with the S-RQ.

7014	<i>pz</i>	The Protection Zone Handle specified to create the S-RQ.
7015 7016	<i>max_rcv_dtos</i>	The maximum number of Receive DTOs that the Consumer can safely have outstanding on the S-RQ.
7017 7018	<i>max_rcv_segs</i>	The maximum number of data segments for a local buffer that the Consumer may specify for a Receive DTO posted to the S-RQ.
7019 7020 7021	<i>low_watermark</i>	The current value of the S-RQ Low Watermark threshold. If this value is zero, the S-RQ Low Watermark mechanism is either not supported by the IA associated with the S-RQ, or the mechanism is not armed.

7022 **RETURN VALUE**

7023 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

7024	IT_ERR_INVALID_SRQ	The Shared Receive Queue Handle (<i>srq_handle</i>) was invalid.
7025	IT_ERR_INVALID_MASK	The <i>mask</i> contained invalid flag values.
7026 7027 7028	IT_ERR_IA_CATASTROPHE	The IA has experienced a catastrophic error and is in the disabled state. None of the output parameters from this routine are valid. See it_ia_info_t for a description of the disabled state.

7029 **SEE ALSO**

7030 [it_srq_create\(\)](#), [it_srq_free\(\)](#), [it_srq_modify\(\)](#)

it_ud_service_reply()

7031

7032 NAME

7033 it_ud_service_reply – return the information necessary to communicate via Unreliable Datagram
7034 (UD) messages with the entity specified by the Connection Qualifier in the UD Service Request
7035 Event

7036 SYNOPSIS

```
7037 #include <it_api.h>
7038
7039 it_status_t it_ud_service_reply (
7040     IN          it_ud_svc_req_identifrier_t  ud_svc_req_id,
7041     IN          it_ud_svc_req_status_t      status,
7042     IN          it_remote_ep_info_t         ep_info,
7043     IN const    unsigned char               *private_data,
7044     IN          size_t                       private_data_length
7045 );
7046
7047 typedef uint64_t it_ud_svc_req_identifrier_t;
```

7048 APPLICABILITY

7049 *it_ud_service_reply* is applicable only to the UD service type.

7050 DESCRIPTION

7051	<i>ud_svc_req_id</i>	Unique identifier from the IT_CM_REQ_UD_SERVICE_REQUEST_
7052		EVENT Event generated from the UD Service Request that is being
7053		responded to with this invocation of <i>it_ud_service_reply</i> .
7054	<i>status</i>	Status to return in the IT_CM_MSG_UD_SERVICE_REPLY_EVENT
7055		data indicating the outcome of the UD Service Request.
7056	<i>ep_info</i>	End-point information to be used by the UD Service Requester to
7057		communicate with this UD Service.
7058	<i>private_data</i>	Opaque Private Data provided by the Consumer which will be sent as part
7059		of the <i>it_ud_service_reply</i> . If the IA does not support Private Data,
7060		<i>private_data_length</i> must be 0.
7061	<i>private_data_length</i>	Length of the <i>private_data</i> provided by the Consumer. If the IA does not
7062		support Private Data, this field must be 0.

7063 The *it_ud_service_reply* routine will be called by the Consumer to respond to an
7064 IT_CM_REQ_UD_SERVICE_REQUEST_EVENT Event. The IT_CM_REQ_UD_SERVICE_
7065 REQUEST_EVENT Event data (*it_ud_svc_request_event_t*) contains a unique Service Request
7066 Handle, the Connection Qualifier of interest, Source address information, and optional Private
7067 Data. The recipient of the IT_CM_REQ_UD_SERVICE_REQUEST_EVENT Event needs to
7068 respond to the request by calling *it_ud_service_reply*.

7069 The *ud_svc_req_id* is a unique identifier allowing this response to be correlated to the request
7070 being responded to. The *ud_svc_req_id* should be copied from the IT_CM_REQ_UD_
7071 SERVICE_REQUEST_EVENT Event data, *ud_svc_req_id* field. Once *it_ud_service_reply* has

7072 been successfully invoked, the supplied *ud_svc_req_id* is no longer valid. The resources
7073 associated with the *ud_svc_req_id* are released and the *ud_svc_req_id* cannot be re-used.

7074 Valid status codes for the *status* field are defined in *it_ud_svc_req_status_t*. A valid status code
7075 must be provided. IT_UD_REQ_REDIRECTED cannot be supplied as input for this parameter,
7076 even though it may appear in the Event given to the requester. The Implementation is
7077 responsible for redirection, not the Consumer.

7078 The *ep_info* is only used by this routine if the *status* field is set to IT_UD_SVC_EP_
7079 INFO_VALID. See *it_ud_svc_req_status_t* for details.

7080 The IA can be queried via *it_ia_query* to determine whether it supports the transfer of Private
7081 Data. This is indicated by the *private_data_support* field of the *it_ia_info_t* structure. If Private
7082 Data is not supported, *private_data_length* must be 0. The maximum length of *private_data* can
7083 be determined by examining the *ud_rep_private_data_len* member of the *it_ia_info_t* structure.

7084 EXTENDED DESCRIPTION

7085 *it_ud_service_reply* is called by the Consumer in response to receiving an IT_CM_REQ_UD_
7086 SERVICE_REQUEST_EVENT Event. The Consumer chooses how to respond to the Service
7087 Request and makes that choice known via the value of *status* passed into the *it_ud_service_reply*
7088 call. The value of *status* determines whether the Implementation uses the *ep_info* input
7089 parameter. The table below describes the meaning of each *status* value, and whether the
7090 Implementation uses the *ep_info* input parameter when that *status* value is present.

<i>status</i> Value	Implication of the <i>status</i> Value
IT_UD_SVC_EP_INFO_VALID	The supplied <i>ep_info</i> (<i>it_remote_ep_info_t</i>) is valid and can be used by the recipient of the <i>it_ud_service_reply</i> to communicate with this service via UD messages. The Consumer must supply an <i>it_remote_ep_info_t</i> structure containing a valid <i>ud_ep_id</i> and <i>ud_ep_key</i> .
IT_UD_SVC_ID_NOT_SUPPORTED	The service described by the <i>conn_qual</i> (<i>it_conn_qual_t</i>) in the <i>it_ud_svc_request_event_t</i> is not supported by this service. The Implementation does not use the <i>ep_info</i> parameter.
IT_UD_SVC_REQ_REJECTED	Rejects the request for UD Service information. The Implementation does not use the <i>ep_info</i> parameter.
IT_UD_NO_EP_AVAILABLE	The Consumer responding via <i>it_ud_service_reply</i> does not have any Endpoints available for UD communication. The Implementation does not use the <i>ep_info</i> parameter.
IT_UD_REQ_REDIRECTED	The Consumer cannot set this status. This status can only be set by the Implementation.

7091 In order for the Implementation to be able to correctly correlate this *it_ud_service_reply* call
7092 with the request Event being responded to, the Consumer must supply the *ud_svc_req_id* from
7093 the *it_ud_svc_request_event_t* as the *ud_svc_req_id* passed into the *it_ud_service_reply* call.
7094

7095 It is possible to receive duplicate UD Service Requests as a result of the active side retrying an
7096 *it_ud_service_request* operation. It is the Consumer's responsibility to detect and handle
7097 duplicate requests. Requests are uniquely identified by a combination of the *ud_svc_req_id* and

7098 the *source_addr* from the *it_ud_svc_request_event_t* data. This combination can be used to
7099 detect duplicate UD Service Requests.

7100 **RETURN VALUE**

7101 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

7102 IT_ERR_PDATA_NOT_SUPPORTED Private Data was supplied by the Consumer, but this
7103 Interface Adapter does not support Private Data.

7104 IT_ERR_INVALID_PDATA_LENGTH The Interface Adapter supports Private Data, but the
7105 length specified exceeded the Interface Adapter's
7106 capabilities.

7107 IT_ERR_INVALID_UD_SVC_REQ_ID The Unreliable Datagram Service Request ID
7108 (*ud_svc_req_id*) was invalid.

7109 IT_ERR_INVALID_UD_STATUS The Unreliable Datagram Service Request *status* was
7110 invalid.

7111 IT_ERR_IA_CATASTROPHE The Interface Adapter has experienced a catastrophic
7112 error and is in the disabled state. None of the output
7113 parameters from this routine are valid. See *it_ia_info_t*
7114 for a description of the disabled state.

7115 **SEE ALSO**

7116 *it_ia_query()*, *it_ud_service_request()*, *it_ep_attributes_t*, *it_cm_msg_events*, *it_cm_req_events*

7117

7118

it_ud_service_request()

7119 **NAME**

7120 *it_ud_service_request* – request that the recipient of this message return the information
7121 necessary to communicate via Unreliable Datagram (UD) messages to the entity specified by the
7122 UD Service Handle

7123 **SYNOPSIS**

```
7124 #include <it_api.h>  
7125  
7126 it_status_t it_ud_service_request (  
7127     IN it_ud_svc_req_handle_t ud_svc_handle  
7128 );
```

7129 **APPLICABILITY**

7130 *it_ud_service_request* is applicable only to the UD service type.

7131 **DESCRIPTION**

7132 *ud_svc_handle* UD Service Request Handle created by a call to *it_ud_service_request_*
7133 *handle_create*. This Handle uniquely identifies this UD Service Request
7134 operation. The UD Service Request Handle is associated with a specific UD
7135 Service described during the creation of the UD Service Request Handle.

7136 The *it_ud_service_request* routine is called by a Consumer to request a remote entity specified
7137 by the UD Service Handle to return information necessary to communicate via Unreliable
7138 Datagram messages.

7139 The *ud_svc_handle* provides the Consumer with a means of correlating this
7140 *it_ud_service_request* with the IT_CM_MSG_UD_SERVICE_REPLY_EVENT Event that the
7141 Consumer will receive when the remote Endpoint responds to this UD Service Request. See
7142 *it_cm_msg_events*.

7143 Due to the nature of Unreliable Datagrams, even though an invocation of *it_ud_service_request*
7144 returns success, the target of the UD Service Request may not receive it. Therefore, the
7145 Consumer may have to call *it_ud_service_request* multiple times with the same *ud_svc_handle*
7146 before the recipient actually receives the request and is able to reply to it. In addition, if the
7147 Consumer issues multiple requests with the same *ud_svc_handle*, the Consumer may receive
7148 multiple replies. It is up to the Consumer to detect and handle duplicate replies.

7149 The *ud_svc_req_id* (*it_ud_svc_req_identifier_t*) associated with a given *ud_svc_handle* does not
7150 change. Therefore, all retries using a given *ud_svc_handle* will result in the same *ud_svc_req_id*
7151 being presented to the recipient of the IT_CM_REQ_UD_SERVICE_REQUEST_EVENT Event
7152 in the Event data (*it_ud_svc_request_event_t*).

7153 Upon a successful invocation and transmission of the *it_ud_service_request*, once the recipient
7154 of the request replies via *it_ud_service_reply*, the Consumer will receive an
7155 IT_CM_MSG_UD_SERVICE_REPLY_EVENT Event. The IT_CM_MSG_UD_SERVICE_
7156 REPLY_EVENT Event data (*it_ud_svc_reply_event_t*) contains the results of the Service
7157 Request query. The *status* field of the *it_ud_svc_reply_event_t* structure in the
7158 IT_CM_MSG_UD_SERVICE_REPLY_EVENT indicates the state of the information in the
7159 *it_ud_svc_reply_event_t* structure. See *it_cm_msg_events*.

7160 **EXTENDED DESCRIPTION**

7161 The *ud_service_request* call requests information from the remote UD Service. Once that remote
 7162 UD Service responds, an IT_CM_MSG_UD_SERVICE_REPLY Event will be generated. The
 7163 data associated with the Event, *it_ud_svc_reply_event_t*, contains information the Consumer
 7164 needs in order to perform Data Transfer Operations with the remote UD Service. The *status*
 7165 (*it_ud_svc_req_status_t*) field of the *it_ud_svc_reply_event_t* indicates the validity of other
 7166 fields in the structure. The *status* field should be checked by the Consumer prior to making any
 7167 assumptions about the data in the rest of the structure. The table below summarizes the *status*
 7168 values and the implications on the data in the *it_ud_svc_reply_event_t* structure:

<i>status</i> Value	Implication for <i>it_ud_svc_reply_event_t</i> Data
IT_UD_SVC_EP_INFO_VALID	The <i>ep_info</i> (<i>it_remote_ep_info_t</i>) is valid. The <i>ud_ep_id</i> and <i>ud_ep_key</i> from the <i>it_remote_ep_info_t</i> structure, combined with the <i>it_path_t</i> from the <i>ud_svc_handle</i> provides the Consumer with the necessary information to perform Data Transfer Operations with the remote UD Service. All fields except <i>destination_path</i> contain valid data.
IT_UD_SVC_ID_NOT_SUPPORTED	The Service described by the <i>connection_qualifier</i> (<i>it_conn_qual_t</i>) in the <i>ud_svc_handle</i> is not supported on the Spigot to which the <i>it_ud_service_request</i> was sent. All fields except <i>ep_info</i> and <i>destination_path</i> contain valid data.
IT_UD_SVC_REQ_REJECTED	The recipient of the <i>it_ud_service_request</i> rejected the UD Service Request operation. All fields except <i>ep_info</i> and <i>destination_path</i> contain valid data.
IT_UD_NO_EP_AVAILABLE	The recipient of the <i>it_ud_service_request</i> does support the UD Service requested, but is out of Endpoint resources. That is, the remote node does not have any Endpoints that can be used to perform Data Transfer Operations with the UD Consumer. All fields except <i>ep_info</i> and <i>destination_path</i> contain valid data.
IT_UD_REQ_REDIRECTED	The Implementation on the receiving side of the <i>it_ud_service_request</i> has requested that the Consumer redirect the Service Request operation to a new Destination. The <i>destination_path</i> (<i>it_path_t</i>) contains valid data. The <i>destination_path</i> should be used to create a new <i>ud_svc_handle</i> to be used in another call to <i>it_ud_service_request</i> . All fields except <i>ep_info</i> , <i>private_data</i> , and <i>private_data_length</i> contain valid data.

7169 **RETURN VALUE**

7170 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

7171 IT_ERR_INVALID_UD_SVC The Unreliable Datagram Service Handle (*ud_svc_handle*) was
7172 invalid.

7173 IT_ERR_IA_CATASTROPHE The Interface Adapter has experienced a catastrophic error and is
7174 in the disabled state. None of the output parameters from this
7175 routine are valid. See *it_ia_info_t* for a description of the disabled
7176 state.

7177 **SEE ALSO**

7178 *it_ud_service_request_handle_create()*, *it_ud_service_reply()*, *it_cm_msg_events*, *it_path_t*,
7179 *it_ep_attributes_t*

7180

7181 **it_ud_service_request_handle_create()**

7182 **NAME**

7183 `it_ud_service_request_handle_create` – create an Unreliable Datagram (UD) Service Request
7184 Handle

7185 **SYNOPSIS**

```
7186 #include <it_api.h>
7187
7188 it_status_t it_ud_service_request_handle_create (
7189     IN  const  it_conn_qual_t      *conn_qual,
7190     IN      it_evd_handle_t        reply_evd,
7191     IN  const  it_path_t           *destination_path,
7192     IN  const  unsigned char       *private_data,
7193     IN      size_t                 private_data_length,
7194     OUT      it_ud_svc_req_handle_t *ud_svc_handle
7195 );
```

7196 **APPLICABILITY**

7197 `it_ud_service_request_handle_create` is applicable only to the UD service type.

7198 **DESCRIPTION**

7199 `conn_qual` The Connection Qualifier describing the UD Service for which the
7200 Consumer is requesting information.

7201 `reply_evd` The Simple EVD on which the `IT_CM_MSG_UD_SERVICE_`
7202 `REPLY_EVENT` Event will be received. `reply_evd` must be of the
7203 `IT_CM_MSG_EVENT_STREAM` Event Stream Type. See
7204 [it_cm_msg_events](#).

7205 `destination_path` `destination_path` specifies a Path to the Destination of the
7206 `it_ud_service_request` operation.

7207 `private_data` Opaque Private Data provided by the Consumer which will be sent as part
7208 of the `it_ud_service_request`. If the IA does not support Private Data,
7209 `private_data_length` must be 0.

7210 `private_data_length:` Length of the `private_data` provided by the Consumer. If the IA does not
7211 support Private Data, this field must be 0.

7212 `ud_svc_handle` UD Service Request Handle created by this call. This Handle will be used
7213 in a call to `it_ud_service_request`.

7214 The `it_ud_service_request_handle_create` routine is called by the Consumer to create an
7215 Unreliable Datagram Service Request Handle to be used in a call to `it_ud_service_request`.

7216 The `destination_path` can be obtained by calling `it_get_pathinfo`. The `spigot_id` in the `it_path_t`
7217 will be the Spigot Identifier used for this UD Service Request.

7218 The IA can be queried via `it_ia_query` to determine whether it supports the transfer of Private
7219 Data. This is indicated by the `private_data_support` field of the `it_ia_info_t` structure. If Private

7220 Data is not supported, *private_data_length* must be 0. The maximum length of *private_data* can
 7221 be determined by examining the *ud_req_private_data_len* member of the *it_ia_info_t* structure.

7222 The returned *ud_svc_handle* is used to identify the UD Service Request. It provides the
 7223 Consumer with a means of correlating this *it_ud_service_request* with the
 7224 IT_CM_MSG_UD_SERVICE_REPLY_EVENT Event that the Consumer will receive when the
 7225 remote Endpoint responds to this UD Service Request.

7226 The *ud_svc_req_id* (*it_ud_svc_req_identifier_t*) associated with a given *ud_svc_handle* does not
 7227 change. Therefore, all retries using a given *ud_svc_handle* will result in the same *ud_svc_req_id*
 7228 being presented to the recipient of the IT_CM_REQ_UD_SERVICE_REQUEST_EVENT Event
 7229 in the Event data (*it_ud_svc_request_event_t*).

7230 **EXTENDED DESCRIPTION**

7231 The members of the *it_path_t* structure that are pertinent for creating a UD Service Request
 7232 Handle are listed in the table below:

<i>it_path_t</i> Member	Description
<i>spigot_id</i>	Spigot Identifier
<i>ib.partition_key</i>	Partition Key
<i>ib.local_port_lid</i>	Source LID
<i>ib.remote_port_lid</i>	Destination LID
<i>ib.sl</i>	Service Level

7233 **RETURN VALUE**

7234 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

7235 7236	IT_ERR_INVALID_CONN_EVD	The Connection Simple Event Dispatcher Handle was invalid.
7237 7238	IT_ERR_INVALID_EVD_TYPE	The Event Stream Type for the Event Dispatcher was invalid.
7239 7240	IT_ERR_PDATA_NOT_SUPPORTED	Private Data was supplied by the Consumer, but this Interface Adapter does not support Private Data.
7241 7242 7243	IT_ERR_INVALID_PDATA_LENGTH	The Interface Adapter supports Private Data, but the length specified exceeded the Interface Adapter's capabilities.
7244	IT_ERR_INVALID_CONN_QUAL	The Connection Qualifier (<i>conn_qual</i>) was invalid.
7245 7246	IT_ERR_INVALID_SOURCE_PATH	One of the components of the Source portion of the supplied Path was invalid.
7247 7248	IT_ERR_INVALID_SPIGOT	An invalid Spigot ID was specified (<i>spigot_id</i> member of the <i>destination_path</i>).
7249 7250	IT_ERR_RESOURCES	The requested operation failed due to insufficient resources.

7251 IT_ERR_IA_CATASTROPHE The Interface Adapter has experienced a catastrophic
7252 error and is in the disabled state. None of the output
7253 parameters from this routine are valid. See *it_ia_info_t*
7254 for a description of the disabled state.

7255 **APPLICATION USAGE**

7256 The resulting *ud_svc_handle* (*it_ud_svc_req_handle_t*) produced by this call will be used in
7257 calls to *it_ud_service_request* to obtain information describing how to communicate with the
7258 remote UD Service described by *conn_qual* (*it_conn_qual_t*).

7259 The *it_ud_service_request* call requests information from the remote UD Service. Once that
7260 remote UD Service responds, an IT_CM_MSG_UD_SERVICE_REPLY Event will be
7261 generated. The data associated with the Event, *it_ud_svc_reply_event_t*, contains an
7262 *it_remote_ep_info_t* structure and other information. The *ud_ep_id* and *ud_ep_key* from the
7263 *it_remote_ep_info_t*, combined with the information from the *destination_path* (*it_path_t*),
7264 provides the Consumer the necessary information to perform Data Transfer Operations with the
7265 remote UD Service.

7266 Note that the *spigot_id* of the Endpoint that will be used for Data Transfer Operations with the
7267 UD Service being requested must match the *spigot_id* in the *destination_path*.

7268 See *it_ud_service_request* and *it_ud_service_reply* for more information.

7269 **SEE ALSO**

7270 *it_ud_service_request_handle_free()*, *it_ud_request_handle_query()*, *it_ia_query()*,
7271 *it_ud_service_request()*, *it_get_pathinfo()*, *it_path_t*, *it_cm_msg_events*, *it_ep_attributes_t*

7272

7273 **it_ud_service_request_handle_free()**

7274 **NAME**

7275 *it_ud_service_request_handle_free* – free a previously created *it_ud_svc_req_handle_t*

7276 **SYNOPSIS**

```
7277 #include <it_api.h>
7278
7279 it_status_t it_ud_service_request_handle_free (
7280     IN it_ud_svc_req_handle_t ud_svc_handle
7281 );
```

7282 **APPLICABILITY**

7283 *it_ud_service_request_handle_free* is applicable only to the UD service type.

7284 **DESCRIPTION**

7285 *ud_svc_handle* Unreliable Datagram (UD) Service Request Handle previously created by a
7286 call to *it_ud_service_request_handle_create*.

7287 *it_ud_service_request_handle_free* removes an existing UD Service Request Handle and frees
7288 all associated underlying resources. Once *it_ud_service_request_handle_free* returns,
7289 *ud_svc_handle* can no longer be used in UD Service Request operations. In addition, once
7290 *it_ud_service_request_handle_free* returns, any replies to outstanding UD Service Request
7291 operations associated with this *ud_svc_handle* will be silently dropped.

7292 Any IT_CM_MSG_UD_SERVICE_REPLY_EVENT Events associated with this request that
7293 have been enqueued on the Event Dispatcher (EVD) will not be removed. It is the Consumer's
7294 responsibility to dequeue and dispose of them.

7295 **RETURN VALUE**

7296 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

7297 IT_ERR_INVALID_UD_SVC The Unreliable Datagram Service Handle (*ud_svc_handle*) was
7298 invalid.

7299 IT_ERR_IA_CATASTROPHE The Interface Adapter has experienced a catastrophic error and
7300 is in the disabled state. None of the output parameters from this
7301 routine are valid. See *it_ia_info_t* for a description of the
7302 disabled state.

7303 **SEE ALSO**

7304 *it_ud_service_request_handle_create()*, *it_ud_service_request_handle_query()*,
7305 *it_cm_msg_events*

7306

7307

it_ud_service_request_handle_query()

7308 NAME

7309 it_ud_service_request_handle_query – return information about a specified
7310 *it_ud_svc_req_handle_t*

7311 SYNOPSIS

```
7312 #include <it_api.h>
7313
7314 it_status_t it_ud_service_request_handle_query (
7315     IN  it_ud_svc_req_handle_t    ud_svc_handle,
7316     IN  it_ud_svc_req_param_mask_t mask,
7317     OUT it_ud_svc_req_param_t     *ud_svc_handle_info
7318 );
7319
7320 typedef enum {
7321     IT_UD_PARAM_ALL                = 0x00000001,
7322     IT_UD_PARAM_IA_HANDLE          = 0x00000002,
7323     IT_UD_PARAM_REQ_ID             = 0x00000004,
7324     IT_UD_PARAM_REPLY_EVD         = 0x00000008,
7325     IT_UD_PARAM_CONN_QUAL         = 0x00000010,
7326     IT_UD_PARAM_DEST_PATH         = 0x00000020,
7327     IT_UD_PARAM_PRIV_DATA         = 0x00000040,
7328     IT_UD_PARAM_PRIV_DATA_LENGTH = 0x00000080
7329 } it_ud_svc_req_param_mask_t;
7330
7331 /*
7332  * The it_ud_svc_req_param_mask_t value in the comment above
7333  * each attribute in the it_ud_svc_req_param_t structure below
7334  * is the mask value used to select that attribute in a call
7335  * to it_ud_service_request_handle_query.
7336  */
7337 typedef struct {
7338     it_ia_handle_t    ia;                /* IT_UD_PARAM_IA_HANDLE */
7339     uint32_t          request_id;        /* IT_UD_PARAM_REQ_ID */
7340     it_evd_handle_t  reply_evd;         /* IT_UD_PARAM_REPLY_EVD */
7341     it_conn_qual_t   conn_qual;         /* IT_UD_PARAM_CONN_QUAL */
7342     it_path_t        destination_path;  /* IT_UD_PARAM_DEST_PATH */
7343     unsigned char    private_data[IT_MAX_PRIV_DATA];
7344                                     /* IT_UD_PARAM_PRIV_DATA */
7345     size_t           private_data_length;
7346                                     /* IT_UD_PARAM_PRIV_DATA_LENGTH */
7347 } it_ud_svc_req_param_t;
```

7348 APPLICABILITY

7349 *it_ud_service_request_handle_query* is applicable only to the UD service type.

7350 DESCRIPTION

7351 ud_svc_handle Unreliable Datagram (UD) Service Request Handle previously created by a
7352 call to *it_ud_service_request_handle_create*.

7353	mask	Bitwise OR of flags for the requested UD Service Request Handle parameters.
7354		
7355	ud_svc_handle_info	Data structure containing information about the UD Service Request Handle, <i>ud_svc_handle</i> .
7356		
7357	<i>it_ud_service_request_handle_query</i> collects the desired information about the <i>ud_svc_handle</i>	
7358	passed in and returns that information in the <i>it_ud_svc_req_param_t</i> structure provided in	
7359	<i>ud_svc_handle_info</i> . On return, each field of <i>ud_svc_handle_info</i> is only valid if the	
7360	corresponding flag is set in the <i>mask</i> argument. The flag values for the <i>mask</i> appear in the	
7361	comments above each of the fields in the <i>it_ud_svc_req_param_t</i> structure. The <i>mask</i> value	
7362	IT_UD_PARAM_ALL causes all fields to be returned.	
7363	The definition of each field in the <i>it_ud_svc_req_param_t</i> structure is as follows:	
7364	<i>ia</i>	Handle for the Interface Adapter associated with this UD Service Request.
7365		
7366	<i>request_id</i>	Unique identifier associated with the <i>it_ud_svc_req_handle_t</i> .
7367	<i>reply_evd</i>	The Simple EVD for reply Events associated with the
7368		<i>it_ud_svc_req_handle_t</i> .
7369	<i>conn_qual</i>	Connection Qualifier describing the UD Service associated with the
7370		<i>it_ud_svc_req_handle_t</i> .
7371	<i>destination_path</i>	Path to the Destination of the <i>it_ud_service_request</i> operation associated
7372		with the <i>it_ud_svc_req_handle_t</i> .
7373	<i>private_data</i>	Opaque Private Data provided by the Consumer if the IA supports
7374		Private Data.
7375	<i>private_data_length</i>	Length of the Private Data supplied by the Consumer.
7376	RETURN VALUE	
7377	A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:	
7378	IT_ERR_INVALID_UD_SVC	The Unreliable Datagram Service Handle (<i>ud_svc_handle</i>) was
7379		invalid.
7380	IT_ERR_INVALID_MASK	The <i>mask</i> contained invalid flag values.
7381	IT_ERR_IA_CATASTROPHE	The Interface Adapter has experienced a catastrophic error and is
7382		in the disabled state. None of the output parameters from this
7383		routine are valid. See <i>it_ia_info_t</i> for a description of the disabled
7384		state.
7385	SEE ALSO	
7386	<i>it_ud_service_request_handle_create()</i> , <i>it_ud_service_request_handle_free()</i> ,	
7387	<i>it_ud_service_request()</i>	

7 Data Type Reference Pages

<i>it_addr_mode_t</i>	Addressing mode of a Local Memory Region or Remote Memory Region
<i>it_aevd_notification_event_t</i>	Aggregate Event Dispatcher Notification Event type
<i>it_affiliated_event_t</i>	Affiliated Asynchronous Event type
<i>it_boolean_t</i>	The Boolean type used by the IT-API
<i>it_cm_msg_events</i>	Communication Management Message Events
<i>it_cm_req_events</i>	Communication Management Request Events
<i>it_conn_qual_t</i>	Encapsulates all supported Connection Qualifier types
<i>it_context_t</i>	Structure describing a Consumer Context
<i>it_dg_remote_ep_addr_t</i>	DatagramTransport Endpoint address
<i>it_dto_cookie_t</i>	DTO Cookie type
<i>it_dto_events</i>	Completion Event types
<i>it_dto_flags_t</i>	Flags for Send, Receive, RDMA Read & Write, RMR Link & Unlink
<i>it_dto_status_t</i>	Definition of DTO and RMR completion asynchronous status
<i>it_ep_attributes_t</i>	Endpoint attributes
<i>it_ep_state_t</i>	RC and UD Endpoint state type definition
<i>it_event_t</i>	Definition of Event data structures
<i>it_handle_t</i>	Enumeration and type definitions for IT Handles
<i>it_ia_info_t</i>	Encapsulates all Interface Adapter attributes and Spigot information
<i>it_lmr_triplet_t</i>	Structure describing a DTO buffer in a Local Memory Region
<i>it_mem_priv_t</i>	Memory access privileges for Local and Remote Memory Regions
<i>it_net_addr_t</i>	Encapsulates all supported Network Address types
<i>it_path_t</i>	Describes the Path between a pair of Spigots
<i>it_rmr_triplet_t</i>	Structure describing a DTO buffer in a Remote Memory Region
<i>it_rmr_type_t</i>	Definition of RMR type
<i>it_software_event_t</i>	Software Event type
<i>it_status_t</i>	Definition of IT-API call return status
<i>it_unaffiliated_event_t</i>	Unaffiliated Asynchronous Event type

7390

it_addr_mode_t

7391 NAME

7392 it_addr_mode_t – definition of addressing mode of a Local Memory Region or Remote Memory
7393 Region

7394 SYNOPSIS

```
7395 #include <it_api.h>  
7396  
7397 typedef enum {  
7398     IT_ADDR_MODE_ABSOLUTE = 0,  
7399     IT_ADDR_MODE_RELATIVE = 1  
7400 } it_addr_mode_t;
```

7401 DESCRIPTION

7402 Addressing mode of an LMR or RMR, with two possible values as follows:

7403	IT_ADDR_MODE_ABSOLUTE	Indicates Absolute Addressing. The <i>addr</i> attribute (requested starting address) of an LMR with Absolute Addressing is interpreted as the Base Address of the LMR. When an LMR with Absolute Addressing is accessed by a DTO, the <i>addr.abs</i> member of an LMR Triplet or the <i>rdma_addr</i> parameter passed to an RDMA DTO is interpreted as the Base Address of the LMR plus a byte offset. When an RMR is linked to an LMR with Absolute Addressing, the <i>addr</i> parameter of the RMR Link operation and the <i>addr</i> attribute of the LMR are used for identifying the offset of the first byte of the RMR from the first byte of the LMR. The <i>addr</i> attribute (starting address) of a linked RMR with Absolute Addressing is interpreted as the Base Address of the RMR. When a linked RMR with Absolute Addressing is accessed by a DTO, the <i>addr.abs</i> member of an RMR Triplet or the <i>rdma_addr</i> parameter passed to an RDMA DTO is interpreted as the Base Address of the RMR plus a byte offset.
7419	IT_ADDR_MODE_RELATIVE	Indicates Relative Addressing. The <i>addr</i> attribute (requested starting address) of an LMR with Relative Addressing is interpreted as the Base Address of the LMR, unless it equals IT_NO_ADDR. This attribute can be used for specifying or registering an LMR but is ignored in case of Relative Addressing when the LMR is accessed by a DTO. When an LMR with Relative Addressing is accessed by a DTO, the <i>addr.rel</i> member of an LMR Triplet or the <i>rdma_addr</i> parameter passed to an RDMA DTO is interpreted as a byte offset relative to the first byte of the LMR. When an RMR is linked to an LMR, with Relative Addressing selected for the RMR, the <i>addr</i> parameter of the RMR Link operation and the <i>addr</i> attribute of the LMR (which must use Absolute Addressing) are used for identifying the offset of the first byte of the RMR from the first byte of the LMR. The <i>addr</i> attribute (starting address) of a linked RMR with Relative Addressing is interpreted as the Base Address of the RMR. When a linked RMR with Relative Addressing is accessed by a DTO, the <i>addr.rel</i> member of an RMR Triplet or the <i>rdma_addr</i> parameter passed to an RDMA DTO is interpreted as a byte offset

7437 relative to the first byte of the RMR. The total offset relative to the
7438 first byte of the underlying LMR is obtained by adding to this byte
7439 offset the difference between the *addr* attributes of the RMR and the
7440 LMR established at RMR Link time.

7441 **EXTENDED DESCRIPTION**

7442 On InfiniBand, Relative Addressing is supported only if the Verb Extensions are supported
7443 including the “ZBVA” option. Moreover, Relative Addressing may be available only for Narrow
7444 RMRs (Type 2 Memory Windows).

7445 **APPLICATION USAGE**

7446 The desired addressing mode of an LMR must be specified when the LMR is created or linked.
7447 The desired addressing mode of an RMR must be specified when the RMR is linked.

7448 The *addr_mode_relative_support* field of the *it_ia_info_t* structure, which can be queried
7449 through *it_ia_query*, indicates whether the IA supports Relative Addressing.

7450 **SEE ALSO**

7451 *it_lmr_create()*, *it_rmr_link()*, *it_post_send()*, *it_post_sendto()*, *it_post_rcv()*,
7452 *it_post_rcvfrom()*, *it_post_rdma_write()*, *it_post_rdma_read()*, *it_lmr_triplet_t*, *it_rmr_triplet_t*

it_aevd_notification_event_t

7453

7454 NAME

7455 it_aevd_notification_event_t – Aggregate Event Dispatcher Notification Event type

7456 SYNOPSIS

```
7457 #include <it_api.h>
7458
7459 typedef struct {
7460     it_event_type_t event_number;
7461     it_evd_handle_t aevd;
7462     it_evd_handle_t sevd;
7463 } it_aevd_notification_event_t;
```

7464 DESCRIPTION

7465 *event_number*

Identifier of the Event type.

7466 Valid values: IT_AEVD_NOTIFICATION_EVENT

7467 *aevd*

Handle for the Aggregate Event Dispatcher (AEVD) where the Event was queued.

7468

7469 *sevd*

Handle to the Simple Event Dispatcher (SEVD) that experienced a Notification Event.

7470

7471 An IT_AEVD_NOTIFICATION_EVENT_STREAM Event is generated when a Notification
7472 has occurred on an SEVD associated with an AEVD with the IT_EVD_DEQUEUE_
7473 NOTIFICATIONS *evd_flag* set (see *it_evd_create*).

7474 The AEVD Notification Event passes the Handle for the associated SEVD on which a
7475 Notification Event has occurred.

7476 The AEVD Notification Event only applies to AEVDs. AEVDs do not overflow.

7477 SEE ALSO

7478 *it_event_t*, *it_evd_create()*, *it_evd_wait()*

7479

it_affiliated_event_t

7480

7481 NAME

7482 it_affiliated_event_t – Affiliated Asynchronous Event type

7483 SYNOPSIS

```
7484 #include <it_api.h>
7485
7486 typedef struct {
7487     it_event_type_t event_number;
7488     it_evd_handle_t evd;
7489
7490     union {
7491         it_evd_handle_t sevd;
7492         it_ep_handle_t ep;
7493         it_srq_handle_t srq;
7494     } cause;
7495 } it_affiliated_event_t;
```

7496 DESCRIPTION

7497	<i>event_number</i>	Identifier of the Event type. Valid values: IT_ASYNC_AFF_SEVD_ENQUEUE_FAILURE, 7498 IT_ASYNC_AFF_EP_FAILURE, 7499 IT_ASYNC_AFF_EP_BAD_TRANSPORT_OPCODE, 7500 IT_ASYNC_AFF_EP_REQ_DROPPED, 7501 IT_ASYNC_AFF_EP_RDMAW_ACCESS_VIOLATION, 7502 IT_ASYNC_AFF_EP_RDMAW_CORRUPT_DATA, 7503 IT_ASYNC_AFF_EP_RDMAR_ACCESS_VIOLATION, 7504 IT_ASYNC_AFF_EP_LOCAL_ACCESS_VIOLATION, 7505 IT_ASYNC_AFF_EP_L_ACCESS_VIOLATION, 7506 IT_ASYNC_AFF_EP_L_RECV_ACCESS_VIOLATION, 7507 IT_ASYNC_AFF_EP_L_IRRQ_ACCESS_VIOLATION, 7508 IT_ASYNC_AFF_EP_L_TRANSPORT_ERROR, 7509 IT_ASYNC_AFF_EP_L_LLQ_ERROR, IT_ASYNC_AFF_EP_R_ERROR, 7510 IT_ASYNC_AFF_EP_R_ACCESS_VIOLATION, 7511 IT_ASYNC_AFF_EP_R_RECV_ACCESS_VIOLATION, 7512 IT_ASYNC_AFF_EP_R_RECV_LENGTH_ERROR, 7513 IT_ASYNC_AFF_EP_SOFT_HI_WATERMARK 7514 IT_ASYNC_AFF_SRQ_LOW_WATERMARK 7515
7516	<i>evd</i>	Handle for the Event Dispatcher where the Event was queued.
7517	<i>sevd</i>	The Handle for the SEVD for which the Implementation failed to enqueue an 7518 Event. Valid only for the IT_ASYNC_AFF_SEVD_ENQUEUE_FAILURE Event 7519 type.
7520	<i>ep</i>	The Handle for the Endpoint that experienced the Event. Valid for all 7521 asynchronous errors affiliated with Endpoint other than 7522 IT_ASYNC_AFF_SEVD_ENQUEUE_FAILURE and 7523 IT_ASYNC_AFF_SRQ_LOW_WATERMARK.
7524	<i>srq</i>	The Handle for the S-RQ whose Low Watermark threshold has just been crossed.

7525
7526
7527

IT_ASYNC_AFF_EVENT_STREAM Events are generated when an Affiliated Asynchronous Event occurs. There are several types of Affiliated Asynchronous Events, and each type is identified by *event_number*.

7528
7529

The Consumer asks for Affiliated Asynchronous Events to be delivered when it uses *it_evd_create* to create an EVD associated with the Affiliated Asynchronous Event Stream.

7530
7531

The following table maps the Affiliated Asynchronous Error values in the *it_event_type_t* enumeration to a transport-independent description.

<i>it_event_type_t</i> Value	Generic Event Description
IT_ASYNC_AFF_SEVD_ENQUEUE_FAILURE	The Implementation was unable to enqueue an entry into the SEVD. Applies to all SEVD Event Streams except for IT_ASYNC_AFF_EVENT_STREAM and IT_ASYNC_UNAFF_EVENT_STREAM.
IT_ASYNC_AFF_EP_FAILURE	The local Endpoint experienced a failure when attempting to enqueue on an EVD in the <i>it_evd_overflowed</i> state or on an EVD in an error state.
IT_ASYNC_AFF_EP_BAD_TRANSPORT_OPCODE	The local Endpoint detected an invalid transport opcode in an incoming request it was processing.
IT_ASYNC_AFF_EP_REQ_DROPPED	The local Endpoint could not process an incoming Send operation because the Receive Queue was empty.
IT_ASYNC_AFF_EP_RDMAW_ACCESS_VIOLATION	The remote Endpoint connected to the local Endpoint that is furnished via this Event detected an access violation while processing an RDMA Write operation.
IT_ASYNC_AFF_EP_RDMAW_CORRUPT_DATA	The remote Endpoint connected to the local Endpoint that is furnished via this Event detected corruption in the incoming data.
IT_ASYNC_AFF_EP_RDMAR_ACCESS_VIOLATION	The remote Endpoint connected to the local Endpoint that is furnished via this Event detected an access violation while processing an RDMA Read operation.
IT_ASYNC_AFF_EP_LOCAL_ACCESS_VIOLATION = IT_ASYNC_AFF_EP_L_ACCESS_VIOLATION	The local Endpoint detected an access violation while processing an incoming request. Note that not all incoming requests that cause an access violation will cause an Affiliated Asynchronous Event to be generated.

<i>it_event_type_t</i> Value	Generic Event Description
IT_ASYNC_AFF_EP_L_RECV_ACCESS_VIOLATION	The Receive Queue or Shared Receive Queue of the local Endpoint detected an access violation while processing an incoming Send. Note that not all incoming Sends that cause an access violation will cause an Affiliated Asynchronous Event to be generated.
IT_ASYNC_AFF_EP_L_IRRQ_ACCESS_VIOLATION	The Inbound RDMA Read Queue of the local Endpoint detected an access violation while processing an incoming Read Request.
IT_ASYNC_AFF_EP_L_TRANSPORT_ERROR	The local Endpoint detected a transport error while processing an incoming request. Note that not all incoming requests that cause a transport error will cause an Affiliated Asynchronous Event to be generated.
IT_ASYNC_AFF_EP_L_LLP_ERROR	The local Endpoint detected a LLP error while processing an incoming request.
IT_ASYNC_AFF_EP_R_ERROR	The local Endpoint received an indication that the connected remote Endpoint detected an error while processing an incoming request.
IT_ASYNC_AFF_EP_R_ACCESS_VIOLATION	The local Endpoint received an indication that the connected remote Endpoint detected an access violation while processing an incoming RDMA request.
IT_ASYNC_AFF_EP_R_RECV_ACCESS_VIOLATION	The local Endpoint received an indication that the connected remote Endpoint could not place an incoming Send due to an MSN inconsistency or the lack of a Receive buffer.
IT_ASYNC_AFF_EP_R_RECV_LENGTH_ERROR	The local Endpoint received an indication that the connected remote Endpoint could not place an incoming Send due to insufficient length of the Receive buffer.
IT_ASYNC_AFF_EP_SOFT_HI_WATERMARK	The total number of Receive DTOs being processed by the Endpoint has exceeded the Endpoint Soft High Watermark threshold that was established by either the <i>it_ep_rc_create()</i> or <i>it_ep_modify()</i> routine.

<i>it_event_type_t</i> Value	Generic Event Description
IT_ASYNC_AFF_SRQ_LOW_WATERMARK	The total number of outstanding Receive DTOs for the Shared Receive Queue has dipped below the S-RQ Low Watermark threshold that was established by calling <i>it_srq_modify()</i> .

7532
7533
7534

All Events on an IT_ASYNC_AFF_EVENT_STREAM SEVD cause Notification. See [it_evd_create](#) for details of Notification.

7535
7536
7537
7538
7539
7540
7541

Default overflow behavior of an IT_ASYNC_AFF_EVENT_STREAM SEVD is overflow Notification enabled with automatic rearming. This default behavior of the SEVD is equivalent to IT_EVD_OVERFLOW_DEFAULT cleared and IT_EVD_OVERFLOW_NOTIFY set and IT_EVD_OVERFLOW_AUTO_RESET set. See [it_evd_create](#) for details of overflow detection. Note that overflow of an IT_ASYNC_AFF_EVENT_STREAM SEVD generates an IT_ASYNC_UNAFF_SEVD_ENQUEUE_FAILURE Event on the Unaffiliated Asynchronous Event SEVD of the IA.

7542
7543
7544
7545

EXTENDED DESCRIPTION

For the InfiniBand Transport, the following table maps the Affiliated Asynchronous Error values in the *it_event_type_t* enumeration to their corresponding “Affiliated Asynchronous Errors” as specified in Chapter 11 of [IB-R1.1] or [IB-R1.2].

<i>it_event_type_t</i> Value	IB “Affiliated Asynchronous Error” Name
IT_ASYNC_AFF_SEVD_ENQUEUE_FAILURE	CQ Error
IT_ASYNC_AFF_EP_FAILURE	Local Work Queue Catastrophic Error
IT_ASYNC_AFF_EP_BAD_TRANSPORT_OPCODE	Invalid Request Local Work Queue Error
IT_ASYNC_AFF_EP_REQ_DROPPED	(Not applicable to the IB transport.)
IT_ASYNC_AFF_EP_RDMAW_ACCESS_VIOLATION	(Not applicable to the IB transport.)
IT_ASYNC_AFF_EP_RDMAW_CORRUPT_DATA	(Not applicable to the IB transport.)
IT_ASYNC_AFF_EP_RDMAR_ACCESS_VIOLATION	(Not applicable to the IB transport.)
IT_ASYNC_AFF_EP_LOCAL_ACCESS_VIOLATION = IT_ASYNC_AFF_EP_L_ACCESS_VIOLATION	Local Access Violation Work Queue Error
IT_ASYNC_AFF_EP_L_RECV_ACCESS_VIOLATION	(Not applicable to the IB transport.)
IT_ASYNC_AFF_EP_L_IRRQ_ACCESS_VIOLATION	(Not applicable to the IB transport.)
IT_ASYNC_AFF_EP_L_TRANSPORT_ERROR	(Not applicable to the IB transport.)
IT_ASYNC_AFF_EP_L_LLQ_ERROR	(Not applicable to the IB transport.)
IT_ASYNC_AFF_EP_R_ERROR	(Not applicable to the IB transport.)
IT_ASYNC_AFF_EP_R_ACCESS_VIOLATION	(Not applicable to the IB transport.)
IT_ASYNC_AFF_EP_R_RECV_ACCESS_VIOLATION	(Not applicable to the IB transport.)

7546
7547
7548
7549

<i>it_event_type_t</i> Value	IB “Affiliated Asynchronous Error” Name
IT_ASYNC_AFF_EP_R_RECV_LENGTH_ERROR	(Not applicable to the IB transport.)
IT_ASYNC_AFF_EP_SOFT_HI_WATERMARK	(Not applicable to the IB transport.)
IT_ASYNC_AFF_SRQ_LOW_WATERMARK	S-RQ Limit Reached

For the iWARP Transport, the following table maps the Affiliated Asynchronous Error values in the *it_event_type_t* enumeration to their corresponding “Asynchronous Event Identifiers” in Section 9.5.3 of [VERBS-RDMAC].

<i>it_event_type_t</i> Value	iWARP “Asynchronous Event Identifier” Name(s)
IT_ASYNC_AFF_SEVD_ENQUEUE_FAILURE	<ul style="list-style-type: none"> • CQ/SQ Error • CQ/RQ Error
IT_ASYNC_AFF_EP_FAILURE	Local QP Catastrophic Error
IT_ASYNC_AFF_EP_BAD_TRANSPORT_OPCODE	Unexpected Opcode
IT_ASYNC_AFF_EP_REQ_DROPPED	(Not applicable to the iWARP Transport.)
IT_ASYNC_AFF_EP_RDMAW_ACCESS_VIOLATION	(Not applicable to the iWARP Transport.)
IT_ASYNC_AFF_EP_RDMAW_CORRUPT_DATA	(Not applicable to the iWARP Transport.)
IT_ASYNC_AFF_EP_RDMAR_ACCESS_VIOLATION	(Not applicable to the iWARP Transport.)
IT_ASYNC_AFF_EP_L_ACCESS_VIOLATION	<ul style="list-style-type: none"> • Invalid STag • Base and bounds violation • Access Rights violation • Invalid PD ID • Wrap error (Locally detected access violation due to incoming RDMA Write or RDMA Read Response message.)
IT_ASYNC_AFF_EP_L_RECV_ACCESS_VIOLATION	<ul style="list-style-type: none"> • Invalid MSN – no buffer available • Invalid MSN – MSN range not valid/gap in MSN (Locally detected access violation due to incoming Send message; includes MSN errors associated with an RQ or S-RQ.)

it_event_type_t Value	iWARP “Asynchronous Event Identifier” Name(s)
IT_ASYNC_AFF_EP_L_IRRQ_ACCESS_VIOLATION	<ul style="list-style-type: none"> • Invalid STag • Base and bounds violation • Access Rights violation • Invalid PD ID • Wrap error • Invalid MSN – MSN range not valid/gap in MSN (Locally detected access violation due to incoming RDMA Read Request.)
IT_ASYNC_AFF_EP_L_TRANSPORT_ERROR	<ul style="list-style-type: none"> • Invalid DDP Queue Number • Invalid RDMA Read Request • No ‘L’ bit when expected (Locally detected operation error or protocol error due to incoming requests.)
IT_ASYNC_AFF_EP_L_LLQ_ERROR	<ul style="list-style-type: none"> • LLP Connection Lost • LLP Connection Reset • LLP Integrity Error: Segment size invalid • LLP Integrity Error: Invalid CRC • Bad FPDU (Locally detected LLP error.)
IT_ASYNC_AFF_EP_R_ERROR	Terminate Message Received (Remotely detected error.)
IT_ASYNC_AFF_EP_R_ACCESS_VIOLATION	Terminate Message Received (Remotely detected “Remote Protection Error” (access violation) due to an incoming RDMA Write, RDMA Read Request, or RDMA Read Response message.)

it_event_type_t Value	iWARP “Asynchronous Event Identifier” Name(s)
IT_ASYNC_AFF_EP_R_RECV_ACCESS_VIOLATION	Terminate Message Received (Remotely detected): <ul style="list-style-type: none"> Invalid MSN – no buffer available Invalid MSN – MSN range not valid/gap in MSN DDP error due to an incoming Send message. Includes MSN errors associated with a remote RQ or S-RQ.)
IT_ASYNC_AFF_EP_R_RECV_LENGTH_ERROR	Terminate Message Received (Remotely detected): <ul style="list-style-type: none"> DDP Message too long for available buffer DDP error due to an incoming Send message.)
IT_ASYNC_AFF_EP_SOFT_HI_WATERMARK	QP RQ Limit Reached
IT_ASYNC_AFF_SRQ_LOW_WATERMARK	Shared Receive Queue Limit Reached

7550
7551
7552
7553

For the VIA transport, the following table maps the Affiliated Asynchronous Error values in the *it_event_type_t* enumeration to their corresponding descriptions in the “VipErrorCallback” reference page in the Appendix of [VIA-V1.0].

it_event_type_t Value	VIA “VipErrorCallback” Name
IT_ASYNC_AFF_SEVD_ENQUEUE_FAILURE	(Not applicable to the VIA transport.)
IT_ASYNC_AFF_EP_FAILURE	Completion Protection Error
IT_ASYNC_AFF_EP_BAD_TRANSPORT_OPCODE	RDMA Write Packet Abort
IT_ASYNC_AFF_EP_REQ_DROPPED	Receive Queue Empty
IT_ASYNC_AFF_EP_RDMAW_ACCESS_VIOLATION	RDMA Write Protection Error
IT_ASYNC_AFF_EP_RDMAW_CORRUPT_DATA	RDMA Write Data Error
IT_ASYNC_AFF_EP_RDMAR_ACCESS_VIOLATION	RDMA Read Protection Error
IT_ASYNC_AFF_EP_LOCAL_ACCESS_VIOLATION = IT_ASYNC_AFF_EP_L_ACCESS_VIOLATION	(Not applicable to the VIA transport.)
IT_ASYNC_AFF_EP_L_RECV_ACCESS_VIOLATION	(Not applicable to the VIA transport.)
IT_ASYNC_AFF_EP_L_IRRQ_ACCESS_VIOLATION	(Not applicable to the VIA transport.)
IT_ASYNC_AFF_EP_L_TRANSPORT_ERROR	(Not applicable to the VIA transport.)
IT_ASYNC_AFF_EP_L_LLQ_ERROR	(Not applicable to the VIA transport.)
IT_ASYNC_AFF_EP_R_ERROR	(Not applicable to the VIA transport.)

IT_ASYNC_AFF_EP_R_ACCESS_VIOLATION	(Not applicable to the VIA transport.)
IT_ASYNC_AFF_EP_R_RECV_ACCESS_VIOLATION	(Not applicable to the VIA transport.)
IT_ASYNC_AFF_EP_R_RECV_LENGTH_ERROR	(Not applicable to the VIA transport.)
IT_ASYNC_AFF_EP_SOFT_HI_WATERMARK	(Not applicable to the VIA transport.)
IT_ASYNC_AFF_SRQ_LOW_WATERMARK	(Not applicable to the VIA transport.)

7554 **SEE ALSO**

7555 *it_event_t, it_evd_wait(), it_evd_create(), it_srq_modify(), it_ep_attributes*

it_boolean_t

7556

7557 NAME

7558 it_boolean_t – the Boolean type used by the API

7559 SYNOPSIS

```
7560 #include <it_api.h>
7561
7562 typedef enum {
7563     IT_FALSE = 0,
7564     IT_TRUE  = 1
7565 } it_boolean_t;
```

7566 DESCRIPTION

7567 The *it_boolean_t* type is used in several data structures in the API to describe a value that can
7568 exist in one of two different states: true (IT_TRUE), or false (IT_FALSE).

7569 SEE ALSO

7570 *it_cm_msg_events*, *it_cm_req_events*, *it_ep_attributes_t*, *it_evd_create()*, *it_evd_modify()*,
7571 *it_evd_query()*, *it_ia_info_t*, *it_rmr_query()*

7572

it_cm_msg_events

7573

7574 NAME

7575 Communication Management Message Events – definitions for communication management
7576 Events other than Connection Requests and definition of Unreliable Datagram service resolution
7577 reply Event

7578 SYNOPSIS

```
7579 #include <it_api.h>
7580
7581 #define IT_MAX_PRIV_DATA 256
7582
7583 typedef enum {
7584     IT_CN_REJ_OTHER           = 0,
7585     IT_CN_REJ_TIMEOUT        = 1,
7586     IT_CN_REJ_BAD_PATH       = 2,
7587     IT_CN_REJ_STALE_CONN     = 3,
7588     IT_CN_REJ_BAD_ORD        = 4,
7589     IT_CN_REJ_RESOURCES      = 5,
7590     IT_CN_REJ_BAD_CONN_PARMS = 6,
7591 } it_conn_reject_code_t;
7592
7593 typedef struct {
7594     it_event_type_t           event_number;
7595     it_evd_handle_t           evd;
7596     it_cn_est_identifier_t    cn_est_id;
7597     it_ep_handle_t            ep;
7598     uint32_t                  rdma_read_ird;
7599     uint32_t                  rdma_read_ord;
7600     it_path_t                 dst_path;
7601     it_conn_reject_code_t     reject_reason_code;
7602     unsigned char             private_data[IT_MAX_PRIV_DATA];
7603     it_boolean_t              private_data_present;
7604 } it_connection_event_t;
7605
7606 typedef enum {
7607     IT_UD_SVC_EP_INFO_VALID   = 0,
7608     IT_UD_SVC_ID_NOT_SUPPORTED = 1,
7609     IT_UD_SVC_REQ_REJECTED    = 2,
7610     IT_UD_NO_EP_AVAILABLE    = 3,
7611     IT_UD_REQ_REDIRECTED      = 4
7612 } it_ud_svc_req_status_t;
7613
7614 typedef struct {
7615     it_event_type_t           event_number;
7616     it_evd_handle_t           evd;
7617     it_ud_svc_req_handle_t    ud_svc;
7618     it_ud_svc_req_status_t    status;
7619     it_remote_ep_info_t       ep_info;
7620     it_path_t                 dst_path;
7621     unsigned char             private_data[IT_MAX_PRIV_DATA];
7622     it_boolean_t              private_data_present;
```

7623 } *it_ud_svc_reply_event_t*;

7624 **DESCRIPTION**

7625 The Communication Management Message Event Stream, *IT_CM_MSG_EVENT_STREAM*,
7626 generates Events for all of the possible state transitions following a Connection Request as well
7627 as for Unreliable Datagram Service Resolution replies. These Events are all the Communication
7628 Management Events except those invoked by incoming requests (see *it_cm_req_events* for
7629 those).

7630 Only one Event will be generated when a Connection is destroyed for any reason, either the
7631 *IT_CM_MSG_CONN_DISCONNECT_EVENT* or the *IT_CM_MSG_CONN_BROKEN_*
7632 *EVENT*, but not both. The Consumer should be ready to handle either of these Events being
7633 generated even when the local or remote Consumer called *it_ep_disconnect*.

7634 The Connection Events are represented by the *it_connection_event_t* structure and the
7635 Unreliable Datagram Service Resolution replies are represented by the *it_ud_svc_reply_event_t*
7636 structure.

7637 The *it_connection_event_t* structure has the following members:

7638	<i>event_number</i>	Identifier of the Event type. Valid values: 7639 <i>IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT</i> , 7640 <i>IT_CM_MSG_CONN_ESTABLISHED_EVENT</i> , 7641 <i>IT_CM_MSG_CONN_PEER_REJECT_EVENT</i> , 7642 <i>IT_CM_MSG_CONN_NONPEER_REJECT_EVENT</i> , 7643 <i>IT_CM_MSG_CONN_DISCONNECT_EVENT</i> , 7644 <i>IT_CM_MSG_CONN_BROKEN_EVENT</i>
7645	<i>evd</i>	Handle for the Event Dispatcher where the Event was queued.
7646	<i>cn_est_id</i>	Identifier for the Connection establishment Event.
7647	<i>ep</i>	Endpoint Handle associated with Connection in progress.
7648	<i>rdma_read_ird</i>	Maximum number of incoming simultaneous RDMA Read 7649 operations supported by the remote EP. Only valid if the 7650 <i>it_ia_info.ird_ord_ia_support</i> value is <i>IT_TRUE</i> and as described 7651 under Application Usage below. Not valid if 7652 <i>IT_LISTEN_SUPPRESS_IRD_ORD</i> is set for <i>it_listen_create</i> or 7653 if <i>IT_CONNECT_SUPPRESS_IRD_ORD</i> is set for 7654 <i>it_ep_connect</i> .
7655	<i>rdma_read_ord</i>	Maximum number of outgoing simultaneous RDMA Read 7656 operations supported by the remote EP. Only valid if the 7657 <i>it_ia_info.ird_ord_ia_support</i> value is <i>IT_TRUE</i> and as described 7658 under Application Usage below. Not valid if 7659 <i>IT_LISTEN_SUPPRESS_IRD_ORD</i> is set for <i>it_listen_create</i> or 7660 if <i>IT_CONNECT_SUPPRESS_IRD_ORD</i> is set for 7661 <i>it_ep_connect</i> .
7662	<i>dst_path</i>	Path to Destination node supporting Service. Valid only if remote 7663 has rejected the proposed Path (<i>reject_reason_code</i> is 7664 <i>IT_CN_REJ_BAD_PATH</i>). Consumer should use <i>dst_path</i> if they 7665 wish to retry Connection attempt.

7666	<i>reject_reason_code</i>	Reason for rejection of Connection attempt.
7667	<i>private_data</i>	Private Data buffer.
7668	<i>private_data_present</i>	When it has the value IT_TRUE then Private Data is present in the
7669		<i>private_data</i> buffer above.
7670	The <i>it_ud_svc_reply_event_t</i> structure has the following members:	
7671	<i>event_number</i>	Identifier of the Event type. Valid values:
7672		IT_CM_MSG_UD_SERVICE_REPLY_EVENT
7673	<i>evd</i>	Handle for the Event Dispatcher where the Event was queued.
7674	<i>ud_svc</i>	Handle for the corresponding Service Request.
7675	<i>status</i>	Completion status for Service Request.
7676	<i>ep_info</i>	Resolution of Connection Qualifier for the UD service to a
7677		specific remote Endpoint. Only valid if <i>status</i> is
7678		IT_UD_SVC_EP_INFO_VALID. See <i>it_ep_attributes_t</i> for the
7679		definition of the <i>it_remote_ep_info_t</i> structure.
7680	<i>dst_path</i>	Path to Destination node supporting Service. Valid only if remote
7681		has redirected (<i>status</i> is IT_UD_REQ_REDIRECTED). Path
7682		returned is complete.
7683	<i>private_data</i>	Private Data buffer.
7684	<i>private_data_present</i>	When it has the value IT_TRUE then Private Data is present in the
7685		<i>private_data</i> buffer above.

7686 **EXTENDED DESCRIPTION**

7687 Connection Events are described in the following table:

Event Type	Description	Notes
IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT	The passive side of a three-way Connection establishment has issued an <i>it_ep_accept</i> for the specified Connection establishment request identifier.	Only applies to three-way Connection establishment. Second phase of three. The Endpoint is in IT_EP_STATE_ACTIVE2_CONNECTION_PENDING state.
IT_CM_MSG_CONN_ESTABLISHED_EVENT	The Connection identified has been established and Data Transfer Operations can be performed. This Event is generated on both the passive and active sides of a Connection.	Applies to Connection management through both the TII (two-way and three-way) and the TDI. Second phase on two-way, third phase on three-way. The Endpoint is in CONNECTED state.

Event Type	Description	Notes
IT_CM_MSG_CONN_DISCONNECT_EVENT	The Connection identified has been disconnected, either by the local or remote side, through a call to <i>it_ep_disconnect</i> . No more Data Transfer Operations posted on the Endpoint will complete successfully.	Applies to Connection management through both the TII (two-way and three-way) and the TDI. The Endpoint is in IT_EP_STATE_NONOPERATIONAL state. All posted DTOs and RMRs are flushed.
IT_CM_MSG_CONN_PEER_REJECT_EVENT	The remote side of a Connection establishment request has issued <i>it_reject</i> for the specified Connection establishment request.	Applies to Connection management through the TII (two-way and three-way). The Endpoint is in IT_EP_STATE_NONOPERATIONAL state. All preposted DTOs and RMRs are flushed.
IT_CM_MSG_CONN_NONPEER_REJECT_EVENT	This Event includes all other reasons for not establishing a Connection that are not related to the remote Consumer issuing <i>it_reject</i> . This includes reasons that were detected remotely and communicated to the local Endpoint, as well as locally detected reasons. Such reasons include overflow of the remote EVD for Connection Events, timeouts of the Connection attempt, the passive side rejecting the proposed Path for the Connection attempt, and a Non-permissive AV-RNIC/IETF being unable to interoperate with an RDMAC AV-RNIC.	Applies to Connection management through both the TII (two-way and three-way) and the TDI. The Endpoint is in IT_EP_STATE_NONOPERATIONAL state. All preposted DTOs and RMRs are flushed.

Event Type	Description	Notes
IT_CM_MSG_CONN_BROKEN_EVENT	The Connection identified has been disconnected by the Implementation. Causes include transport errors.	Applies to Connection management through both the TII (two-way and three-way) and the TDI. The Endpoint is in IT_EP_STATE_NONOPERATIONAL state. All posted DTOs and RMRs are flushed.

7688

Table 3: Connection Management Event Definitions

7689

The following table identifies which fields are valid in each of the Connection Management Message Events. For any Event, *event_number* and *evd* are always valid.

7690

Event Type	Valid Fields
IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT	<i>cn_est_id</i> , <i>ep</i> , <i>rdma_read_ird</i> , <i>rdma_read_ord</i> , <i>private_data</i> , <i>private_data_present</i> . The <i>cn_est_id</i> may not be valid if the Consumer called <i>it_ep_disconnect</i> on the associated Endpoint at any time before the <i>cn_est_id</i> is used.
IT_CM_MSG_CONN_ESTABLISHED_EVENT	<i>ep</i> , <i>private_data</i> , <i>private_data_present</i> , <i>rdma_read_ird</i> (IRD), <i>rdma_read_ord</i> (ORD). On the passive side, the IRD/ORD values are reported as those initially sent by the active side and may differ from the actual values used on the connection.
IT_CM_MSG_CONN_DISCONNECT_EVENT	<i>ep</i> , <i>private_data</i> , <i>private_data_present</i>
IT_CM_MSG_CONN_PEER_REJECT_EVENT	<i>ep</i> , <i>private_data</i> , <i>private_data_present</i>
IT_CM_MSG_CONN_NONPEER_REJECT_EVENT	<i>ep</i> , <i>reject_reason_code</i> If the <i>reject_reason_code</i> is IT_CN_REJ_BAD_PATH, then <i>dst_path</i> is also valid.
IT_CM_MSG_CONN_BROKEN_EVENT	<i>ep</i>

7691

Table 4: Event Management Event Fields

7692

The following table describes the meaning of the various *reject_reason_code* values that can be present in an IT_CM_MSG_CONN_NONPEER_REJECT_EVENT.

7693

reject_reason_code Value	Description
IT_CN_REJ_OTHER	The Connection establishment attempt was rejected for some reason other than those listed below.

reject_reason_code Value	Description
IT_CN_REJ_TIMEOUT	The Connection could not be established within the timeout period defined by the timeout member of the <i>it_path_t</i> that was input to <i>it_ep_connect</i> . This <i>reject_reason_code</i> is only returned when the local timeout period has elapsed; a timeout that occurs at the remote peer does not cause this status to be returned.
IT_CN_REJ_BAD_PATH	The passive side replied to the request to establish a Connection by rejecting the proposed Path. If the Consumer wishes to retry the Connection establishment attempt, the <i>dst_path</i> member of the Event structure contains the suggested Path to use.
IT_CN_REJ_STALE_CONN	The remote peer detected a stale Connection using the local Endpoint that the Consumer furnished as part of the Connection establishment attempt, and has initiated the cleanup process for that stale Connection. If the Consumer wishes to retry the Connection establishment attempt with the remote peer, they should either use a different Endpoint when they retry, or wait for the stale Connection cleanup process to complete before doing the retry. (The duration of the stale Connection cleanup process is Implementation-dependent.)
IT_CN_REJ_BAD_ORD	When this Event is received on the passive side of a Connection establishment attempt, it means that the active side was unwilling to accept the <i>rdma_read_ird</i> limit in the passive-side Endpoint.
IT_CN_REJ_RESOURCES	The remote peer was unable to allocate resources necessary to establish the Connection.
IT_CN_REJ_BAD_CONN_PARMS	During Connection establishment, the local side did not recognize the version or format of the Connection parameter header received from the remote side. See Chapter 5.

Table 5: reject_reason_code Descriptions

7695 The UD Service Resolution Reply Event is described in the following table:

Event Type	Description
IT_CM_MSG_UD_SERVICE_REPLY_EVENT	The passive side of a UD service has responded to the request for Connection Qualifier resolution.

7696 **Table 6: UD Service Resolution Reply Event Definitions**

7697 UD service resolution replies return *status* in the Event data structure as described in the
7698 following table:

Status	Description
IT_UD_SVC_EP_INFO_VALID	Reply is valid. <i>ep_info</i> resolves the remote Endpoint associated with Connection Qualifier.
IT_UD_SVC_ID_NOT_SUPPORTED	Service is not supported by remote.
IT_UD_SVC_REQ_REJECTED	Request is rejected by remote.
IT_UD_NO_EP_AVAILABLE	Remote is out of resources.
IT_UD_REQ_REDIRECTED	Remote redirected the request.

7699 **Table 7: Service Resolution Reply Status**

7700 All Events on an IT_CM_MSG_EVENT_STREAM SEVD cause Notification. See
7701 [it_evd_create](#) for details of Notification.

7702 Default overflow behavior of an IT_CM_MSG_EVENT_STREAM SEVD is automatic
7703 rearming. This default behavior of the SEVD is equivalent to IT_EVD_OVERFLOW_
7704 DEFAULT cleared and IT_EVD_OVERFLOW_NOTIFY set and IT_EVD_OVERFLOW_
7705 AUTO_RESET set. See [it_evd_create](#) for details of overflow detection.

7706 **EXTENDED DESCRIPTION**

7707 For the Infiniband transport, the following table maps the values in the *reject_reason_code* field
7708 to their corresponding “Rejection Reason Code” for the REJ message as specified in Volume 1,
7709 Chapter 12 of the Infiniband specification. Rejection Reason Codes that are not listed in this
7710 table should never be received by a Consumer that is using this API.

reject_reason_code Value	Infiniband Rejection Reason Code Number
IT_CN_REJ_OTHER	4-9, 29-31
IT_CN_REJ_TIMEOUT	None. This code is generated based upon failure to establish the Connection within a given amount of time, not upon receiving a REJ message.
IT_CN_REJ_BAD_PATH	12-17, 24-26, 32
IT_CN_REJ_STALE_CONN	10
IT_CN_REJ_BAD_ORD	27

reject_reason_code Value	Infiniband Rejection Reason Code Number
IT_CN_REJ_RESOURCES	1, 3
IT_CN_REJ_BAD_CONN_PARMS	None. This code is not applicable to the InfiniBand Transport.

Table 8: InfiniBand reject_reason_code Mapping

For the VIA transport, the following table maps the values in the *reject_reason_code* field to their corresponding return values from the reference pages for “VipConnectRequest” and “VipConnectAccept” in the Appendix of the VIA specification. Return values that are not listed in the table below are manifest to Consumers of the IT-API through a mechanism other than a *IT_CM_MSG_CONN_NONPEER_REJECT_EVENT*.

reject_reason_code Value	VIA Return Code
IT_CN_REJ_OTHER	VipConnectAccept – VIP_INVALID_RELIABILITY_LEVEL, VIP_INVALID_QOS, VIP_TIMEOUT, VIP_ERROR_RESOURCE VipConnectRequest – VIP_NO_MATCH
IT_CN_REJ_TIMEOUT	VipConnectAccept – VIP_NOT_REACHABLE VipConnectRequest – VIP_TIMEOUT, VIP_NOT_REACHABLE
IT_CN_REJ_BAD_PATH	None. This code is not applicable to the VIA transport.
IT_CN_REJ_STALE_CONN	None. This code is not applicable to the VIA transport.
IT_CN_REJ_BAD_ORD	None. This code is not applicable to the VIA transport.
IT_CN_REJ_RESOURCES	VipConnectRequest – VIP_ERROR_RESOURCE
IT_CN_REJ_BAD_CONN_PARMS	None. This code is not applicable to the VIA transport.

Table 9: VIA reject_reason_code Mapping

For the iWARP Transport, the following table maps the values in the *reject_reason_code* field to their corresponding return values from the sockets API *connect(2)* call.

reject_reason_code Value	iWARP Return Code
IT_CN_REJ_OTHER	All other possible return values from <i>connect(2)</i> that are not enumerated in the following table entries.
IT_CN_REJ_TIMEOUT	Due to the following LLP connection attempt failure indications: ETIMEDOUT. This code is also generated based upon failure to establish the Connection within the amount of time specified in the <i>timeout</i> member of the <i>it_path_t</i> structure passed to <i>it_ep_connect</i> .

reject_reason_code Value	iWARP Return Code
IT_CN_REJ_BAD_PATH	Due to the following LLP connection attempt failure indications: ENETUNREACH, EADDRNOTAVAIL, ECONNREFUSED.
IT_CN_REJ_STALE_CONN	Due to LLP stale connection conditions such as EADDRINUSE.
IT_CN_REJ_BAD_ORD	None. However, the Implementation may support <i>Reject_Codes</i> as described in the IRD/ORD Header section in Appendix B.
IT_CN_REJ_RESOURCES	Due to the following LLP resource failure indications: ENOMEN and ENOBUFS.
IT_CN_REJ_BAD_CONN_PARMS	During Connection establishment, the local side did not recognize the version or format of the Connection parameter header received from the remote side. See Chapter 5.

Table 8: iWARP reject_reason_code Mapping

For the InfiniBand Transport, the following table maps the *ep_info* field elements in the UD Service Resolution Event data type to InfiniBand concepts as specified in Volume 1 of the Infiniband specification.

ep_info Element	IB Concept
<i>it_ud_ep_id_t</i>	Queue Pair Number (QPN)
<i>it_ud_ep_key_t</i>	Queue Key

Table 9: ep_info Element Mapping

APPLICATION USAGE

The Consumer should use the *it_event_t* structure if it is desired to wait for both communication management Events (*it_connection_event_t*) and Unreliable Datagram service resolution reply Events (*it_ud_svc_reply_event_t*) via the same EVD. The *it_event_t* structure is of sufficient size to hold either Event type.

When using three-way Connection establishment, the Consumer may receive an IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT containing *rdma_read_ird* and *rdma_read_ord* values that differ from those of the Endpoint in use. The Consumer should use *it_ep_modify* to adjust the values associated with the Endpoint to agree with those from this Event before issuing the *it_ep_accept* call to complete the Connection establishment.

When using two-way Connection establishment, the active side Consumer may receive an IT_CM_MSG_CONN_ESTABLISHED_EVENT containing *rdma_read_ird* and *rdma_read_ord* values that differ from those of the Endpoint in use. The Consumer may use *it_ep_modify* to adjust *rdma_read_ord* associated with the Endpoint if it is desired to conserve outbound RDMA resources.

7740 When using two-way Connection establishment, the passive side Consumer may receive an
7741 IT_CM_MSG_CONN_ESTABLISHED_EVENT Event. The *rdma_read_ird* and
7742 *rdma_read_ord* values found in the Event should be ignored by the Consumer (rationale – the
7743 Implementation could, at best, expose the original IRD/ORD values sent by the active side, but
7744 the active side may choose to change ORD on receipt of the passive side thus invalidating the
7745 original values).

7746 With the three-way handshake Connection establishment method, there is also a potential race
7747 condition between the Implementation generating the IT_CM_MSG_CONN_ACCEPT_
7748 ARRIVAL_EVENT Event and the Consumer calling *it_ep_disconnect* or *it_ep_free*. The
7749 Consumer should not use the *cn_est_id* if the IT_CM_MSG_CONN_ACCEPT_
7750 ARRIVAL_EVENT Event arrives after *it_ep_disconnect* or *it_ep_free* was called, regardless of
7751 whether the call returned yet, and regardless of whether the Event was dequeued before or after
7752 the call was made. If the Consumer does use the *cn_est_id* then the Implementation generates an
7753 IT_ERR_INVALID_CN_EST_ID error, or it may generate a segmentation fault, or other error.

7754 Neither the Active nor the Passive side Consumer should rely upon the
7755 IT_CM_MSG_CONN_ESTABLISHED_EVENT Event containing any Private Data, even if
7756 Private Data is input to the final *it_ep_accept* call that causes the Connection to be established.
7757 The side that makes the final call to *it_ep_accept* will never see any Private Data in the
7758 IT_CM_MSG_CONN_ESTABLISHED_EVENT Event, and because of races and unreliability
7759 inherent to the Connection establishment process of many of the transports that the IT-API
7760 supports, Private Data can sometimes be dropped on the other side as well.

7761 See the *it_ep_disconnect* reference page for details on the guarantee of delivery of Private Data
7762 when disconnecting Connections.

7763 The Consumer is advised to structure their ULP so that the Active side sends the first message
7764 after a Connection has been established. This is good practice because under some circumstances
7765 the completion of the first Receive operation is what causes the IT_CM_MSG_CONN_
7766 ESTABLISHED_EVENT Event to be generated on the Passive side. Depending upon the
7767 Passive side to send the first message after a Connection has been established can potentially
7768 result in the Connection establishment process timing out rather than completing successfully.

7769 Consult the Application Usage section of *it_cm_req_events* for the discussion of use of Private
7770 Data.

7771 When the Consumer receives an IT_CM_MSG_UD_SERVICE_REPLY_EVENT where the
7772 *status* is IT_UD_REQ_REDIRECTED, the Consumer can retry the attempt to retrieve UD
7773 service information. In order to do so the Consumer should free up its old UD Service Request
7774 Handle (by calling *it_ud_service_request_handle_free*), and create a new UD Service Request
7775 Handle by passing the *dst_path* returned in the IT_CM_MSG_UD_SERVICE_REPLY_EVENT
7776 to *it_ud_service_request_handle_create* to create a new Handle.

7777 No ordering guarantee exists between generation of IT_CM_MSG_EVENT_STREAM Events
7778 and generation of IT_DTO_EVENT_STREAM Events. For example, when an Endpoint is
7779 disconnected, the IT_CM_MSG_CONN_DISCONNECT_EVENT can be generated on the
7780 IT_CM_MSG_EVENT_STREAM Event Stream either before or after the flushed Events (i.e.,
7781 Events with an IT_DTO_ERR_FLUSHED status) for pending DTOs are generated on the
7782 IT_DTO_EVENT_STREAM Event Stream.

7783 **SEE ALSO**

7784 *it_event_t, it_evd_create(), it_evd_wait(), it_ep_modify(), it_listen_create(), it_ep_accept(),*
7785 *it_ep_disconnect(), it_reject(), it_address_handle_create(),*
7786 *it_ud_service_request_handle_free(), it_path_t, it_ia_info_t, it_ep_attributes_t*

it_cm_req_events

7787

7788 NAME

7789 Communication Management Request Events – definitions for Connection Request and
7790 Unreliable Datagram service resolution request communication management Events

7791 SYNOPSIS

```
7792 #include <it_api.h>
7793
7794 typedef struct {
7795     it_event_type_t          event_number;
7796     it_evd_handle_t         evd;
7797     it_cn_est_identifier_t   cn_est_id;
7798     it_conn_qual_t          conn_qual;
7799     it_net_addr_t           source_addr;
7800     size_t                   spigot_id;
7801     uint32_t                 max_message_size;
7802     uint32_t                 rdma_read_ird;
7803     uint32_t                 rdma_read_ord;
7804     unsigned char            private_data[IT_MAX_PRIV_DATA];
7805     it_boolean_t             private_data_present;
7806 } it_conn_request_event_t;
7807
7808 typedef struct {
7809     it_event_type_t          event_number;
7810     it_evd_handle_t         evd;
7811     it_ud_svc_req_identifier_t ud_svc_req_id;
7812     it_conn_qual_t          conn_qual;
7813     it_net_addr_t           source_addr;
7814     size_t                   spigot_id;
7815     unsigned char            private_data[IT_MAX_PRIV_DATA];
7816     it_boolean_t             private_data_present;
7817 } it_ud_svc_request_event_t;
```

7818 DESCRIPTION

7819 The Communication Management Request Event Stream, IT_CM_REQ_EVENT_STREAM,
7820 generates Events when an incoming Connection Request Event or incoming Unreliable
7821 Datagram service resolution request occurs.

7822 Incoming Connection Request Events are represented by the *it_conn_request_event_t* structure
7823 and incoming Unreliable Datagram Service Resolution requests are represented by the
7824 *it_ud_svc_request_event_t* structure.

7825 The *it_conn_request_event_t* structure has the following members:

7826	<i>event_number</i>	Identifier of the Event type. Valid values: 7827 IT_CM_REQ_CONN_REQUEST_EVENT
7828	<i>evd</i>	Handle for the Event Dispatcher where the Event was queued.
7829	<i>cn_est_id</i>	Identifier for the Connection establishment Event.
7830	<i>conn_qual</i>	Connection Qualifier on which request was received.

7831	<i>source_addr</i>	Source address of requestor.
7832	<i>spigot_id</i>	Local Spigot on which the request was received.
7833	<i>max_message_size</i>	Largest message supported on Connection by the requesting remote EP. Only valid if <i>it_ia_info.max_message_size_support</i> is IT_TRUE.
7834		
7835		
7836	<i>rdma_read_ird</i>	Maximum number of incoming simultaneous RDMA Read operations supported by the requesting remote EP. Only valid if <i>it_ia_info.ird_ord_ia_support</i> is IT_TRUE. Not valid if IT_LISTEN_SUPPRESS_IRD_ORD is set for <i>it_listen_create</i> .
7837		
7838		
7839		
7840	<i>rdma_read_ord</i>	Maximum number of outgoing simultaneous RDMA Read operations supported by the requesting remote EP. Only valid if <i>it_ia_info.ird_ord_ia_support</i> is IT_TRUE. Not valid if IT_LISTEN_SUPPRESS_IRD_ORD is set for <i>it_listen_create</i> .
7841		
7842		
7843		
7844	<i>private_data</i>	Private Data buffer.
7845	<i>private_data_present</i>	When it has the value IT_TRUE, then Private Data is present in the <i>private_data</i> buffer above.
7846		
7847	The <i>it_ud_svc_request_event_t</i> structure has the following members:	
7848	<i>event_number</i>	Identifier of the Event type. Valid values: IT_CM_REQ_UD_SERVICE_REQUEST_EVENT
7849		
7850	<i>evd</i>	Handle for the Event Dispatcher where the Event was queued.
7851	<i>ud_svc_req_id</i>	Identifier for the Service Request. Must be passed into the <i>it_ud_service_reply</i> call used to respond.
7852		
7853	<i>conn_qual:</i>	Connection Qualifier on which request was received.
7854	<i>source_addr</i>	Source address of requestor.
7855	<i>spigot_id</i>	Local Spigot on which request was received.
7856	<i>private_data</i>	Private Data buffer.
7857	<i>private_data_present:</i>	When it has the value IT_TRUE then Private Data is present in the <i>private_data</i> buffer above.
7858		

Event Type	Description
IT_CM_REQ_CONN_REQUEST_EVENT	An incoming request for Connection establishment. This Connection Request is identified by the Connection establishment request identifier (<i>cn_est_id</i>) in the Event.
IT_CM_REQ_UD_SERVICE_REQUEST_EVENT	An incoming request for Unreliable Datagram service resolution. This request is identified by the <i>ud_svc_req_id</i> in the Event.

7859 **Table 10: Communication Management Request Event Definitions**

7860 All Events on an IT_CM_REQ_EVENT_STREAM SEVD cause Notification. See *it_evd_create*
7861 for details of Notification.

7862 Default overflow behavior of an IT_CM_REQ_EVENT_STREAM SEVD is overflow
7863 Notification disabled. This default behavior of the SEVD is equivalent to IT_EVD_
7864 OVERFLOW_DEFAULT cleared and IT_EVD_OVERFLOW_NOTIFY cleared. See
7865 *it_evd_create* for details of overflow detection.

7866 **APPLICATION USAGE**

7867 The Consumer should use the *it_event_t* structure if it is desired to wait for both Connection
7868 Request Events (*it_conn_request_event_t*) and Unreliable Datagram service resolution request
7869 Events (*it_ud_svc_request_event_t*) via the same EVD. The *it_event_t* structure is of sufficient
7870 size to hold either Event type.

7871 The Consumer must *it_evd_create* an IT_CM_REQ_EVENT_STREAM Simple EVD and pass
7872 the new EVD and a Connection Qualifier to *it_listen_create* in order to receive
7873 IT_CM_REQ_EVENT_STREAM Events via the *it_evd_wait* or *it_evd_dequeue* calls.

7874 The *private_data_present* field indicates whether Private Data is present in the *private_data*
7875 buffer. It is the Consumer's responsibility to convey the size of the data contained in the Private
7876 Data buffer using their own ULP. Each communication management Event type may have a
7877 different maximum Private Data buffer size. The Consumer can determine the maximum
7878 possible sizes for the Private Data buffers corresponding to each of the Event types from the
7879 *it_ia_info_t* structure (for instance, *connect_private_data_len*).

7880 **SEE ALSO**

7881 *it_event_t*, *it_evd_create()*, *it_evd_wait()*, *it_evd_dequeue()*, *it_ep_modify()*, *it_listen_create()*,
7882 *it_ep_accept()*, *it_ep_disconnect()*, *it_reject()*, *it_ud_service_reply()*, *it_ia_info_t*

7883

7884

it_conn_qual_t

7885 **NAME**

7886 it_conn_qual_t – encapsulates all supported Connection Qualifier types

7887 **SYNOPSIS**

```
7888 #include <it_api.h>
7889
7890 /* Enumerates all the possible Connection Qualifier types supported by
7891    the API. */
7892 typedef enum {
7893
7894     /* IANA (TCP/UDP) Port Number */
7895     IT_IANA_PORT = 0x01,
7896
7897     /* InfiniBand Service ID, as described in Section 12.7.3 of
7898        Volume 1 of the InfiniBand specification. */
7899     IT_IB_SERVICEID = 0x02,
7900
7901     /* VIA Connection Discriminator */
7902     IT_VIA_DISCRIMINATOR = 0x04,
7903
7904     /* iWARP local and remote IP (IANA) port object */
7905     IT_IANA_LR_PORT = 0x08
7906
7907 } it_conn_qual_type_t;
7908
7909 /* Defines the Connection Qualifier format for a VIA "connection
7910    discriminator". The API imposes a fixed upper bound on the
7911    discriminator size. */
7912 #define IT_MAX_VIA_DISC_LEN 64
7913
7914 typedef struct {
7915
7916     /* The total number of bytes in the array below */
7917     /* that are significant. */
7918     uint16_t len;
7919
7920     /* VIA connection discriminator, which is an array of bytes. */
7921     unsigned char discriminator[IT_MAX_VIA_DISC_LEN];
7922
7923 } it_via_discriminator_t;
7924
7925 /* This defines the Connection Qualifier for InfiniBand, which is the
7926    64-bit Service ID. */
7927 typedef uint64_t it_ib_serviceid_t;
7928
7929 /* Defines an additional Connection Qualifier for iWARP that
7930    allows specification of both local and remote IANA ports when
7931    passing this structure to it_ep_connect. */
7932 typedef struct {
7933     uint16_t local;
```

```

7935         uint16_t remote;
7936     } it_iana_lr_port_t;
7937
7938     /* This describes a Connection Qualifier suitable for input to
7939        several routines in the API. */
7940     typedef struct {
7941
7942         /* The discriminator for the union below. */
7943         it_conn_qual_type_t type;
7944
7945         union {
7946
7947             /* IANA Port Number, in network byte order. */
7948             uint16_t port;
7949
7950             /* InfiniBand Service ID, in network byte order. */
7951             it_ib_serviceid_t serviceid;
7952
7953             /* VIA connection discriminator. */
7954             it_via_discriminator_t discriminator;
7955
7956             /* IANA local/remote Port numbers, in network byte order. */
7957             it_iana_lr_port_t lr_port;
7958
7959         } conn_qual;
7960     } it_conn_qual_t;
7961

```

7962 DESCRIPTION

7963 The *it_conn_qual_t* type is used by several routines in the API to encapsulate a Connection
7964 Qualifier. A Connection Qualifier is used by a Consumer on the Active side of the Connection
7965 establishment process in the *it_ep_connect* routine to target the remote Consumer that should be
7966 responding to the Connection establishment attempt. It is used on the Passive side of the
7967 Connection establishment process in the *it_listen_create* routine to steer incoming Connection
7968 Requests to an appropriate EVD for further processing.

7969 Each Spigot on an IA can support one or more types of Connection Qualifier. All Spigots will
7970 support the IANA Port Number type of Connection Qualifier, regardless of which transport the
7971 IA that houses the Spigot is using. Which types of Connection Qualifier a Spigot supports can be
7972 determined using the *it_ia_query* routine.

7973 In order to aid Consumers in writing portable applications that span platforms with different
7974 native byte orders, all Connection Qualifiers that are supported by the API with the exception of
7975 the VIA “connection discriminator” are required to be input to the API in network byte order,
7976 and will be output from the API in network byte order. (The VIA “connection discriminator” is
7977 defined to be an array of bytes, and hence is not affected by which native byte order a platform
7978 uses.)

7979 SEE ALSO

7980 *it_ep_connect()*, *it_listen_create()*, *it_ia_query()*

7981

it_context_t

7982

7983 NAME

7984 `it_context_t` – structure describing a Consumer Context

7985 SYNOPSIS

```
7986 #include <it_api.h>
7987
7988 typedef union {
7989     void *    ptr;
7990     uint64_t  index;
7991 } it_context_t;
```

7992 DESCRIPTION

7993 The `it_context_t` union describes storage definitions for the Consumer Context associated with
7994 an IT Object Handle.

7995 *ptr* Storage space for an address pointer.

7996 *index* Storage space for an unsigned 64-bit integer.

7997 SEE ALSO

7998 [*it_get_consumer_context\(\)*](#), [*it_set_consumer_context\(\)*](#)

7999

8000

it_dg_remote_ep_addr_t

8001 **NAME**

8002 `it_dg_remote_ep_addr_t` – datagram transport Endpoint address

8003 **SYNOPSIS**

```
8004 #include <it_api.h>
8005
8006 typedef struct
8007 {
8008     it_addr_handle_t    addr;
8009     it_remote_ep_info_t ep_info;
8010 } it_ib_ud_addr_t;
8011
8012 typedef enum
8013 {
8014     IT_DG_TYPE_IB_UD
8015 } it_dg_type_t;
8016
8017 typedef struct
8018 {
8019     it_dg_type_t  type; /* IT_DG_TYPE_IB_UD */
8020     union {
8021         it_ib_ud_addr_t  ud;
8022     } addr;
8023 } it_dg_remote_ep_addr_t;
```

8024 **APPLICABILITY**

8025 The `it_dg_remote_ep_addr_t` data type is applicable only to Endpoints created for the UD
8026 Service Type.

8027 **DESCRIPTION**

8028 For datagram transports, the Endpoint address is specified in DTO operations using the
8029 `it_dg_remote_ep_addr_t` data structure. The structure is intended to allow support of more than
8030 one datagram transport type.

8031 The datagram transport type is specified as `type` in the `it_dg_remote_ep_addr_t` structure. In this
8032 revision of the API, only the InfiniBand Unreliable Datagram transport, `IT_DG_TYPE_IB_UD`,
8033 is supported.

8034 For InfiniBand Unreliable Datagram, the transport-specific Endpoint address is contained in the
8035 `it_ib_ud_addr_t` sub-structure of the `it_dg_remote_ep_addr_t`. The components of the
8036 InfiniBand Unreliable Datagram Endpoint address are:

8037 `addr` An Address Handle created by the Consumer using `it_address_handle_create`.

8038 `ep_info` An Endpoint Info structure (see `it_ep_attributes_t`) containing the Endpoint ID,
8039 `ud_ep_id`, and the Endpoint Key, `ud_ep_key`. The Consumer may make use of
8040 `it_ud_service_request` to obtain `ud_ep_id` and `ud_ep_key` or may obtain them by
8041 their own means.

8042 **EXTENDED DESCRIPTION**

8043 For InfiniBand Unreliable Datagram, the Endpoint ID is equivalent to an InfiniBand QP number
8044 and the Endpoint Key is equivalent to an InfiniBand *Q_key*.

8045 **FUTURE DIRECTIONS**

8046 Support for Reliable Datagram Service Type may be provided in a future revision of this API.

8047 **SEE ALSO**

8048 *it_post_sendto()*, *it_post_recvfrom()*, *it_address_handle_create()*, *it_ud_service_request()*,
8049 *it_ep_attributes_t*

8050

8051

it_dto_cookie_t

8052 **NAME**

8053 `it_dto_cookie_t` – definition of implementation-opaque Consumer cookie

8054 **SYNOPSIS**

8055 `#include <it_api.h>`

8056

8057 `typedef uint64_t it_dto_cookie_t;`

8058 **DESCRIPTION**

8059 *it_dto_cookie_t* is an object that can be provided by the Consumer on every DTO or RMR
8060 operation and is returned to the Consumer in the corresponding DTO Completion Event (see
8061 [it_dto_events](#)) if a DTO Completion Event is generated (see [it_dto_flags_t](#)). The *it_dto_cookie_t*
8062 object is opaque to the Implementation and is returned unchanged to the Consumer in the DTO
8063 Completion Event corresponding to the posted DTO or RMR.

8064 **SEE ALSO**

8065 [it_dto_events](#), [it_dto_flags_t](#), [it_post_send\(\)](#), [it_post_sendto\(\)](#), [it_post_recv\(\)](#),

8066 [it_post_recvfrom\(\)](#), [it_post_rdma_read\(\)](#), [it_post_rdma_write\(\)](#), [it_rmr_link\(\)](#), [it_rmr_unlink\(\)](#)

8067

8068

8069 **NAME**

8070 DTO and RMR Link/Unlink Completion Event types

8071 **SYNOPSIS**

```

8072 #include <it_api.h>
8073
8074 typedef enum {
8075     IT_UD_IB_GRH_PRESENT = 0x01
8076 } it_dto_ud_flags_t;
8077
8078 typedef struct {
8079     it_event_type_t event_number;
8080     it_evd_handle_t evd;
8081     it_ep_handle_t ep;
8082     it_dto_cookie_t cookie;
8083     it_dto_status_t dto_status;
8084     uint32_t transferred_length;
8085 } it_dto_cmpl_event_t;
8086
8087 typedef struct {
8088     it_event_type_t event_number;
8089     it_evd_handle_t evd;
8090     it_ep_handle_t ep;
8091     it_dto_cookie_t cookie;
8092     it_dto_status_t dto_status;
8093     uint32_t transferred_length;
8094     it_dto_ud_flags_t flags;
8095     it_ud_ep_id_t ud_ep_id;
8096     it_path_t src_path;
8097 } it_all_dto_cmpl_event_t;

```

8098 **APPLICABILITY**

8099 The *it_dto_cmpl_event_t* data type is applicable only to Endpoints created for the RC Service
8100 Type. The *it_all_dto_cmpl_event_t* data type is applicable to Endpoints of any Service Type.

8101 **DESCRIPTION**

8102 The DTO Completion Event Stream, *IT_DTO_EVENT_STREAM*, generates Events for all Data
8103 Transfer Operation completions as well as RMR Link and Unlink completions.

8104 Unreliable Datagram Receive Completion Events provide additional data beyond that of the
8105 other DTO Completion Events. The additional data is large enough to warrant defining a much
8106 smaller Event structure for all other DTO operations usable by Consumers interested in
8107 conserving the memory footprint of their application.

8108 With the exception of UD Receive completions, all DTO completions, including RMR Links
8109 and Unlinks, can be represented by the *it_dto_cmpl_event_t* structure.

8110 UD Receive completions require use of the *it_all_dto_cmpl_event_t* structure. Consumers
8111 wishing to receive UD Receive and Send Completion Events on one Simple EVD or wishing to
8112 handle all possible DTO completions with one Simple EVD must use the

8113 *it_all_dto_cmpl_event_t* structure or the encompassing *it_event_t* structure (see *it_event_t*).
8114 Failure to use the *it_all_dto_cmpl_event_t* structure or *it_event_t* structure for UD Receive
8115 Completion Events can result in program termination.

8116 The *it_dto_cmpl_event_t* structure has the following members:

8117 *event_number* Identifier of the Event type. Valid values:
8118 IT_DTO_SEND_CMPL_EVENT,
8119 IT_DTO_RC_RECV_CMPL_EVENT,
8120 IT_DTO_RDMA_WRITE_CMPL_EVENT,
8121 IT_DTO_RDMA_READ_CMPL_EVENT,
8122 IT_RMR_BIND_CMPL_EVENT,
8123 IT_RMR_LINK_CMPL_EVENT

8124 *evd* Handle for the Event Dispatcher where the Event was queued.

8125 *ep* For all Event types except IT_DTO_RC_RECV_CMPL_EVENT, this is
8126 the Handle for the Endpoint on which the DTO was posted. For the
8127 IT_DTO_RC_RECV_CMPL_EVENT, this is the Handle for the Endpoint
8128 targeted by an incoming Send operation that resulted in the completion of a
8129 Receive DTO.

8130 *cookie* Cookie that the Consumer associated with the DTO when it was posted.
8131 See *it_dto_cookie_t* for details.

8132 *dto_status* Status of completed DTO.

8133 *transferred_length* Length of transferred message.

8134 See *it_dto_status_t* for values and definition of *dto_status*.

8135 The *transferred_length* field indicates the amount of data transferred in a Receive operation. The
8136 content of this field is undefined for Send, RDMA Read, RDMA Write, RMR Link, and RMR
8137 Unlink operations. This field is also only valid if *dto_status* is IT_DTO_SUCCESS; otherwise,
8138 the contents are undefined.

8139 The *it_all_dto_cmpl_event_t* structure has the following additional members:

8140 *flags:* Flags indicating additional service specific information.

8141 *ud_ep_id* Remote Endpoint ID from incoming datagram.

8142 *src_path* Partial Source Path information from incoming datagram.

8143 The IT_DTO_UD_RECV_CMPL_EVENT *event_number* is an additional valid value only for
8144 the *it_all_dto_cmpl_event_t* structure or *it_event_t* structure.

8145 The *flags* parameter indicates whether or not the InfiniBand Global Routing Header (GRH) is
8146 present in the first 40 bytes of the message payload. If the GRH is present, the IT_UD_IB_
8147 GRH_PRESENT bit will be set in *flags*. If the GRH is not present (IT_UD_IB_GRH_PRESENT
8148 bit cleared in *flags*), the first 40 bytes of the payload are undefined.

8149 For an IT_DTO_UD_RECV_CMPL_EVENT, the *transferred_length* field includes the length of
8150 the transferred message plus 40 bytes regardless of the IT_UD_IB_GRH_PRESENT bit value.

8151 The remote Endpoint ID, *ud_ep_id*, is derived from the incoming datagram. See
8152 [it_ep_attributes_t](#) for more details.

8153 Partial Source address information is returned in a datagram Completion Event in the *src_path*
8154 structure element. See Application Usage below.

8155 The *src_path* can hold more information than is returned in the Completion Event. The members
8156 of the *it_path_t* structure that are pertinent to a datagram Completion Event are listed in the table
8157 below. For each member, the corresponding Infiniband datagram addressing information that the
8158 member corresponds to is also identified. For a detailed explanation of the semantics associated
8159 with the Datagram addressing information, see Chapter 11.4.2.1 Poll For Completion in [IB-
8160 R1.2].

it_path_t Member	Unreliable Datagram Completion Addressing Information
<i>ib.sl</i>	Service level
<i>ib.remote_port_lid</i>	Source LID

8161
8162 IT_DTO_EVENT_STREAM Events may or may not cause Notification depending on the use of
8163 DTO flags (see [it_dto_flags_t](#)). See [it_evd_create](#) for details of Notification.

8164 Default overflow behavior of an IT_DTO_EVENT_STREAM SEVD is overflow Notification
8165 enabled (for an IA that supports DTO EVD overflow detection, see below). The behavior of the
8166 SEVD is equivalent to IT_EVD_OVERFLOW_DEFAULT cleared and
8167 IT_EVD_OVERFLOW_NOTIFY set. Once an IT_DTO_EVENT_STREAM SEVD overflows,
8168 it cannot be rearmed. See [it_evd_create](#) for details of overflow detection.

8169 Overflow of an IT_DTO_EVENT_STREAM SEVD is catastrophic for the associated Endpoint
8170 or Endpoints. Each Endpoint is transitioned to the IT_EP_STATE_NONOPERATIONAL state
8171 as defined in [it_ep_state_t](#). The Endpoints are unrecoverable; [it_ep_free](#) must be called for all
8172 Endpoints sharing the same SEVD on which the overflow occurred.

8173 Overflow detection of DTO EVDs is only supported when the *dto_evd_overflow_detection* flag
8174 found in [it_ia_info_t](#) is IT_TRUE.

8175 APPLICATION USAGE

8176 Within an IT_DTO_UD_RECV_CMPL_EVENT Event, the *src_path* member returned contains
8177 insufficient information to identify the remote Endpoint. To resolve the remote Endpoint Path,
8178 the user should pass *src_path* returned in this Event to [it_address_handle_create](#) with the
8179 IT_AH_PATH_COMPLETE bit cleared. [it_address_handle_create](#) will complete the resolution
8180 of the Path.

8181 Overflow of an EVD containing the DTO Event Stream can result in indeterminate behavior
8182 when the IA does not support overflow detection (*dto_evd_overflow_detection* is IT_FALSE).
8183 On such an IA, the Consumer should structure their ULP to avoid any possibility of this
8184 occurring. Indeterminate behavior is constrained to the EVD concerned; overflow on a DTO
8185 Event Stream associated with a particular EVD must not affect any other EVD or Endpoint
8186 associated with other EVDs.

8187 No ordering guarantee exists between generation of IT_CM_MSG_EVENT_STREAM Events
8188 and generation of IT_DTO_EVENT_STREAM Events. For example, when an Endpoint is
8189 disconnected, the IT_CM_MSG_CONN_DISCONNECT_EVENT can be generated on the

8190 IT_CM_MSG_EVENT_STREAM Event Stream either before or after the flushed Events (i.e.,
8191 Events with an IT_DTO_ERR_FLUSHED status) for pending DTOs are generated on the
8192 IT_DTO_EVENT_STREAM Event Stream.

8193 **FUTURE DIRECTIONS**

8194 An additional detailed status code may be added to the DTO Completion Event data structures.

8195 **SEE ALSO**

8196 *it_post_send(), it_post_sendto(), it_post_recv(), it_post_recvfrom(), it_post_rdma_read(),*
8197 *it_post_rdma_write(), it_dto_status_t, it_dto_flags_t, it_event_t, it_evd_create(), it_evd_wait(),*
8198 *it_address_handle_create(), it_ep_state_t, it_ep_free(), it_ep_reset(), it_path_t, it_dto_cookie_t,*
8199 *it_ia_info_t*

8200

8201

it_dto_flags_t

8202 **NAME**

8203 *it_dto_flags_t* – DTO flags for Send, Receive, RDMA Read, RDMA Write, RMR Link, and
8204 RMR Unlink operations

8205 **SYNOPSIS**

```
8206 #include <it_api.h>
8207
8208 typedef enum
8209 {
8210     /* If flag set, completion generates a local event. */
8211     IT_COMPLETION_FLAG = 0x01,
8212
8213     /* If flag set, completion causes local Notification. */
8214     IT_NOTIFY_FLAG = 0x02,
8215
8216     /* If flag set, receipt of DTO at remote will cause Notification
8217        at remote. */
8218     IT_SOLICITED_WAIT_FLAG = 0x04,
8219
8220     /* If flag set, DTO processing will not start if
8221        previously posted RDMA Reads are not complete. */
8222     IT_BARRIER_FENCE_FLAG = 0x08,
8223 } it_dto_flags_t;
```

8224 **DESCRIPTION**

8225 *it_dto_flags* Flags for posted DTOs: Send, Receive, RDMA Read, RDMA Write, RMR Link, and
8226 RMR Unlink.

8227 Values for *it_dto_flags* are constructed by a bitwise-inclusive OR of flags from the following
8228 discussion.

8229 Any combination of the following may be used subject to restrictions as noted:

8230 **IT_COMPLETION_FLAG**

8231 If set, generate a Completion Event for this DTO, else do not generate a Completion Event
8232 unless there is an error.

8233 If not set, then the completion of a subsequent DTO on the same Work Queue of the same
8234 Endpoint with this flag set or with error completion will indicate the successful completion of
8235 prior DTO(s) with this flag cleared.

8236 **Restrictions**

8237 *IT_COMPLETION_FLAG* may be set or cleared only for Send, RDMA Write, RDMA Read,
8238 RMR Link, and RMR Unlink operations on a Reliable Connection Service Type, and may be set
8239 or cleared only for Send operation on an Unreliable Datagram Service Type.

8240 IT_COMPLETION_FLAG must be set for all Receive DTO operations on all Service Types
8241 (Reliable Connection and Unreliable Datagram). Posting a Receive DTO operation with
8242 IT_COMPLETION_FLAG cleared is an error.

8243 **IT_NOTIFY_FLAG**

8244 If set, generate Notification of completion of the DTO/RMR operation. If there is an error,
8245 Notification of completion will be generated regardless of the IT_NOTIFY_FLAG value.

8246 **Restrictions**

8247 IT_NOTIFY_FLAG may be set or cleared on all DTO and RMR operations on a Reliable
8248 Connected Service Type, and may be set or cleared on Send and Receive operations on an
8249 Unreliable Datagram Service Type. It is an error to set IT_NOTIFY_FLAG if
8250 IT_COMPLETION_FLAG is clear.

8251 A completion will be generated with the Notification for a Receive DTO if the matching
8252 received Send DTO had been posted at the remote with the IT_SOLICITED_WAIT_FLAG set
8253 regardless of the IT_NOTIFY_FLAG of the posted Receive DTO.

8254 **IT_SOLICITED_WAIT_FLAG**

8255 If set, the Send DTO operation will request completion Notification for the matching Receive on
8256 the other side of the Connection or, for Unreliable Datagram, for the matching Receive at the
8257 remote datagram Endpoint.

8258 **Restrictions**

8259 IT_SOLICITED_WAIT_FLAG is supported only for Send operations for all Service Types. It is
8260 an error to specify IT_SOLICITED_WAIT_FLAG on other operations.

8261 If set, requests Notification of completion of the matching remote Receive DTO regardless of
8262 the value of the IT_NOTIFY_FLAG on the Receive DTO.

8263 **IT_BARRIER_FENCE_FLAG**

8264 If set, then the DTO/RMR operation will not be started until all previously posted RDMA Read
8265 requests to the Endpoint have been completed.

8266 **Restrictions**

8267 If the service does not support RDMA Read, it is an error to set this flag. Specifically, it is an
8268 error to set IT_BARRIER_FENCE_FLAG on a DTO on UD service.

8269 IT_BARRIER_FENCE_FLAG must be cleared for all Receive DTO operations on any Service
8270 Types. Posting a Receive DTO operation with IT_BARRIER_FENCE_FLAG set is an error.

8271 **EXTENDED DESCRIPTION**

8272

8273

The following table lists all DTO and RMR operations and details the legal *it_dto_flags* values for each.

DTO or RMR Operation	Legal <i>it_dto_flags</i> Combinations
<i>it_post_send</i>	All possible combinations subject to the constraint that IT_NOTIFY_FLAG may only be set if IT_COMPLETION_FLAG is also set. All flags cleared is a legal value.
<i>it_post_sendto</i>	IT_BARRIER_FENCE_FLAG may not be used. All other possible combinations of the remaining flags are legal subject to the constraint that IT_NOTIFY_FLAG may only be set if IT_COMPLETION_FLAG is also set. All flags cleared is a legal value.
<i>it_post_recv</i>	IT_COMPLETION_FLAG must be specified. IT_BARRIER_FENCE_FLAG may not be used. IT_SOLICITED_WAIT_FLAG may not be used. All other possible combinations of the remaining flags are legal.
<i>it_post_recvfrom</i>	IT_COMPLETION_FLAG must be specified. IT_BARRIER_FENCE_FLAG may not be used. IT_SOLICITED_WAIT_FLAG may not be used. All other possible combinations of the remaining flags are legal.
<i>it_post_rdma_read</i>	IT_SOLICITED_WAIT_FLAG may not be used. All other possible combinations are legal subject to the constraint that IT_NOTIFY_FLAG may only be set if IT_COMPLETION_FLAG is also set. All flags cleared is a legal value.
<i>it_post_rdma_read_to_rmr</i>	IT_SOLICITED_WAIT_FLAG may not be used. All other possible combinations are legal subject to the constraint that IT_NOTIFY_FLAG may only be set if IT_COMPLETION_FLAG is also set. All flags cleared is a legal value.
<i>it_post_rdma_write</i>	IT_SOLICITED_WAIT_FLAG may not be used. All other possible combinations are legal subject to the constraint that IT_NOTIFY_FLAG may only be set if IT_COMPLETION_FLAG is also set. All flags cleared is a legal value.
<i>it_rmr_link</i>	IT_SOLICITED_WAIT_FLAG may not be used. All other possible combinations are legal subject to the constraint that IT_NOTIFY_FLAG may only be set if IT_COMPLETION_FLAG is also set. All flags cleared is a legal value.
<i>it_rmr_unlink</i>	IT_SOLICITED_WAIT_FLAG may not be used. All other possible combinations are legal subject to the constraint that IT_NOTIFY_FLAG may only be set if IT_COMPLETION_FLAG is also set. All flags cleared is a legal value.

8274

8275

The following table lists the DTOs on each Service Type on which each flag value is supported.

it_dto_flags_t Value	Supported DTO on RC	Supported DTO on UD
IT_COMPLETION_FLAG	Send, RDMA Read, RDMA Write, RMR Link, RMR Unlink	<i>Recvfrom</i>

it_dto_flags_t Value	Supported DTO on RC	Supported DTO on UD
IT_NOTIFY_FLAG	Send, Recv, RDMA Read, RDMA Write, RMR Link, RMR Unlink	<i>Sendto, Recvfrom</i>
IT_SOLICITED_WAIT_FLAG	Send	<i>Sendto</i>
IT_BARRIER_FENCE_FLAG	Send, RDMA Read, RDMA Write, RMR Link, RMR Unlink	N/A

8276
8277
8278

As stated in the Description section, IT_COMPLETION_FLAG must be set on *Recv* and *Recvfrom* DTOs.

8279
8280
8281

As stated in the Description section, IT_BARRIER_FENCE_FLAG must be cleared on *Recv* DTOs for RC and must be cleared on *Sendto* DTOs as well as cleared on *Recvfrom* DTOs for UD.

8282
8283

For the Infiniband transport, the following table maps the values in the *it_dto_flags_t* enumeration to their corresponding concepts as specified in [IB-R1.2].

it_dto_flags_t Value	IB Concept
IT_COMPLETION_FLAG	May be implemented using signaled/unsigaled Completions concept.
IT_NOTIFY_FLAG	None but can be supported by Implementation.
IT_SOLICITED_WAIT_FLAG	Solicited Event
IT_BARRIER_FENCE_FLAG	Fence Indicator

8284
8285
8286

For the VIA transport, the following table maps the values in the *it_dto_flags_t* enumeration to their corresponding concepts as documented in [VIA-V1.0].

it_dto_flags_t Value	VIA Concept
IT_COMPLETION_FLAG	May be implemented by associating completion queues with work queues
IT_NOTIFY_FLAG	VipSendNotify, VipRecvNotify, VipCQNotify
IT_SOLICITED_WAIT_FLAG	Not applicable
IT_BARRIER_FENCE_FLAG	Queue Fence Bit

8287
8288
8289

For the iWARP Transport, the following table maps the values in the *it_dto_flags_t* enumeration to their corresponding concepts as documented in [RDMAC-VERBS].

it_dto_flags_t Value	iWARP Concept
IT_COMPLETION_FLAG	May be implemented using signaled/unsigaled Completions concept.

it_dto_flags_t Value	iWARP Concept
IT_NOTIFY_FLAG	None but can be supported by Implementation.
IT_SOLICITED_WAIT_FLAG	Solicited Event
IT_BARRIER_FENCE_FLAG	Read Fence Indicator

8290 **APPLICATION USAGE**

8291 The IT_COMPLETION_FLAG allows “per-Work-Request Completion Suppression”. That is, it
8292 controls whether or not a Completion Event is generated and posted to the Event Dispatcher for
8293 a successful Work Request (DTO).

8294 The IT_NOTIFY_FLAG allows “per-Work-Request Notification Suppression”. That is, it
8295 controls whether or not Notification occurs once a Completion Event has been generated for a
8296 successful Work Request (DTO). It is possible to generate a Completion Event and not generate
8297 a Notification.

8298 See Application Usage in *it_ep_state_t* for discussion of flushing DTO completions when the
8299 DTOs have IT_COMPLETION_FLAG cleared.

8300 When posting Send DTOs with Completion Suppression (IT_COMPLETION_FLAG cleared) to
8301 an Endpoint, the Consumer is advised to enqueue at least one DTO with
8302 IT_COMPLETION_FLAG set in every *max_request_dtos* number of postings to the Endpoint,
8303 in order to preserve the capability to recover from failures.

8304 **SEE ALSO**

8305 *it_post_send()*, *it_post_sendto()*, *it_post_recv()*, *it_post_recvfrom()*, *it_post_rdma_read()*,
8306 *it_post_rdma_write()*, *it_rmr_link()*, *it_rmr_unlink()*, *it_dto_status_t*, *it_dto_events*,
8307 *it_ep_state_t*

8308

8309

it_dto_status_t8310 **NAME**

8311 it_dto_status_t – definition of DTO and RMR completion status

8312 **SYNOPSIS**

```

8313 #include <it_api.h>
8314
8315 typedef enum {
8316     IT_DTO_SUCCESS                = 0,
8317     IT_DTO_ERR_LOCAL_LENGTH       = 1,
8318     IT_DTO_ERR_LOCAL_EP          = 2,
8319     IT_DTO_ERR_LOCAL_PROTECTION  = 3,
8320     IT_DTO_ERR_FLUSHED           = 4,
8321     IT_RMR_OPERATION_FAILED       = 5,
8322     IT_DTO_ERR_BAD_RESPONSE      = 6,
8323     IT_DTO_ERR_REMOTE_ACCESS     = 7,
8324     IT_DTO_ERR_REMOTE_RESPONDER  = 8,
8325     IT_DTO_ERR_TRANSPORT         = 9,
8326     IT_DTO_ERR_RECEIVER_NOT_READY = 10,
8327     IT_DTO_ERR_PARTIAL_PACKET    = 11,
8328     IT_DTO_ERR_LOCAL_MM_OPERATION = 12
8329 } it_dto_status_t;

```

8330 **DESCRIPTION**

8331 Any successfully initiated Data Transfer Operation (i.e., Send, Receive, RDMA Read, or RDMA
 8332 Write) or RMR operation (i.e., RMR Link or RMR Unlink) can return its completion status
 8333 asynchronously via an Event enqueued on an SEVD. For some DTOs, the Consumer can control
 8334 whether an Event is generated via the IT_COMPLETION_FLAG (see *it_dto_flags_t*). If an
 8335 Event is generated, the completion status is contained in the *it_dto_status_t*.

8336 If the completion status is anything other than IT_DTO_SUCCESS for a Reliable Connected
 8337 Endpoint, the Connection will be broken.

8338 The table below enumerates all of the allowed values for *it_dto_status_t*. For each value, a
 8339 description of what the value means and the applicable operations on RC and on UD is shown.

it_dto_status_t Value	Description	Applicable to Operations	
		RC	UD
IT_DTO_SUCCESS	The DTO completed successfully.	Send Recv RDMA Read RDMA Write RMR Link RMR Unlink	Sendto Recvfrom
IT_DTO_ERR_LOCAL_LENGTH	The length of the incoming DTO was larger than <i>max_dto_payload_size</i> for the Endpoint	Recv	Recvfrom

it_dto_status_t Value	Description	Applicable to Operations	
		RC	UD
IT_DTO_ERR_LOCAL_LENGTH	The length of the outgoing DTO was larger than <i>max_dto_payload_size</i> for the Endpoint.	Send RDMA Read RDMA Write	Sendto
IT_DTO_ERR_LOCAL_LENGTH	The total length of the buffers associated with a Receive DTO was too small to hold all the incoming data from a Send DTO.	Recv	Recvfrom
IT_DTO_ERR_LOCAL_EP	An internal local Endpoint consistency error was detected while processing a DTO.	Send Recv RDMA Read RDMA Write RMR Link RMR Unlink	Sendto Recvfrom
IT_DTO_ERR_LOCAL_PROTECTION	One of the local segments in the DTO or the local Endpoint caused a protection violation when the DTO was processed, and the possible causes are as follows: <ul style="list-style-type: none"> • The LMR handle or local RMR handle was invalid. • The LMR or local RMR was not in the linked state. • The address range specified by <i>addr</i> and <i>length</i> was outside the bounds of the LMR or local RMR. • The Protection Zone associated with the LMR or local RMR didn't match the Protection Zone of the Endpoint to which the DTO was posted. • For <i>it_post_rdma_read_to_rmr</i>, the local Narrow RMR was associated with an Endpoint different from the Endpoint to which the DTO was posted. • An attempt was made to access the LMR or local RMR in a way that conflicted with its access permissions. • For the RC service, the local Endpoint was not enabled for the incoming RDMA operation. 	Send Recv RDMA Read RDMA Write	Sendto Recvfrom
IT_DTO_ERR_FLUSHED	The Endpoint entered the IT_EP_STATE_NONOPERATIONAL state before processing of the DTO could begin.	Send Recv RDMA Read RDMA Write RMR Link RMR Unlink	Sendto Recvfrom

it_dto_status_t Value	Description	Applicable to Operations	
		RC	UD
IT_RMR_OPERATION_FAILED	<p>An RMR operation failed due to a protection violation. Possible causes for this error are as follows.</p> <p>For <i>it_rmr_link</i>:</p> <ul style="list-style-type: none"> • The RMR handle was invalid. • The Protection Zones associated with the RMR, LMR, and EP to which the operation was posted didn't match. • The RMR was a Narrow RMR that was already linked. • The address range specified by the <i>addr</i> and <i>length</i> arguments was outside the bounds of the LMR. • Remote write access was requested for the RMR, but the LMR did not allow local write access (InfiniBand only). • An attempt was made to grant access through the RMR that conflicted with the access allowed by either the LMR or the EP. <p>For <i>it_rmr_unlink</i>:</p> <ul style="list-style-type: none"> • The RMR handle was invalid • The Protection Zones associated with the RMR and EP to which the operation was posted didn't match. • The RMR was a linked Narrow RMR and the EP to which the operation was posted didn't match the EP associated with the RMR. 	RMR Link RMR Unlink	N/A
IT_DTO_ERR_BAD_RESPONSE	The DTO operation that was posted to the Work queue was responded to with an unexpected transport opcode.	Send RDMA Read RDMA Write	N/A

it_dto_status_t Value	Description	Applicable to Operations	
		RC	UD
IT_DTO_ERR_REMOTE_ACCESS	<p>For Implementations supporting end-to-end completions only. For iWARP, end-to-end completions are supported for RDMA Read only. A protection violation was detected at the remote end when processing an RDMA DTO operation. Possible causes for this error are as follows:</p> <ul style="list-style-type: none"> • The Protection Zone associated with the remote buffer didn't match the Protection Zone of the remote Endpoint. • The remote buffer was a Narrow RMR that was linked via a remote Endpoint not associated with the Connection. • The address range specified by <i>rdma_addr</i> and the length implicitly given by <i>local_segments</i> and <i>num_segments</i> was outside the bounds of the remote buffer. • An attempt was made to access the remote buffer in a way that conflicted with its access permissions. • An attempt was made to access the remote buffer in a way that conflicted with the access permissions of the remote Endpoint. • The remote Endpoint was not enabled for the incoming RDMA operation. 	RDMA Read RDMA Write	N/A
IT_DTO_ERR_REMOTE_RESPONDER	<p>For Implementations supporting end-to-end completions only. A DTO operation could not be completed at the remote end. Possible causes for this error include the remote Endpoint experiencing a condition causing an IT_DTO_ERR_LOCAL_EP or IT_DTO_ERR_LOCAL_LENGTH error to be returned.</p>	Send RDMA Read RDMA Write	N/A
IT_DTO_ERR_TRANSPORT	<p>The underlying transport could not successfully transfer the data for the DTO operation. Possible causes for this error include the remote IA not responding, the DTO data was corrupted in the process of transmission, a LLP error, or the network fabric being used by the IA is broken.</p>	Send Receive RDMA Read RDMA Write	N/A
IT_DTO_ERR_RECEIVER_NOT_READY	<p>The DTO operation could not be processed because the responding side repeatedly indicated that it had no resources to do so.</p>	Send RDMA Read RDMA Write	N/A
IT_DTO_ERR_PARTIAL_PACKET	<p>The data delivered by the Receive DTO was truncated. The contents of the receiver's buffer are unspecified.</p>	Receive	N/A

it_dto_status_t Value	Description	Applicable to Operations	
		RC	UD
IT_DTO_ERR_LOCAL_MM_OPERATION	<p>A local memory management operation failed due to one of the following causes.</p> <p>For <i>it_rmr_link</i>:</p> <ul style="list-style-type: none"> The LMR to link to used Relative Addressing. A Narrow RMR was provided and the <i>length</i> argument was zero (only for InfiniBand). 	RMR Link	N/A

8340 **EXTENDED DESCRIPTION**

8341 For the InfiniBand Transport, the following table maps the values in the *it_dto_status_t*
8342 enumeration to their corresponding “Completion Return Status” values as specified in Chapter
8343 11 of [IB-R1.1] or [IB-R1.2].

it_dto_status_t Value	IB “Completion Return Status” Name
IT_DTO_SUCCESS	Success
IT_DTO_ERR_LOCAL_LENGTH	Local Length Error
IT_DTO_ERR_LOCAL_EP	Local QP Operation Error
IT_DTO_ERR_LOCAL_PROTECTION	Local Protection Error
IT_DTO_ERR_FLUSHED	Work Request Flushed Error
IT_RMR_OPERATION_FAILED	Memory Window Bind Error
IT_DTO_ERR_BAD_RESPONSE	Bad Response Error
IT_DTO_ERR_REMOTE_ACCESS	Remote Access Error
IT_DTO_ERR_REMOTE_RESPONDER	Remote Operation Error
IT_DTO_ERR_TRANSPORT	Transport Retry Counter Exceeded
IT_DTO_ERR_RECEIVER_NOT_READY	RNR Retry Counter Exceeded
IT_DTO_ERR_PARTIAL_PACKET	(Not applicable to the IB transport.)
IT_DTO_ERR_LOCAL_MM_OPERATION	Memory Management Operation Error

8344 For the iWARP Transport, the following table maps the values in the *it_dto_status_t*
8345 enumeration to the Completion Status Codes in Section 9.5.2 of [VERBS-RDMAC].
8346

it_dto_status_t Value	iWARP “Completion Return Status” Name
IT_DTO_SUCCESS	Success

it_dto_status_t Value	iWARP “Completion Return Status” Name
IT_DTO_ERR_LOCAL_LENGTH	Local Length Error or Base and bounds violation (Receive)
IT_DTO_ERR_LOCAL_EP	Local QP Catastrophic Error, Zero RDMA Read Resources
IT_DTO_ERR_LOCAL_PROTECTION	Invalid STag (DTOs only), Invalid PD ID, Access Rights violation, Base and bounds Violation, Wrap Error
IT_DTO_ERR_FLUSHED	Flushed
IT_RMR_OPERATION_FAILED	Invalid Region (Bind), Invalid Window (Bind), Invalid PD ID (Bind), Access Rights violation (Bind), Base and bounds Violation (Bind)
IT_DTO_ERR_BAD_RESPONSE	(Not applicable to the iWARP Transport.)
IT_DTO_ERR_REMOTE_ACCESS	Remote Termination Error
IT_DTO_ERR_REMOTE_RESPONDER	Remote Termination Error
IT_DTO_ERR_TRANSPORT	Transport Retry Counter Exceeded, or LLP Error
IT_DTO_ERR_RECEIVER_NOT_READY	(Not applicable to the iWARP Transport.)
IT_DTO_ERR_PARTIAL_PACKET	(Not applicable to the iWARP Transport.)
IT_DTO_ERR_LOCAL_MM_OPERATION	STag to Invalidate had Invalid PD or Access Rights, STag Not In Invalid State, Invalid Region (Bind), Invalid Window (Bind)

8347
8348
8349
8350

For the VIA transport, the following table maps the values in the *it_dto_status_t* enumeration to their corresponding bits in the Descriptor Control Segment “Status” field, as documented in the Appendix of [\[VIA-V1.0\]](#).

it_dto_status_t Value	VIA “Status Bit” Name
IT_DTO_SUCCESS	Done
IT_DTO_ERR_LOCAL_LENGTH	Local Length Error
IT_DTO_ERR_LOCAL_EP	Local Format Error
IT_DTO_ERR_LOCAL_PROTECTION	Local Protection Error
IT_DTO_ERR_FLUSHED	Descriptor Flushed
IT_RMR_OPERATION_FAILED	(There is no operation corresponding to RMR Link or RMR Unlink in VIA, but this error can still be returned from an IA that is utilizing the VIA transport. The Implementation synthesizes the RMR operation for VIA.)

it_dto_status_t Value	VIA “Status Bit” Name
IT_DTO_ERR_BAD_RESPONSE	(Not applicable to the VIA transport.)
IT_DTO_ERR_REMOTE_ACCESS	RDMA Protection Error
IT_DTO_ERR_REMOTE_RESPONDER	(Not applicable to the VIA transport.)
IT_DTO_ERR_TRANSPORT	Transport Error
IT_DTO_ERR_RECEIVER_NOT_READY	(Not applicable to the VIA transport.)
IT_DTO_ERR_PARTIAL_PACKET	Partial Packet Error
IT_DTO_ERR_LOCAL_MM_OPERATION	(Not applicable to the VIA transport.)

8351 **SEE ALSO**

8352 *it_post_send(), it_post_sendto(), it_post_recv(), it_post_recvfrom(), it_post_rdma_read(),*
8353 *it_post_rdma_write(), it_rmr_link(), it_rmr_unlink()*

8354

8355

it_ep_attributes_t

8356 **NAME**

8357 it_ep_attributes – Endpoint attributes

8358 **SYNOPSIS**

```
8359 #include <it_api.h>
8360
8361 typedef uint32_t it_ud_ep_id_t;
8362 typedef uint32_t it_ud_ep_key_t;
8363
8364 typedef enum {
8365     IT_EP_PARAM_ALL           = 0x00000001,
8366     IT_EP_PARAM_IA           = 0x00000002,
8367     IT_EP_PARAM_SPIGOT       = 0x00000004,
8368     IT_EP_PARAM_STATE        = 0x00000008,
8369     IT_EP_PARAM_SERV_TYPE    = 0x00000010,
8370     IT_EP_PARAM_PATH         = 0x00000020,
8371     IT_EP_PARAM_PZ           = 0x00000040,
8372     IT_EP_PARAM_REQ_SEVD     = 0x00000080,
8373     IT_EP_PARAM_RECV_SEVD    = 0x00000100,
8374     IT_EP_PARAM_CONN_SEVD    = 0x00000200,
8375     IT_EP_PARAM_RDMA_RD_ENABLE = 0x00000400,
8376     IT_EP_PARAM_RDMA_WR_ENABLE = 0x00000800,
8377     IT_EP_PARAM_MAX_RDMA_READ_SEG = 0x00001000,
8378     IT_EP_PARAM_MAX_RDMA_WRITE_SEG = 0x00002000,
8379     IT_EP_PARAM_MAX_IRD      = 0x00004000,
8380     IT_EP_PARAM_MAX_ORD      = 0x00008000,
8381     IT_EP_PARAM_EP_ID        = 0x00010000,
8382     IT_EP_PARAM_EP_KEY       = 0x00020000,
8383     IT_EP_PARAM_MAX_PAYLOAD   = 0x00040000,
8384     IT_EP_PARAM_MAX_REQ_DTO   = 0x00080000,
8385     IT_EP_PARAM_MAX_RECV_DTO  = 0x00100000,
8386     IT_EP_PARAM_MAX_SEND_SEG  = 0x00200000,
8387     IT_EP_PARAM_MAX_RECV_SEG  = 0x00400000,
8388     IT_EP_PARAM_SRQ          = 0x00800000,
8389     IT_EP_PARAM_SOFT_HI_WATERMARK = 0x01000000,
8390     IT_EP_PARAM_HARD_HI_WATERMARK = 0x02000000
8391 } it_ep_param_mask_t;
8392
8393 /*
8394  * The it_ep_param_mask_t value in the comment beside or
8395  * following each attribute is the mask value used to select
8396  * the attribute in the it_ep_query and it_ep_modify calls
8397  */
8398 typedef struct {
8399     it_boolean_t rdma_read_enable;
8400     /* IT_EP_PARAM_RDMA_RD_ENABLE */
8401     it_boolean_t rdma_write_enable;
8402     /* IT_EP_PARAM_RDMA_WR_ENABLE */
8403     size_t max_rdma_read_segments;
8404     /* IT_EP_PARAM_MAX_RDMA_READ_SEG */
8405     size_t max_rdma_write_segments;
```

```

8406         /* IT_EP_PARAM_MAX_RDMA_WRITE_SEG */
8407         uint32_t rdma_read_ird;
8408         /* IT_EP_PARAM_MAX_IRD */
8409         uint32_t rdma_read_ord;
8410         /* IT_EP_PARAM_MAX_ORD */
8411         it_srq_handle_t srq;
8412         /* IT_EP_PARAM_SRQ */
8413         size_t soft_hi_watermark;
8414         /* IT_EP_PARAM_SOFT_HI_WATERMARK */
8415         size_t hard_hi_watermark;
8416         /* IT_EP_PARAM_HARD_HI_WATERMARK */
8417
8418     } it_rc_only_attributes_t;
8419
8420     #define IT_HARD_HI_WATERMARK_DISABLE ((size_t) -1)
8421
8422     typedef struct {
8423         it_ud_ep_id_t ud_ep_id; /* IT_EP_PARAM_EP_ID */
8424         it_ud_ep_key_t ud_ep_key; /* IT_EP_PARAM_EP_KEY */
8425     } it_remote_ep_info_t;
8426
8427     typedef struct {
8428         it_remote_ep_info_t ep_info;
8429
8430     } it_ud_only_attributes_t;
8431
8432     typedef union {
8433         it_rc_only_attributes_t rc;
8434         it_ud_only_attributes_t ud;
8435     } it_service_attributes_t;
8436
8437     typedef struct {
8438         size_t max_dto_payload_size; /* IT_EP_PARAM_MAX_PAYLOAD */
8439         size_t max_request_dtos; /* IT_EP_PARAM_MAX_REQ_DTO */
8440         size_t max_recv_dtos; /* IT_EP_PARAM_MAX_RECV_DTO */
8441         size_t max_send_segments; /* IT_EP_PARAM_MAX_SEND_SEG */
8442         size_t max_recv_segments; /* IT_EP_PARAM_MAX_RECV_SEG */
8443
8444         it_service_attributes_t srv;
8445     } it_ep_attributes_t;

```

8446 DESCRIPTION

8447 `it_ep_attributes` List of Endpoint attributes. The `it_service_attributes_t` union elements are
8448 discriminated by `service_type` found in the `it_ep_param_t` structure in the
8449 [it_ep_query](#) reference page. Mask values for query and modify of Endpoint
8450 attributes appear as comments to each attribute.

Attribute	Description	Service Type	Modifiable?
<i>max_dto_payload_size</i>	<p>Maximum message transfer size for the Endpoint. It specifies the maximum amount of payload data that Consumer will transfer in a single DTO Send or Receive message in either direction on the Endpoint.</p> <p>For RC only, it also specifies the maximum payload data size for RDMA Reads and Writes posted on the Endpoint.</p>	UD and RC	For RC, only when Endpoint is in the IT_EP_STATE_NONOPERATIONAL or in the IT_EP_STATE_UNCONNECTED states. For UD, only on creation.
<i>max_request_dtos</i>	<p>Maximum number of outstanding Send, Sendto, RDMA Read, RDMA Write DTOs, RMR Link, and RMR Unlink operations combined that a Consumer can submit to the Endpoint. If the Consumer attempts to post more than this number of request DTOs simultaneously, an error will be returned from the <i>it_post_send</i>, <i>it_post_rdma_read</i>, etc., routines.</p>	UD and RC	Subject to the setting of the <i>ep_work_queues_resizable</i> field in the <i>it_ia_info_t</i> for this IA.
<i>max_rcv_dtos</i>	<p>Maximum number of outstanding <i>Recv</i> or <i>Recvfrom</i> DTOs that a Consumer can submit to the Endpoint. If the Consumer attempts to post more than this number of Receive DTOs simultaneously, an error will be returned from the <i>it_post_rcv</i> or <i>it_post_rcvfrom</i> routines.</p> <p>If an Endpoint is created with an associated S-RQ, this attribute is ignored. For such an Endpoint, this attribute shall be returned as zero by <i>it_ep_query</i> and any attempt to modify the attribute with <i>it_ep_modify</i> shall return an error.</p> <p>For an Endpoint having an associated S-RQ, a corresponding <i>max_rcv_dtos</i> attribute exists as an S-RQ attribute.</p>	UD and RC	Subject to the setting of the <i>ep_work_queues_resizable</i> field in the <i>it_ia_info_t</i> for this IA.
<i>max_send_segments</i>	<p>Maximum number of data segments for a local buffer that the Consumer specifies for a posted Send or Sendto DTO for the Endpoint.</p>	UD and RC	Only on creation.

Attribute	Description	Service Type	Modifiable?
<i>max_rcv_segments</i>	<p>Maximum number of data segments for a local buffer that the Consumer specifies for a posted <i>Recv</i> or <i>Recvfrom</i> DTO for the Endpoint.</p> <p>If an Endpoint is created with an associated S-RQ, this attribute is ignored. For such an Endpoint, this attribute shall be returned as zero by <i>it_ep_query</i> and any attempt to modify the attribute with <i>it_ep_modify</i> shall return an error.</p> <p>For an Endpoint having an associated S-RQ, a corresponding <i>max_rcv_segments</i> attribute exists as an S-RQ attribute.</p>	UD and RC	Only on creation.
<i>ud_ep_id</i>	Local Endpoint ID for this Endpoint.	UD only	Never – this is a READ-ONLY attribute.
<i>ud_ep_key</i>	Local Endpoint key for this Endpoint.	UD only	In any state.
<i>rdma_read_enable</i>	Flag allowing Consumer to enable or disable incoming RDMA Read operations on this Endpoint.	RC only	If the <i>it_ia_info_t</i> boolean <i>ep_rdma_enables_modifiable</i> is IT_TRUE, then may be modified in any state. Else, only on creation.
<i>rdma_write_enable</i>	<p>Flag allowing Consumer to enable or disable incoming RDMA Write operations on this Endpoint.</p> <p>For posting RDMA Read DTOs to this Endpoint, this attribute must be IT_TRUE if the IA attribute <i>rdma_read_requires_remote_write</i> equals IT_TRUE.</p>	RC only	If the <i>it_ia_info_t</i> boolean <i>ep_rdma_enables_modifiable</i> is IT_TRUE, then may be modified in any state. Else, only on creation.
<i>max_rdma_read_segments</i>	Maximum number of data segments for a local buffer that the Consumer specifies for a posted RDMA Read DTO for the Endpoint.	RC only	Only on creation.
<i>max_rdma_write_segments</i>	Maximum number of data segments for a local buffer that the Consumer specifies for a posted RDMA Write DTO for the Endpoint.	RC only	Only on creation.

Attribute	Description	Service Type	Modifiable?
<i>rdma_read_ird</i>	Maximum number of incoming RDMA Reads from the remote side of the connected Endpoint that can be outstanding simultaneously.	RC only	When Endpoint is in the IT_EP_STATE_UNCONNECTED or IT_EP_STATE_ACTIVE2_CONNECTION_PENDING state.
<i>rdma_read_ord</i>	Maximum number of outgoing RDMA Reads of the connected Endpoint that can be outstanding simultaneously. May be changed after Endpoint has reached the connected state.	RC only	When Endpoint is in the IT_EP_STATE_UNCONNECTED or IT_EP_STATE_ACTIVE2_CONNECTION_PENDING or IT_EP_STATE_CONNECTED state.
<i>srq</i>	The handle for the Shared Receive Queue (if any) associated with the Endpoint. If the Endpoint has no S-RQ, <i>it_ep_query</i> will return the value IT_NULL_HANDLE for this attribute.	RC only	Only on creation.
<i>soft_hi_watermark</i>	<p>The current value of the Endpoint Soft High Watermark threshold.</p> <p>If the IA does not support the Endpoint Soft High Watermark mechanism, then the Consumer must set this attribute to zero when creating an Endpoint with an associated S-RQ. If the IA does support the Endpoint Soft High Watermark mechanism, if the Consumer sets this attribute to zero when creating the Endpoint the Endpoint Soft High Watermark mechanism shall be disabled so that no Endpoint Soft High Watermark Event shall be generated.</p> <p>If the Consumer sets this attribute to a non-zero value the Endpoint Soft High Watermark mechanism shall be armed so that when/if the number of Receive DTOs in progress for an Endpoint exceeds this value an Endpoint Soft High Watermark Event shall be generated. (See the Extended Description section for a definition of “number of Receive</p>	RC only	In any state.

Attribute	Description	Service Type	Modifiable?
	<p>DTOs in progress for an Endpoint”.)</p> <p>If the Consumer modifies this to any non-zero value (including the existing non-zero value if the Endpoint Soft High Watermark mechanism is already armed) and <i>it_ep_modify</i> returns success, the Endpoint Soft High Watermark mechanism shall be armed/rearmed. Once the mechanism is armed, it shall remain armed until either the Endpoint is destroyed by a call to <i>it_ep_free</i>, or the Endpoint Soft High Watermark threshold is exceeded. If the mechanism is disarmed because the Endpoint Soft High Watermark threshold is exceeded, an Affiliated Asynchronous Event shall be enqueued on the Affiliated Asynchronous Event Stream for the IA that the Endpoint is associated with; see <i>it_affiliated_event_t</i> for details. Once the Endpoint Soft High Watermark mechanism is disarmed, it can only be rearmed by calling <i>it_ep_modify</i> to establish a new <i>soft_hi_watermark</i> value.</p> <p>If the IA does not support the Endpoint Soft High Watermark mechanism, or if the Endpoint does not have an associated S-RQ, <i>it_ep_query</i> shall return zero for this attribute.</p> <p>An error will be returned if the Consumer attempts to set this attribute to a value that is larger than the maximum number of Receive DTOs that can be posted to the associated S-RQ.</p> <p>If the Consumer attempts to set this attribute to a value of zero while the Endpoint Soft High Watermark mechanism is armed, the Implementation shall ignore the attempt to change the value, and the mechanism shall stay armed.</p>		

Attribute	Description	Service Type	Modifiable?
<i>hard_hi_watermark</i>	<p>The current value of the Endpoint Hard High Watermark.</p> <p>If the IA does not support the Endpoint Hard High Watermark mechanism, then the Consumer must set this attribute to zero when creating an Endpoint with an associated S-RQ. If the IA does support the Endpoint Hard High Watermark mechanism, if the Consumer sets this attribute to the distinguished value <code>IT_HARD_HI_WATERMARK_DISABLE</code> the mechanism shall be disabled and the normal rules for when a connection broken Event gets generated for an Endpoint shall apply. If the Consumer sets this attribute to a value other than <code>IT_HARD_HI_WATERMARK_DISABLE</code> when the IA associated with the Endpoint supports the Endpoint Hard High Watermark mechanism, in addition to the normal rules for when a connection broken Event gets generated a connection broken Event shall be generated on the Connection Event Stream associated with the Endpoint when/if the Number of Receive DTOs in progress for the Endpoint exceeds this value, and the Endpoint shall enter the <code>IT_EP_STATE_NONOPERATIONAL</code> state.</p> <p>If the IA does not support the Endpoint Hard High Watermark mechanism, or if the Endpoint does not have an associated S-RQ, <i>it_ep_query</i> shall return zero for this attribute.</p>	RC only	In any state.
<i>ep_state</i>	The current state of the Endpoint.	UD and RC	Never – this is a READ-ONLY attribute.

8451
8452
8453
8454
8455

Since the Implementation is at liberty to allocate more resources than requested by the Consumer, the Consumer is advised to use *it_ep_query* to determine the Implementation-assigned values. All guarantees and warnings are with respect to the Implementation-assigned values.

8456 Exceeding *max_request_dtos* or *max_rcv_dtos* using the post DTO and post RMR operations
8457 will result in the post operation returning an error or completing in error.

8458 Posting more RDMA Read operations than specified in *rdma_read_ord* is not an error and will
8459 have no adverse effects.

8460 **EXTENDED DESCRIPTION**

8461 In the discussion above, the term “number of Receive DTOs in progress for an Endpoint” is
8462 defined to be the number of Receive DTOs which have been consumed on behalf of the
8463 Endpoint and for which data has been placed from incoming Send DTOs, but which have not yet
8464 completed due to out-of-order Send DTO arrival. For the InfiniBand Transport, this number can
8465 be no greater than one. For the iWARP Transport, this number can be greater than one.

8466 **FUTURE DIRECTIONS**

8467 Some new Service Types may be added in the future.

8468 **SEE ALSO**

8469 *it_ep_rc_create()*, *it_ep_ud_create()*, *it_ep_query()*, *it_ep_modify()*, *it_ia_info_t*

it_ep_state_t

8470

8471 **NAME**

8472 it_ep_state_t – RC and UD Endpoint state type definition

8473 **SYNOPSIS**

```
8474 #include <it_api.h>
8475
8476 typedef enum
8477 {
8478     IT_EP_STATE_UNCONNECTED                = 0,
8479     IT_EP_STATE_ACTIVE1_CONNECTION_PENDING = 1,
8480     IT_EP_STATE_ACTIVE2_CONNECTION_PENDING = 2,
8481     IT_EP_STATE_PASSIVE_CONNECTION_PENDING = 3,
8482     IT_EP_STATE_CONNECTED                  = 4,
8483     IT_EP_STATE_NONOPERATIONAL             = 5,
8484     IT_EP_STATE_PASSIVE_WAIT_RDMA_TRANS_REQ = 6
8485 } it_ep_state_rc_t;
8486
8487 typedef enum
8488 {
8489     IT_EP_STATE_UD_NONOPERATIONAL = 0,
8490     IT_EP_STATE_UD_OPERATIONAL    = 1
8491 } it_ep_state_ud_t;
8492
8493 typedef union
8494 {
8495     it_ep_state_rc_t rc;
8496     it_ep_state_ud_t ud;
8497 } it_ep_state_t;
```

8498 **DESCRIPTION**

8499 The following table identifies and describes the RC Endpoint states. The Transport-Independent
8500 Interface (TII) and the Transport-Dependent Interface (TDI) use these states in a consistent way.
8501 For each state, the table lists the API routines that can be legally applied to an RC Endpoint in
8502 that state.

8503 Whenever an Endpoint transitions its state, at most one Event is generated for that transition.
8504 The Endpoints state transitions before the Communication Management Message Event is
8505 enqueued. This guarantees that the Consumer can only dequeue Events after the state transition
8506 has occurred. Subsequent state transitions and their related Events will occur regardless of
8507 whether the Consumer is dequeuing the Events.

8508

Reliable Connection Endpoint States	Description of State	Allowed Calls
IT_EP_STATE_UNCONNECTED	The Endpoint is not Connected to another nor is there a pending Connection establishment related to the Endpoint. The Endpoint is available to be used in a Connection establishment. When an Endpoint is first created by calling <i>it_ep_rc_create</i> it is in this state.	<i>it_ep_accept</i> , <i>it_ep_connect</i> , <i>it_ep_free</i> , <i>it_ep_modify</i> , <i>it_ep_query</i> , <i>it_get_consumer_context</i> , <i>it_get_handle_type</i> , <i>it_post_rcv</i> , <i>it_set_consumer_context</i> <i>it_socket_convert</i>
IT_EP_STATE_ACTIVE1_CONNECTION_PENDING	The Active side Endpoint has initiated a Connection establishment.	<i>it_ep_free</i> , <i>it_ep_modify</i> , <i>it_ep_query</i> , <i>it_get_consumer_context</i> , <i>it_get_handle_type</i> , <i>it_post_rcv</i> , <i>it_set_consumer_context</i> , <i>it_ep_disconnect</i>
IT_EP_STATE_ACTIVE2_CONNECTION_PENDING	The Active side Endpoint has initiated a Connection establishment and has received an IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT Event because the Passive side has accepted the Connection Request. This state is only used in three-way Connection establishments.	<i>it_ep_accept</i> , <i>it_ep_free</i> , <i>it_ep_modify</i> , <i>it_ep_query</i> , <i>it_get_consumer_context</i> , <i>it_get_handle_type</i> , <i>it_post_rcv</i> , <i>it_reject</i> , <i>it_set_consumer_context</i> , <i>it_ep_disconnect</i>
IT_EP_STATE_PASSIVE_CONNECTION_PENDING	The Passive side Consumer has called <i>it_ep_accept</i> in response to the IT_CM_REQ_CONN_REQUEST_EVENT Event. For iWARP only, this is also a transient state during Socket Conversion on the Conversion Initiator side.	<i>it_ep_free</i> , <i>it_ep_modify</i> , <i>it_ep_query</i> , <i>it_get_consumer_context</i> , <i>it_get_handle_type</i> , <i>it_post_rcv</i> , <i>it_set_consumer_context</i> , <i>it_ep_disconnect</i>

Reliable Connection Endpoint States	Description of State	Allowed Calls
IT_EP_STATE_CONNECTED	The Endpoint is Connected and is ready for all types of Data Transfer and Link Operations.	<i>it_ep_disconnect</i> , <i>it_ep_free</i> , <i>it_ep_modify</i> , <i>it_ep_query</i> , <i>it_get_consumer_context</i> , <i>it_get_handle_type</i> , <i>it_post_rdma_read</i> , <i>it_post_rdma_read_to_rmr</i> , <i>it_post_rdma_write</i> , <i>it_post_recv</i> , <i>it_post_send</i> , <i>it_rmr_link</i> , <i>it_rmr_unlink</i> , <i>it_set_consumer_context</i>
IT_EP_STATE_NONOPERATIONAL	The Endpoint is in the process of disconnecting. Any pending Data Transfer Operations on the Endpoint will be flushed. Any well-formed operation subsequently posted in this state will complete with Flushed or error Status.	<i>it_ep_disconnect</i> , <i>it_ep_free</i> , <i>it_ep_modify</i> , <i>it_ep_query</i> , <i>it_ep_reset</i> , <i>it_get_consumer_context</i> , <i>it_get_handle_type</i> , <i>it_post_rdma_read</i> , <i>it_post_rdma_read_to_rmr</i> , <i>it_post_rdma_write</i> , <i>it_post_recv</i> , <i>it_post_send</i> , <i>it_rmr_link</i> , <i>it_rmr_unlink</i> , <i>it_set_consumer_context</i>
IT_EP_STATE_PASSIVE_WAIT_RDMA_TRANS_REQ	The Conversion Initiator has called <i>it_socket_convert</i> , and its IT-API Implementation is expecting a request for transitioning to RDMA mode from the remote side. This transient state is only used for the iWARP Transport.	<i>it_ep_free</i> , <i>it_ep_modify</i> , <i>it_ep_query</i> , <i>it_get_consumer_context</i> , <i>it_get_handle_type</i> , <i>it_post_recv</i> , <i>it_set_consumer_context</i> , <i>it_ep_disconnect</i>

8509
8510
8511
8512

The following table identifies the RC Endpoint state transitions:

State	Event	Transition to
IT_EP_STATE_UNCONNECTED	<i>it_ep_connect</i> called	IT_EP_STATE_ACTIVE1_CONNECTION_PENDING

State	Event	Transition to
	<i>it_ep_accept</i> called	IT_EP_STATE_PASSIVE_CONNECTION_PENDING
IT_EP_STATE_ACTIVE1_CONNECTION_PENDING	Completion of two-way Connection establishment	IT_EP_STATE_CONNECTED
	IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT enqueued	IT_EP_STATE_ACTIVE2_CONNECTION_PENDING
	Local or Remote error, or <i>it_reject</i> called on Remote side	IT_EP_STATE_NONOPERATIONAL
	<i>it_ep_disconnect</i> called	IT_EP_STATE_NONOPERATIONAL
IT_EP_STATE_ACTIVE2_CONNECTION_PENDING	Completion of three-way Connection establishment	IT_EP_STATE_CONNECTED
	Local or Remote error	IT_EP_STATE_NONOPERATIONAL
	<i>it_ep_disconnect</i> called	IT_EP_STATE_NONOPERATIONAL
IT_EP_STATE_PASSIVE_CONNECTION_PENDING	Completion of Connection establishment	IT_EP_STATE_CONNECTED
	Local or Remote error, or <i>it_reject</i> called on Active side	IT_EP_STATE_NONOPERATIONAL
	<i>it_ep_disconnect</i> called	IT_EP_STATE_NONOPERATIONAL
IT_EP_STATE_PASSIVE_WAIT_RDMA_TRANS_REQ	Reception of a request for transitioning to RDMA mode after initiating Socket Conversion.	IT_EP_STATE_PASSIVE_CONNECTION_PENDING
	Local or Remote error	IT_EP_STATE_NONOPERATIONAL
	<i>it_ep_disconnect</i> called	IT_EP_STATE_NONOPERATIONAL
IT_EP_STATE_CONNECTED	Local Error, or <i>it_ep_disconnect</i> called, or Remote error or Remote disconnect	IT_EP_STATE_NONOPERATIONAL
IT_EP_STATE_NONOPERATIONAL	<i>it_ep_reset</i> called	IT_EP_STATE_UNCONNECTED

8513
8514
8515

The following table identifies and describes the UD Endpoint states. For each state, the table lists the API routines that can be legally applied to a UD Endpoint in that state.

Unreliable Datagram Endpoint States	Description of State	Allowed Calls
IT_EP_STATE_UD_NONOPERATIONAL	Any pending Data Transfer Operations on the Endpoint will be flushed. Any well-formed operation subsequently posted in this state will complete with a Flushed status.	<i>it_ep_free</i> , <i>it_ep_modify</i> , <i>it_ep_query</i> , <i>it_get_consumer_context</i> , <i>it_get_handle_type</i> , <i>it_post_rcvfrom</i> , <i>it_post_sendto</i> , <i>it_set_consumer_context</i> , <i>it_ep_reset</i>
IT_EP_STATE_UD_OPERATIONAL	Data Transfer Operations can be posted to the Endpoint. When an Endpoint is first created by calling <i>it_ep_ud_create</i> it is in this state.	<i>it_ep_free</i> , <i>it_ep_modify</i> , <i>it_ep_query</i> , <i>it_get_consumer_context</i> , <i>it_get_handle_type</i> , <i>it_post_rcvfrom</i> , <i>it_post_sendto</i> , <i>it_set_consumer_context</i> , <i>it_ep_reset</i>

8516
8517
8518
8519
8520
8521
8522
8523

An Endpoint in any state can be destroyed by calling *it_ep_free*. However, calling *it_ep_free* may result in pending Completion Events for the Endpoint being silently discarded by the Implementation. Once a Reliable Connected Endpoint is referenced by either *it_ep_connect* or *it_ep_accept* if for any reason the Connection is not established the Endpoint will transition into the IT_EP_STATE_NONOPERATIONAL state from any state. If a Connection is established and then the Connection is broken for any reason, the Endpoint will transition into the IT_EP_STATE_NONOPERATIONAL state.

8524

IT_EP_STATE_UNCONNECTED

8525
8526
8527

When Endpoints are created they are in the IT_EP_STATE_UNCONNECTED state. Only Receive Data Transfer Operations can be posted to an unconnected Endpoint. An Endpoint must be in this state to be used in either an *it_ep_connect* or an *it_ep_accept* call.

8528

IT_EP_STATE_ACTIVE1_CONNECTION_PENDING

8529
8530
8531

Once an Active side Endpoint is referenced by a Connection establishment it transitions into the IT_EP_STATE_ACTIVE1_CONNECTION_PENDING state. Receive Data Transfer Operations may be posted in this state.

8532
8533
8534
8535

In the case of two-way Connection establishment, the IT_EP_STATE_ACTIVE1_CONNECTION_PENDING state is transient, and the Endpoint will transition to the IT_EP_STATE_CONNECTED state once the Passive side accepts the Connection. If the Passive side rejects the Connection, the Active side will receive an IT_CM_MSG_CONN_

8536 PEER_REJECT_EVENT Event and the Endpoint will transition into the IT_EP_
8537 STATE_NONOPERATIONAL state.

8538 In the case of three-way Connection establishment, the Active side Endpoint will transition to
8539 IT_EP_STATE_ACTIVE2_CONNECTION_PENDING when an IT_CM_MSG_CONN_
8540 ACCEPT_ARRIVAL_EVENT Event is enqueued for the Active side Consumer. If the Active
8541 Consumer calls *it_reject* after processing the IT_CM_MSG_CONN_ACCEPT_
8542 ARRIVAL_EVENT Event, the Endpoint will transition into the IT_EP_STATE_
8543 NONOPERATIONAL state. If the Passive side rejects the Connection, the Active side will
8544 receive an IT_CM_MSG_CONN_PEER_REJECT_EVENT Event and the Endpoint will
8545 transition into the IT_EP_STATE_NONOPERATIONAL state.

8546 **IT_EP_STATE_ACTIVE2_CONNECTION_PENDING**

8547 In the case of three-way Connection establishment, the Endpoint will transition to the
8548 IT_EP_STATE_CONNECTED state when the Active side Consumer successfully calls
8549 *it_ep_accept*, or the Endpoint will transition to the IT_EP_STATE_NONOPERATIONAL state
8550 if the Active side Consumer successfully calls *it_reject*.

8551 In the case of two-way Connection establishment, this state is transient, and the Endpoint will
8552 transition to the IT_EP_STATE_CONNECTED state when the Connection is successfully
8553 established.

8554 **IT_EP_STATE_PASSIVE_CONNECTION_PENDING**

8555 The Passive side Endpoint transitions into this state when the Consumer calls *it_ep_accept*
8556 during Connection establishment. In this state only Receive Data Transfer Operations can be
8557 posted. For iWARP only, this is also a transient state during Socket Conversion on the
8558 Conversion Initiator side.

8559 From this state the Endpoint will transition into the IT_EP_STATE_CONNECTED state when
8560 Connection establishment completes successfully.

8561 **IT_EP_STATE_CONNECTED**

8562 This state is entered when Connection establishment completes. Upon transition to this state, the
8563 Implementation delivers an IT_CM_MSG_CONN_ESTABLISHED_EVENT Event. All types
8564 of Data Transfer and Link Operations can be posted and will be processed in this state.

8565 If either the Local or Remote Consumer disconnects, the Endpoint will transition into the
8566 IT_EP_STATE_NONOPERATIONAL state, and the Implementation will deliver an
8567 IT_CM_MSG_CONN_DISCONNECT_EVENT Event.

8568 Local or Remote errors (e.g., protection violations) also cause the Endpoint to transition to the
8569 IT_EP_STATE_NONOPERATIONAL state, and the Implementation will deliver an
8570 IT_CM_MSG_CONN_BROKEN_EVENT Event.

8571 The transition out of the IT_EP_STATE_CONNECTED state is surfaced by either the
8572 IT_CM_CONN_DISCONNECT_EVENT Event or the IT_CM_MSG_CONN_BROKEN_
8573 EVENT Event, but never both.

8574 **IT_EP_STATE_NONOPERATIONAL**

8575 In this state no requests will be processed by the Endpoint and any well-formed requests posted
8576 will generate Completions with a failing Completion Status. The Endpoint will remain in this
8577 state until *it_ep_reset* is used to put the Endpoint back into the
8578 IT_EP_STATE_UNCONNECTED state.

8579 **IT_EP_STATE_PASSIVE_WAIT_RDMA_TRANS_REQ**

8580 For iWARP only, the Passive side Endpoint; i.e., the Endpoint of the RDMA Responder or,
8581 equivalently, of the Conversion Initiator, transitions into this state when the Conversion Initiator
8582 calls *it_socket_convert*. In this state only Receive Data Transfer Operations can be posted. From
8583 this state, the Endpoint will transition into the transient IT_EP_STATE_
8584 PASSIVE_CONNECTION_PENDING state when a request for transitioning to RDMA mode is
8585 received from the remote side. When Connection establishment completes successfully, the
8586 Endpoint will finally transition to the IT_EP_STATE_CONNECTED state.

8587 **IT_EP_STATE_UD_OPERATIONAL**

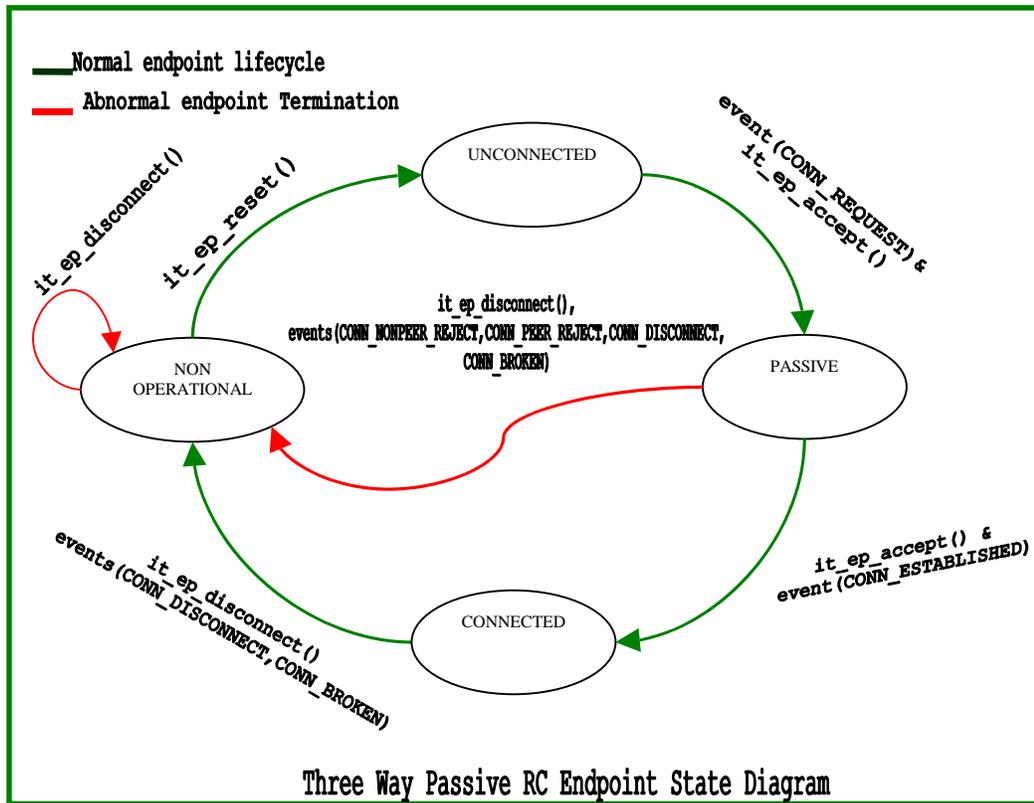
8588 In this state requests will be processed by the Endpoint.

8589 **IT_EP_STATE_UD_NONOPERATIONAL**

8590 In this state no requests will be processed by the Endpoint and any well-formed requests posted
8591 will generate Completions with a failing Completion Status. Once an Unreliable Datagram
8592 Endpoint enters this state it can only be destroyed with *it_ep_free*.

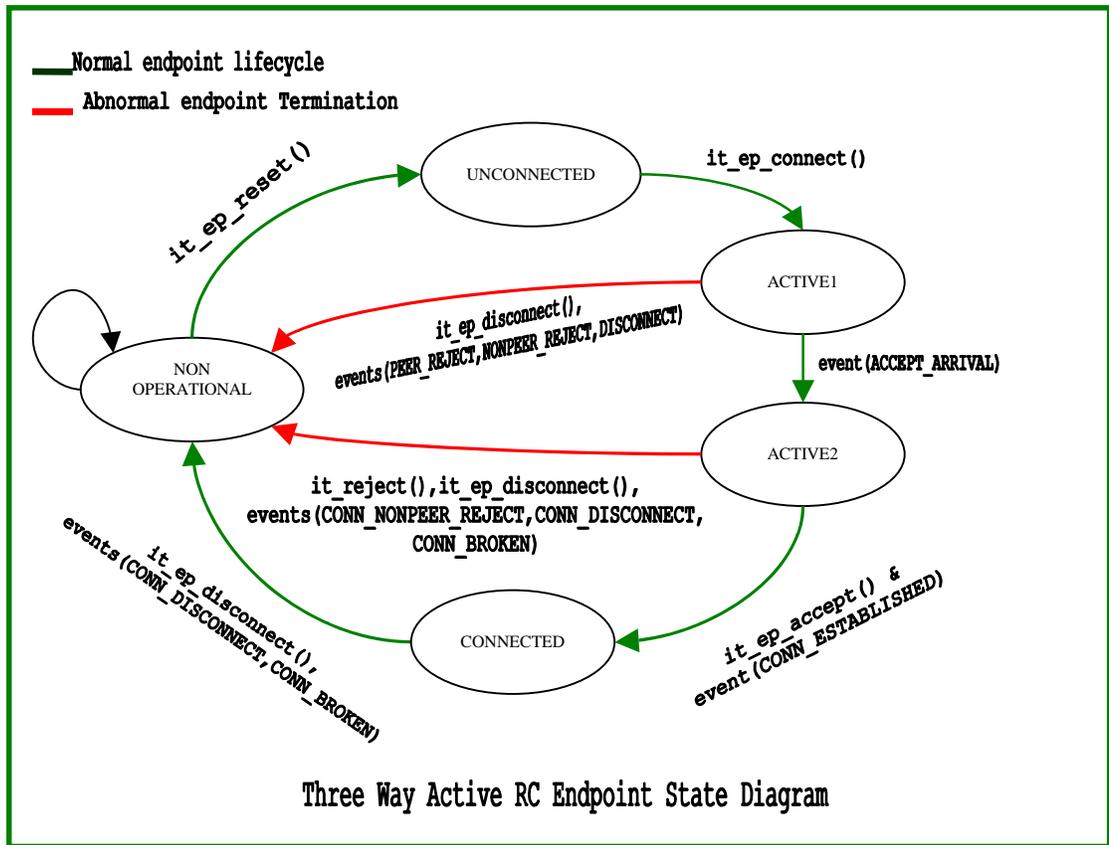
8593 **EXTENDED DESCRIPTION**

8594 For the TII, the state diagrams are shown below, separately for three-way and two-way
8595 Connection establishment, and for the active and passive sides.



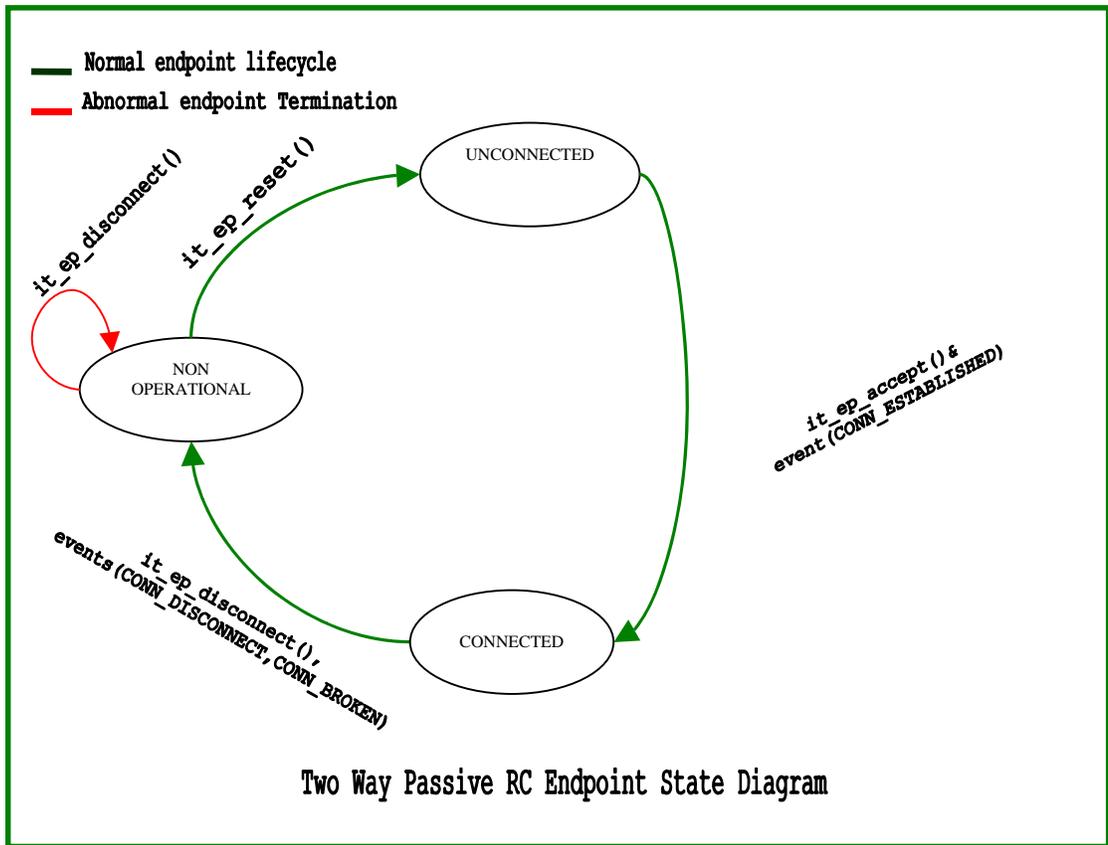
8596
8597

Figure 8: Three Way Passive RC Endpoint State Diagram



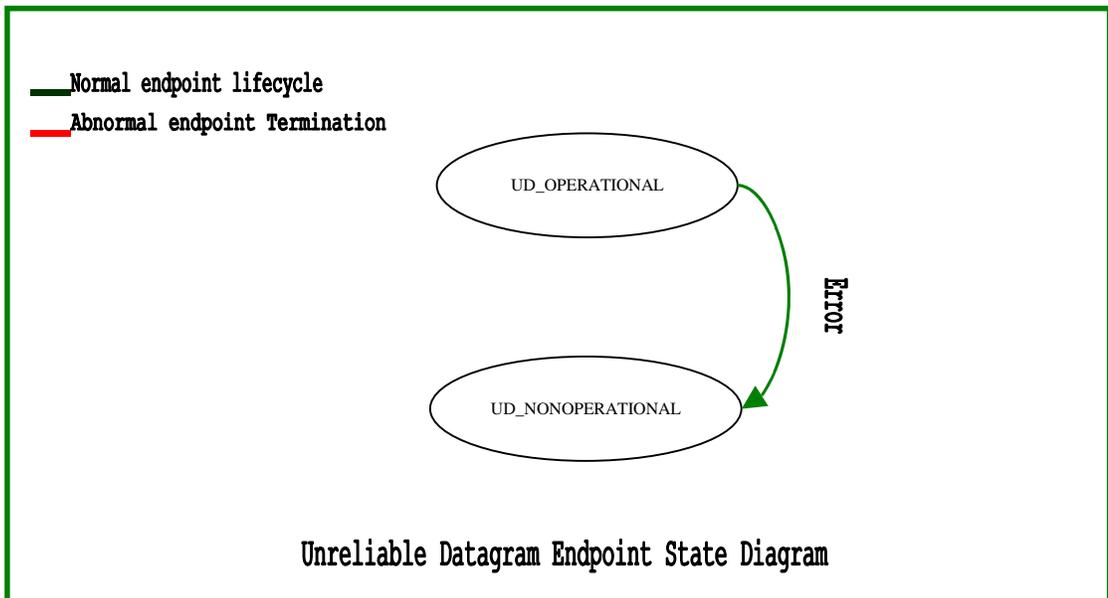
8598
8599

Figure 9: Three Way Active RC Endpoint State Diagram



8602
8603

Figure 11: Two Way Passive RC Endpoint State Diagram



8604
8605
8606

Figure 12: Unreliable Datagram Endpoint State Diagram

For the state diagrams applicable to the TDI, see [it_socket_convert](#).

8607 **APPLICATION USAGE**

- 8608
8609
8610
8611
8612
8613
8614
8615
1. If the Consumer cares about the Completion Status of posted Data Transfer or Link Operations after an Endpoint transitions into the `IT_EP_STATE_NONOPERATIONAL` state, then the Consumer can Post Send and Receive DTOs to the Endpoint to serve as markers in the Endpoint associated EVD. The API guarantees that any posting in `IT_EP_STATE_NONOPERATIONAL` state will be immediately flushed to the EVDs. The Consumer can reap Completions from the associated EVDs until Completions for the marker DTOs are returned. This way the Consumer can be guaranteed that all Completions associated with the Endpoint have been reaped.
 2. The Consumer is responsible for coordinating the use of functions that free a Connection establishment Identifier (*cn_est_id*) such as *it_ep_accept*, *it_reject*, *it_ep_disconnect*, and *it_handoff*. The behavior of functions that are passed as invalid Connection establishment Identifier is indeterminate.
- 8616
8617
8618
8619

8620 **SEE ALSO**

8621 *it_ep_accept()*, *it_reject()*, *it_ep_disconnect()*, *it_handoff()*, *it_ep_reset()*

8622

8623

8624 **NAME**

8625 it_event – definition of Event data structures

8626 **SYNOPSIS**

```

8627 #include <it_api.h>
8628
8629 #define IT_EVENT_STREAM_MASK 0xff000
8630 #define IT_TIMEOUT_INFINITE ((uint64_t)(-1))
8631
8632 typedef enum
8633 {
8634     /*
8635      * Event Stream for WR/DTO completions
8636      */
8637     IT_DTO_EVENT_STREAM           = 0x00000,
8638     IT_DTO_SEND_CMPL_EVENT       = 0x00001,
8639     IT_DTO_RC_RECV_CMPL_EVENT    = 0x00002,
8640     IT_DTO_UD_RECV_CMPL_EVENT    = 0x00003,
8641     IT_DTO_RDMA_WRITE_CMPL_EVENT = 0x00004,
8642     IT_DTO_RDMA_READ_CMPL_EVENT  = 0x00005,
8643     IT_RMR_BIND_CMPL_EVENT       = 0x00006,
8644     IT_RMR_LINK_CMPL_EVENT       = 0x00006,
8645
8646     /*
8647      * Event Stream for Communication Management Request Events
8648      */
8649     IT_CM_REQ_EVENT_STREAM        = 0x01000,
8650     IT_CM_REQ_CONN_REQUEST_EVENT  = 0x01001,
8651     IT_CM_REQ_UD_SERVICE_REQUEST_EVENT = 0x01002,
8652
8653     /*
8654      * Event Stream for Communication Management Message Events
8655      */
8656     IT_CM_MSG_EVENT_STREAM        = 0x02000,
8657     IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT = 0x02001,
8658     IT_CM_MSG_CONN_ESTABLISHED_EVENT  = 0x02002,
8659     IT_CM_MSG_CONN_DISCONNECT_EVENT   = 0x02003,
8660     IT_CM_MSG_CONN_PEER_REJECT_EVENT  = 0x02004,
8661     IT_CM_MSG_CONN_NONPEER_REJECT_EVENT = 0x02005,
8662     IT_CM_MSG_CONN_BROKEN_EVENT       = 0x02006,
8663     IT_CM_MSG_UD_SERVICE_REPLY_EVENT   = 0x02007,
8664
8665     /* Event Stream for Affiliated Asynchronous Events */
8666     IT_ASYNC_AFF_EVENT_STREAM         = 0x04000,
8667     IT_ASYNC_AFF_SEVD_ENQUEUE_FAILURE = 0x04001,
8668     IT_ASYNC_AFF_EP_FAILURE           = 0x04002,
8669     IT_ASYNC_AFF_EP_BAD_TRANSPORT_OPCODE = 0x04003,
8670     IT_ASYNC_AFF_EP_REQ_DROPPED      = 0x04005,
8671     IT_ASYNC_AFF_EP_RDMAW_ACCESS_VIOLATION = 0x04006,
8672     IT_ASYNC_AFF_EP_RDMAW_CORRUPT_DATA = 0x04007,
8673     IT_ASYNC_AFF_EP_RDMAW_ACCESS_VIOLATION = 0x04008,

```

```

8674     IT_ASYNC_AFF_EP_LOCAL_ACCESS_VIOLATION    = 0x04020,
8675     IT_ASYNC_AFF_EP_L_ACCESS_VIOLATION       = 0x04020,
8676     IT_ASYNC_AFF_EP_L_RECV_ACCESS_VIOLATION  = 0x04021,
8677     IT_ASYNC_AFF_EP_L_IRRQ_ACCESS_VIOLATION  = 0x04022,
8678     IT_ASYNC_AFF_EP_L_TRANSPORT_ERROR        = 0x04023,
8679     IT_ASYNC_AFF_EP_L_LLP_ERROR              = 0x04024,
8680     IT_ASYNC_AFF_EP_R_ERROR                  = 0x04040,
8681     IT_ASYNC_AFF_EP_R_ACCESS_VIOLATION       = 0x04041,
8682     IT_ASYNC_AFF_EP_R_RECV_ACCESS_VIOLATION  = 0x04042,
8683     IT_ASYNC_AFF_EP_R_RECV_LENGTH_ERROR      = 0x04043,
8684     IT_ASYNC_AFF_EP_SOFT_HI_WATERMARK        = 0x04060,
8685     IT_ASYNC_AFF_SRQ_LOW_WATERMARK           = 0x04100,
8686
8687     /* Event Stream for Unaffiliated Asynchronous Events */
8688     IT_ASYNC_UNAFF_EVENT_STREAM               = 0x08000,
8689     /* 0x08001 is deprecated */
8690     IT_ASYNC_UNAFF_SPIGOT_ONLINE             = 0x08002,
8691     IT_ASYNC_UNAFF_SPIGOT_OFFLINE            = 0x08003,
8692     IT_ASYNC_UNAFF_SEVD_ENQUEUE_FAILURE     = 0x08004,
8693
8694     /* Event Stream for Software Events */
8695     IT_SOFTWARE_EVENT_STREAM                 = 0x10000,
8696     IT_SOFTWARE_EVENT                       = 0x10001,
8697
8698     /* Event Stream for AEVD Notifications */
8699     IT_AEVD_NOTIFICATION_EVENT_STREAM        = 0x20000,
8700     IT_AEVD_NOTIFICATION_EVENT              = 0x20001
8701 } it_event_type_t;
8702
8703 typedef struct {
8704     it_event_type_t  event_number;
8705     it_evd_handle_t  evd;
8706 } it_any_event_t;
8707
8708 typedef union
8709 {
8710     /*
8711     * The following two union elements are
8712     * available for programming convenience.
8713     *
8714     * The event_number may be used to determine the
8715     * it_event_type_t of any Event. it_any_event_t
8716     * allows the EVD to be determined as well.
8717     */
8718     it_event_type_t  event_number;
8719     it_any_event_t   any;
8720
8721     /*
8722     * The remaining union elements correspond to
8723     * the various it_event_type_t types.
8724     */
8725
8726     /*
8727     * The following two Event structures

```

```

8728 * support the IT_DTO_EVENT_STREAM Event Stream.
8729 *
8730 * it_dto_cmpl_event_t supports
8731 * only the following events:
8732 *     IT_DTO_SEND_CMPL_EVENT
8733 *     IT_DTO_RC_RECV_CMPL_EVENT
8734 *     IT_DTO_RDMA_WRITE_CMPL_EVENT
8735 *     IT_DTO_RDMA_READ_CMPL_EVENT
8736 *     IT_RMR_BIND_CMPL_EVENT = IT_RMR_LINK_CMPL_EVENT
8737 *
8738 * it_all_dto_cmpl_event_t supports all
8739 * possible DTO and RMR events:
8740 *     IT_DTO_SEND_CMPL_EVENT
8741 *     IT_DTO_RC_RECV_CMPL_EVENT
8742 *     IT_DTO_UD_RECV_CMPL_EVENT
8743 *     IT_DTO_RDMA_WRITE_CMPL_EVENT
8744 *     IT_DTO_RDMA_READ_CMPL_EVENT
8745 *     IT_RMR_BIND_CMPL_EVENT = IT_RMR_LINK_CMPL_EVENT
8746 */
8747 it_dto_cmpl_event_t      dto_cmpl;
8748 it_all_dto_cmpl_event_t all_dto_cmpl;
8749
8750 /*
8751 * The following two Event structures
8752 * support the IT_CM_REQ_EVENT_STREAM Event
8753 * stream:
8754 *
8755 * it_conn_request_event_t supports:
8756 *     IT_CM_REQ_CONN_REQUEST_EVENT
8757 *
8758 * it_ud_svc_request_event_t supports:
8759 *     IT_CM_REQ_UD_SERVICE_REQUEST_EVENT
8760 */
8761 it_conn_request_event_t  conn_req;
8762 it_ud_svc_request_event_t ud_svc_request;
8763
8764 /*
8765 * The following two Event structures
8766 * support the IT_CM_MSG_EVENT_STREAM Event
8767 * stream:
8768 *
8769 * it_connection_event_t supports:
8770 *     IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT
8771 *     IT_CM_MSG_CONN_ESTABLISHED_EVENT
8772 *     IT_CM_MSG_CONN_PEER_REJECT_EVENT
8773 *     IT_CM_MSG_CONN_NONPEER_REJECT_EVENT
8774 *     IT_CM_MSG_CONN_DISCONNECT_EVENT
8775 *     IT_CM_MSG_CONN_BROKEN_EVENT
8776 *
8777 * it_ud_svc_reply_event_t supports:
8778 *     IT_CM_MSG_UD_SERVICE_REPLY_EVENT
8779 */
8780 it_connection_event_t    conn;
8781 it_ud_svc_reply_event_t ud_svc_reply;

```

```

8782
8783      /*
8784      * it_affiliated_event_t supports
8785      * the following Event Stream:
8786      *     IT_ASYNC_AFF_EVENT_STREAM
8787      */
8788      it_affiliated_event_t  aff_async;
8789
8790      /*
8791      * it_unaffiliated_event_t supports
8792      * the following Event Stream:
8793      *     IT_ASYNC_UNAFF_EVENT_STREAM
8794      */
8795      it_unaffiliated_event_t  unaff_async;
8796
8797      /*
8798      * it_software_event_t supports
8799      * the following Event Stream:
8800      *     IT_SOFTWARE_EVENT_STREAM
8801      */
8802      it_software_event_t  sw;
8803
8804      /*
8805      * it_aevd_notification_event_t supports
8806      * the following Event Stream:
8807      *     IT_AEVD_NOTIFICATION_EVENT_STREAM
8808      */
8809      it_aevd_notification_event_t  aevd_notify;
8810  } it_event_t;

```

8811 DESCRIPTION

8812 The *it_event_t* defines the format for Events for IT-APIs. Each Event consists of a Handle to the
8813 EVD where the Event has been queued, and Event type identifier with the Event type-specific
8814 Event data.

8815 Events for a Simple EVD can be fed from only a single Event Stream type.

8816 Multiple Event numbers that can be on the same Event Stream type form an Event group. The
8817 Event data formats for all Event types of the same Event Stream type are defined in one or more
8818 separate reference page(s) specific to each Event group.

8819 APPLICATION USAGE

8820 The Consumer allocates an *it_event_t* object and passes it into the *it_evd_wait* or *it_evd_dequeue*
8821 calls in order to retrieve Events. The *it_event_t* object is a union of all possible Event Stream
8822 data types, thus it is the size of the largest possible Event type.

8823 If the Consumer wishes to conserve memory and use only the minimally-sized Event data
8824 structures found in the *it_event_t* union, they are free to. Use of *it_event_t* structures that are too
8825 small for an Event Stream may cause program termination.

8826 The Consumer may use the `IT_EVENT_STREAM_MASK` to convert from an *it_event_type_t*
8827 *event_number* to Event Stream by masking off the lower bits of the Event number.

8828 **SEE ALSO**

8829 *it_aevd_notification_event_t, it_affiliated_event_t, it_cm_msg_events, it_cm_req_events,*
8830 *it_dto_events, it_software_event_t, it_unaffiliated_event_t, it_evd_wait(), it_evd_dequeue(),*
8831 *it_evd_create()*

8832

it_handle_t

8833

8834 NAME

8835 it_handle_t – enumeration and type definitions for IT Handles

8836 SYNOPSIS

```
8837 #include <it_api.h>
8838
8839 typedef enum {
8840     IT_HANDLE_TYPE_ADDR,
8841     IT_HANDLE_TYPE_EP,
8842     IT_HANDLE_TYPE_EVD,
8843     IT_HANDLE_TYPE_IA,
8844     IT_HANDLE_TYPE_LISTEN,
8845     IT_HANDLE_TYPE_LMR,
8846     IT_HANDLE_TYPE_PZ,
8847     IT_HANDLE_TYPE_RMR,
8848     IT_HANDLE_TYPE_UD_SVC_REQ,
8849     IT_HANDLE_TYPE_SRQ
8850 } it_handle_type_enum_t;
8851
8852 typedef void *it_handle_t;
8853 #define IT_NULL_HANDLE ((it_handle_t) NULL)
8854
8855 typedef struct it_addr_handle_s      *it_addr_handle_t;
8856 typedef struct it_ep_handle_s        *it_ep_handle_t;
8857 typedef struct it_evd_handle_s       *it_evd_handle_t;
8858 typedef struct it_ia_handle_s        *it_ia_handle_t;
8859 typedef struct it_listen_handle_s    *it_listen_handle_t;
8860 typedef struct it_lmr_handle_s       *it_lmr_handle_t;
8861 typedef struct it_pz_handle_s        *it_pz_handle_t;
8862 typedef struct it_rmr_handle_s       *it_rmr_handle_t;
8863 typedef struct it_ud_svc_req_handle_s *it_ud_svc_req_handle_t;
8864 typedef struct it_srq_handle_s       *it_srq_handle_t;
```

8865 DESCRIPTION

8866 The *it_handle_type_enum_t* associates an enumerated value with each type of Handle used in the
8867 API Implementation. The enumeration is used to describe the type of a Handle returned by
8868 [it_get_handle_type](#).

8869 The table below defines the relationship of IT-API Handle types and the associated
8870 *it_handle_type_enum_t* value.

it_handle Type	Returned it_handle_type_enum Value
<i>it_addr_handle_t</i>	IT_HANDLE_TYPE_ADDR
<i>it_ep_handle_t</i>	IT_HANDLE_TYPE_EP
<i>it_evd_handle_t</i>	IT_HANDLE_TYPE_EVD
<i>it_ia_handle_t</i>	IT_HANDLE_TYPE_IA

it_handle Type	Returned it_handle_type_enum Value
<i>it_listen_handle_t</i>	IT_HANDLE_TYPE_LISTEN
<i>it_lmr_handle_t</i>	IT_HANDLE_TYPE_LMR
<i>it_pz_handle_t</i>	IT_HANDLE_TYPE_PZ
<i>it_rmr_handle_t</i>	IT_HANDLE_TYPE_RMR
<i>it_ud_svc_req_handle_t</i>	IT_HANDLE_TYPE_UD_SVC_REQ
<i>it_srq_handle_t</i>	IT_HANDLE_TYPE_SRQ

8871 **SEE ALSO**
8872 [*it_get_handle_type\(\)*](#)

8873

8874 **NAME**

8875 it_ia_info_t – encapsulates all Interface Adapter attributes and Spigot information

8876 **SYNOPSIS**

```

8877 #include <it_api.h>
8878
8879 /* Enumerates all the transport types supported by the API. */
8880 typedef enum {
8881
8882     /* InfiniBand Transport */
8883     IT_IB_TRANSPORT = 1,
8884
8885     /* VIA host Interface using IP transport, supporting
8886      only the Reliable Delivery reliability level */
8887     IT_VIA_IP_TRANSPORT = 2,
8888
8889     /* VIA host Interface, using Fibre Channel transport, supporting
8890      only the Reliable Delivery reliability level */
8891     IT_VIA_FC_TRANSPORT = 3,
8892
8893     /* iWARP over TCP transport */
8894     IT_IWARP_TCP_TRANSPORT = 4,
8895
8896     /* Vendor-proprietary Transport */
8897     IT_VENDOR_TRANSPORT = 1000
8898
8899 } it_transport_type_t;
8900
8901 /* Transport Service Type definitions. */
8902 typedef enum {
8903
8904     /* Reliable Connected Transport Service Type */
8905     IT_RC_SERVICE = 0x1,
8906
8907     /* Unreliable Datagram Transport Service Type */
8908     IT_UD_SERVICE = 0x2,
8909
8910 } it_transport_service_type_t;
8911
8912 /* The following structure describes an Interface Adapter Spigot */
8913 typedef struct {
8914
8915     /* Spigot identifier */
8916     size_t spigot_id;
8917
8918     /* Maximum sized Send operation for the RC service
8919      on this Spigot. */
8920     size_t max_rc_send_len;
8921
8922     /* Maximum sized RDMA Read/Write operation for the RC service on
8923      this Spigot. */

```

```

8924     size_t  max_rc_rdma_len;
8925
8926     /* Maximum sized Send operation for the UD service
8927        on this Spigot. */
8928     size_t  max_ud_send_len;
8929
8930     /* Indicates whether the Spigot is online or offline.
8931        An IT_TRUE value means online. */
8932     it_boolean_t  spigot_online;
8933
8934     /* A mask indicating which Connection Qualifier types this
8935        IA supports for input to it_ep_connect and
8936        it_ud_service_request_handle_create. The bits in the mask are
8937        an inclusive OR of the values for Connection Qualifier types
8938        that this IA supports. */
8939     it_conn_qual_type_t  active_side_conn_qual;
8940
8941     /* A mask indicating which Connection Qualifier types this to
8942        it_listen_create. The bits in the mask are an inclusive OR of
8943        the values for Connection Qualifier types that this IA
8944        supports. */
8945     it_conn_qual_type_t  passive_side_conn_qual;
8946
8947     /* The number of Network Addresses associated with Spigot. */
8948     size_t  num_net_addr;
8949
8950     /* Pointer to array of Network Address addresses. */
8951     it_net_addr_t*  net_addr;
8952
8953 } it_spigot_info_t;
8954
8955 /* The following structure is used to identify the vendor associated
8956    with an IA that uses the IB transport*/
8957 typedef struct {
8958
8959     /* The NodeInfo:VendorID as described in chapter 14 of the
8960        IB spec. */
8961     uint32_t  vendor : 24;
8962
8963     /* The NodeInfo:DeviceID as described in chapter 14 of the
8964        IB spec. */
8965     uint16_t  device;
8966
8967     /* The NodeInfo:Revision as described in chapter 14 of the
8968        IB spec. */
8969     uint32_t  revision;
8970 } it_vendor_ib_t;
8971
8972 /* The following structure is used to identify the vendor associated
8973    with an IA that uses a VIA transport. */
8974 typedef struct {
8975     /* The "Name" member of the VIP_NIC_ATTRIBUTES structure, as
8976        described in the VIA spec. */
8977     char  name[64];

```

```

8978
8979     /* The "HardwareVersion" member of the VIP_NIC_ATTRIBUTES
8980        structure, as described in the VIA spec. */
8981     unsigned long hardware;
8982
8983     /* The "ProviderVersion" member of the VIP_NIC_ATTRIBUTES
8984        structure, as described in the VIA spec. */
8985     unsigned long provider;
8986 } it_vendor_via_t;
8987
8988 /* The following structure is used to identify the vendor associated
8989    with an IA that uses an iWARP TCP transport. */
8990 typedef struct {
8991     /* Indicates whether or not vid field contains valid data. */
8992     it_boolean_t valid_vid;
8993
8994     /* Vendor Identification field - strictly vendor-specific (only
8995        valid if valid_vid field is IT_TRUE). */
8996     unsigned char vid[64];
8997 } it_vendor_iwarp_tcp_t;
8998
8999 /* The following structure is returned by the it_ia_query function. */
9000 typedef struct {
9001
9002     /* Interface Adapter name, as specified in it_ia_create. */
9003     char* ia_name;
9004
9005     /* The major version number of the latest version of the
9006        IT-API that this IA supports. */
9007     uint32_t api_major_version;
9008
9009     /* The minor version number of the latest version of the
9010        IT-API that this IA supports. */
9011     uint32_t api_minor_version;
9012
9013     /* The major version number for the software being used to control
9014        this IA. The IT-API imposes no structure whatsoever on this
9015        number; its meaning is completely IA-dependent. */
9016     uint32_t sw_major_version;
9017
9018     /* The minor version number for the software being used to control
9019        this IA. The IT-API imposes no structure whatsoever on this
9020        number; its meaning is completely IA-dependent. */
9021     uint32_t sw_minor_version;
9022
9023     /* The vendor associated with the IA. This information is useful
9024        if the Consumer wishes to do device-specific programming. This
9025        union is discriminated by transport_type. No vendor
9026        identification is provided for transports not listed below. */
9027     union {
9028
9029         /* Used if transport_type is IT_IB_TRANSPORT. */
9030         it_vendor_ib_t ib;
9031

```

```

9032         /* Used if transport_type is IT_VIA_IP_TRANSPORT or
9033            IT_VIA_FC_TRANSPORT. */
9034         it_vendor_via_t   via;
9035
9036         /* Used if transport_type is IT_IWARP_TCP_TRANSPORT. */
9037         it_vendor_iwarp_tcp_t iwarp;
9038
9039     } vendor;
9040
9041     /* The Interface Adapter and platform provide a data alignment hint
9042        to the Consumer to help the Consumer align their data transfer
9043        buffers in a way that is optimal for the performance of the IA.
9044        For example, if the best throughput is obtained by aligning
9045        buffers to 128-byte boundaries, dto_alignment_hint will have the
9046        value 128. The Consumer may choose to ignore the alignment hint
9047        without any adverse functional impact. (There may be an adverse
9048        performance impact.) */
9049     uint32_t  dto_alignment_hint;
9050
9051     /* The transport type (e.g., InfiniBand) supported by Interface
9052        Adapter. An Interface Adapter supports precisely one transport
9053        type. */
9054     it_transport_type_t  transport_type;
9055
9056     /* The Transport Service Types supported by this IA. This is
9057        constructed by doing an inclusive OR of the Transport Service
9058        Type values.*/
9059     it_transport_service_type_t  supported_service_types;
9060
9061     /* Indicates whether Work Queues are resizable. */
9062     it_boolean_t  ep_work_queues_resizable;
9063
9064     /* Indicates whether the underlying transport used by this IA uses
9065        a three-way handshake for doing Connection establishment. Note
9066        that if the underlying transport supports a three-way handshake
9067        the Consumer can choose whether to use two handshakes or three
9068        when establishing the Connection. If the underlying transport
9069        supports a two-way handshake for establishing a Connection, the
9070        Consumer can only use two handshakes when establishing the
9071        Connection. */
9072     it_boolean_t  three_way_handshake_support;
9073
9074     /* Indicates whether Private Data is supported on Connection
9075        establishment or UD service resolution operations. */
9076     it_boolean_t  private_data_support;
9077
9078     /* Indicates whether the max_message_size field in the
9079        IT_CM_REQ_CONN_REQUEST_EVENT is valid for this IA. */
9080     it_boolean_t  max_message_size_support;
9081
9082     /* Indicates whether or not the IA supports IRD/ORD. Affects
9083        whether the rdma_read_ird or rdma_read_ord fields in the
9084        IT_CM_REQ_CONN_REQUEST_EVENT or the
9085        IT_CM_MSG_CONN_ESTABLISHED_EVENT are valid for this IA.

```

```

9086         Also affects whether IRD/ORD suppression is an option.
9087         Deprecates IT-API 1.0 values "ird_support" and "ord_support". */
9088 it_boolean_t ird_ord_ia_support;
9089
9090 /* Indicates whether IRD/ORD suppression is supported
9091    for this IA. If this member has a value of IT_TRUE, the
9092    Consumer can control IRD/ORD suppression in it_ep_connect
9093    and it_listen_create. Otherwise they cannot. */
9094 it_boolean_t ird_ord_suppressible;
9095
9096 /* Indicates whether the IA generates IT_ASYNC_UNAFF_SPIGOT_ONLINE
9097    Events. See it_unaffiliated_event_t for details. */
9098 it_boolean_t spigot_online_support;
9099
9100 /* Indicates whether the IA generates IT_ASYNC_UNAFF_SPIGOT_OFFLINE
9101    Events. See it_unaffiliated_event_t for details. */
9102 it_boolean_t spigot_offline_support;
9103
9104 /* The maximum number of bytes of Private Data supported for the
9105    it_ep_connect routine. This will be less than or equal to
9106    IT_MAX_PRIV_DATA. */
9107 size_t connect_private_data_len;
9108
9109 /* The maximum number of bytes of Private Data supported for the
9110    it_ep_accept routine. This will be less than or equal to
9111    IT_MAX_PRIV_DATA. */
9112 size_t accept_private_data_len;
9113
9114 /* The maximum number of bytes of Private Data supported for the
9115    it_reject routine. This will be less than or equal to
9116    IT_MAX_PRIV_DATA. */
9117 size_t reject_private_data_len;
9118
9119 /* The maximum number of bytes of Private Data supported for the
9120    it_ep_disconnect routine. This will be less than or equal to
9121    IT_MAX_PRIV_DATA. */
9122 size_t disconnect_private_data_len;
9123
9124 /* The maximum number of bytes of Private Data supported for the
9125    it_ud_service_request_handle_create routine. This will be
9126    less than or equal to IT_MAX_PRIV_DATA. */
9127 size_t ud_req_private_data_len;
9128
9129 /* The maximum number of bytes of Private Data supported for the
9130    it_ud_service_reply routine. This will be less than or equal to
9131    IT_MAX_PRIV_DATA. */
9132 size_t ud_rep_private_data_len;
9133
9134 /* Specifies the number of Spigots associated with this Interface
9135    Adapter. */
9136 size_t num_spigots;
9137
9138 /* An array of Spigot information data structures. The array
9139    contains num_spigots elements. */

```

```

9140     it_spigot_info_t*  spigot_info;
9141
9142     /* The Handle for the EVD that contains the affiliated async Event
9143        Stream. If no EVD contains the Affiliated Async Event Stream,
9144        this member will have the distinguished value IT_NULL_HANDLE. */
9145     it_evd_handle_t  affiliated_err_evd;
9146
9147     /* The Handle for the EVD that contains the Unaffiliated Async
9148        Event Stream. If no EVD contains the Unaffiliated Async Event
9149        Stream, this member will have the distinguished value
9150        IT_NULL_HANDLE. */
9151     it_evd_handle_t  unaffiliated_err_evd;
9152
9153     /* Indicates whether the IA supports the S-RQ feature. */
9154     it_boolean_t  srq_support;
9155
9156     /* Indicates whether the IA supports the Endpoint Hard
9157        High Watermark mechanism for limiting the number of Receive
9158        DTOs that can be in progress on an Endpoint that has an
9159        associated S-RQ. */
9160     it_boolean_t  hard_hi_watermark_support;
9161
9162     /* Indicates whether the IA supports the Endpoint Soft
9163        High Watermark mechanism for generating an Affiliated
9164        Asynchronous Event when the number of Receive DTOs in
9165        progress on an Endpoint that has an associated S-RQ exceeds
9166        the Endpoint Soft High Watermark. */
9167     it_boolean_t  soft_hi_watermark_support;
9168
9169     /* Indicates whether an S-RQ can be resized after it is created. */
9170     it_boolean_t  srq_resizable;
9171
9172     /* Indicates that an iWARP V-RNIC supports a modified qp state
9173        diagram (outside the RDMAC verbs). */
9174     it_boolean_t  extended_iwarp_qp_states;
9175
9176     /* Indicates that Implementation supports socket conversion (TDI).
9177        This attribute is IT_TRUE if and only if transport_type is
9178        IT_IWARP_TCP_TRANSPORT. */
9179     it_boolean_t  socket_conversion_support;
9180
9181     /* Indicates that IA supports increasing ORD
9182        (decreasing ORD is mandatory for all RNICs). */
9183     it_boolean_t  rdma_read_ord_increasable;
9184
9185     /* Indicates that IA supports modifying IRD. */
9186     it_boolean_t  rdma_read_ird_modifiable;
9187
9188     /* Bit set indicating which RMR types are supported.
9189        Possible values are IT_RMR_TYPE_NARROW, IT_RMR_TYPE_WIDE, and
9190        (IT_RMR_TYPE_NARROW|IT_RMR_TYPE_WIDE). See also it_rmr_type_t.*/
9191     it_rmr_type_t  rmr_types_supported;
9192
9193     /* Indicates whether the IA supports Relative Addressing. See also

```

```

9194         it_addr_mode_t. */
9195     it_boolean_t  addr_mode_relative_support;
9196
9197     /* Indicates whether the Destination buffer for an RDMA Read DTO
9198        must have remote or local write permission, and whether or not
9199        the Endpoint to which an RDMA Read DTO is posted must have RDMA
9200        Write access enabled. See also it_post_rdma_read. */
9201     it_boolean_t  rdma_read_requires_remote_write;
9202
9203     /* Indicates whether the IA supports changing the RDMA enables
9204        after EP creation. */
9205     it_boolean_t  ep_rdma_enables_modifiable;
9206
9207     /* Indicates whether the IA supports it_post_rdma_read_to_rmr. */
9208     it_boolean_t  rdma_read_local_extensions;
9209
9210     /* Indicates whether the IA supports DTO EVD overflow detection. */
9211     it_boolean_t  dto_evd_overflow_detection;
9212 } it_ia_info_t;

```

9213 DESCRIPTION

9214 The *it_ia_info_t* structure is returned by the *it_ia_query* routine.

9215 The *it_ia_info_t* structure specifies the capabilities and attributes of an Interface Adapter. It also
9216 identifies the Interface Adapter's Spigots and their Network Addresses.

9217 Spigot identifiers are required inputs to the *it_get_pathinfo* and *it_listen_create* routines. Spigot
9218 Network Addresses may be used in the advertisement of local Consumer-provided services to
9219 remote Consumers.

9220 The IA can be in one of two states: enabled or disabled. An IA will be in the enabled state when
9221 it is created (via *it_ia_create*). Normally an IA will remain in the enabled state, but if it
9222 encounters a catastrophic error it will move into the disabled state. The Implementation
9223 guarantees that no unreported data corruption has occurred as a result of the IA entering the
9224 disabled state. If the Consumer calls any API routine other than *it_ia_free* while the IA is in the
9225 disabled state, neither the IA nor any of the Implementation data structures associated with it
9226 will be modified in any way. Instead, the routine will return the error code
9227 IT_ERR_IA_CATASTROPHE, and none of the output parameters from the routine will be
9228 valid. Once an IA has entered the disabled state the only recovery action that the Consumer can
9229 perform is to free the IA.

9230 FUTURE DIRECTIONS

9231 Quality of Service control for VIA and other transports may be added in the future.

9232 SEE ALSO

9233 *it_ia_query()*, *it_get_pathinfo()*, *it_listen_create()*

9234

it_lmr_triplet_t

9235

9236 NAME

9237 `it_lmr_triplet_t` – structure describing a DTO buffer in a Local Memory Region

9238 SYNOPSIS

```
9239 #include <it_api.h>
9240
9241 typedef struct {
9242     it_lmr_handle_t lmr;
9243     union {
9244         void *abs;
9245         it_length_t rel;
9246     } addr;
9247     it_length_t length;
9248 } it_lmr_triplet_t;
```

9249 DESCRIPTION

9250 The `it_lmr_triplet_t` structure describes a local Source or Destination buffer segment for Data
9251 Transfer Operations. Its members are defined as follows:

9252 *lmr* Handle of LMR in which the local buffer resides.

9253 *addr* Starting address of the local buffer segment, interpreted according to the *addr_mode*
9254 (addressing mode) attribute of the LMR (see *it_addr_mode_t*) either as an absolute address
9255 (*abs*) or as an address offset (*rel*) relative to the Base Address of the LMR.

9256 *length* Length in bytes of the local buffer segment.

9257 APPLICATION USAGE

9258 The LMR Triplet is an input parameter describing a local buffer segment for DTOs such as
9259 *it_post_send*.

9260 SEE ALSO

9261 *it_post_send()*, *it_post_sendto()*, *it_post_recv()*, *it_post_recvfrom()*, *it_post_rdma_write()*,
9262 *it_post_rdma_read()*, *it_addr_mode_t*.

9263

it_mem_priv_t

9264

9265 NAME

9266 it_mem_priv_t – Memory access privileges for Local Memory Regions and Remote Memory
9267 Regions

9268 SYNOPSIS

```
9269 #include <it_api.h>
9270
9271 typedef enum {
9272     IT_PRIV_LOCAL_READ      = 0x0001,
9273     IT_PRIV_LOCAL_WRITE    = 0x0002,
9274     IT_PRIV_LOCAL          = 0x0003,
9275     IT_PRIV_REMOTE_READ    = 0x0004,
9276     IT_PRIV_REMOTE_WRITE   = 0x0008,
9277     IT_PRIV_REMOTE        = 0x000c,
9278     IT_PRIV_ALL            = 0x000f
9279 } it_mem_priv_t;
```

9280 DESCRIPTION

9281 *privs* Memory access privileges for an LMR or RMR, formed by a bitwise-inclusive OR
9282 of values from *it_mem_priv_t*.

9283 Individual bit values defined in *it_mem_priv_t* are as follows:

9284 IT_PRIV_LOCAL_READ Grants local read access to the IA, enabling the use of an LMR as
9285 a Source buffer for locally posted Send or RDMA Write DTOs.

9286 IT_PRIV_LOCAL_WRITE Grants local write access to the IA, enabling the use of the LMR as
9287 a Destination buffer for locally posted Receive or RDMA Read
9288 DTOs.

9289 IT_PRIV_REMOTE_READ Grants remote read access to the IA, enabling the use of an LMR
9290 or RMR as a Source buffer for incoming RDMA Read requests.

9291 IT_PRIV_REMOTE_WRITE Grants remote write access to the IA, enabling the use of an LMR
9292 or RMR as a Destination buffer for incoming RDMA Write
9293 operations or RDMA Read Responses.

9294 Predefined bit combinations defined in *it_mem_priv_t* are as follows:

9295 IT_PRIV_LOCAL Local read and local write access, for an LMR.

9296 IT_PRIV_REMOTE Remote read and remote write access, for an LMR or RMR.

9297 IT_PRIV_ALL All access privileges, for an LMR.

9298 APPLICATION USAGE

9299 Memory access privileges of a Local Memory Region are specified when the LMR is created.
9300 Memory access privileges of a Remote Memory Region are specified when the RMR is linked to
9301 an LMR.

9302 **SEE ALSO**
9303 *it_lmr_create(), it_lmr_query(), it_rmr_link(), it_rmr_query()*

it_net_addr_t

9304

9305 NAME

9306

it_net_addr_t – encapsulates all supported Network Address types

9307 SYNOPSIS

9308

```
#include <it_api.h>
```

9309

```
/* Enumerates all the possible Network Address types supported  
by the API. */
```

9312

```
typedef enum {
```

9313

```
    /* IPv4 address */
```

9315

```
    IT_IPV4 = 0x1,
```

9316

```
    /* IPv6 address */
```

9318

```
    IT_IPV6 = 0x2,
```

9319

```
    /* InfiniBand GID */
```

9321

```
    IT_IB_GID = 0x3,
```

9322

```
    /* VIA Network Address */
```

9324

```
    IT_VIA_HOSTADDR = 0x4
```

9325

```
} it_net_addr_type_t;
```

9327

```
/* Defines the Network Address format for a VIA "host address"  
The API has a fixed upper bound on the maximum sized VIA  
address it will support. */
```

9330

```
#define IT_MAX_VIA_ADDR_LEN 64
```

9333

```
typedef struct {
```

9335

```
    /* The number of bytes in the array below that are  
significant. */
```

9338

```
    uint16_t len;
```

9339

```
    /* VIA host address, which is an array of bytes. */
```

9341

```
    unsigned char hostaddr[IT_MAX_VIA_ADDR_LEN];
```

9342

```
} it_via_net_addr_t;
```

9344

```
/* This defines the Network Address format for the InfiniBand  
GID, which is just an IPv6 address. */
```

9347

```
typedef struct in6_addr it_ib_gid_t;
```

9348

```
/* This describes a Network Address suitable for input to several  
routines in the API. */
```

9351

```
typedef struct {
```

9352

```
    /* The discriminator for the union below. */
```

9353

```
    it_net_addr_type_t addr_type;
```

9354

```

9355
9356         union {
9357
9358             /* IPv4 address, in network byte order. */
9359             struct in_addr  ipv4;
9360
9361             /* IPv6 address, in network byte order. */
9362             struct in6_addr  ipv6;
9363
9364             /* InfiniBand GID, in network byte order. */
9365             it_ib_gid_t  gid;
9366
9367             /* VIA Network Address. */
9368             it_via_net_addr_t  via;
9369
9370         } addr;
9371
9372     } it_net_addr_t;

```

9373 **DESCRIPTION**

9374 The *it_net_addr_t* type is used by several routines in the API to encapsulate a Network Address
9375 associated with a Spigot on an IA. The *it_net_addr_t* is the name for a Spigot when it is being
9376 accessed remotely. (When it is being accessed locally, it is named by a Spigot identifier, not by
9377 an *it_net_addr_t*.) Each Spigot on an IA has at least one *it_net_addr_t* associated with it. A
9378 Spigot can have more than one Network Address associated with it, and these Network
9379 Addresses can be of different types. (For example, an InfiniBand HCA might have both an IPv4
9380 address and an InfiniBand GID associated with one of its Spigots.) The set of Network
9381 Addresses that can be used to refer to a Spigot on an IA can be determined using the *it_ia_query*
9382 routine.

9383 In order to aid Consumers in writing portable applications that span platforms with different
9384 native byte orders, all Network Addresses that are supported by the API with the exception of
9385 the VIA “host address” are required to be input to the API in network byte order, and will be
9386 output from the API in network byte order. (The VIA “host address” is defined to be an array of
9387 bytes, and hence is not affected by which native byte order a platform uses.)

9388 **SEE ALSO**

9389 *it_get_pathinfo()*, *it_ia_query()*

9390

9391

9392 **NAME**

9393 it_path_t – describes the Path between a pair of Spigots

9394 **SYNOPSIS**

```

9395 #include <it_api.h>
9396
9397 /* This is the Path information for the InfiniBand Transport. */
9398 typedef struct {
9399
9400     /* Partition Key, as defined in the REQ message for the IB
9401        CM protocol. */
9402     uint16_t  partition_key;
9403
9404     /* Path Packet Payload MTU, as defined in the REQ message
9405        for the IB CM protocol. */
9406     uint8_t   path_mtu : 4;
9407
9408     /* PacketLifeTime, as defined in the PathRecord in IB
9409        specification. This field is useful for Consumers that
9410        wish to use timeout values other than the default ones
9411        for doing Connection establishment. */
9412     uint8_t   packet_lifetime : 6;
9413
9414     /* Local Port LID, as defined in the REQ message for the IB
9415        CM protocol. The low-order bits of this value also
9416        constitute the "Source Path Bits" that are used to
9417        create an Address Handle. */
9418     uint16_t  local_port_lid;
9419
9420     /* Remote Port LID, as defined in the REQ message for the
9421        IB CM protocol. This is also the "Destination LID" used
9422        to create an Address Handle. */
9423     uint16_t  remote_port_lid;
9424
9425     /* Local Port GID in network byte order, as defined in the
9426        REQ message for the IB CM protocol. This is also used to
9427        determine the appropriate "Source GID Index" to be used
9428        when creating an Address Handle. */
9429     it_ib_gid_t  local_port_gid;
9430
9431     /* Remote Port GID in network byte order, as defined in the
9432        REQ message for the IB CM protocol. This is also the
9433        "Destination GID or MGID" used to create an Address
9434        Handle. */
9435     it_ib_gid_t  remote_port_gid;
9436
9437     /* Packet Rate, as defined in the REQ message for the IB CM
9438        protocol. This is also the "Maximum Static Rate" to be
9439        used when creating an Address Handle. */
9440     uint8_t   packet_rate : 6;
9441

```

```

9442     /* SL, as defined in the REQ message for the IB CM
9443        protocol. This is also the "Service Level" to be used
9444        when creating an Address Handle. */
9445     uint8_t  sl : 4;
9446
9447     /* Subnet Local, as defined in the REQ message for the IB
9448        CM protocol. When creating an Address Handle, setting
9449        this bit causes a GRH to be included as part of any
9450        Unreliable Datagram sent using the Address Handle. */
9451     uint8_t  subnet_local : 1;
9452
9453     /* Flow Label, as defined in the REQ message for the IB CM
9454        protocol. This is also the "Flow Label" to be used when
9455        creating an Address Handle. This is only valid if
9456        subnet_local is clear. */
9457     uint32_t flow_label : 20;
9458
9459     /* Traffic Class, as defined in the REQ message for the IB
9460        CM protocol. This is also the "Traffic Class" to be
9461        used when creating an Address Handle. This is only
9462        valid if subnet_local is clear. */
9463     uint8_t  traffic_class;
9464
9465     /* Hop Limit, as defined in the REQ message for the IB CM
9466        protocol. This is also the "Hop Limit" to be used when
9467        creating an Address Handle. This is only valid if
9468        subnet_local is clear. */
9469     uint8_t  hop_limit;
9470
9471 } it_ib_net_endpoint_t;
9472
9473 /* This is the Path information for the VIA transport. */
9474 typedef it_via_net_addr_t it_via_net_endpoint_t;
9475
9476 /* This is the Path information for the iWARP Transport. */
9477
9478 /* enum to discriminate path element types in
9479    it_iwarp_net_endpoint_t. */
9480 typedef enum {
9481     IT_IP_VERS_IPV4 = 0x1,
9482     IT_IP_VERS_IPV6 = 0x2
9483 } it_ip_vers_t;
9484
9485 typedef struct {
9486     /* Designates the type of IP address that is found in
9487        both the laddr and raddr unions below. */
9488     it_ip_vers_t  ip_vers;
9489
9490     /* Local path element */
9491     union {
9492         struct in_addr  ipv4;
9493         struct in6_addr ipv6;
9494     } laddr;
9495

```

```

9496         /* Remote path element */
9497         union {
9498             struct in_addr  ipv4;
9499             struct in6_addr ipv6;
9500         } raddr;
9501     } it_iwarp_net_endpoint_t;
9502
9503     /* This is the Path data structure used by several routines in
9504        the API. */
9505     typedef struct {
9506
9507         /* Identifier for the Spigot to be used on the local IA.
9508            Note that this data structure is always used in a
9509            Context where the IA associated with the Spigot can be
9510            deduced. */
9511         size_t  spigot_id;
9512
9513         /* The transport-independent timeout parameter for how long
9514            to wait, in microseconds, before timing out a Connection
9515            establishment attempt using this Path. The timeout
9516            period for establishing a Connection
9517            can only be specified on the Active side; the timeout
9518            period cannot be changed on the Passive side. */
9519         uint64_t  timeout;
9520
9521         /* The remote component of the Path. */
9522         union {
9523
9524             /* For use with InfiniBand. */
9525             it_ib_net_endpoint_t  ib;
9526
9527             /* For use with VIA. */
9528             it_via_net_endpoint_t  via;
9529
9530             /* For use with iWARP. */
9531             it_iwarp_net_endpoint_t  iwarp;
9532
9533         } remote;
9534
9535     } it_path_t;

```

9536 DESCRIPTION

9537 The *it_path_t* type is used by several routines in the API to encapsulate a Path between two
9538 Spigots. The *it_path_t* contains a local Spigot identifier, a remote Spigot address, and a
9539 specification of all information that determines the properties of the Path that messages will take
9540 between the two Spigots. The local Spigot to be used is identified in a transport-independent
9541 manner, but the remote Spigot and the Path to that Spigot are specified in a transport-dependent
9542 manner.

9543 The *it_path_t* structure also contains a *timeout* member. This *timeout* member defines how long
9544 the local Consumer is willing to wait for a response to its attempt to establish a Connection when
9545 the *it_path_t* is used with the *it_ep_connect* routine. If the Consumer retrieves the Path using the
9546 *it_get_pathinfo* routine, the Implementation will provide a *timeout* that should be sufficiently

9547 long to establish a Connection under most circumstances, and so the Consumer should have no
9548 need to modify this value. If the Consumer chooses to provide their own value for the *timeout*
9549 member, the Consumer should take care to choose a value that is compatible with any
9550 underlying transport-specific timeout values governing Connection establishment that may be
9551 present in the transport-specific portion of the *it_path_t*. Choosing a value of *timeout* that is
9552 incompatible with the transport-specific timeout values governing Connection establishment will
9553 result in an error being returned from the *it_ep_connect* routine.

9554 All data contained within the *it_path_t* data structure appears in host byte order unless otherwise
9555 noted in the comments associated with the members of the data structure. The contents of the
9556 *path_t* returned from the *it_get_pathinfo* routine are only valid on the node on which the routine
9557 was invoked.

9558 **SEE ALSO**

9559 *it_get_pathinfo()*, *it_ep_connect()*, *it_address_handle_create()*,
9560 *it_ud_service_request_handle_create()*

9561

it_rmr_triplet_t

9562
9563 **NAME**
9564 `it_rmr_triplet_t` – structure describing a DTO buffer in a Remote Memory Region

9565 **SYNOPSIS**
9566

```
#include <it_api.h>
```


9567
9568

```
typedef struct {
```


9569

```
    it_rmr_handle_t  rmr;
```


9570

```
    union {
```


9571

```
        void          *abs;
```


9572

```
        it_length_t   rel;
```


9573

```
    } addr;
```


9574

```
    it_length_t       length;
```


9575

```
} it_rmr_triplet_t;
```

9576 **APPLICABILITY**
9577 Remote Memory Regions can be used only for the RC service type. See [it_rmr_create](#).

9578 **DESCRIPTION**
9579 The `it_rmr_triplet_t` structure describes a local Destination buffer segment for an RDMA Read
9580 DTO initiated by the `it_post_rdma_read_to_rmr()` call. Its members are defined as follows:

9581 *rmr* Handle of RMR in which the local buffer resides.

9582 *addr* Starting address of the local buffer segment, interpreted according to the *addr_mode*
9583 (addressing mode) attribute of the RMR (see [it_addr_mode_t](#)) either as an absolute address
9584 (*abs*) or as an address offset (*rel*) relative to the Base Address of the RMR.

9585 *length* Length in bytes of the local buffer segment.

9586 **APPLICATION USAGE**
9587 The RMR Triplet is an input parameter describing a local buffer segment for RDMA Read
9588 DTOs initiated by the `it_post_rdma_read_to_rmr()` call only.

9589 **SEE ALSO**
9590 [it_post_rdma_read_to_rmr\(\)](#), [it_addr_mode_t](#)

9591

it_rmr_type_t

9592

9593 NAME

9594 `it_rmr_type_t` – definition of RMR type

9595 SYNOPSIS

```
9596 #include <it_api.h>
9597
9598 typedef enum {
9599     IT_RMR_TYPE_DEFAULT = 0,
9600     IT_RMR_TYPE_NARROW = 1,
9601     IT_RMR_TYPE_WIDE = 2
9602 } it_rmr_type_t;
```

9603 APPLICABILITY

9604 Remote Memory Regions can be used only for the RC service type. See [it_rmr_create](#).

9605 DESCRIPTION

9606 Type of a Remote Memory Region, with three possible values as follows:

9607 **IT_RMR_TYPE_DEFAULT**

9608 This value can be used only for creating an RMR. Creating an RMR of the default type results in
9609 an RMR of the Narrow or Wide type, depending on the Implementation.

9610 **IT_RMR_TYPE_NARROW**

9611 An unlinked Narrow RMR may be used as a target for Link operations via all Endpoints in the
9612 Protection Zone of the RMR. A linked Narrow RMR may be used as a target for DTOs and
9613 Unlink operations only via the Endpoint through which it was linked.

9614 **IT_RMR_TYPE_WIDE**

9615 An unlinked Wide RMR may be used as a target for Link operations via all Endpoints in the
9616 Protection Zone of the RMR.

9617 A linked Wide RMR may be used as a target for DTOs and Link/Unlink operations via all
9618 Endpoints in the Protection Zone of the RMR.

9619 APPLICATION USAGE

9620 The desired type of a Remote Memory Region must be specified at RMR creation time. The
9621 allowable RMR types are Implementation-dependent. However, InfiniBand Implementations at
9622 least support Wide RMRs and, if Verb Extensions (BMM) are provided, also Narrow RMRs;
9623 iWARP Implementations at least support Narrow RMRs. The `rmr_types_supported` field of the
9624 [it_ia_info_t](#) structure, which can be queried through [it_ia_query](#), indicates which RMR types are
9625 supported by the IA; i.e., Narrow RMRs, Wide RMRs, or both.

9626 SEE ALSO

9627 [it_rmr_create\(\)](#), [it_rmr_link\(\)](#), [it_rmr_query\(\)](#)

it_software_event_t

9628

9629 NAME

9630 it_software_event_t – Software Event type

9631 SYNOPSIS

```
9632 #include <it_api.h>
9633
9634 typedef struct {
9635     it_event_type_t event_number;
9636     it_evd_handle_t evd;
9637     void *data;
9638 } it_software_event_t;
```

9639 DESCRIPTION

9640 *event_number* Identifier of the Event type. Valid values:
9641 IT_SOFTWARE_EVENT

9642 *evd* Handle for the Event Dispatcher where the Event was queued.

9643 *data* The pointer that the Consumer furnished to the *it_evd_post_se* routine.

9644 An IT_SOFTWARE_EVENT_STREAM Event is generated when the Consumer calls the
9645 *it_evd_post_se* routine to post a Software Event.

9646 The IT_SOFTWARE_EVENT_STREAM Event Stream supports Events with the *event_number*
9647 IT_SOFTWARE_EVENT (see *it_event_t*).

9648 The Software Event type passes back the same pointer that the Consumer furnished to the
9649 *it_evd_post_se* routine.

9650 All Events on an IT_SOFTWARE_EVENT_STREAM SEVD cause Notification. See
9651 *it_evd_create* for details of Notification.

9652 The Software Event type EVD does not overflow. Instead, *it_evd_post_se* will generate an
9653 immediate error if the Consumer attempts to queue a software Event on a full Software Event
9654 EVD.

9655 SEE ALSO

9656 *it_evd_post_se()*, *it_evd_create()*, *it_evd_wait()*, *it_event_t*

it_status_t

9657

9658 **NAME**

9659 it_status_t – definition of IT-API call return status

9660 **SYNOPSIS**

```
9661 #include <it_api.h>
9662
9663 typedef enum {
9664     IT_SUCCESS = 0,
9665     IT_ERR_ABORT,
9666     IT_ERR_ACCESS,
9667     IT_ERR_ADDRESS,
9668     IT_ERR_AEVD_NOT_ALLOWED,
9669     IT_ERR_ASYNC_AFF_EVD_EXISTS,
9670     IT_ERR_ASYNC_UNAFF_EVD_EXISTS,
9671     IT_ERR_CANNOT_RESET,
9672     IT_ERR_CONN_QUAL_BUSY,
9673     IT_ERR_EP_TIMEWAIT,
9674     IT_ERR_EVD_BUSY,
9675     IT_ERR_EVD_QUEUE_FULL,
9676     IT_ERR_FAULT,
9677     IT_ERR_IA_CATASTROPHE,
9678     IT_ERR_INTERRUPT,
9679     IT_ERR_INVALID_ADDRESS,
9680     IT_ERR_INVALID_AEVD,
9681     IT_ERR_INVALID_AH,
9682     IT_ERR_INVALID_ETIMEOUT,
9683     IT_ERR_INVALID_CM_RETRY,
9684     IT_ERR_INVALID_CN_EST_FLAGS,
9685     IT_ERR_INVALID_CN_EST_ID,
9686     IT_ERR_INVALID_CONN_EVD,
9687     IT_ERR_INVALID_CONN_QUAL,
9688     IT_ERR_INVALID_CONVERSION,
9689     IT_ERR_INVALID_DTO_FLAGS,
9690     IT_ERR_INVALID_EP,
9691     IT_ERR_INVALID_EP_ATTR,
9692     IT_ERR_INVALID_EP_KEY,
9693     IT_ERR_INVALID_EP_STATE,
9694     IT_ERR_INVALID_EP_TYPE,
9695     IT_ERR_INVALID_EVD,
9696     IT_ERR_INVALID_EVD_STATE,
9697     IT_ERR_INVALID_EVD_TYPE,
9698     IT_ERR_INVALID_FLAGS,
9699     IT_ERR_INVALID_HANDLE,
9700     IT_ERR_INVALID_IA,
9701     IT_ERR_INVALID_LENGTH,
9702     IT_ERR_INVALID_LISTEN,
9703     IT_ERR_INVALID_LMR,
9704     IT_ERR_INVALID_LTIMEOUT,
9705     IT_ERR_INVALID_MAJOR_VERSION,
9706     IT_ERR_INVALID_MASK,
9707     IT_ERR_INVALID_MINOR_VERSION,
```

9708	IT_ERR_INVALID_NAME,
9709	IT_ERR_INVALID_NETADDR,
9710	IT_ERR_INVALID_NUM_SEGMENTS,
9711	IT_ERR_INVALID_PDATA_LENGTH,
9712	IT_ERR_INVALID_PRIVS,
9713	IT_ERR_INVALID_PZ,
9714	IT_ERR_INVALID_QUEUE_SIZE,
9715	IT_ERR_INVALID_RECV_EVD,
9716	IT_ERR_INVALID_RECV_EVD_STATE,
9717	IT_ERR_INVALID_REQ_EVD,
9718	IT_ERR_INVALID_REQ_EVD_STATE,
9719	IT_ERR_INVALID_RETRY,
9720	IT_ERR_INVALID_RMR,
9721	IT_ERR_INVALID_RNR_RETRY,
9722	IT_ERR_INVALID_RTIMEOUT,
9723	IT_ERR_INVALID_SOFT_EVD,
9724	IT_ERR_INVALID_SOURCE_PATH,
9725	IT_ERR_INVALID_SPIGOT,
9726	IT_ERR_INVALID_THRESHOLD,
9727	IT_ERR_INVALID_UD_STATUS,
9728	IT_ERR_INVALID_UD_SVC,
9729	IT_ERR_INVALID_UD_SVC_REQ_ID,
9730	IT_ERR_LMR_BUSY,
9731	IT_ERR_MISMATCH_FD,
9732	IT_ERR_NO_CONTEXT,
9733	IT_ERR_NO_PERMISSION,
9734	IT_ERR_PAYLOAD_SIZE,
9735	IT_ERR_PDATA_NOT_SUPPORTED,
9736	IT_ERR_PZ_BUSY,
9737	IT_ERR_QUEUE_EMPTY,
9738	IT_ERR_RANGE,
9739	IT_ERR_RESOURCES,
9740	IT_ERR_RESOURCE_IRD,
9741	IT_ERR_RESOURCE_LMR_LENGTH,
9742	IT_ERR_RESOURCE_ORD,
9743	IT_ERR_RESOURCE_QUEUE_SIZE,
9744	IT_ERR_RESOURCE_RECV_DTO,
9745	IT_ERR_RESOURCE_REQ_DTO,
9746	IT_ERR_RESOURCE_RRSEG,
9747	IT_ERR_RESOURCE_RSEG,
9748	IT_ERR_RESOURCE_RWSEG,
9749	IT_ERR_RESOURCE_SSEG,
9750	IT_ERR_TIMEOUT_EXPIRED,
9751	IT_ERR_TOO_MANY_POSTS,
9752	IT_ERR_WAITER_LIMIT,
9753	IT_ERR_INVALID_SRQ,
9754	IT_ERR_SOFT_HI_WATERMARK,
9755	IT_ERR_HARD_HI_WATERMARK,
9756	IT_ERR_INVALID_WATERMARK,
9757	IT_ERR_INVALID_RECV_DTO,
9758	IT_ERR_INVALID_SRQ_SIZE,
9759	IT_ERR_SRQ_LOW_WATERMARK,
9760	IT_ERR_SRQ_BUSY,
9761	IT_ERR_SRQ_NOT_SUPPORTED,

```

9762         IT_ERR_INVALID_ADDR_MODE,
9763         IT_ERR_INVALID_RMR_TYPE,
9764         IT_ERR_OP_NOT_SUPPORTED,
9765         IT_ERR_EP_BUSY
9766     } it_status_t;

```

9767 **DESCRIPTION**

9768 Most IT-API function calls return *it_status_t* on function completion. IT_SUCCESS indicates
9769 that an IT-API operation was invoked successfully; otherwise, the return code indicates the
9770 reason for failure. See each individual reference page for the meaning of a return code in the
9771 Context of the function.

9772 Some API function calls are used to initiate asynchronous operations. For those function calls, a
9773 return value of IT_SUCCESS indicates only that the operation was successfully initiated; it does
9774 not indicate that it was successfully completed. To determine whether an asynchronous
9775 operation was successfully completed, the Completion Event for the asynchronous operation
9776 should be examined.

9777 **SEE ALSO**

9778 *it_address_handle_create()*, *it_address_handle_free()*, *it_address_handle_modify()*,
9779 *it_address_handle_query()*, *it_convert_net_addr()*, *it_ep_accept()*, *it_ep_connect()*,
9780 *it_ep_disconnect()*, *it_ep_free()*, *it_ep_modify()*, *it_ep_query()*, *it_ep_rc_create()*, *it_ep_reset()*,
9781 *it_ep_ud_create()*, *it_evd_create()*, *it_evd_dequeue()*, *it_evd_free()*, *it_evd_modify()*,
9782 *it_evd_post_se()*, *it_evd_query()*, *it_evd_wait()*, *it_get_consumer_context()*,
9783 *it_get_handle_type()*, *it_get_pathinfo()*, *it_handoff()*, *it_ia_create()*, *it_ia_free()*, *it_ia_query()*,
9784 *it_listen_create()*, *it_listen_free()*, *it_listen_query()*, *it_lmr_create()*, *it_lmr_free()*,
9785 *it_lmr_modify()*, *it_lmr_query()*, *it_lmr_sync_rdma_read()*, *it_lmr_sync_rdma_write()*,
9786 *it_post_rdma_read()*, *it_post_rdma_read_to_rmr()*, *it_post_rdma_write()*, *it_post_recv()*,
9787 *it_post_recvfrom()*, *it_post_send()*, *it_post_sendto()*, *it_pz_create()*, *it_pz_free()*, *it_pz_query()*,
9788 *it_reject()*, *it_rmr_link()*, *it_rmr_create()*, *it_rmr_free()*, *it_rmr_query()*, *it_rmr_unlink()*,
9789 *it_set_consumer_context()*, *it_socket_convert()*, *it_srq_create()*, *it_srq_free()*, *it_srq_modify()*,
9790 *it_srq_query()*, *it_ud_service_reply()*, *it_ud_service_request()*,
9791 *it_ud_service_request_handle_create()*, *it_ud_service_request_handle_free()*,
9792 *it_ud_service_request_handle_query()*, *it_dto_events*

9793

it_unaffiliated_event_t

9794

9795 NAME

9796 it_unaffiliated_event_t – Unaffiliated Asynchronous Event type

9797 SYNOPSIS

```
9798 #include <it_api.h>
9799
9800 typedef struct {
9801     it_event_type_t  event_number;
9802     it_evd_handle_t  evd;
9803     it_ia_handle_t   ia;
9804
9805     size_t  spigot_id;
9806 } it_unaffiliated_event_t;
```

9807 DESCRIPTION

9808 *event_number* Identifier of the Event type. Valid values:
9809 IT_ASYNC_UNAFF_SPIGOT_ONLINE,
9810 IT_ASYNC_UNAFF_SPIGOT_OFFLINE,
9811 IT_ASYNC_UNAFF_SEVD_ENQUEUE_FAILURE

9812 *evd* Handle for the Event Dispatcher where the Event was queued.

9813 *ia* The Handle for the IA that experienced the Unaffiliated Event.

9814 *spigot_id* The identifier for the Spigot that changed state on the IA. Valid only for the
9815 IT_ASYNC_UNAFF_SPIGOT_ONLINE and
9816 IT_ASYNC_UNAFF_SPIGOT_OFFLINE Events.

9817 IT_ASYNC_UNAFF_EVENT_STREAM Events are generated when an Unaffiliated
9818 Asynchronous Event occurs. There are several types of Unaffiliated Asynchronous Events, and
9819 each type is identified by *event_number*. The Consumer asks for Unaffiliated Asynchronous
9820 Events to be delivered when it creates an EVD for the Unaffiliated Asynchronous Event Stream
9821 using the *it_evd_create* call.

9822 The following table maps the values in the Unaffiliated Asynchronous Events *it_event_type_t*
9823 enumeration to a transport-independent description.

it_event_type_t Value	Generic Event Description
IT_ASYNC_UNAFF_SPIGOT_ONLINE	A Spigot on the IA that was previously offline is now online. The Implementation will only generate this Event if the <i>it_ia_info.spigot_online_support</i> value is IT_TRUE.
IT_ASYNC_UNAFF_SPIGOT_OFFLINE	A Spigot on the IA that was previously online is now offline. The Implementation will only generate this Event if the <i>it_ia_info.spigot_offline_support</i> value is IT_TRUE.

it_event_type_t Value	Generic Event Description
IT_ASYNC_UNAFF_SEVD_ENQUEUE_FAILURE	The API Implementation was unable to enqueue an entry into an Affiliated Asynchronous Event SEVD.

9824 **EXTENDED DESCRIPTION**

9825 For the Infiniband transport, the following table maps the values in the Unaffiliated
 9826 Asynchronous Errors *it_event_type_t* enumeration to their corresponding “Unaffiliated
 9827 Asynchronous Events” and “Unaffiliated Asynchronous Errors” as specified in Chapter 11 of
 9828 [IB-R1.2].

it_event_type_t Value	IB “Unaffiliated Asynchronous Event/Error” Name
IT_ASYNC_UNAFF_SPIGOT_ONLINE	Port Active
IT_ASYNC_UNAFF_SPIGOT_OFFLINE	Port Error

9829 For the VIA transport, the following table maps the values in the Unaffiliated Asynchronous
 9830 Errors *it_event_type_t* enumeration to their corresponding descriptions in the
 9831 “VipErrorCallback” reference page in the Appendix of [VIA-V1.0].
 9832

it_event_type_t Value	VIA “VipErrorCallback” Name(s)
IT_ASYNC_UNAFF_SPIGOT_ONLINE	(Not applicable to the VIA transport.)
IT_ASYNC_UNAFF_SPIGOT_OFFLINE	(Not applicable to the VIA transport.)

9833 All Events on an IT_ASYNC_UNAFF_EVENT_STREAM SEVD cause Notification. See
 9834 *it_evd_create* for details of Notification.
 9835

9836 Overflow of an IT_ASYNC_UNAFF_EVENT_STREAM SEVD is not visible to the Consumer;
 9837 all subsequent IT_ASYNC_UNAFF_EVENT_STREAM Events are silently dropped until the
 9838 Consumer dequeues at least one Event from the EVD that contains the
 9839 IT_ASYNC_UNAFF_EVENT_STREAM.

9840 When a Consumer has created more than one IA corresponding to an underlying physical
 9841 adapter (say in different processes), then every Unaffiliated Event is replicated to every IA
 9842 instance.

9843 **SEE ALSO**

9844 *it_ia_create()*, *it_event_t*, *it_evd_create()*, *it_evd_wait()*

9845

9846

A Implementer's Guide

9847
9848
9849
9850
9851

The IT-API Standard does not prohibit any implementation from providing functionality beyond that specified in the standard. However, we urge that implementations and authors of code using IT-API avoid the “it_” prefix for any function name or data structure name not defined by the standard. This is to preserve the option for future enhancement of IT-API without concern that a new IT-API name will conflict with a name used in an existing Implementation or application.

9852

it_address_handle_create

9853
9854
9855
9856
9857

The Infiniband Create Address Handle verb does not take a Source GID as input; it takes a Source GID index. The Implementation therefore needs to use the Query HCA verb to get access to the GID table associated with the port identified by *spigot_id*, and match the input *ib.local_port_gid* field to an entry in the GID table to determine the appropriate Source GID index to use.

9858
9859
9860
9861
9862

The Infiniband Create Address Handle verb does not take a Source LID as input, it takes the Source Path Bits instead and uses them in conjunction with the Base LID to create the appropriate SLID to use. The Implementation therefore needs to use the Query HCA verb to retrieve the LMC associated with the port identified by *spigot_id* to determine how many of the low-order bits in the input *ib.local_port_lid* need to be extracted as the Source Path Bits.

9863
9864
9865

When running over the InfiniBand Transport, if the Consumer provides a Path to *it_address_handle_create* that contains a *P_Key* that is not in the HCA's *P_Key* table, the Implementation shall return `IT_ERR_INVALID_SOURCE_PATH`.

9866

it_affiliated_event_t

9867
9868
9869
9870

Asynchronous Events should be copied from hardware resources into per-process software queues. The effect of overflow of the software queue should be isolated to the owning process. When overflow of the Affiliated Event EVD occurs, hardware resources should still be dequeued and discarded.

9871
9872

`IT_ASYNC_AFF_SEVD_ENQUEUE_FAILURE` should be generated for overflow on all Simple EVDs with the exception of the Affiliated and Unaffiliated Event EVDs.

9873

it_cm_msg_events

9874
9875
9876
9877
9878

`IT_CM_MSG_CONN_BROKEN_EVENT` Events should be synthesized by the Implementation from asynchronous Events, etc., generated by the underlying transport. The Consumer has the option of ignoring asynchronous Events (by not creating an EVD for the Affiliated or Unaffiliated Asynchronous Event Streams) but still needs warning of state changes affecting their Endpoints.

9879 In general, all transport-specific Connection Management rejection Events not explicitly defined
9880 in this API should be implemented as IT_CM_MSG_CONN_NONPEER_REJECT_EVENT
9881 with IT_CN_REJ_OTHER reject reason code Events.

9882 When running over the InfiniBand Transport, Implementations have the option to “chew up” a
9883 REJ that is returned with reject reason code 1 (No QP available), 3 (No resources available), or 4
9884 (Timeout) rather than immediately posting an Event with status IT_CN_REJ_RESOURCES (for
9885 REJ codes 1 and 3) or IT_CN_REJ_OTHER (for REJ code 4). The Implementation may wait
9886 until the timeout specified by the Consumer in the *it_path_t* structure input to *it_ep_connect*
9887 expires and then enqueue a non-peer reject Event with an IT_CN_REJ_TIMEOUT status.
9888 Alternatively, within the specified timeout period the Implementation may retry the Connection
9889 establishment attempt on the Consumer's behalf. If a Connection could not be established within
9890 the Consumer-specified timeout period, the Implementation should enqueue a non-peer reject
9891 Event with an IT_CN_REJ_TIMEOUT status after the timeout period has expired.

9892 When running on the IB transport, there are two different things that can signal the
9893 Implementation that it should generate the IT_CM_MSG_CONN_ESTABLISHED_EVENT on
9894 the Passive side: receiving an RTU message, or receiving a “Communication Established”
9895 Affiliated Asynchronous Event from the HCA. (Due to inherent races in the IB Connection
9896 establishment process, it is also possible that both of these conditions could be present.) If the
9897 Implementation receives an RTU message while the Endpoint is in the
9898 IT_EP_STATE_PASSIVE_CONNECTION_PENDING state and within the timeout period
9899 advertised to the Passive side in the REQ message, it should generate an
9900 IT_CM_MSG_CONN_ESTABLISHED_EVENT Event and use the Private Data from the RTU
9901 as part of that Event. If the Implementation receives the “Communication Established” Affiliated
9902 Asynchronous Event without receiving an RTU, the Implementation should generate the
9903 IT_CM_MSG_CONN_ESTABLISHED_EVENT Event with a Private Data size of zero, and
9904 when/if the RTU for the Connection subsequently arrives the Implementation should ignore it.

9905 For the InfiniBand Transport, there is a potential race condition in the three-way handshake
9906 Connection establishment method between the Implementation generating the
9907 IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT Event and the Consumer calling
9908 *it_ep_disconnect* (or *it_ep_free*).

9909 The conditions for the race arise when a Consumer has called *it_ep_connect*, but before the
9910 Connection is successfully established, the Consumer calls *it_ep_disconnect* or *it_ep_free* on the
9911 Endpoint. Within the timeframe of this single Connection attempt, the Implementation must
9912 order Events as follows.

9913 It is acceptable for the Implementation to generate and queue the IT_CM_MSG_CONN_
9914 ACCEPT_ARRIVAL_EVENT Event to the SEVD prior to the IT_CM_MSG_CONN_
9915 DISCONNECT_EVENT Event. But, when the IT_CM_MSG_CONN_DISCONNECT_EVENT
9916 Event is generated, the Implementation must invalidate the *cn_est_id* found in the
9917 IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT Event. Likewise, if the Consumer calls
9918 *it_ep_free*, the Implementation must also invalidate the *cn_est_id*. (Note that it is possible that
9919 the *cn_est_id* may have already been invalidated by a Consumer call to *it_ep_accept*, *it_reject* or
9920 *it_handoff*.)

9921 On the other hand, if the Implementation has generated an IT_CM_MSG_CONN_
9922 DISCONNECT_EVENT Event, and subsequently the Implementation receives indication that

9923 the Connection Request has been accepted by the remote side, the Implementation shall not
 9924 generate an IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT Event. In this situation, the
 9925 Implementation is still responsible for generating any necessary transport-specific response to
 9926 the arrived acceptance message.

9927 **it_conn_qual_t**

9928 When the IANA Port Number Connection Qualifier type is used with the VIA transport, the
 9929 IANA Port Number is mapped into a 2-byte VIA connection discriminator, with byte 0 of the
 9930 connection discriminator containing the upper 8 bits of the 16-bit IANA Port Number, and byte
 9931 1 containing the lower 8 bits of the 16-bit IANA Port Number.

9932 When the IANA Port Number Connection Qualifier type is used with the InfiniBand Transport,
 9933 the IANA Port Number is mapped into the 64-bit Service ID as follows:

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
0x10	0x000CE1			0x0	0x0	IANA Port Number	

9934 **it_dto_cmpl_event_t**

9935 The handling of remotely detected errors is Implementation-dependent.

9936 On the iWARP Transport, when a locally posted RDMA DTO results in a remote access
 9937 violation, the remote iWARP Implementation tears down the connection abortively. For
 9938 implementing the IT_DTO_ERR_REMOTE_ACCESS Completion Error on iWARP, the
 9939 RDMAC Verbs provide a “remote termination error” completion status that allows on-the-fly
 9940 conversion of a received Terminate message to a Completion Error. Due to delays in the
 9941 reception of the Terminate message, it may not always be possible to use this mechanism.

9942 **it_dto_flags_t**

9943 The Implementation should attempt to support the use of IT_NOTIFY_FLAG on Receive DTOs.
 9944 Where the underlying transport does not support Receive DTO Notification Suppression it may
 9945 be necessary for the Implementation to generate Receive Notifications regardless of the setting
 9946 of the IT_NOTIFY_FLAG on the Receive DTOs.

9947 **it_ep_accept**

9948 In all cases where a communication manager message changes the state of an Endpoint the
 9949 Implementation must first transition the Endpoint state before generating the Event. This closes a
 9950 race condition where the Consumer may see the Event and call a function expecting the
 9951 Endpoint to be in the state that the Event results in. For example, when the Consumer reaps the
 9952 IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT the Endpoint should be in the
 9953 IT_EP_STATE_ACTIVE2_CONNECTION_PENDING state and ready to be accepted.

9954 **it_ep_attributes_t**

9955 The Implementation must allocate resources needed for the support of the Consumer-requested
9956 attributes. In general, the Implementation can allocate more resources than requested by the
9957 Consumer (a few exceptions are noted in the next paragraph). Some of these resources may
9958 actually be allocated at Connection establishment time for RC, but Connection establishment
9959 cannot fail because resources requested by the Consumer at Endpoint creation time are not
9960 available. Connection establishment may fail when the attributes associated with the local and
9961 remote Endpoints are mismatched. For example, for the InfiniBand and iWARP Transports, if
9962 the local Endpoint's attributes have an *rdma_read_ird* that is less than the remote Endpoint's
9963 *rdma_read_ord*, the Connection establishment attempt will fail.

9964 The attributes that the Implementation must allocate in exactly the quantity that the Consumer
9965 specified are:

9966 `it_rc_only_attributes_t.rdma_read_ord`
9967 `it_ep_attributes_t.max_dto_payload_size`

9968 The reason *rdma_read_ord* is listed above is that for InfiniBand you cannot establish a
9969 Connection unless the ORD value for one side of the Connection is less than or equal to the IRD
9970 value for the other side.

9971 The reason *max_dto_payload_size* is listed above is that for VIA you cannot establish a
9972 Connection unless the passive side and the active side Endpoints have matching values for this
9973 attribute.

9974 Whether the *max_dto_payload_size* limit is actually enforced for data transfers is IA-specific. (It
9975 might not be transport-specific; but it is not clear if all VIA Implementations do this checking.)

9976 An attempt to change *max_request_dtos* or *max_rcv_dtos* for an Endpoint of an Interface
9977 Adapter whose *it_ia_info_ep_work_queues_resizable* is clear must not be successful and
9978 IT_ERR_INVALID_EP_STATE is the return value for this case.

9979 When running over the InfiniBand or iWARP Transports, the Implementation must set Signaling
9980 type to Selectable in order to support *it_dto_flags*.

9981 **it_ep_connect**

9982 When running over the InfiniBand Transport, if the Consumer provides a Path to *it_ep_connect*
9983 that contains a *P_Key* that is not in the HCA's *P_Key* table, the Implementation shall return
9984 IT_ERR_INVALID_SOURCE_PATH.

9985 **it_ep_disconnect**

9986 If the EP is already in the IT_EP_STATE_NONOPERATIONAL state, no messages or Events
9987 should be generated. In this case, the transport-level Disconnect Request should have been sent
9988 when the EP transitioned into the non-operational state.

9989 When running over the InfiniBand Transport, if *it_ep_disconnect* is called while the Endpoint is
9990 in the IT_EP_STATE_CONNECTED state, the Implementation should send CM DREQ
9991 (Disconnect Request) message.

9992

it_ep_free

9993

The *it_ep_free* call is equivalent to the destruction of the underlying transport Endpoint. Except as noted below for the *cn_est_id*, the Implementation is at liberty to retain resources until such a time as it is capable of freeing them. For IB this means that Completion Events may be left on the CQ after QP destruction, and CM-generated Events may be left on connect EVD.

9994

9995

9996

9997

This call must destroy the *cn_est_id* associated with the Endpoint if it has not been destroyed before. If the *cn_est_id* was destroyed it should not cause any problem for the Implementation. For the InfiniBand Transport, the *it_ep_free* call when the Endpoint is in IT_EP_STATE_ACTIVE1_CONNECTION_PENDING may be racing with the Implementation generating IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT that creates *cn_est_id*.

9998

9999

10000

10001

10002

When a remote Endpoint involved in Connection establishment is destroyed, locally the IT_CM_MSG_CONN_NONPEER_REJECT_EVENT shall be generated with either IT_CN_REJ_OTHER, or IT_CN_REJ_TIMEOUT *reject_code* reason.

10003

10004

10005

it_ep_rc_create

10006

The Implementation should ignore the IT_EP_REUSEADDR *it_ep_rc_creation_flag_t* parameter on transports where the timewait state is not applicable.

10007

10008

For the InfiniBand Transport, QP creation does not allow specifying RDMA Read/Write privileges. Therefore the QP must be modified after creation to ensure that QP RDMA Read/Write privileges match those requested in the *ep_attr* data structure.

10009

10010

10011

For the iWARP transport, the QP attribute for MW bind operations shall be set to “enabled”.

10012

it_ep_reset

10013

This operation must hide any internal Implementation waiting for timeout expiration that the Endpoint may be in due to an *it_ep_disconnect* call during Connection set up.

10014

10015

it_ep_state_t

10016

In all cases where a communication manager message changes the state of an Endpoint the Implementation must first transition the Endpoint state before generating the Event. This closes a race condition where the Consumer may see the Event and call a function expecting the Endpoint to be in the state in which the Event results. For example, when the Consumer reaps the IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT the Endpoint should be in the IT_EP_STATE_ACTIVE2_CONNECTION_PENDING state and ready to be accepted.

10017

10018

10019

10020

10021

10022

The Connection establishment identifier (*conn_est_id*) object should not be destroyed by the Implementation when Endpoint transition state occurs due to a communication manager message or error. They should only be destroyed by explicit Consumer-initiated functions such as *it_ep_accept*, *it_reject*, and *it_ep_disconnect*.

10023

10024

10025

10026 **it_evd_create**

10027 An IT-API Implementation should meet the following rules for EVD behavior.

10028 **Definitions**

- 10029 1. An arriving (queued on SEVD) Event is a *notification event* if any one of the following is
10030 true:
- 10031 a. It is an Event for a DTO with IT_NOTIFY_FLAG set when posted.
 - 10032 b. It is an Event for a Recv DTO where matching Send DTO was originally posted
10033 with the IT_SOLICITED_WAIT_FLAG set (regardless of IT_NOTIFY_FLAG on
10034 Recv DTO).
 - 10035 c. It is the *N*th Event to arrive where threshold is set to *N*.
 - 10036 d. It is an Event for a DTO completing in error regardless of IT_NOTIFY_FLAG and
10037 IT_COMPLETION_FLAG.
 - 10038 e. It is a non-DTO Completion Event.

10039 The above are called *arriving notification events*.

10040 The Events of type a, b, d, and e are also called *plain notification events* and retain their
10041 notification status⁹ on the SEVD queue.

- 10042 2. An arriving Event is a *non-notification event* if none of the above 1a. to 1e. criteria are
10043 met. These are called both *arriving non-notification events* and *plain non-notification*
10044 *events*.
- 10045 3. IT_THRESHOLD_DISABLED stands for no threshold or threshold == infinity.
- 10046 4. The desired semantic for IT-API is: *it_evd_wait* returns only:
- 10047 a. For SEVD – when there is a Notification Event of 1a, 1b, 1d, or 1e type (above) or
10048 when there are a number of Events on the SEVD equal to or more than the
10049 threshold on the SEVD queue.
 - 10050 b. For AEVD – when any one of the associated SEVDs satisfies 4a.

10051 **Rules**

- 10052 1. *it_evd_wait* will block¹⁰ when:
- 10053 a. For SEVD – queue is empty.
 - 10054 b. For AEVD – all associated SEVDs are empty.

⁹ InfiniBand does not retain the notion of *notification status* for types *a*, *b*, and *d* after arrival. Hence, Implementations that do not rely on this notion, and only rely on *arriving notification events* must be permitted. Events of type *e* have their own Event Stream types and their own SEVDs. Hence, Implementation can retain the *notification status* for them based on SEVD Event Stream type.

¹⁰ This is really an Implementation issue. Semantically, Consumer does not care if it is blocked or not.

- 10055
10056
2. *it_evd_wait* may block if there are no notification events and number of Events is below the threshold:
- 10057
- a. For SEVD – on the SEVD queue.
- 10058
- b. For AEVD – on all associated SEVD queues.
- 10059
10060
3. *it_evd_wait* **will** return if there is a *notification event*¹¹ or the number of Events is greater than or equal to the *threshold*:
- 10061
- a. For SEVD – on the SEVD queue.
- 10062
- b. For AEVD – on any associated SEVDs.
- 10063
10064
4. If threshold > 1, *it_evd_wait* **should** block¹² if less than threshold number of Events and no *notification events* are¹³:
- 10065
- a. For SEVD – on the SEVD queue.
- 10066
- b. For AEVD – on all SEVD queues of AEVD.
- 10067
10068
5. Each *arriving notification event* **must** unblock at least one waiter¹⁴, but **should** unblock only one waiter.
- 10069
6. An *arriving notification event* **can** unblock as many waiters as there are Events available.
- 10070
7. *it_evd_dequeue* **will** return an Event if one exists regardless of waiters from:
- 10071
- a. For SEVD – the SEVD queue.
- 10072
- b. For AEVD with IT_EVD_DEQUEUE_NOTIFICATIONS set – the AEVD queue of IT_AEVD_NOTIFICATION_EVENTS
- 10073
- c. For AEVD with IT_EVD_DEQUEUE_NOTIFICATIONS cleared – any of its SEVDs.¹⁵
- 10074
10075

¹¹ This semantic mandates that events retain their notification status on the EVD (a “sticky notification” semantic). However, the event that passes a threshold number of events should not be considered to be a notification event – only the number of events on the SEVD(s) should be considered at the time *it_evd_wait* is called (if any SEVD has threshold number of events on it, then *it_evd_wait* must return).

¹² This terminology allows an Implementation that can support thresholds as well as other notification events in hardware to do so while *not* mandating that those Implementations that cannot use suboptimal schemes.

¹³ With *sevd_threshold* = IT_THRESHOLD_DISABLED (= infinity), *it_evd_wait* will block if no notification event is on the EVD. If a notification event is on the EVD, *it_evd_wait* will return each event until the notification event is dequeued; from then on, *it_evd_wait* may block.

¹⁴ Events that arrive when there are no blocked *it_evd_wait* calls *should* retain their notification status. Events that had IT_SOLICITED_WAIT_FLAG or IT_NOTIFY_FLAG set when originally posted or completions with errors that arrive at a time when no waiters are waiting will cause *it_evd_wait* to return when later called on the EVD. An Implementation *can* be over-eager and return from an *it_evd_wait* call without checking the notification status of events on the EVD.

¹⁵ If there is an event on any of the feeding SEVDs, *it_evd_dequeue* must return it regardless of the notification status of the SEVD and even if the SEVD is disabled.

10076 **Heuristic for Wakeup of Multiple Waiters**

10077 Rules 5 and 6 in the EVD Rules (above) require that multiple waiters be awakened on every
10078 Notification Event because of the possibility of Notification coalescing.

10079 The following heuristic is recommended: In the Notification Handler, check the queue length¹⁶,
10080 and wake up that many waiters and no more. This limits the number of threads that wake up, and
10081 reduces the number that wake up and find no Event because some other thread(s) has consumed
10082 them.

10083 The handler algorithm should be something like:

```
10084 evd_handler() {  
10085     for (;;) {  
10086         nToUnblock = min(nmore(),nwaiters());  
10087         for (n = 0; n != nToUnblock; ++n)  
10088             unblockWaiter();  
10089         if (nwaiters() == 0) break;  
10090         rearmHandler();  
10091         if (nmore() == 0) break;  
10092     }  
10093 }
```

10094 **Additional Issues**

10095 An AEVD does not have a queue as such. Potentially, an AEVD can be implemented using a bit
10096 array with an entry for each feeding SEVD. An SEVD bit "set" for an AEVD with
10097 IT_EVD_DEQUEUE_NOTIFICATIONS set means that the SEVD Handle can be returned in an
10098 IT_AEVD_NOTIFICATION_EVENT Event from the AEVD wait or dequeue. The SEVD bit
10099 "set" for an AEVD with IT_EVD_DEQUEUE_NOTIFICATIONS cleared means that the actual
10100 underlying SEVD Event can be returned from the AEVD wait or dequeue.

10101 Ideally, the SEVD bit is cleared as soon as the SEVD is not in the Notification criteria. But it can
10102 be cleared as soon as the SEVD becomes empty or, instead, only when both it becomes empty
10103 and there is direct (waiter on SEVD) or indirect waiter (AEVD waiter or FD) on the SEVD.

10104 If the SEVD is enabled, then an arriving SEVD Event that causes the SEVD to reach
10105 Notification criteria or maintain Notification criteria of the SEVD will set the SEVD bit for the
10106 AEVD.

10107 It is recommended that the Implementation employ a "starvation-free" algorithm in returning
10108 Events via an AEVD from underlying SEVDs. That is, the Implementation should ensure
10109 forward progress on all SEVDs feeding an AEVD.

10110 For IT_EVD_DEQUEUE_NOTIFICATION set, the Implementation should eventually select
10111 every feeding SEVD that has reached Notification status when generating the next AEVD
10112 Notification Event.

10113 For IT_EVD_DEQUEUE_NOTIFICATION cleared, the Implementation of *it_evd_wait* should
10114 eventually select the first Event from every feeding SEVD that has reached Notification status.

¹⁶ The queue length function is not Verbs-compliant, but will be commonly available.

10115 When the requested stream type is `IT_ASYNC_AFF_EVENT_STREAM` or
10116 `IT_ASYNC_UNAFF_EVENT_STREAM`, the Implementation creates the requested EVD, and
10117 fills in the appropriate *evd* field in the associated *it_ia_info_t* structure for that IA.

10118 The Implementation shall not provide to the Consumer Unaffiliated and Affiliated Events that
10119 happened before the Consumer created SEVDs for an Unaffiliated and Affiliated Event Stream.
10120 The Implementation can provide Events that happened during SEVD creation time.

10121 **it_evd_dequeue**

10122 The Implementation should abide by the EVD rules defined in the *it_evd_create* section of this
10123 Implementers Guide.

10124 All synchronization issues between multiple waiters and/or dequeue-ers from the same Event
10125 Dispatcher simultaneously are left to the Implementation.

10126 The Implementation is not required to check that the *Event* structure that the Consumer provides
10127 is sufficient to hold a returned Event.

10128 **it_evd_post_se**

10129 All the synchronization issues between multiple Consumer Contexts trying to post software
10130 Events to an Event Dispatcher instance simultaneously are left to the Implementation.

10131 **it_evd_wait**

10132 The Implementation should abide by the EVD rules defined in the *it_evd_create* section of this
10133 Implementers Guide.

10134 All synchronization issues between multiple waiters and/or dequeue-ers from the same Event
10135 Dispatcher simultaneously are left to the Implementation.

10136 The Implementation is not required to check that the *event* structure that the Consumer provides
10137 is sufficient to hold a returned Event.

10138 **it_event_t**

10139 Each Event Stream has a designated contiguous range of Event numbers with common most-
10140 significant bits (as masked by `IT_EVENT_STREAM_MASK`) representing the Event Stream.

10141 **it_handle_t**

10142 The definition of all object Handles is Implementation-specific, but all object Handles can be
10143 typecast to *it_handle_t* and *vice versa*.

10144 **it_handoff**

10145 Once the IA is open, and until all processes close it, the *ia_name* and Spigot identifier need to
10146 reference the same objects for all processes that are referencing the IA.

10147 **it_ia_create**

10148 The Implementation shall not provide to the Consumer Unaffiliated and Affiliated Events that
10149 happened before the Consumer created SEVDs for the Unaffiliated and Affiliated Event Stream.
10150 The Implementation can provide Events that happened during SEVD creation time.

10151 **it_ia_free**

10152 The Implementation should free the IT Objects in the reverse order of their construction in order
10153 to guarantee that all underlying transport resources will be successfully freed.

10154 **it_ia_info_t**

10155 The 1000+ number range in the *it_transport_type_t* is intended to facilitate the short-term
10156 prototyping efforts of vendors who are developing new transports that work with the IT-API. A
10157 vendor that wishes to productize their prototyping effort should contact the ICSC in order to be
10158 assigned a permanent transport number in the <1000 range. The Implementation is not
10159 responsible for ensuring that two different vendors utilizing the 1000+ range of transport
10160 numbers do not collide.

10161 For the IB transport, when *it_ep_disconnect* is called there are two possible underlying CM
10162 messages that the Private Data could travel with: DREQ or REJ. The value of
10163 *disconnect_private_data_len* for IB should be the lesser of the maximum Private Data supported
10164 in a DREQ message and the maximum Private Data supported in a REJ message.

10165 **it_ia_query**

10166 The Implementation must track any allocation(s) of *it_ia_info_t* structures due to *it_ia_query*
10167 calls so that any allocated *it_ia_info_t* structure(s) can later be freed if necessary when
10168 *it_ia_info_free* or *it_ia_free* is called.

10169 **it_listen_create**

10170 If the EVD where the CM Request Events stream (IT_CM_REQ_EVENT_STREAM) is routed
10171 on the passive side is full, then the Active side shall receive an
10172 IT_CM_MSG_CONN_NONPEER_REJECT_EVENT Event with a reason code of
10173 IT_CN_REJ_TIMEOUT.

10174 **it_lmr_create**

10175 On the InfiniBand Transport, *it_lmr_create* can be implemented by using the Register Physical
10176 Memory Region or Register Memory Region verb. Use of the Register Physical Memory Region
10177 verb on InfiniBand incurs a risk that the InfiniBand Transport may return a differing address
10178 from what was requested by the Consumer (for instance, possibly due to misaligned addresses).

10179 However, using the Register Physical Memory Region verb has the advantage that control over
10180 Memory Region pinning/unpinning remains in generic, OS-provided code, while using the
10181 Register Memory Region verb would pass such control to IHV-private code. Using the Register
10182 Physical Memory Region verb is thus preferred, since it (i) retains sufficient OS control over

10183 pinned memory, (ii) supports multiple registrations of Memory Regions in a multi-IHV
10184 environment, and (iii) reduces duplication of pinning code in IHV verbs provider drivers.

10185 On the iWARP Transport, *it_lmr_create* can be implemented with the Register Non-Shared
10186 Memory Region verb.

10187 For both InfiniBand and iWARP, the Implementation of *it_lmr_create* must include the Enable
10188 Memory Window Binding input modifier to allow subsequent calls to *it_rmr_link* using this
10189 LMR. The IT_LMR_FLAG_SHARED option can be implemented by registering either a new or
10190 a shared Memory Region. The Implementation should search for a matching LMR. If a match is
10191 found, call the Register Shared Memory Region Verb; if not, call the Register (Physical)
10192 Memory Region verb for InfiniBand and the Register Non-Shared Memory Region verb for
10193 iWARP.

10194 The Implementation should protect against race conditions between multiple Consumers and
10195 avoid calling memory registration verbs multiple times for the same memory region.

10196 See the advice under *it_rmr_link* for comments concerning *rmr_context* and byte order.

10197 **it_lmr_free**

10198 For InfiniBand and iWARP, *it_lmr_free* corresponds to the Deregister Memory Region verb and
10199 the Deallocate STag verb, respectively.

10200 Beware that determining when it is permissible to unlock or unpin physical memory is tricky.
10201 The Implementation must handle multiple LMRs with overlapping ranges, LMRs on different
10202 IAs with overlapping ranges, and LMRs created in overlapping regions of shared memory by
10203 different Consumers. In addition, any of these LMRs may have been created with the
10204 IT_LMR_FLAG_SHARED flag set.

10205 **it_lmr_modify**

10206 On the InfiniBand Transport, *it_lmr_modify* corresponds to the Reregister Physical Memory
10207 Region or Reregister Memory Region verb.

10208 On the iWARP Transport, *it_lmr_modify* corresponds to the Reregister Non-Shared Memory
10209 Region verb. The Implementation should modify the 8-bit STag Key when modifying a Memory
10210 Region to reduce the potential for problems with stale STags.

10211 **it_lmr_query**

10212 For both InfiniBand and iWARP, *it_lmr_query* can be implemented with the Query Memory
10213 Region verb.

10214 For InfiniBand, the *actual_addr* and *actual_length* fields in *params* should be derived from the
10215 Actual Remote Protection Bounds returned by the verb, because the Consumer will be most
10216 concerned with the degree of exposure of the region to remote Consumers. For a transport
10217 Implementation that does not provide Memory Region bounds checking with byte-level
10218 granularity, the remote bounds are obtained by rounding the bounds requested by the Consumer
10219 to 4K byte boundaries, and the local bounds are obtained by rounding the bounds requested by

10220 the Consumer to page boundaries. The Actual Remote Protection Bounds will be contained
10221 within the Actual Local Protection Bounds for any IB transport Implementation. This means the
10222 Consumer may safely use the returned *actual_addr* and *actual_length* as both local and remote
10223 bounds.

10224 For iWARP and any transport supporting Relative Addressing, LMR creation and bounds
10225 checking must be performed with byte-level granularity. For InfiniBand 1.2, LMR creation and
10226 bounds checking should be performed with byte-level granularity. In case of byte-level
10227 granularity, the actual starting address and actual length must equal the requested starting
10228 address and requested length, respectively.

10229 **it_lmr_sync_rdma_read**
10230 **it_lmr_sync_rdma_write**

10231 There is no defined error code for the case where some portion of the *local_segments* array lies
10232 outside the Consumer's valid address space. It is expected that the Implementation will signal the
10233 application with the appropriate platform-dependent signal in this case, as would happen for any
10234 dereference of an invalid pointer.

10235 **it_post_rdma_read**

10236 The Implementation should avoid resource allocation as part of *it_post_rdma_read* to ensure that
10237 this operation is non-blocking and thread-safe. This operation cannot fail due to insufficient
10238 resources. All resource allocation required must be done at Endpoint creation time to ensure that
10239 all necessary resources are available at post time.

10240 The Implementation should support zero-copy data transfers and kernel bypass for the RDMA
10241 Read operation.

10242 For iWARP, certain access violations pertaining to the local sink buffer or Endpoint can be
10243 predicted by the Implementation before an RDMA Read Request is issued via the underlying
10244 transport protocol. An optimized Implementation will avoid issuing an RDMA Read Request in
10245 this case, although it is required to check incoming RDMA Read Responses regarding access
10246 rights.

10247 **it_post_rdma_read_to_rmr**

10248 This call is available for iWARP only.

10249 The Implementation should avoid resource allocation as part of *it_post_rdma_read_to_rmr* to
10250 ensure that this operation is non-blocking and thread-safe. This operation cannot fail due to
10251 insufficient resources. All resource allocation required must be done at Endpoint creation time to
10252 ensure that all necessary resources are available at post time.

10253 The Implementation should support zero-copy data transfers and kernel bypass for the RDMA
10254 Read operation.

10255 Certain access violations pertaining to the local sink buffer or Endpoint can be predicted by the
10256 Implementation before an RDMA Read Request is issued via the underlying transport protocol.

10257 An optimized Implementation will avoid issuing an RDMA Read Request in this case, although
10258 it is required to check incoming RDMA Read Responses regarding access rights.

10259 **it_post_rdma_write**

10260 The Implementation should avoid resource allocation as part of *it_post_rdma_write* to ensure
10261 that this operation is non-blocking and thread-safe. This operation cannot fail due to insufficient
10262 resources. All resource allocation required must be done at Endpoint creation time to ensure that
10263 all necessary resources are available at post time.

10264 The Implementation should support zero-copy data transfers and kernel bypass for the RDMA
10265 Write operation.

10266 **it_post_recv**

10267 The Implementation should avoid resource allocation as part of *it_post_recv* to ensure that this
10268 operation is non-blocking and thread-safe. This operation cannot fail due to insufficient
10269 resources. All resource allocation required must be done at Endpoint or Shared Receive Queue
10270 creation time to ensure that all necessary resources are available at post time.

10271 The Implementation should support zero-copy data transfers and kernel bypass for the Receive
10272 operation.

10273 **it_post_recvfrom**

10274 The Implementation should avoid resource allocation as part of *it_post_recvfrom* to ensure that
10275 this operation is non-blocking and thread-safe. This operation cannot fail due to insufficient
10276 resources. All resource allocation required must be done at Endpoint creation time to ensure that
10277 all necessary resources are available at post time.

10278 The Implementation should support zero-copy data transfers and kernel bypass for the
10279 ReceiveFrom operation.

10280 **it_post_send**

10281 The Implementation should avoid resource allocation as part of *it_post_send* to ensure that this
10282 operation is non-blocking and thread-safe. This operation cannot fail due to insufficient
10283 resources. All resource allocation required must be done at Endpoint creation time to ensure that
10284 all necessary resources are available at post time.

10285 The Implementation should support zero-copy data transfers and kernel bypass for the Send
10286 operation.

10287 **it_post_sendto**

10288 The Implementation should avoid resource allocation as part of *it_post_sendto* to ensure that this
10289 operation is non-blocking and thread-safe. This operation cannot fail due to insufficient
10290 resources. All resource allocation required must be done at Endpoint creation time to ensure that
10291 all necessary resources are available at post time.

10292 For details on handling IT_ERR_INVALID_AH under InfiniBand see compliance statement
10293 o10-2.1.1 in [IB-R1.2].

10294 The Implementation should support zero-copy data transfers and kernel bypass for the *SendTo*
10295 operation.

10296 **it_rmr_create**

10297 For both InfiniBand and iWARP, *it_rmr_create* corresponds to the Allocate Memory Window
10298 verb.

10299 **it_rmr_free**

10300 For InfiniBand and iWARP, *it_rmr_free* corresponds to the Deallocate Memory Window verb
10301 and the Deallocate STag verb, respectively.

10302 **it_rmr_link**

10303 On the InfiniBand Transport, *it_rmr_link* must immediately select the appropriate verb for
10304 linking an RMR, namely Bind MW for a Wide RMR and Post Send Bind MW for a Narrow
10305 RMR.

10306 On the iWARP Transport, *it_rmr_link* translates to the PostSQ verb's Bind Memory Window
10307 operation. The Implementation should modify the 8-bit STag Key when binding a Memory
10308 Window to reduce the potential for problems with stale STags.

10309 The *rmr_context* returned by *it_rmr_link* is defined to be returned in network byte order; i.e., in
10310 big endian format. The intent is that no further reordering of the bytes of the *rmr_context* will be
10311 performed by either the local or remote Consumer, or the local or remote Implementation. When
10312 the remote Consumer passes the *rmr_context* to a call such as *it_post_rdma_write*, the
10313 Implementation of *it_post_rdma_write* will put the first byte of *rmr_context* on the network wire
10314 first, the second byte of *rmr_context* second, and so on. Thus, the Implementation of *it_rmr_link*
10315 should return the *rmr_context* in the byte order that the IA expects to see on the wire, so that the
10316 IA may correctly interpret the incoming *rmr_context*.

10317 **it_rmr_query**

10318 For InfiniBand and iWARP, *it_rmr_query* can be implemented with the Query Memory Window
10319 verb.

10320 Alternatively, in order to return correct values for any RMR attributes that may change after
10321 RMR creation, the Implementation may track the RMR Create/Free operations and the
10322 completion status of RMR Link and RMR Unlink operations, including the parameters passed to
10323 these operations. To do so, the Implementation can replace the Consumer's cookie passed to
10324 *it_rmr_link* or *it_rmr_unlink* with an Implementation cookie that points to a structure. This
10325 structure stores the original Consumer cookie and all relevant parameters to the Link or Unlink
10326 operation. In *it_evd_dequeue*, the Implementation would peek at the operation type of the
10327 dequeued Completion Event. In addition to tracking RMR Create/Free operations, an
10328 Implementation would track the Bind MW, Post Send Type-2 MW Bind, and Post Send Local

10329 Invalidate operations for InfiniBand (note that InfiniBand does not provide an Unlink MW
10330 operation), and the PostSQ Bind Memory Window and PostSQ Invalidate Local STag operations
10331 for iWARP. If such an operation completes successfully, then extract the Implementation
10332 cookie, and restore the Consumer's cookie. Read the structure, and copy the parameters to
10333 storage associated with the RMR object. The next *it_rmr_query* call can obtain its parameters
10334 from this storage. This alternative requires that the Implementation always requests signaled
10335 completions, regardless of whether the Consumer includes IT_COMPLETION_FLAG in the
10336 *dto_flags* of an RMR Link/Unlink operation. For RMR Link/Unlink operations with
10337 IT_COMPLETION_FLAG cleared, the Implementation must dequeue the completion as if none
10338 was generated.

10339 **it_rmr_unlink**

10340 For InfiniBand, *it_rmr_unlink* must immediately select the appropriate verb for unlinking an
10341 RMR, namely Bind MW (with a length of zero) for a Wide RMR and Post Send Invalidate MW
10342 for a Narrow RMR.

10343 For iWARP, *it_rmr_unlink* corresponds to the PostSQ verb's Invalidate Local STag operation.

10344 **it_ud_service_request_handle_create**

10345 The Implementation only needs to validate the components of the *it_path_t* structure that refer to
10346 local entities, specifically *spigot_id*, *local_port_lid*, and *local_port_gid*.

10347 When running over the InfiniBand Transport, if the Consumer provides a Path to
10348 *it_ud_service_request_handle_create* that contains a *P_Key* that is not in the HCA's *P_Key*
10349 table, the Implementation shall return IT_ERR_INVALID_SOURCE_PATH.

10350 **it_unaffiliated_event_t**

10351 Asynchronous Events should be copied from hardware resources into per-process software
10352 queues. The effect of overflow of the software queue should be isolated to the owning process.
10353 When overflow of the Unaffiliated Event EVD occurs, hardware resources should still be
10354 dequeued and discarded.

10355 In the case of Unaffiliated Events, the underlying Implementation should copy every Event into
10356 each process-specific queue.

10357

10358 **B** Implementer's Guide to Connection Management for 10359 **iWARP**

10360 **B.1** Overview

10361 The IT-API provides RDMA services for several underlying RDMA transports, including
10362 InfiniBand and iWARP¹⁷. The specifics of these RDMA transports as well as different
10363 application requirements led to the design of two different Connection management interfaces
10364 for Reliable Connected (RC) RDMA transports, as introduced below.

10365 The Transport-Independent Interface (TII) provides a unified RDMA Connection management
10366 service abstraction for any RC transport supported by the IT-API. The TII includes the
10367 *it_ep_connect*, *it_listen_create*, *it_ep_accept*, *it_reject*, and *it_ep_disconnect* calls. The TII
10368 allows Consumers to negotiate the IRD and ORD parameters and to exchange Private Data
10369 during Connection establishment. Consumers are expected to preconfigure the IRD and ORD
10370 parameters of their Endpoints prior to calling *it_ep_connect* or *it_ep_accept*. From the
10371 Consumer's viewpoint, the RDMA service is available immediately after Connection
10372 establishment. For iWARP, Connection establishment with the TII involves creating and
10373 connecting a socket within the API Implementation, as described in Section 5.2. This socket is
10374 not exposed through a Lower Layer Protocol (LLP) handle to the Consumer.

10375 The Transport-Dependent Interface (TDI) is available for iWARP only and allows the
10376 Conversion of an unconnected Endpoint of the RC type and a connected socket to a connected
10377 Endpoint. The TDI includes the *it_socket_convert* and *it_ep_disconnect* calls. It enables
10378 Consumers to exchange an arbitrary (but non-zero) amount of streaming-mode data via a TCP
10379 socket (and possibly via an SCTP socket in the future) prior to performing a Conversion, which
10380 is a requirement for several TCP/IP-based applications. This prior exchange of streaming-mode
10381 data occurs outside the IT-API. In contrast to Connection establishment with the TII, the
10382 *it_socket_convert* call neither supports the negotiation of IRD and ORD parameters, nor the
10383 exchange of Private Data. The main reason for this simplification is that the peers have an
10384 opportunity to negotiate IRD and ORD and to exchange Private Data prior to Conversion;
10385 moreover, they have an opportunity to modify ORD after the Conversion¹⁸, i.e., while already in
10386 RDMA mode. The Implementation of the TDI is described in Section 5.3.

10387 Additional details are found in Chapter 5.

¹⁷ IT-API Version 2 will support TCP-based iWARP. SCTP support may be added in a later issue.

¹⁸ Unfortunately, there is no opportunity to modify IRD after the Conversion because this is an optional capability according to [VERBS-RDMAC].

10388 **B.2 Miscellaneous**

10389 **Private Data**

10390 The IT-API must report Private Data length supported for the *it_ep_connect*, *it_ep_accept*, and
10391 *it_reject* calls as 256 bytes and report the Private Data length supported for the *it_ep_disconnect*
10392 routine as zero.

10393 The IT-API must report Private Data length supported for the Unreliable Datagram service as
10394 zero as well even though the iWARP Transport does not support the IT_UD_SERVICE service
10395 type.

10396 The Private Data fields are fixed-size and it is the Consumer's responsibility to define which
10397 portion of the fields contain their own valid data. Such a requirement is legacy behavior from
10398 other RDMA transports (specifically InfiniBand).

10399 **Handshake**

10400 The IT-API supports only two-way Connection establishment for iWARP. The
10401 *three_way_handshake_support* boolean found in the *it_ia_info_t* structure must be reported as
10402 IT_FALSE.

10403 **B.3 Interoperability Considerations**

10404 The IETF has released a draft document describing modifications to the MPA startup wire
10405 protocol [INTEROP-IETF] intended to aid in achieving interoperability between the RDMAC
10406 and IETF versions of iWARP.

10407 Where interoperability is not possible – i.e., between Non-permissive AV-RNIC/IETFs and
10408 RDMAC AV-RNICS – Connection attempts via the TII or TDI should result in
10409 IT_CM_MSG_CONN_NONPEER_REJECT_EVENT Events with the reject reason code
10410 IT_CN_REJ_BAD_CONN_PARAMS or IT_CM_MSG_CONN_BROKEN_EVENT Events. See
10411 the Interoperability Ladder Diagrams found in both the TII and TDI sections of this document.

10412 Where the MPA startup protocol is not used – i.e., between RDMAC AV-RNICs –
10413 interoperability concerns are left to the ULP.

10414 **B.4 MPA Marker Control**

10415 The RDMAC version of MPA [MPA-RDMAC] mandated the use of MPA framing (Markers)
10416 for the use of recovering placement information for DDP frames. The IETF version of MPA
10417 [MPA-IETF] made the use of Markers optional and as specified by the receiver.

10418 The IT-API implementations should implement the recommendations of [INTEROP-IETF].

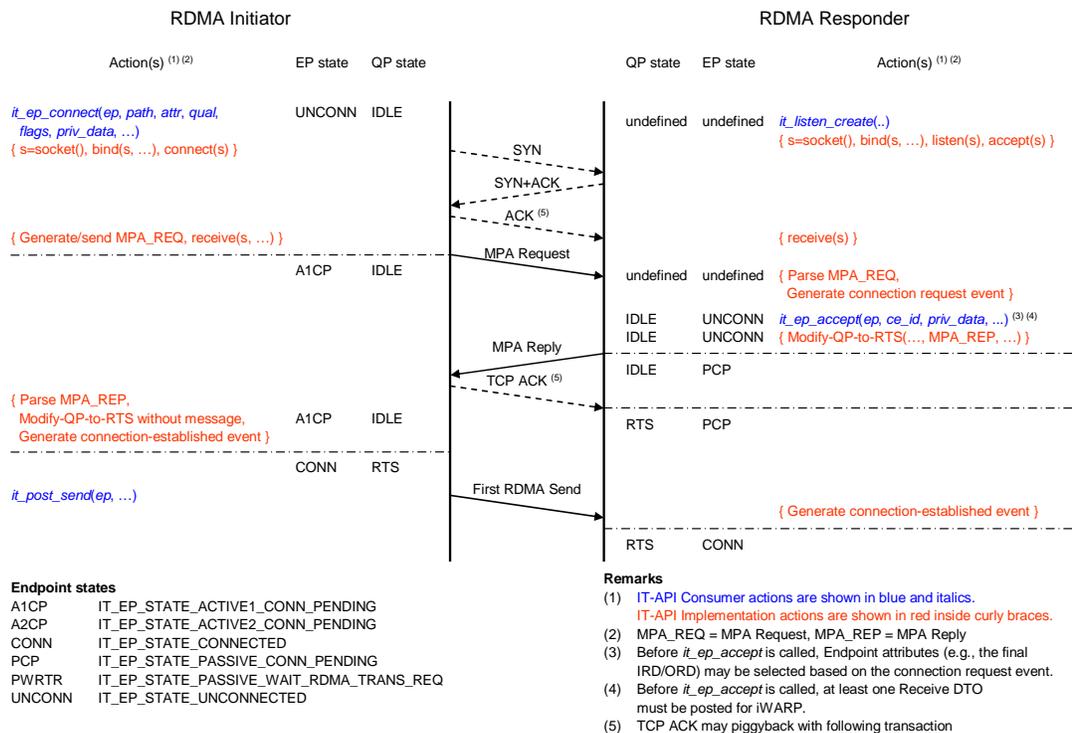
10419 B.5 Transport-Independent Interface

10420 Overview

10421 For the *Transport-Independent Interface (TII)*, the Consumer calling *it_ep_connect* and the
 10422 corresponding side of the Connection is referred to as the *RDMA Initiator*, while the Consumer
 10423 calling *it_listen_create* followed by *it_ep_accept* (or *it_reject*) and the corresponding side of the
 10424 Connection is referred to as the *RDMA Responder*.

10425 The Endpoint finite-state machine uses the active (passive) Endpoint states for the RDMA
 10426 Initiator (Responder).

10427 Connection establishment for the TII in case of an RNIC without RTR state (see below) is
 10428 illustrated in Figure 13.



10429
 10430 **Figure 13: Connection Establishment with MPA Startup (TII): RNIC without RTR State**

10431 The RDMA Responder uses *it_listen_create* to create a Listen Point, which causes the (API)
 10432 Implementation to create and bind a socket, to listen for an incoming LLP connection request,
 10433 and to accept such a request. The RDMA Initiator invokes *it_ep_connect* for connecting an
 10434 Endpoint of the RC type, which causes the Implementation to create and bind a socket and to
 10435 attempt a connection. As soon as the LLP connection is established, the Implementation
 10436 completes TII Connection establishment by performing an Immediate RDMA Transition. For

10437 TCP, the transition to RDMA mode corresponds to the *MPA Startup phase*, during which an
10438 MPA Request/Reply handshake takes place in streaming mode [MPA-IETF].

10439 The RDMA Transition is visible to the Consumers only through the exchange of Private Data; in
10440 particular, the Implementation handles the details of the MPA Startup.

10441 The RDMA Initiator generates and sends an MPA Request message with private data containing
10442 the IRD/ORD Header (IOH) and the private data provided by the Consumer with *it_ep_connect*.
10443 After receiving a valid MPA Reply message with the MPA Reject bit not set, the RDMA
10444 Initiator invokes the Modify-QP-to-RTS verb with a message length of zero and generates a
10445 Connection established event for the Consumer providing the remote Endpoint's IRD and ORD
10446 parameters as well as the remote Consumer's private data.

10447 The RDMA Responder's Listen Point expects an MPA Request. Upon receiving a valid MPA
10448 Request message, it generates a Connection Request event for the Consumer containing the
10449 remote Endpoint's IRD and ORD parameters as well as the remote Consumer's Private Data.
10450 The Consumer on the RDMA Responder side now invokes either *it_ep_accept* or *it_reject*,
10451 optionally with Private Data.

10452 Calling *it_ep_accept* causes the Implementation to associate the Connection establishment ID
10453 (and the corresponding LLP connection) with an Endpoint. It then causes the Implementation to
10454 generate an MPA Reply message with Private Data containing the IOH and the Private Data
10455 provided by the Consumer. Next, the Implementation invokes the Modify-QP-to-RTS verb,
10456 providing the MPA Reply message as the Last Streaming-Mode Message. Finally, upon
10457 detection of the first iWARP payload, the Implementation of *it_ep_accept* generates a
10458 Connection established event for the Consumer providing the remote Endpoint's IRD and ORD
10459 parameters as well as the remote Consumer's Private Data, and transitions the Endpoint into
10460 IT_EP_STATE_CONNECTED state.

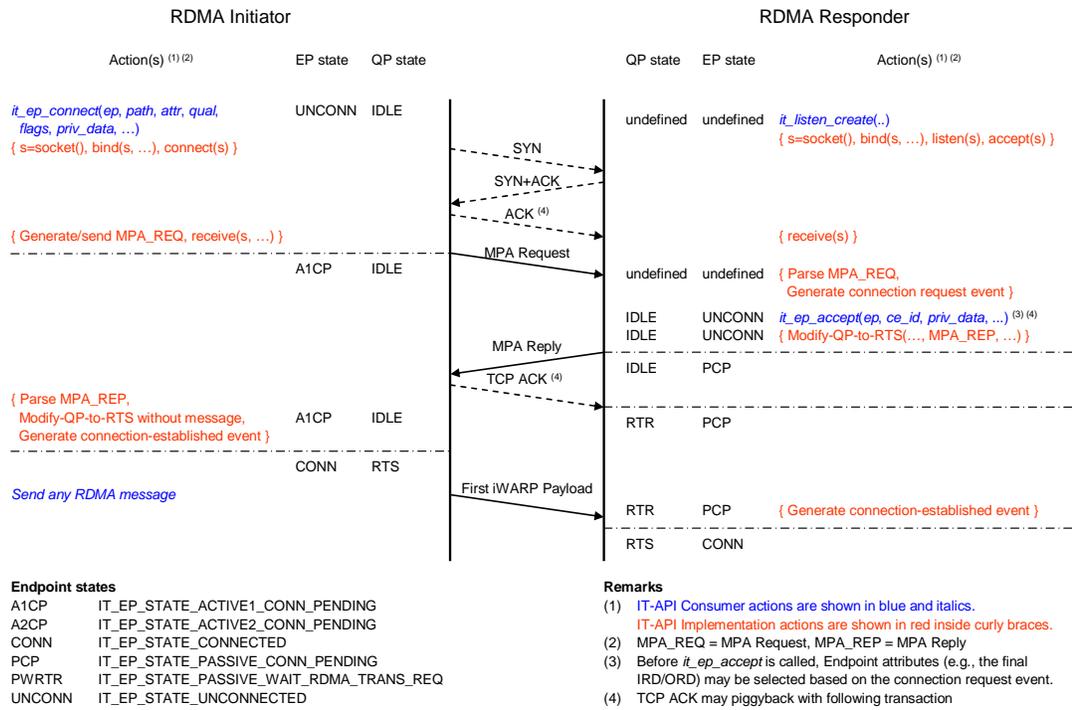
10461 For Figure 13, the Modify-QP-to-RTS verb is assumed to transition the QP to RTS state without
10462 waiting for the reception of the first iWARP payload (FPDU or RDMA message). While this
10463 behavior complies with [VERBS-RDMAC], it will not guarantee that the underlying iWARP
10464 Implementation stalls the sending of iWARP payload before the first iWARP payload has been
10465 received, as required by [MPA-IETF]. The fact that premature sending of the iWARP payload
10466 could result in synchronization problems and that receiving a Completion for an RDMAP Post
10467 Receive work request is the only way to detect the arrival of the first iWARP payload on an
10468 RNIC complying with [VERBS-RDMAC] has the following, far-reaching consequences:

- 10469 • For the ULP, after reaping the Connection established event, the RDMA Initiator must
10470 post a Send DTO to allow the RDMA Responder to detect the first iWARP payload.
- 10471 • For the ULP, the RDMA Responder must post at least one Receive DTO prior to calling
10472 *it_ep_accept*.
- 10473 • For the Implementation of *it_ep_accept*, the reception of the first iWARP payload must be
10474 detected by noting the Completion of the Consumer's first Receive DTO.

10475 In order to minimize transport-dependent ULP requirements, the IT-API offers an alternative for
10476 RNICs that implement an additional *Ready-To-Receive (RTR)* QP state to wait for the first
10477 iWARP payload, as well as an additional asynchronous event to indicate the RTR-to-RTS
10478 transition when the first iWARP payload is received on the RDMA Responder side. Connection

10479
10480

establishment for the TII in case of an RNIC with an RTR state (extended QP state machine) is depicted in Figure 14.



10481
10482

Figure 14: Connection Establishment with MPA Startup (TII): RNIC with RTR State

10483

This alternative approach has the following consequences:

10484
10485
10486

- For the ULP, after reaping the connection-established event, the RDMA Initiator must post a meaningful DTO (not necessarily a Send DTO) to allow the RDMA Responder to detect the first iWARP payload.

10487
10488

- For the Implementation of *it_ep_accept*, the reception of the first iWARP payload must be detected by noting the asynchronous event that indicates the RTR-to-RTS transition.

10489

IT-API ULP and IRD/ORD Header

10490
10491
10492
10493

RDMA transitions are not fully specified by the iWARP protocol suite. For instance, [MPA-IETF] leaves it up to its ULP (i) when to enter MPA Startup, (ii) which side initiates the MPA Startup (i.e., sends MPA Request), and (iii) how to use the private data in the MPA Request/Reply messages.

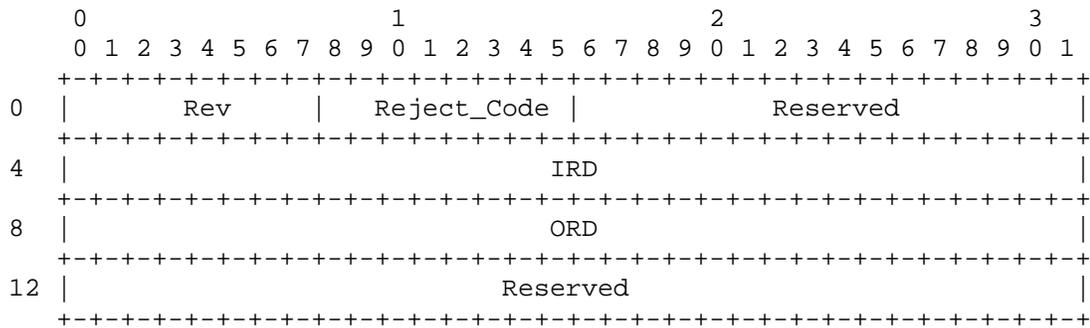
10494
10495

The TII of the IT-API fills this gap by specifying an *Immediate RDMA Transition* and thereby defines a (minimal) ULP with the following properties:

- 10496 • The MPA Startup Phase shall start immediately after LLP connection setup.
- 10497 • The peer initiating the LLP connection establishment shall initiate the MPA Startup.
- 10498 • Due to the lack of standardized IRD/ORD fields in the MPA Request/Reply messages and
10499 the requirement to exchange IRD/ORD during Connection establishment, the first 16
10500 bytes of the MPA Private Data buffer shall contain an *IRD/ORD Header (IOH)* specified
10501 by the IT-API for IRD/ORD negotiation. Note that there is an effort proposed for the
10502 IETF to standardize an equivalent header to the IOH but as of this writing details are not
10503 available. Due consideration of incorporation of the IETF proposal shall be taken.

10504 **IRD/ORD Header**

10505 The IT-API specifies the layout of the IRD/ORD Header (IOH) according to Figure 15.



10517 Rev: This field contains the Revision of the IRD/ORD header. For this
10518 version of the specification senders MUST set this field to '1'.
10519 Receivers compliant with this version of the specification MUST check
10520 this field for '1', and close the connection and report an error
10521 locally if any other value is detected.

10522 Reject_Code: This field is read in combination with the MPA reject bit
10523 and supplies additional information (if available). The field contains
10524 one of the following values: 0x00 means no additional information. 0x01
10525 means iWARP version mismatch (surfaced as IT_CN_REJ_BAD_CONN_PARAMS in
10526 IT-API). 0x02 means IRD too small (called IT_CN_REJ_BAD_ORD in IT-API).
10527 All other values are reserved. This field only has validity if the MPA
10528 reject bit is set.

10529 Reserved: This field is reserved for future use. It must be set to zero
10530 when sending, and must not be checked on reception.

10531 IRD: This field contains the sender's Inbound RDMA Read Queue Depth as
10532 a 32-bit word in network byte order.

10533 ORD: This field contains the sender's Outbound RDMA Read Queue Depth as
10534 a 32-bit word in network byte order.

10535 Reserved: This field is reserved for future use. It must be set to zero
10536 when sending, and not checked on reception.

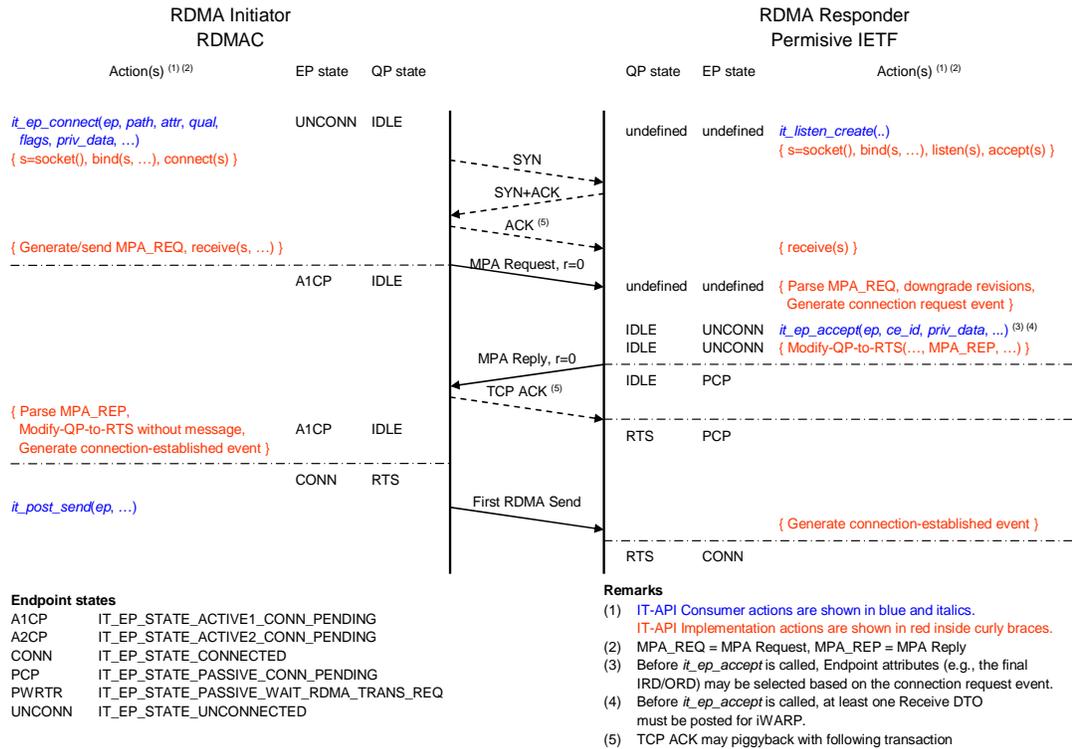
10543 **Figure 15: IRD/ORD Header**

10544
10545
10546

The IOH shall be conveyed as the first 16 bytes of Private Data in the MPA Request and MPA Reply frames. The Consumer's Private Data shall follow the IOH. The IOH shall not be visible to the Consumer.

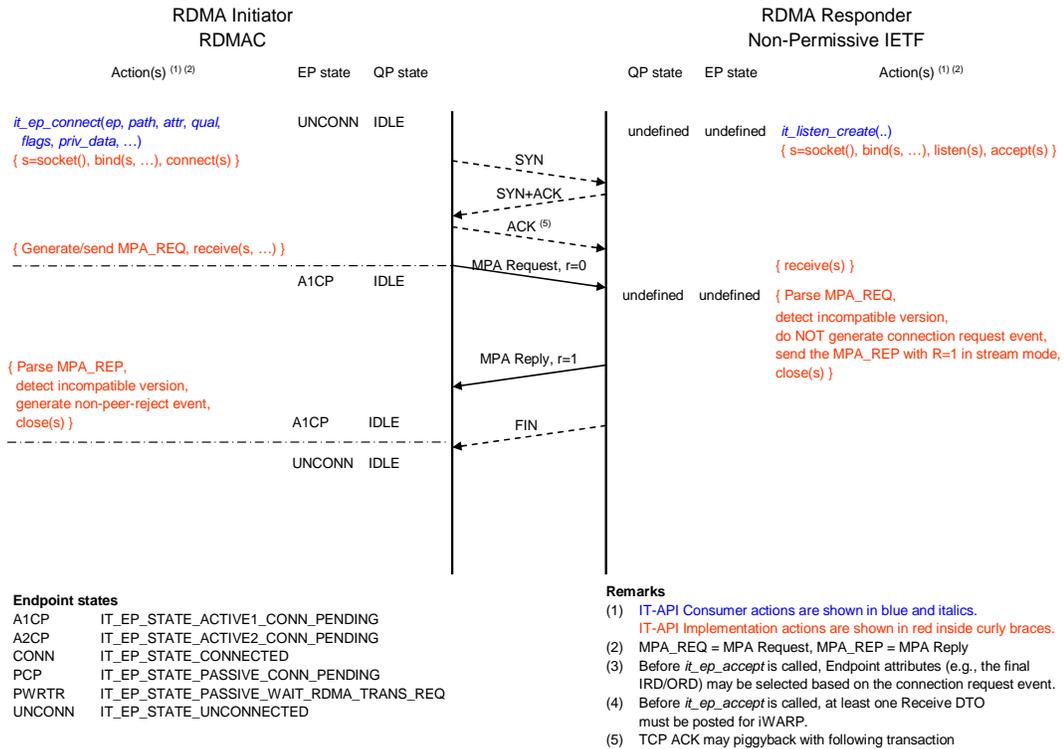
10547

TII Interoperability Ladder Diagrams



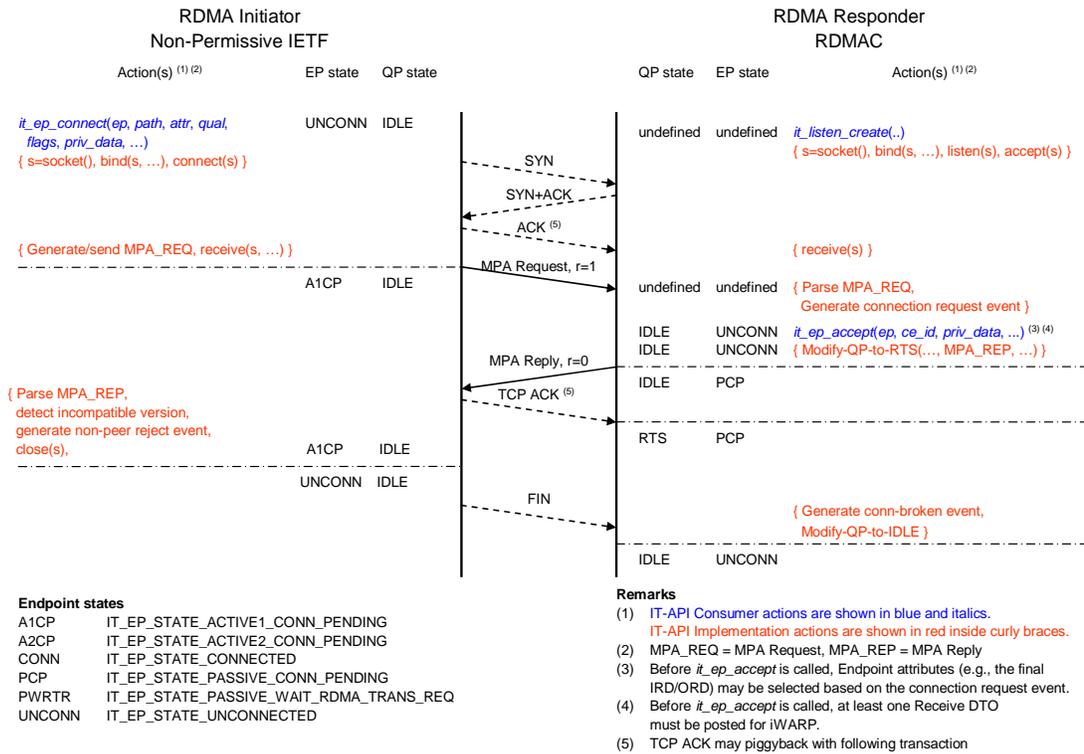
10548
10549

Figure 16: RDMAC Initiator to Permissive IETF Responder



10552
10553

Figure 18: RDMAC Initiator to Non-Permissive IETF Responder



10554
10555

Figure 19: Non-Permissive IETF Initiator to RDMAC Responder

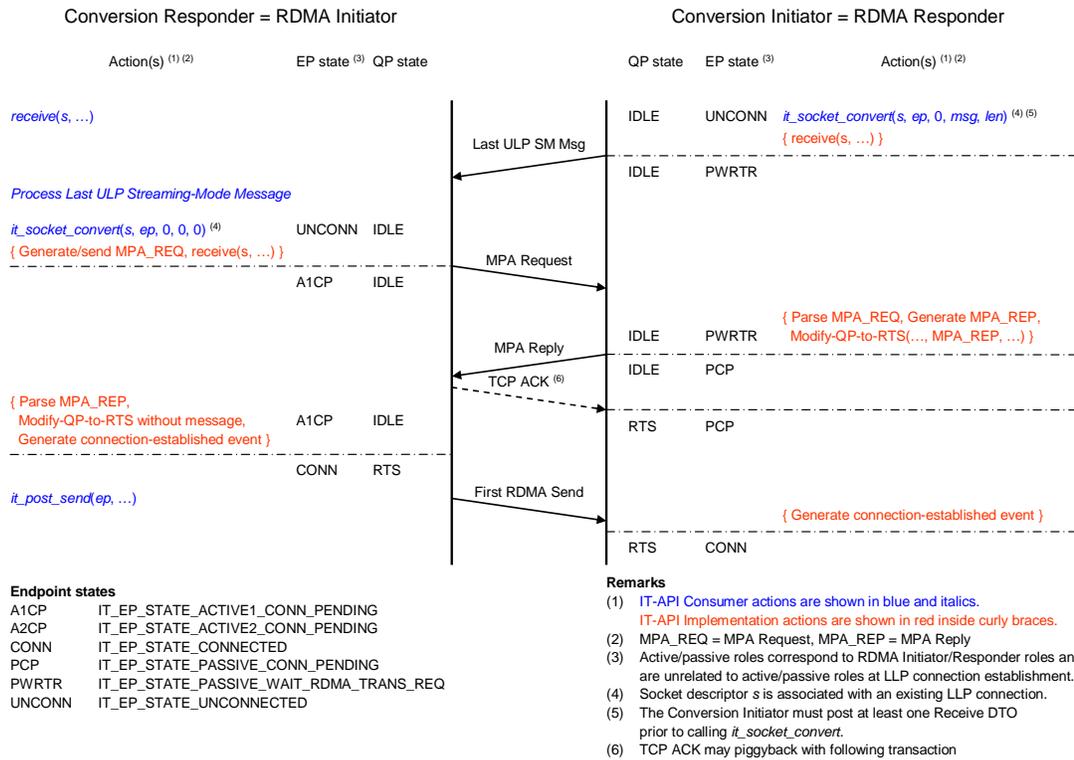
10556 **B.6 Transport-Dependent Interface**

10557 **Overview**

10558 For the *Transport-Dependent Interface (TDI)*, the Consumer calling *it_socket_convert* with a
 10559 *Last ULP Streaming-Mode (SM) Message* of length greater than zero is referred to as the
 10560 *Conversion Initiator*, while the Consumer calling *it_socket_convert* with a message length of
 10561 zero is referred to as the *Conversion Responder*.

10562 As for the TII, the Endpoint finite-state machine uses the active (passive) Endpoint states for the
 10563 RDMA Initiator (Responder). In order to avoid an inconsistency in the meaning of Endpoint
 10564 states between TII and TDI, the TDI uses the convention that the Conversion Initiator and the
 10565 corresponding side of the connection is in the *RDMA Responder* role, while the Conversion
 10566 Responder and the corresponding side of the connection is in the *RDMA Initiator* role.

10567 The Conversion process in case of an RNIC without RTR state is illustrated in Figure 20.



10568

10569

Figure 20: Conversion Process with MPA Startup (TDI) - RNIC without RTR State

10570

The Conversion process corresponds to a Deferred RDMA Transition as described, for instance, in [MPA-IETF]; for the Conversion Initiator, however, it includes the transmission of the Last ULP Streaming-Mode Message.

10571

10572

10573

For TCP, the transition to RDMA mode corresponds to the MPA Startup phase, during which an MPA Request/Reply handshake takes place in streaming mode [MPA-IETF]. The RDMA Transition is invisible to the Consumers; in particular, the Implementation handles the details of the MPA Startup.

10574

10575

10576

10577

The Conversion Initiator invokes *it_socket_convert*, passing an unconnected Endpoint of the RC type, a socket handle corresponding to an established LLP connection, a flags argument, and a Last ULP Streaming-Mode Message of length greater than zero, indicating the Conversion Initiator role. The (default) flag RTH indicates that the RDMA Transition Handshake – i.e. MPA Startup for MPA/TCP – shall not be suppressed. The (API) Implementation sends the Last ULP Streaming-Mode Message over the connection associated with the socket handle and expects to receive an MPA Request.

10578

10579

10580

10581

10582

10583

10584

Upon receiving a valid MPA Request message, it generates an MPA Reply message without any private data and invokes the Modify-QP-to-RTS verb, providing the MPA Reply message as the Last Streaming-Mode Message. Finally, upon detection of the first iWARP payload, the Implementation generates a connection-established event for the Consumer providing neither IRD/ORD parameters nor any Private Data from the remote Consumer, and transitions the

10585

10586

10587

10588

10589
10590

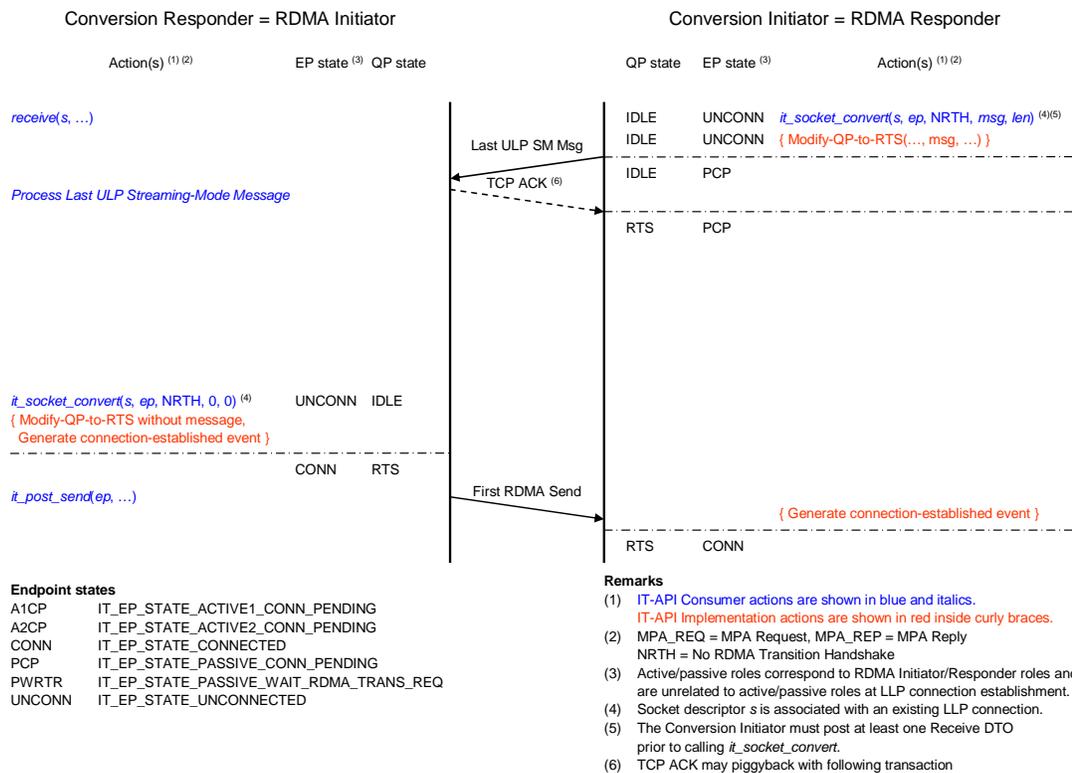
Endpoint into IT_EP_STATE_CONNECTED state. The IRD/ORD values shall appear as zero in the Event and the *private_data_present* flag shall be in the cleared state.

10591
10592
10593
10594
10595
10596
10597
10598
10599
10600
10601

The Conversion Responder now expects to receive the Last ULP Streaming-Mode Message. Upon receiving that message, it invokes *it_socket_convert*, passing an unconnected Endpoint of the RC type, a socket handle corresponding to an established LLP connection, a flags argument RTH, and two trailing NULL values, where the message length zero indicates the Conversion Responder role. The Implementation now generates an MPA Request message without any Private Data, sends it over the connection associated with the socket handle, and expects to receive an MPA Reply message. After receiving a valid MPA Reply message with the MPA Reject bit not set, the Implementation invokes the Modify-QP-to-RTS verb with a message length of zero and generates a Connection established event for the Consumer providing neither IRD/ORD parameters nor any Private Data from the remote Consumer, and transitions the Endpoint into IT_EP_STATE_CONNECTED state.

10602
10603
10604
10605

In order to provide interoperability with remote devices in the class AV-RNIC/RDMAC without MPA Startup, the TDI allows suppressing the RDMA Transition Handshake in the Conversion process by setting the flag NRTH in the *it_socket_convert* call. The Conversion process with MPA Startup suppression in case of an RNIC without RTR state is shown in Figure 21.



10606
10607

Figure 21: Conversion Process with MPA Startup Suppression (TDI): RNIC without RTR State

TDI Interoperability Ladder Diagrams

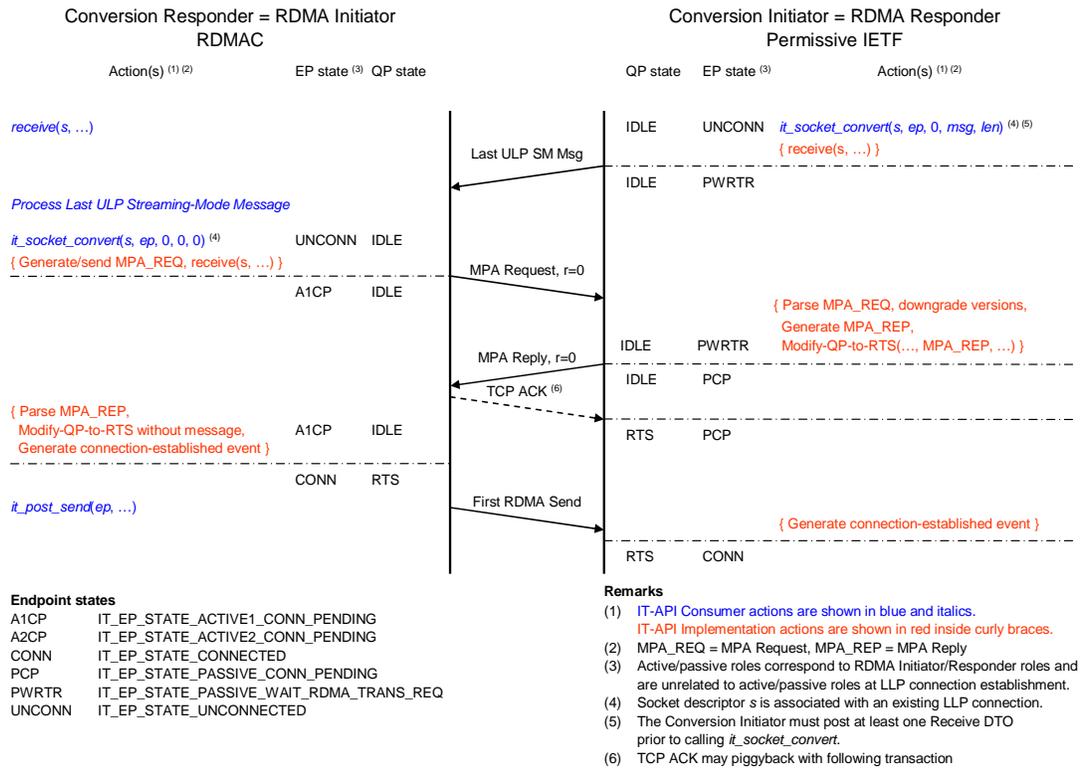
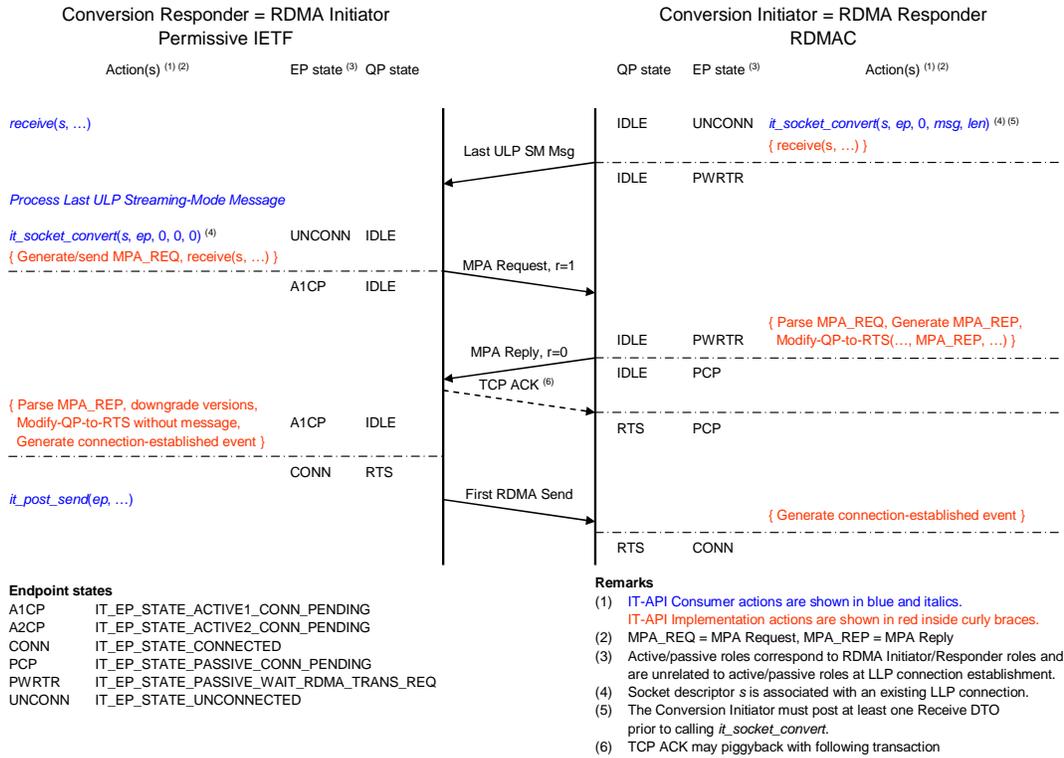
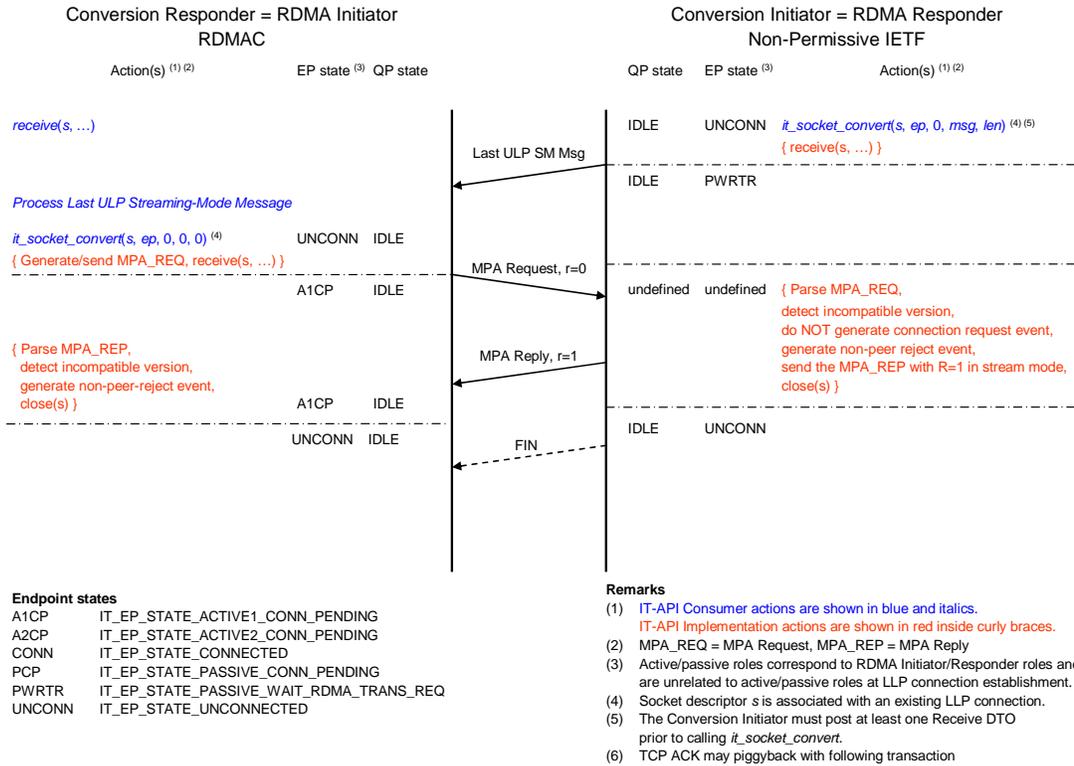


Figure 22: Permissive IETF Conversion Initiator to RDMAC Conversion Responder



10611
10612

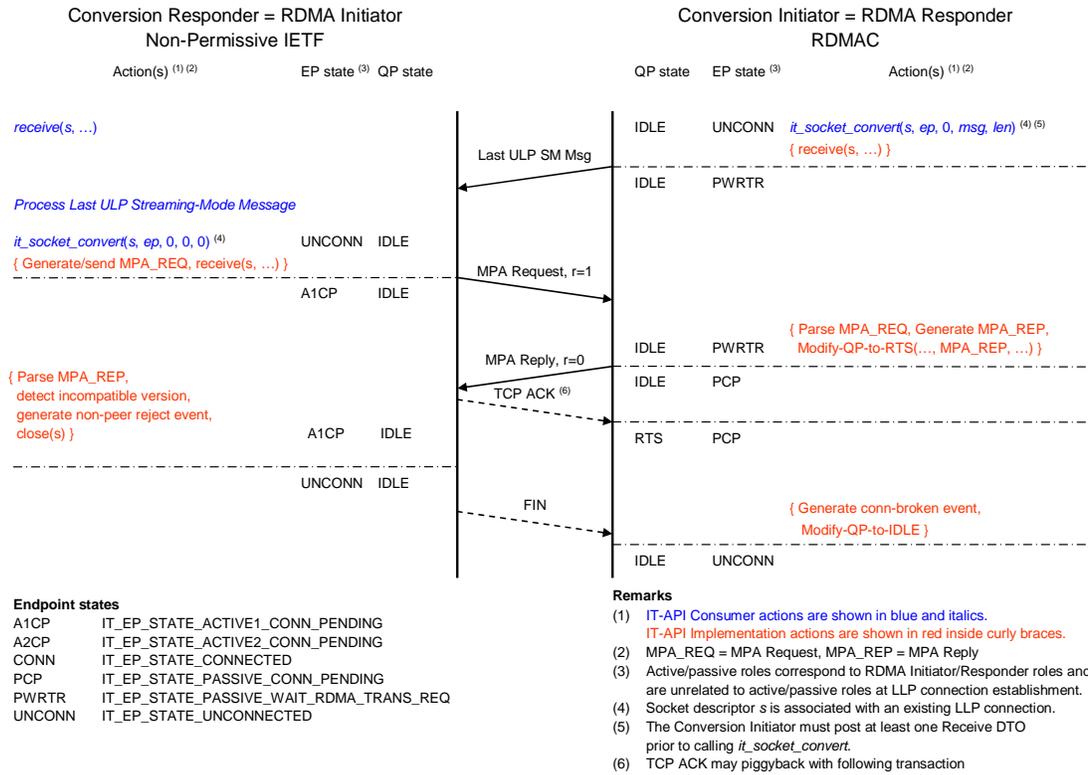
Figure 23: RDMAC Conversion Initiator to Permissive IETF Conversion Responder



10613

10614

Figure 24: Non-Permissive IETF Conversion Initiator to RDMAC Conversion Responder



10615
10616

Figure 25: RDMAC Conversion Initiator to Non-Permissive IETF Conversion Responder

10617

C Backwards Compatibility with Earlier IT-API Versions

10618
10619
10620
10621
10622

IT-API Version 2.0 is designed to minimize source-level incompatibility with IT-API Version 1.0. For several Version 1.0 calls, the functionality extensions for Version 2.0 required a modification of the function signature from Version 1.0 to Version 2.0; e.g., for *it_lmr_create*. Such functions will be called *modified functions* below. Backwards-compatibility is provided through the means described in the sections that follow.

10623

IT-API Version 2.0 Implementation Requirements

10624
10625

A Version 2.0 Implementation with Version 1.0 backwards-compatibility must provide the following:

10626
10627

- A set of macros in the `<it_api.h>` header file as shown below for mapping the names of deprecated structure fields

10628
10629

- Additional enumeration entries in *it_mem_priv_t* and *it_rmr_param_mask_t* as identified below

10630
10631

- A set of macros in the `<it_api.h>` header file as shown below for mapping the names of modified functions

10632
10633
10634
10635

For each modified function *it_fun*, implementations of both the Version 1.0 and the Version 2.0 functions are explicitly called *it_fun10* and *it_fun20*, respectively. The former can be a simple inline call to the latter, with default arguments as necessary. For *it_rmr_bind* (v1.0), Version 2.0 uses the new function name *it_rmr_link*; an *it_rmr_bind20* shall not be provided.

10636

Application Requirements

10637
10638

The compile-time flag `ITAPI_ENABLE_V20_BINDINGS` (see `<it_api.h>`) controls the backwards-compatibility mode.

10639
10640
10641
10642
10643

A Consumer wishing to compile a program with Version 1.0 bindings need not change anything and does not define the `ITAPI_ENABLE_V20_BINDINGS` flag. In this case, Version 1.0 functions continue to use Version 1.0 signatures. The `<it_api.h>` header file converts function names for modified functions to the *explicit* Version 1.0 function names *it_fun10* provided by the Implementation.

10644
10645
10646
10647
10648

A Consumer wishing to take advantage of Version 2.0 functionality, particularly in the area of memory management, must define the flag `ITAPI_ENABLE_V20_BINDINGS` before including `<it_api.h>`. In this case, modified functions must be used with the new Version 2.0 signatures. The `<it_api.h>` header file converts function names for modified functions to the *explicit* Version 2.0 function names *it_fun20* provided by the Implementation.

10649
10650

In any case, it is not recommended to use explicit Version 1.0 function names directly, as the functionality may be incomplete for certain transports and explicit Version 1.0 function names

10651 may be removed in a future IT-API version. Similarly, it is not recommended to use explicit
10652 Version 2.0 function names directly, as these may be renamed in a future IT-API version by
10653 dropping the suffix “20”.

10654 **Backwards-Compatibility Support in Header File**

```
10655 /*  
10656  * Macros restoring some IT-API 1.0 field names:  
10657  */  
10658  
10659 /* Define IT-API 1.0 name mappings for it_rc_only_attributes_t */  
10660 #define rdma_read_inflight_incoming rdma_read_ird  
10661 #define rdma_read_inflight_outgoing rdma_read_ord  
10662  
10663 /* Define IT-API 1.0 name mappings for it_ia_info_t */  
10664 #define ird_support ird_ord_ia_support  
10665 #define ord_support ird_ord_ia_support  
10666  
10667 /* Define IT-API 1.0 name mapping for it_make_rdma_addr. */  
10668 #define it_make_rdma_addr it_make_rdma_addr_absolute  
10669  
10670 /*  
10671  * Enums requiring some old IT-API 1.0 values:  
10672  */  
10673  
10674 /* Add IT_PRIV_NONE and IT_PRIV_DEFAULT into it_mem_priv_t  
10675    to support IT-API 1.0 applications.  
10676  */  
10677 typedef enum {  
10678     IT_PRIV_NONE           = 0x0000, /* for IT-API 1.0 compatibility */  
10679     IT_PRIV_LOCAL_READ    = 0x0001,  
10680     IT_PRIV_LOCAL_WRITE   = 0x0002,  
10681     IT_PRIV_LOCAL         = 0x0003,  
10682     IT_PRIV_DEFAULT       = 0x0003, /* now IT_PRIV_LOCAL */  
10683     IT_PRIV_REMOTE_READ   = 0x0004,  
10684     IT_PRIV_REMOTE_WRITE  = 0x0008,  
10685     IT_PRIV_REMOTE        = 0x000c,  
10686     IT_PRIV_ALL           = 0x000f  
10687 } it_mem_priv_t;  
10688  
10689 /* Add IT_RMR_PARAM_BOUND into it_rmr_param_mask_t to support  
10690    IT-API 1.0 applications.  
10691  */  
10692 typedef enum {  
10693     IT_RMR_PARAM_ALL       = 0x000001,  
10694     IT_RMR_PARAM_IA       = 0x000002,  
10695     IT_RMR_PARAM_PZ       = 0x000004,  
10696     IT_RMR_PARAM_LINKED   = 0x000008,  
10697     IT_RMR_PARAM_BOUND    = 0x000008, /* now IT_RMR_PARAM_LINKED */  
10698     IT_RMR_PARAM_LMR      = 0x000010,  
10699     IT_RMR_PARAM_ADDR     = 0x000020,  
10700     IT_RMR_PARAM_LENGTH   = 0x000040,  
10701     IT_RMR_PARAM_MEM_PRIV = 0x000080,  
10702     IT_RMR_PARAM_RMR_CONTEXT = 0x000100,
```

```

10703         IT_RMR_PARAM_TYPE           = 0x000200,
10704         IT_RMR_PARAM_ADDR_MODE      = 0x000400
10705     } it_rmr_param_mask_t;
10706
10707     #ifndef ITAPI_ENABLE_V20_BINDINGS
10708     /*
10709         Backwards compatibility mode:
10710         For functions whose signature changed from v1.0 to v2.0,
10711         <it_api.h> converts v1.0 function names to explicit v1.0 function
10712         names. Functions such as it_lmr_create continue to use v1.0
10713         signatures.
10714     */
10715     #define it_lmr_create   it_lmr_create10
10716     #define it_rmr_create   it_rmr_create10
10717     #define it_rmr_bind     it_rmr_bind10
10718
10719     #else
10720
10721     /*
10722         Full v2.0 functionality:
10723         For functions whose signature changed from v1.0 to v2.0,
10724         <it_api.h> converts v1.0 function names to explicit v2.0 function
10725         names. Functions such as it_lmr_create use the new v2.0
10726         signatures.
10727     */
10728     #define it_lmr_create   it_lmr_create20
10729     #define it_rmr_create   it_rmr_create20
10730     #define it_rmr_bind     it_rmr_link
10731
10732     #endif
10733
10734     /* Sample prototypes for a modified function: */
10735
10736     /*
10737         it_lmr_create10 is provided for backwards-compatibility
10738         and may be dropped in a future IT-API version.
10739
10740         Calls with a suffix "10" can be typically implemented through
10741         direct inlining to the corresponding calls with suffix "20",
10742         providing default arguments as necessary. */
10743     it_status_t it_lmr_create10(
10744         IN      it_pz_handle_t    pz_handle,
10745         IN      void              *addr,
10746         IN      it_length_t       length,
10747         IN      it_mem_priv_t     privs,
10748         IN      it_lmr_flag_t     flags,
10749         IN      uint32_t          shared_id,
10750         OUT     it_lmr_handle_t   *lmr_handle,
10751         IN OUT  it_rmr_context_t  *rmr_context
10752     );
10753
10754     /*
10755         it_lmr_create20 provides the v2.0 functionality and may
10756         be renamed to it_lmr_create in a future IT-API version.

```

```

10757 */
10758 it_status_t it_lmr_create20(
10759     IN     it_pz_handle_t    pz_handle,
10760     IN     void              *addr,
10761     IN     it_length_t      length,
10762     IN     it_addr_mode_t   addr_mode,
10763     IN     it_mem_priv_t    privs,
10764     IN     it_lmr_flag_t    flags,
10765     IN     uint32_t         shared_id,
10766     OUT    it_lmr_handle_t   *lmr_handle,
10767     IN OUT it_rmr_context_t  *rmr_context
10768 );
10769
10770 /*
10771     it_rmr_create10 is provided for backwards-compatibility
10772     and may be dropped in a future IT-API version.
10773
10774     Calls with a suffix "10" can be typically implemented through
10775     direct inlining to the corresponding calls with suffix "20",
10776     providing default arguments as necessary.
10777 */
10778
10779 it_status_t it_rmr_create10(
10780     IN     it_pz_handle_t    pz_handle,
10781     OUT    it_rmr_handle_t   *rmr_handle
10782 );
10783
10784 /*
10785     it_rmr_create20 provides the v2.0 functionality and may
10786     be renamed to it_rmr_create in a future IT-API version.
10787 */
10788
10789 it_status_t it_rmr_create20(
10790     IN     it_pz_handle_t    pz_handle,
10791     IN     it_rmr_type_t     rmr_type,
10792     OUT    it_rmr_handle_t   *rmr_handle
10793 );

```

10793 **D Functional Changes (IT-API Version 2.0 Relative to IT-** 10794 **API Version 1.0)**

10795 This Appendix contains a list of all *functional changes* for every reference page in IT-API
10796 Version 1.0 and for every new reference page in IT-API Version 2.0, including:

- 10797 • Changes in call signatures
- 10798 • New functionality in previously existing calls
- 10799 • New calls
- 10800 • Changes in data structures
- 10801 • New data structures
- 10802 • Changes and clarifications in the programming model such as:
 - 10803 — Additional or modified restrictions and rules
 - 10804 — Clarified restrictions and rules that were previously ambiguous
- 10805 • Errata from Version 1.0 that have been resolved

10806 **it_addr_mode_t**

10807 This is a new data type added for IT-API Version 2.0.

10808 **it_address_handle_create**

10809 Flags now identified as bitwise rather than logical, resolving Erratum GN63=IBM1.

10810 **it_address_handle_free**

10811 Removed bad advice regarding free handles, resolving Erratum GN50=HP10.

10812 **it_address_handle_modify**

10813 Flags now identified as bitwise rather than logically ORed value, resolving Erratum
10814 GN63=IBM1.

10815 Replaced use of IT_ADDR_PATH with IT_ADDR_PARAM_PATH, resolving Erratum
10816 GN32=OG12.

10817 Replaced use of IT_ERR_INVALID_SLID and IT_ERR_INVALID_SGID with
10818 IT_ERR_INVALID_SOURCE_PATH, resolving Erratum GN13=SUN1. Also see *it_status_t*
10819 change listing for related change.

10820 **it_address_handle_query**

10821 Flags now identified as bitwise rather than logically ORed value, resolving Erratum
10822 GN63=IBM1.

10823 Added additional cross-reference text about relevant fields in *it_path_t*, resolving Erratum
10824 GN14=SUN2.

10825 Resolved GN21=SUN9 as defined in Global Behavior section *Output Parameters*.

10826 **it_affiliated_event_t**

10827 Added Affiliated Asynchronous Events for a finer discrimination of locally detected errors.

10828 Added Affiliated Asynchronous Events for a finer discrimination of remotely detected errors,
10829 supporting interpretation of iWARP Terminate messages.

10830 Added Affiliated Asynchronous Events supporting the use of S-RQs.

10831 Provided mappings between iWARP Affiliated Asynchronous Events and IT-API Affiliated
10832 Asynchronous Events.

10833 **it_cm_msg_events**

10834 Added new reject reason code for iWARP (IT_CN_REJ_BAD_CONN_PARMS).

10835 Added text describing manifestation of capability to disable IRD/ORD.

10836 Replaced broken cross-references, resolving Erratum GN60=HP17.

10837 Added text clarifying that Events apply both to the TII (two-way and three-way) as well as the
10838 TDI.

10839 Revised text describing non-peer reject Events to reflect that they can apply both to remotely
10840 detected and locally detected events, and also that a possible cause of a non-peer reject Event is a
10841 non-interoperable RNIC combination.

10842 Removed text specifying that IRD/ORD values in an accept arrival Event are valid only on the
10843 active side in three-way Connection establishment – they are now valid also on the passive side.

10844 Added text specifying that IRD/ORD values are valid in the connection established Event and
10845 may differ from the actual IRD/ORD in use on the connection under certain circumstances.

10846 Added table specifying the mapping of reject reason codes on the iWARP Transport.

10847 Added text defining the lack of ordering relationship between
10848 IT_CM_MSG_EVENT_STREAM Events and IT_DTO_EVENT_STREAM Events, resolving
10849 Erratum GN62=HP19.

10850 **it_cm_req_events**

10851 Fixed typo, resolving Erratum GN38=OG18.

10852 **it_convert_net_addr**

10853 Added text to the IT_ERR_INVALID_CONVERSION error description stating this can also be
10854 returned when the Implementation cannot convert the particular source address that was input,
10855 resolving Erratum GN22=SUN10.

10856 Clarified the text of IT_ERR_INVALID_NETADDR to mean that the type of the Network
10857 Address specified in *source_addr* was not recognized (as opposed to
10858 IT_ERR_INVALID_ADDRESS which means the address was a valid type but an invalid value),
10859 resolving Erratum GN23=SUN11.

10860 Text added to reflect that this call may block.

10861 **it_conn_qual_t**

10862 Removed assigned enumerated values to *it_conn_qual_type_t*. IT-API Version 1.0 specified
10863 these as bit values but they are all mutually-exclusive.

10864 Added a new Connection Qualifier type for iWARP that allows specification of local and remote
10865 IANA ports.

10866 **it_dto_events**

10867 Modified the definition for the *ep* member of *it_dto_cmpl_event_t* so that for Receive
10868 completions it refers to the Endpoint on which the Receive completed, rather than to the
10869 Endpoint on which the Receive was posted. The only time this is different from the IT-API
10870 Version 1.0 behavior is when the Receive operation is posted to an S-RQ rather than to an
10871 Endpoint.

10872 Typos fixed resolving Erratum GN37=OG17 and GN40=OG20.

10873 Added text defining the lack of ordering relationship between
10874 IT_CM_MSG_EVENT_STREAM Events and IT_DTO_EVENT_STREAM Events, resolving
10875 Erratum GN62=HP19.

10876 **it_dto_flags_t**

10877 Moved a sentence in the IT_COMPLETION_FLAG description to instead be in the
10878 IT_NOTIFY_FLAG description. This was not a functional change; it was done only to enhance
10879 readability. This resolves Erratum GN20=SUN8.

10880	Fixed typo. IT_SOLICITED_WAIT_FLAG was changed to IT_SOLICITED_WAIT_FLAG.
10881	This resolves Erratum GN36=OG16.
10882	it_dto_status_t
10883	Added a new DTO status for memory management operation errors.
10884	Added additional causes for most of the existing DTO status values. A DTO status other than
10885	IT.DTO_SUCCSS represents a category of Completion Errors.
10886	Provided mappings between iWARP Completion Errors and IT-API DTO status values.
10887	it_ep_accept
10888	Added text describing use in two-way and three-way Connection Establishment, resolving
10889	Erratum GN6 = HP5.
10890	Removed text erroneously indicating that Private Data delivery is always unreliable, clarifying a
10891	corner case.
10892	Asynchronous Errors:
10893	<ul style="list-style-type: none"> • Specified how lack of interoperability between RDMAC RNIC and Non-permissive IETF
10894	RNIC is manifested.
10895	it_ep_attributes_t
10896	Added the following RC attributes for Shared Receive Queue (S-RQ) support:
10897	<ul style="list-style-type: none"> • <i>srq</i>
10898	<ul style="list-style-type: none"> • <i>soft_hi_watermark</i>
10899	<ul style="list-style-type: none"> • <i>hard_hi_watermark</i>
10900	Modified description of <i>max_recv_dtos</i> and <i>max_recv_segments</i> attributes for S-RQ support.
10901	Modified description of <i>rdma_read_ord</i> to reflect new capability of IT-API Version 2.0 to allow
10902	Consumer to change ORD after Endpoint is connected.
10903	it_ep_connect
10904	Replaced erroneous description implying that Private Data is delivered in an
10905	IT_CM_MSG_CONN_PEER_REJECT_EVENT and corrected stating that it appears in the
10906	IT_CM_REQ_CONN_REQUEST_EVENT, resolving Erratum GN7=HP6.
10907	Removed text erroneously indicating that Private Data delivery is unreliable, resolving Erratum
10908	GN8=HP7.
10909	Added text explaining that the connected event occurs both on Active and Passive sides,
10910	resolving Erratum GN9=HP8.

- 10911 Added dummy entries to attributes structures to aid in ANSI compilation.
- 10912 Added IT_CONNECT_SUPPRESS_IRD_ORD flags to *it_cn_est_flags_t*. This flag allows the
10913 Consumer to suppress use of IRD/ORD negotiation by the Implementation.
- 10914 Flags now identified as bitwise rather than logically ORed value, resolving Erratum
10915 GN63=IBM1.
- 10916 Clarified the mutual exclusivity of two-way *versus* three-way connection flags.
- 10917 Asynchronous Errors:
- 10918 • Specified how lack of interoperability between RDMAC RNIC and Non-permissive IETF
10919 RNIC is manifested.
- 10920 **it_ep_disconnect**
- 10921 Added text clarifying the relationship between Private Data delivery and generation of
10922 disconnect events.
- 10923 Fixed bad grammar, resolving Erratum GN61=HP18.
- 10924 **it_ep_free**
- 10925 Clarified that when *it_ep_free* returns, the handle can no longer be used; the previous statement
10926 was that an error was guaranteed to be returned if an attempt was made to use a handle that had
10927 been freed, which is not necessarily true. This resolves Erratum GN52=HP12.
- 10928 Reorganized and expanded the paragraph in the Application Usage section that talks about
10929 marker operations to make it clearer how they are used. This resolves Erratum GN54=HP14.
- 10930 Clarified that the Events that can be lost from the EVDs associated with the Endpoint are only
10931 those Events that pertain to the Endpoint. This resolves Erratum GN55=HP15.
- 10932 **it_ep_modify**
- 10933 Added the Endpoint Hard High Watermark and the Endpoint Soft High Watermark to the set of
10934 Endpoint parameters that can be modified. These new parameters are only used if the Endpoint
10935 is associated with a Shared Receive Queue.
- 10936 Made it illegal to use this routine to modify the total number of entries in the Receive Queue if
10937 the Endpoint is associated with a Shared Receive Queue.
- 10938 **it_ep_query**
- 10939 Reworded description for the *spigot_id* field to help clarify that this field is not valid if the
10940 Endpoint is in the IT_EP_STATE_UNCONNECTED state. This resolves Erratum GN25=OG5.

10941 Reworded description for the *dst_path* field to help clarify that this field is not valid if the
10942 Endpoint is in the IT_EP_STATE_UNCONNECTED state, and that it is not valid if this
10943 Endpoint is of the UD Service Type. This resolves Erratum GN26=OG6.

10944 Resolved GN21=SUN9 as defined in Global Behavior section *Output Parameters*.

10945 **it_ep_rc_create**

10946 Added the IT_EP_SRQ bit to the set of flags passed to this routine to allow the Consumer to
10947 specify that they want to associate a Shared Receive Queue with the Endpoint that they are
10948 creating.

10949 **it_ep_reset**

10950 No functional changes.

10951 **it_ep_state_t**

10952 Added the IT_EP_STATE_PASSIVE_WAIT_RDMA_TRANS_REQ state. This new state is
10953 used by the TDI (*it_socket_convert*).

10954 Removed text that erroneously implies Private Data delivery is always unreliable.

10955 **it_ep_ud_create**

10956 Fixed typo in description text. IT_EP_STATE_OPERATIONAL was changed to
10957 IT_EP_STATE_UD_OPERATIONAL. This resolves Erratum GN33=OG13.

10958 Fixed typo resolving Erratum GN40=OG20.

10959 **it_evd_create**

10960 Flags now identified as bitwise rather than logical, resolving Erratum GN63=IBM1.

10961 IT_SOLICITED_WAIT changed to IT_SOLICITED_WAIT_FLAG, resolving Erratum
10962 GN35=OG15.

10963 Invalid threshold value defined (must be strictly less than or equal to the actual Implementation-
10964 allocated queue size), resolving Erratum GN27=OG7.

10965 Revised text describing unblocking, resolving Erratum GN57=SUN16.

10966 Numerous grammatical fixes applied to text.

10967 **it_evd_dequeue**

10968 Typo fixed resolving Erratum GN28=OG8.

- 10969 **it_evd_free**
- 10970 Removed bad advice regarding freed handles, resolving Erratum GN53=HP13.
- 10971 **it_evd_modify**
- 10972 Clarified text regarding conditions under which the AEVD associated with an SEVD may be
10973 modified.
- 10974 Flags now identified as bitwise rather than logical, resolving Erratum GN63=IBM1.
- 10975 IT_EVD_CREAT_FD changed to IT_EVD_CREATE_FD, resolving Erratum GN34=OG14.
- 10976 **it_evd_post_se**
- 10977 No functional changes.
- 10978 **it_evd_wait**
- 10979 Corrected grammar, resolving Erratum GN29=OG9.
- 10980 Text added to reflect that this call may block.
- 10981 **it_event_t**
- 10982 The Event Stream for RMR Link completions is now called IT_RMR_LINK_CMPL_EVENT.
10983 The old name IT_RMR_BIND_CMPL_EVENT remains valid for source backwards-
10984 compatibility.
- 10985 Added Event Streams for the new Affiliated Asynchronous Events defined in
10986 *it_affiliated_event_t*.
- 10987 Removed (and marked deprecated) IT_ASYNC_UNAFF_IA_CATASTROPHIC_ERROR
10988 resolving Erratum GN4=HP4.
- 10989 **it_get_consumer_context**
- 10990 Added text documenting that an error is returned if the Consumer Context was never set,
10991 resolving Erratum GN15=SUN3.
- 10992 **it_get_pathinfo**
- 10993 Added IT_ERR_INTERRUPT as a return value, resolving Erratum GN1=HP1.
- 10994 Clarified the text of IT_ERR_INVALID_NETADDR to mean the type of *it_net_addr_t* passed
10995 in is not recognized.
- 10996 Added IT_ERR_INVALID_ADDRESS to mean the address was of a legal type but was invalid.
- 10997 Text added to reflect that this call may block.

10998	it_handle_t
10999	No functional changes.
11000	it_handoff
11001	Typos fixed resolving Erratum GN10=OG2 and GN11=OG3.
11002	Clarifying text added resolving Erratum GN56=HP16.
11003	it_ia_create
11004	Documented that the second release of the IT-API shall have a major version number of 2 and a minor version number of 0.
11005	
11006	Typo fixed resolving Erratum GN5=OG1.
11007	it_ia_free
11008	No functional changes.
11009	it_ia_info_t
11010	Added iWARP Transport type to <i>it_transport_type_t</i> .
11011	Added an iWARP vendor data structure definition.
11012	Added booleans indicating support or not for features as follows:
11013	• S-RQ
11014	• Hard high watermark (S-RQ)
11015	• Soft high watermark (S-RQ)
11016	• Resizable S-RQ
11017	• Extended iWARP state machine
11018	• Support for socket conversion (TDI)
11019	• IRD modifiable
11020	• ORD increasable
11021	• DTO Overflow detection
11022	• RDMA Read local extensions

- 11023 **it_listen_create**
- 11024 Flags now identified as bitwise rather than logically ORed value, resolving Erratum
11025 GN63=IBM1.
- 11026 Added functional capability to suppress use of IRD/ORD.
- 11027 **it_listen_free**
- 11028 Added text constraining use of *listen_handle* after call returns, resolving Erratum GN51 (HP11).
- 11029 **it_lmr_create**
- 11030 Added input modifier to select addressing mode (absolute addressing or relative addressing).
- 11031 *privs* and *flags* now identified as bitwise rather than logically ORed values, resolving Erratum
11032 GN63=IBM1.
- 11033 *it_mem_priv_t* is now specified on a separate reference page.
- 11034 Use of 0 for *privs* is no longer permitted.
- 11035 Clarified restrictions on and use of access privileges, resolving Erratum GN49=SUN15.
- 11036 Deprecated the use of IT_PRIV_DEFAULT (value remains in header file only), resolving
11037 Erratum GN3=HP3.
- 11038 Clarified the semantics of IT_LMR_FLAG_SHARED, resolving Erratum GN16=SUN4.
- 11039 Clarified potential issue with stale RMR Contexts, resolving Erratum GN46=SUN12.
- 11040 New return value IT_ERR_INVALID_ADDR_MODE.
- 11041 **it_lmr_free**
- 11042 Specified that any RMR Context associated with the LMR may no longer be used after freeing
11043 an LMR, resolving Erratum GN46=SUN12.
- 11044 Added missing statement that a failing call due to linked RMR(s) will neither affect an
11045 associated RMR Context, nor any linked RMR, resolving Erratum GN47=SUN13.
- 11046 While not a functional change, the discussion on stale RMR Contexts was moved to the
11047 Application Usage section, resolving Erratum GN18=SUN6.
- 11048 **it_lmr_modify**
- 11049 Added input modifier to select addressing mode (absolute addressing or relative addressing).
- 11050 *mask* now identified as bitwise rather than logically ORed value, resolving Erratum
11051 GN63=IBM1.

- 11052 Clarified potential issue with stale RMR Contexts, resolving Erratum GN46=SUN12.
- 11053 Clarified in which cases an LMR remains unaffected when the operation fails and that the
11054 operation fails if the LMR has any RMRs linked to it, resolving Erratum GN48=SUN14.
- 11055 **it_lmr_query**
- 11056 *mask* now identified as bitwise rather than logically ORed value, resolving Erratum
11057 GN63=IBM1.
- 11058 Added addressing mode (*addr_mode*) LMR attribute.
- 11059 Specified in which cases the LMR must have byte-level granularity. In particular,
11060 Implementations supporting Relative Addressing must provide byte-level granularity.
- 11061 Resolved GN21=SUN9 as defined in Global Behavior section *Output Parameters*.
- 11062 **it_lmr_sync_rdma_read**
- 11063 Added statement that when both normal and non-coherent LMRs are input to this routine that
11064 only the non-coherent segments are affected, resolving Erratum GN17=SUN5.
- 11065 **it_lmr_sync_rdma_write**
- 11066 Added statement that when both normal and non-coherent LMRs are input to this routine that
11067 only the non-coherent segments are affected, resolving Erratum GN17=SUN5.
- 11068 **it_lmr_triplet_t**
- 11069 The *addr* member of LMR Triplets must now be interpreted depending on the addressing mode
11070 (Absolute Addressing or Relative Addressing) of the LMR.
- 11071 Additionally, the *addr* member has been changed from a simple void pointer to a union of
11072 absolute (void pointer) and relative (*it_length_t*) elements.
- 11073 **it_make_rdma_addr**
- 11074 The IT-API 1.0 routine *it_make_rdma_addr* has been replaced with two new routines,
11075 *it_make_rdma_addr_absolute* and *it_make_rdma_addr_relative*. A backward-compatibility
11076 macro is recommended to define *it_make_rdma_addr* as *it_make_rdma_addr_absolute*.
- 11077 **it_mem_priv_t**
- 11078 This is a new reference page for a data type that was specified under *it_lmr_create* in IT-API
11079 Version 1.0.
- 11080 *privs* now identified as bitwise rather than logically ORed value, resolving Erratum
11081 GN63=IBM1.

- 11082 Use of 0 for *privs* is no longer permitted.
- 11083 Clarified restrictions on and use of access privileges, resolving Erratum GN49=SUN15.
- 11084 Deprecated the use of IT_PRIV_DEFAULT (value remains in header file only), resolving
11085 Erratum GN3=HP3.
- 11086 **it_path_t**
- 11087 Removed erroneous assertion that *it_ib_net_endpoint* is the remote component of an InfiniBand
11088 path.
- 11089 Added Path data structures for the iWARP Transport.
- 11090 **it_post_rdma_read**
- 11091 The starting address of the remote source buffer, *rdma_addr*, must be chosen according to the
11092 addressing mode of the remote buffer.
- 11093 Specified required access privileges for remote source buffer.
- 11094 Specified required access privileges for local sink buffer, which are transport-dependent.
- 11095 Clarified that Consumer retains ownership of the array specified by *local_segments* and
11096 *num_segments*, resolving Erratum GN31=OG11.
- 11097 Added ordering rules for subsequent DTOs targeting overlapping sections of local sink buffers.
- 11098 Added ordering rule for Work Requests posted after RMR Link/Unlink, resolving Erratum
11099 GN58=SUN17.
- 11100 Asynchronous Errors:
- 11101 • Described scenarios causing an access violation, data loss, or data corruption at the
11102 remote/local Endpoint.
 - 11103 • Specified how remotely detected errors will manifest remotely and locally.
 - 11104 • Specified how locally detected errors will manifest.
- 11105 **it_post_rdma_write**
- 11106 The starting address of the remote sink buffer, *rdma_addr*, must be chosen according to the
11107 addressing mode of the remote buffer.
- 11108 Specified required access privileges for source and sink buffer.
- 11109 Clarified that the Consumer retains ownership of the array specified by *local_segments* and
11110 *num_segments*, resolving Erratum GN31=OG11.
- 11111 Clarified the lack of end-to-end completions.

- 11112 Clarified the difference between operation ordering and local/remote completion ordering.
- 11113 Added ordering rules for subsequent DTOs targeting overlapping sections of remote sink
11114 buffers.
- 11115 Added ordering rule for Work Requests posted after RMR Link/Unlink, resolving Erratum
11116 GN58=SUN17.
- 11117 Asynchronous Errors:
- 11118 • Described scenarios causing an access violation, data loss, or data corruption at the remote
11119 Endpoint.
 - 11120 • Change in how remotely detected errors will manifest remotely and locally (transport-
11121 dependent).
- 11122 **it_post_rcv**
- 11123 Receive DTOs can now be posted to an Endpoint or Shared Receive Queue.
- 11124 Specified required access privileges for the sink buffer.
- 11125 Clarified that the Consumer retains ownership of the array specified by *local_segments* and
11126 *num_segments*, resolving Erratum GN31=OG11.
- 11127 Added ordering rules for subsequent incoming Send DTOs matching with Receive DTOs whose
11128 Receive buffers overlap.
- 11129 Asynchronous Errors:
- 11130 • Described scenarios causing an access violation, length error, data loss, or data corruption
11131 at the local Endpoint.
 - 11132 • Change in how locally detected errors will manifest locally and remotely (transport-
11133 dependent).
- 11134 Clarified that a Receive DTO can be posted to an EP prior to reaching the
11135 IT_EP_STATE_CONNECTED state.
- 11136 **it_post_rcvfrom**
- 11137 Specified required access privileges for the sink buffer.
- 11138 Clarified that the Consumer retains ownership of the array specified by *local_segments* and
11139 *num_segments*, resolving Erratum GN31=OG11.
- 11140 Asynchronous Errors:
- 11141 • Described scenarios causing an access violation, length error, data loss, or data corruption
11142 at the local Endpoint.

- 11143 **it_post_send**
- 11144 Specified required access privileges for the source buffer.
- 11145 Clarified that the Consumer retains ownership of the array specified by *local_segments* and
11146 *num_segments*, resolving Erratum GN31=OG11.
- 11147 Clarified the lack of end-to-end completions.
- 11148 Clarified the difference between operation ordering and local/remote completion ordering.
- 11149 Added ordering rules for subsequent Send DTOs matching with Receive DTOs whose Receive
11150 buffers overlap.
- 11151 Added ordering rule for Work Requests posted after RMR Link/Unlink, resolving Erratum
11152 GN58=SUN17.
- 11153 Asynchronous Errors:
- 11154
 - Described scenarios causing an access violation, length error, data loss, or data corruption
11155 at the remote Endpoint.
 - Change in how remotely detected errors will manifest remotely and locally (transport-
11156 dependent).
11157
- 11158 **it_post_sendto**
- 11159 Specified required access privileges for the source buffer.
- 11160 Clarified that the Consumer retains ownership of the array specified by *local_segments* and
11161 *num_segments*, resolving Erratum GN31=OG11.
- 11162 Asynchronous Errors:
- 11163
 - Described scenarios causing an access violation or length error at the remote Endpoint.
- 11164 **it_pz_query**
- 11165 Flags now identified as bitwise rather than logical, resolving Erratum GN63=IBM1.
- 11166 Resolved GN21=SUN9 as defined in Global Behavior section *Output Parameters*.
- 11167 **it_reject**
- 11168 Removed text that erroneously implies Private Data delivery is always unreliable and replaced it
11169 with a clarification, resolving Erratum GN12=HP9.
- 11170 **it_rmr_bind**
- 11171 This routine was replaced by *it_rmr_link*. See also Appendix C.

- 11172 **it_rmr_create**
- 11173 Added input modifier to select RMR type (Narrow RMR or Wide RMR).
- 11174 New return value IT_ERR_INVALID_RMR_TYPE.
- 11175 **it_rmr_free**
- 11176 No functional changes.
- 11177 **it_rmr_link**
- 11178 This routine replaces *it_rmr_bind* of IT-API Version 1.0. See also Appendix C.
- 11179 Added input modifier to select addressing mode (absolute addressing or relative addressing).
- 11180 *privs* and *dto_flags* now identified as bitwise rather than logically ORed values, resolving
11181 Erratum GN63=IBM1.
- 11182 Specified additional restrictions for linking a Narrow RMR.
- 11183 Clarified that any local access privileges specified in *privs* are ignored.
- 11184 Added ordering rule for Work Requests posted after RMR Link, resolving Erratum
11185 GN58=SUN17.
- 11186 New return value IT_ERR_INVALID_ADDR_MODE.
- 11187 Detailed description of Asynchronous Errors.
- 11188 Deprecated the use of 0 for *privs* and recommended using *it_rmr_unlink* instead, resolving
11189 Erratum GN19=SUN7.
- 11190 Erratum GN2=HP2 (reference to a inexistant IT_DTO_DEFAULT_FLAG) is resolved – in fact
11191 it was already resolved in Version 1.0.
- 11192 Clarified ways to infer completion of an RMR Link operation, resolving Erratum
11193 GN59=SUN18.
- 11194 Clarified potential issue with stale RMR Contexts.
- 11195 **it_rmr_query**
- 11196 *mask* now identified as bitwise rather than logically ORed value, resolving Erratum
11197 GN63=IBM1.
- 11198 Added RMR type (*type*) RMR attribute.
- 11199 Added addressing mode (*addr_mode*) RMR attribute.
- 11200 Replaced the RMR attribute *bound* by *linked*. If ITAPI_ENABLE_V20_BINDINGS is not
11201 defined, the old attribute name remains valid. See also Appendix C.

- 11202 Resolved GN21=SUN9 as defined in Global Behavior section *Output Parameters*.
- 11203 **it_rmr_triplet_t**
- 11204 This is a new data type added for IT-API Version 2.0.
- 11205 **it_rmr_type_t**
- 11206 This is a new data type added for IT-API Version 2.0.
- 11207 **it_rmr_unbind**
- 11208 This routine was replaced by *it_rmr_unlink*. See also Appendix C.
- 11209 **it_rmr_unlink**
- 11210 This routine replaces *it_rmr_unbind* of IT-API Version 1.0. See also Appendix C.
- 11211 *dto_flags* now identified as bitwise rather than logically ORed value, resolving Erratum
11212 GN63=IBM1.
- 11213 Specified additional restrictions for unlinking a Narrow RMR.
- 11214 Added ordering rule for Work Requests posted after RMR Unlink, resolving Erratum
11215 GN58=SUN17.
- 11216 Detailed description of Asynchronous Errors.
- 11217 Improved guidance for dealing with a linked RMR after a disconnect.
- 11218 Clarified ways to infer completion of an RMR Unlink operation, resolving Erratum
11219 GN59=SUN18.
- 11220 Eliminated typo, resolving Erratum GN39=OG19.
- 11221 **it_socket_convert**
- 11222 This is a new routine added for IT-API Version 2.0.
- 11223 **it_srq_create**
- 11224 This is a new routine added for IT-API Version 2.0.
- 11225 **it_srq_free**
- 11226 This is a new routine added for IT-API Version 2.0.

- 11227 **it_srq_modify**
- 11228 This is a new routine added for IT-API Version 2.0.
- 11229 **it_srq_query**
- 11230 This is a new routine added for IT-API Version 2.0.
- 11231 **it_status_t**
- 11232 Removed IT_ERR_INVALID_SLID and IT_ERR_INVALID_SGID, resolving Erratum
11233 GN13=SUN1. Also see *it_address_handle_modify* Functional Change listing (above) for related
11234 change. Code using IT_ERR_INVALID_SLID or IT_ERR_INVALID_SGID will no longer
11235 compile and should be updated to use IT_ERR_INVALID_SOURCE_PATH.
- 11236 **it_ud_service_reply**
- 11237 Removed text erroneously indicating that Private Data delivery is unreliable.
- 11238 **it_ud_service_request_handle_create**
- 11239 Typo corrected, resolving Erratum GN42=OG22.
- 11240 Typo corrected, resolving Erratum GN41=OG21.
- 11241 **it_ud_service_request_handle_query**
- 11242 Flags now identified as bitwise rather than logically ORed value, resolving Erratum
11243 GN63=IBM1.
- 11244 Typo corrected, resolving Erratum GN24=OG4.
- 11245 Resolved GN21=SUN9 as defined in Global Behavior section *Output Parameters*.
- 11246 **it_unaffiliated_event_t**
- 11247 Typo corrected, resolving Erratum GN30=OG10.

E IT-API 1.0 Errata

Global No.	Company No.	Title	Filename	Problem Description	Suggested Severity
1	HP1	<i>it_get_pathinfo</i> : should return IT_ERR_INTERRUPT	<i>it_get_pathinfo</i>	<i>it_get_pathinfo</i> currently does not specify the return value of IT_ERR_INTERRUPT. Given that it is (may be) a blocking call, it should have this possible return value.	4
2	HP2	<i>it_rmr_bind</i> : IT_DTO_DEFAULT_FLAG needs to be removed	<i>it_rmr_bind</i>	Page 2 line 54 of <i>it_rmr_bind</i> refers to the IT_DTO_DEFAULT_FLAG. The flag no longer exists in the API so the text should be corrected (perhaps suggesting some appropriate flag settings).	3
3	HP3	<i>it_lmr_create</i> : confusing use of IT_PRIV_DEFAULT	<i>it_lmr_create</i>	On the <i>it_lmr_create</i> reference page, it states: "The special value IT_PRIV_DEFAULT or 0 may be used to grant default access, which includes local read and write access". The way this is phrased, it's not clear if it only includes local read and write access, or if it includes local read and write access amongst other things. Not knowing with certainty what the "default" is will lead to portability problems. This should be clarified. Also, IT_PRIV_DEFAULT does not have a value of 0 in the < <i>it_api.h</i> > header file. Either the header file should be fixed, or else we should fix the documentation on this reference page.	3
4	HP4	<i>it_event_t</i> : IT_ASYNC_UNAFF_IA_CATASTROPHIC_ERROR needs to be removed	<i>it_event_t</i>	In the <i>it_event_t</i> reference page in the 0.952 IT-API snapshot, Asynchronous Non-affiliated Event group there is an IT_ASYNC_UNAFF_IA_CATASTROPHIC_ERROR definition. There is no corresponding condition documented on the reference page for Unaffiliated Asynchronous Events, so it is probably left over from some previous revision of the spec and should be removed.	2

Global No.	Company No.	Title	Filename	Problem Description	Suggested Severity
5	OG1	<i>it_ia_create</i> : delete an extra "also"	<i>it_ia_create</i>	Lines 32-33 of the reference page say: "The major version number associated with the first release of the IT-API is 1, and the minor version number associated with the first release of the IT-API is also 0." The "also" at the end of the sentence does not make sense, since the major version number is different, and should be removed.	1
6	HP5	<i>it_ep_accept</i> : description of when this is called could lead to confusion	<i>it_ep_accept</i>	Lines 39-44 of the <i>it_ep_accept</i> reference page currently state: "Calling <i>it_ep_accept</i> is the last Local Consumer step in establishing an Endpoint-to-Endpoint Connection for a Three-way Connection Establishment. The Consumer is notified of an established Connection by an IT_CM_MSG_CONN_ESTABLISHED_EVENT Event being delivered on the connect EVD of the Endpoints. The event is generated on both the active and passive side of the connection establishment." Calling <i>it_ep_accept</i> is also the last step in establishing a connection of a two-way connection establishment, but that is not mentioned above. Calling out the three-way case while excluding the two-way case can lead to confusion; either both should be mentioned, or neither should be mentioned.	2
7	HP6	<i>it_ep_connect</i> : IT_CM_MSG_CONN_PEER_REJECT_EVENT should be IT_CM_REQ_CONN_REQUEST_EVENT	<i>it_ep_connect</i>	The description for the <i>private_data</i> field for <i>it_ep_connect</i> reads as follows: " <i>private_data</i> : Opaque Private Data to be sent in the IT_CM_MSG_CONN_PEER_REJECT_EVENT Event delivered to the Remote Consumer. If the IA does not support Private Data, <i>private_data_length</i> must be zero. The delivery of Private Data to the Remote Endpoint is unreliable." The talk about an IT_CM_MSG_CONN_PEER_REJECT_EVENT is wrong. The Event in question is the IT_CM_REQ_CONN_REQUEST_EVENT. This typo should be fixed in the spec.	1

Global No.	Company No.	Title	Filename	Problem Description	Suggested Severity
8	HP7	<i>it_ep_connect</i> : private data delivery IS reliable	<i>it_ep_connect</i>	The description for the <i>private_data</i> parameter to <i>it_ep_connect</i> states: “ <i>private_data</i> : Opaque Private Data to be sent in the IT_CM_MSG_CONN_PEER_REJECT_EVENT Event delivered to the Remote Consumer. If the IA does not support Private Data, <i>private_data_length</i> must be zero. The delivery of Private Data to the Remote Endpoint is unreliable.” The statement that “delivery of Private Data to the Remote Endpoint is unreliable” is not accurate. There are no known circumstances under which a connection request Event could be enqueued without its associated Private Data. The last sentence above should be removed.	5
9	HP8	<i>it_ep_connect</i> : connection established event is generated on both sides, not just the remote side	<i>it_ep_connect</i>	Lines 129-134 of the <i>it_ep_connect</i> reference page state: “For a two way connection establishment, an IT_CM_MSG_CONN_ESTABLISHED_EVENT Event is generated on the active side after the passive side Consumer accepts the connection and the Endpoint transitions into the (IT_EP_STATE_CONNECTED) connected state. See the < <i>it_ep_state.t.doc</i> > reference page for a complete description on the Endpoint state diagram for both the three-way and two-way connection establishment.” The event in question is generated on both the Active and the Passive sides. Mentioning only the Active side above makes it appear as if it doesn't get generated on the Passive side. The description above should be modified to state that the Event is generated for both the Active and Passive sides.	2
10	OG2	<i>it_handoff</i> : typo (“Cn_est_id”)	<i>it_handoff</i>	Typo: In the Description section, the argument <i>Cn_est_id</i> has the first letter in uppercase; it should be lowercase.	1
11	OG3	<i>it_handoff</i> : Typo (“conn_qual)was”)	<i>it_handoff</i>	Typo: In the RETURN VALUE section, the description of IT_ERR_INVALID_CONN_QUAL is missing a space after the closing parenthesis; i.e., “(conn_qual)was” should be “(conn_qual) was”.	1

Global No.	Company No.	Title	Filename	Problem Description	Suggested Severity
12	HP9	<i>it_reject</i> : private data delivery isn't as unreliable as this makes it sound	<i>it_reject</i>	There is a note in the Application Usage section of the <i>it_reject</i> reference page that states: "The Consumer should be aware that the delivery of Private Data to the Remote Endpoint is unreliable." This is an oversimplification, and can be read to mean that you can't be guaranteed that when you get a peer reject Event that the Private Data that was submitted via <i>it_reject</i> will be a part of it. That's not the case. The unreliability here is that just because you called <i>it_reject</i> doesn't mean the remote end will receive a peer reject Event; they might receive a non-peer reject Event (e.g., because the REJ message got lost on IB and the connection establishment attempt timed out). If you do get a peer reject Event, the Private Data contained within that Event will be exactly what was furnished to <i>it_reject</i> . This needs to be clarified.	3
13	SUN1	<i>it_address_handle_modify</i> : obsolete error codes need to be removed	<i>it_address_handle_modify</i>	Under RETURN VALUE we describe IT_ERR_INVALID_SGID and IT_ERR_INVALID_SLID. These errors were to be combined under IT_ERR_INVALID_SOURCE_PATH, as in <i>it_address_handle_create()</i> . These error codes should be removed from <i>it_ah_modify.pdf</i> , as well as <i>it_status_t.pdf</i> and from wherever else they appear.	2
14	SUN2	<i>it_address_handle_query</i> : what should be returned for an incomplete path	<i>it_address_handle_query</i>	How much of the path can the user expect from a query? Not all fields of the path will be valid for address handles. A specific issue is, if the path were created with IT_AH_PATH_COMPLETE not set, what fields can the user expect to be valid? The compliance test seems to require a path with all the fields listed in <i>it_address_handle_create.pdf</i> be returned for all calls; but a "lazy" implementation might only return <i>ib.sl</i> and <i>ib.remote_port_lid</i> for address handles that were created with IT_AH_PATH_COMPLETE free. Would this lazy implementation be non-compliant? Should we edit the text in <i>it_address_handle_query.pdf</i> to be more explicit?	3

Global No.	Company No.	Title	Filename	Problem Description	Suggested Severity
15	SUN3	<i>it_get_consumer_context</i> : murky description of what happens when no context has been set	<i>it_get_consumer_context</i>	<p>The testable assertion folks make a good point in the specification issues: The error code IT_ERR_NO_CONTEXT has been added to <i>it_get_consumer_context()</i> for the situation where “the handle does not have an associated context”.</p> <p>However, the function description still says: “If the Consumer context was never set ... then the value of the returned Consumer context is 0”. This is confusing, as it suggests two different indications for what is (presumably) the same situation, not to mention that it requires the value of an output parameter to be set when the return value indicates an error condition. Is there a difference between a handle with no context, and a handle with a context of 0? It seems that there is, and the proper thing to do is as follows:</p> <ol style="list-style-type: none"> 1. Handle has a context, context is non zero: return the context and IT_SUCCESS. 2. Handle has a context, context is zero: return the (zero) context and IT_SUCCESS. 3a. Handle has no context at all: return IT_ERR_NO_CONTEXT, set context to 0 (current method). 3b. Handle has no context at all: return IT_ERR_NO_CONTEXT, don't set context at all (suggested fix). <p>The suggested solution makes more sense; however, it's a subtle change to the spec. The spec is not necessarily broken; it's just a little over-eager in the error case.</p>	3
16	SUN4	<i>it_lmr_create</i> : confusing description of IT_LMR_FLAG_SHARED behavior	<i>it_lmr_create</i>	<p>The discussion of IT_LMR_FLAG_SHARED (Lines 68-80) should say a new LMR is created. Currently it says: “If set, then the Implementation will re-use resources from a matching LMR ... refers to the same physical memory pages ... and has the same coherency mode.” It does not explicitly say that a new LMR is created, that is logically distinct from the previous LMR. This caused some confusion in the testable assertions. Suggest rewording the sentence starting at Line 71 as follows: “If set, then the implementation will create a new LMR that re-uses resources from a matching LMR ...”</p>	3

Global No.	Company No.	Title	Filename	Problem Description	Suggested Severity
17	SUN5	<i>it_lmr_sync_rdma_*</i> : behavior when both coherent and incoherent LMRs are passed is unspecified	<i>it_lmr_sync_rdma_read</i> <i>it_lmr_sync_rdma_write</i>	Neither of these reference pages describe what happens if the user passes an array of buffer segments that includes both coherent and non-coherent memory regions. What should happen is the ranges of the non-coherent LMRs should be made ready for <i>rdma</i> reads, while the coherent LMRs should be silently ignored. The lack of clarity here caused some confusion in the testable assertions.	4
18	SUN6	<i>it_rmr_free</i> : description needs to move to Application Usage section	<i>it_rmr_free</i>	The testable assertion folks say: "The discussion on re-use on RMR context values (Lines 18-26) should be placed in the Application Usage section.	2
19	SUN7	<i>ir_rmr_bind</i> : privs of 0 should not be allowed	<i>it_rmr_bind</i>	Why is it that a bind with privs of 0 is allowed, but a bind of length 0 is not? It seems to me that in both of these cases, we should encourage users to use <i>rmr_unbind()</i> instead.	5
20	SUN8	<i>it_dto_flags_t</i> : sentence should be moved	<i>it_dto_flags_t</i>	In the IT_COMPLETION_FLAG section there is a sentence that should more properly be in the IT_NOTIFY_FLAG section: "If there is an error, the completion event will be generated with notification regardless of IT_NOTIFY_FLAG value." (Line 37-38)	2
21	SUN9	<i>it_*_query</i> : behavior not defined if params is NULL	<i>it_lmr_query</i> other query routines	The behaviour of <i>it_lmr_query()</i> if params is NULL is not defined." We either need a new error code or a line saying that if you call this function with a NULL pointer you will blow up.	3

Global No.	Company No.	Title	Filename	Problem Description	Suggested Severity
22	SUN10	<i>it_convert_net_addr</i> : need new error code	<i>it_convert_net_addr</i>	W may want to add an additional error code for this function. Suppose you're an IT-API consumer and you're attached to an IB fabric. And, let's say that some of the hosts on this fabric have IP addresses and some do not. If you call <i>it_convert_net_addr()</i> from GID to IP on some remote address, it seems that sometimes it should succeed just fine, but in some cases there would be no way to go from a GID to an IP. What is the error code that would be returned in that case? It seems right now the only thing the implementation could do is return <code>IT_ERR_INVALID_CONVERSION</code> . That error implies the conversion "was not supported by the Implementation" which is not true in this case. The Implementation supports this kind of conversion, but for this particular request, there is no address to convert to. Do you think it is worth adding another error code, say, <code>IT_ERR_NO_CONVERSION</code> , to cover this case?	3
23	SUN11	<i>it_convert_net_addr</i> : want to rename an error code	<i>it_convert_net_addr</i>	There are two errors here that have very similar names. <code>IT_ERR_INVALID_ADDRESS</code> and <code>IT_ERR_INVALID_NETADDR</code> . Is it possible to rename the latter to a more specific error code such as <code>IT_ERR_INVALID_ADDR_TYPE</code> ? (Perhaps it is too late to do this.)	2
24	OG4	<i>it_ud_service_request_handle_create</i> : typo	<i>it_ud_service_request_handle_create</i>	On Line 41 in the <i>it_ud_service_request_handle_query</i> reference page, the comment against the <i>private_data_length</i> structure member is <code>IT_UD_PARAM_PRIV_DATA_LEN</code> , whereas the actual constant name is <code>IT_UD_PARAM_PRIV_DATA_LENGTH</code> .	1
25	OG5	<i>it_ep_query</i> : what spigot do you get back in unconnected state?	<i>it_ep_query</i>	What value does <i>it_ep_query(ep_handle, mask, params)</i> return when <code>IT_EP_PARAM_SPIGOT</code> is set in <i>mask</i> and <i>ep_handle</i> specifies an RC endpoint in the <code>IT_EP_STATE_UNCONNECTED</code> state?	2
26	OG6	<i>it_ep_query</i> : what path do you get back in unconnected state?	<i>it_ep_query</i>	What value does <i>it_ep_query(ep_handle, mask, params)</i> return when <code>IT_EP_PARAM_PATH</code> is set in <i>mask</i> and <i>ep_handle</i> specifies an RC endpoint in the <code>IT_EP_STATE_UNCONNECTED</code> state, or specifies a UD endpoint?	2

Global No.	Company No.	Title	Filename	Problem Description	Suggested Severity
27	OG7	<i>it_evd_create</i> : what value of <i>sevd_threshold</i> is invalid?	<i>it_evd_create</i>	The specification does not say which values of <i>sevd_threshold</i> are invalid and would therefore cause an IT_ERR_INVALID_THRESHOLD error.	2
28	OG8	<i>it_evd_dequeue</i> : typo	<i>it_evd_dequeue</i>	The text in the 5th paragraph: "...returns a notification event..." should read: "...returns a notification event...".	1
29	OG9	<i>it_evd_wait</i> : typo	<i>it_evd_wait</i>	The sentence: "If <i>sevd_threshold</i> value of <i>evd_handle</i> is 1, then one or more simultaneous waiters can supported for the SEVD." should read: "If <i>sevd_threshold</i> value of <i>evd_handle</i> is 1, then one or more simultaneous waiters can be supported for the SEVD."	1
30	OG10	<i>it_events</i> : typo	<i>it_events</i>	In the <i>it_unaffiliated_event_t</i> reference page, in the table at the end of the 'DESCRIPTION' section, it refers to the fields <i>spigot_online_event_support</i> and <i>spigot_offline_event_support</i> . These should be <i>spigot_online_support</i> and <i>spigot_offline_support</i> respectively. See the <i>it_ia_info_t</i> reference page.	1
31	OG11	<i>it_post_*</i> : segment ownership language is confusing	<i>it_post_rdma_read</i> <i>it_post_rdma_write</i> <i>it_post_send</i> <i>it_post_sendto</i> <i>it_post_rcv</i> <i>it_post_rcvfrom</i>	The Description says: "A Consumer does get back the ownership of the <i>num_segments</i> and <i>local_segments</i> arguments (but not the local buffer identified by them) when <i>it_post_rdma_read</i> returns and is free to use the <i>num_segments</i> and <i>local_segments</i> arguments for other calls, or to modify them, or to destroy them." Since the C language passes all arguments by value, the Consumer always has ownership, in the sense that the specification describes, of the <i>num_segments</i> and <i>local_segments</i> arguments. Therefore it is not meaningful to say that it gets back the ownership of these arguments.	2
32	OG12	<i>it_address_handle_modify</i> : typo	<i>it_address_handle_modify</i>	"IT_ADDR_PATH" should be "IT_ADDR_PARAM_PATH".	1
33	OG13	<i>it_ep_ud_create</i> : typo	<i>it_ep_ud_create</i>	"IT_EP_STATE_OPERATIONAL" should be IT_EP_STATE_UD_OPERATIONAL".	1
34	OG14	<i>it_evd_modify</i> : typo	<i>it_evd_modify</i>	"IT_EVD_CREAT_FD" should be "IT_EVD_CREATE_FD".	1
35	OG15	<i>it_evd_create</i> : typo	<i>it_evd_create</i>	"IT_SOLICITED_WAIT" should be "IT_SOLICITED_WAIT_FLAG".	1

Global No.	Company No.	Title	Filename	Problem Description	Suggested Severity
36	OG16	<i>it_dto_flags_t</i> : typo	<i>it_dto_flags_t</i>	“IT_SOLICITED_WAIT_FLAG” should be “IT_SOLICITED_WAIT_FLAG”.	1
37	OG17	<i>it_dto_events</i> : typo	<i>it_dto_events</i>	“IB_UD_IB_GRH_PRESENT” should be “IT_UD_IB_GRH_PRESENT”.	1
38	OG18	<i>it_cm_req_events</i> : typo	<i>it_cm_req_events</i>	“ <i>it_conn_req_event_f</i> ” should be “ <i>it_conn_request_event_f</i> ”.	1
39	OG19	<i>it_rmr_unbind</i> : typo	<i>it_rmr_unbind</i>	“ <i>it_dto_compl_event_f</i> ” should be “ <i>it_dto_cmpl_event_f</i> ”.	1
40	OG20	<i>it_dto_events</i> : typo	<i>it_dto_events</i> <i>it_ep_ud_create</i>	“ <i>it_ep_states_f</i> ” should be “ <i>it_ep_state_f</i> ”.	1
41	OG21	<i>it_ud_service_request_handle_create</i> : typo	<i>it_ud_service_request_handle_create</i>	“ <i>it_ud_service_request_handle_f</i> ” should be “ <i>it_ud_svc_req_handle_f</i> ”.	1
42	OG22	<i>it_ud_service_request</i> : typo	<i>it_ud_service_request</i> <i>it_ud_service_request_handle_create</i>	“ <i>it_ud_svc_req_identifer_f</i> ” should be “ <i>it_ud_svc_req_identifier_f</i> ”.	1
43	OG23	Can functions be implemented as macros?	<i>itapdx_implementors_guide</i>	There is no mention in the IT-API specification as to whether functions are required/allowed to be implemented as macros as well as or instead of functions.	3
44	OG24	< <i>itapdx_it_api.h</i> > typo	< <i>itapdx_it_api.h</i> >	In the file < <i>itapdx_it_api.h</i> >, the comment against the <i>private_data_length</i> member of <i>it_ud_svc_req_param_t</i> mentions “IT_UD_PARAM_PRIV_DATA_LEN”; it should say “IT_UD_PARAM_PRIV_DATA_LENGTH”.	1
45	OG25	< <i>itapdx_it_api.h</i> > missing prototypes	< <i>itapdx_it_api.h</i> >	The file < <i>itapdx_it_api.h</i> > is missing prototypes for the functions <i>it_make_rdma_addr()</i> and <i>it_hton64()</i> .	5
46	SUN12	<i>it_lmr_free</i> : what happens to the RMR context?	<i>it_lmr_free</i>	Nowhere in <i>it_lmr_free.pdf</i> do we describe what happens to the associated RMR Context (if one exists) when an <i>lmr_handle</i> is freed. It is an error to use it afterwards, much like it's an error to use the <i>rmr</i> context of an unbound MW. We should say something to this effect in <i>it_lmr_free.pdf</i> .	3

Global No.	Company No.	Title	Filename	Problem Description	Suggested Severity
47	SUN13	<i>it_lmr_free</i> : RMR context isn't affected by failure of this routine	<i>it_lmr_free</i>	In Lines 15-16 we say: "A Local Memory Region may not be destroyed if it has an RMR bound to it; an attempt to do so will fail and the LMR will not be affected." We do not say what happens to the RMR in this case. The RMRs are not affected by this failure either. We should say: "A Local Memory Region may not be destroyed if it has an RMR bound to it; an attempt to do so will fail and neither the LMR, RMR, or RMR Context(s) will be affected."	3
48	SUN14	<i>it_lmr_modify</i> : confusing text	<i>it_lmr_modify</i>	<p>If you call <i>it_lmr_modify</i>() and an error occurs:</p> <ul style="list-style-type: none"> • IT_ERR_RESOURCES: your LMR blows up (33-36). • IT_ERR_ACCESS: your LMR also blows up (33-36). • IT_ERR_LMR_BUSY: your LMR is guaranteed fine (26-28). • IT_ERR_INVALID_{PZ,MASK,PRIVS}: who knows??? <p>This is confusing. Furthermore, if you get a BUSY error, your LMR is guaranteed OK but we don't say anything about the bound RMRs (26-28). Do they survive? Right now the spec is silent.</p> <p>We should patch these gaps as follows (according to the IB Spec 1.1 (Page 536):</p> <p>C11-19 if the CI returns either the Invalid HCA handle or Invalid Memory Region handle error, the CI shall make no change to the current registration (assuming that it even exists).</p> <p>C11-20 if the CI returns any other error, the CI shall invalidate both "old" and "new" registrations, and release any associated resources.</p> <p>IT_ERR_INVALID_PRIVS corresponds to the IB "Invalid Access Control specifier" error. This is covered by C11-20.</p> <p>IT_ERR_INVALID_PZ corresponds to the "Invalid Protection Domain" error. That is definitely a C11-20 error.</p> <p>As for IT_ERR_INVALID_MASK, that is not something that maps over to IB. This is an error the implementation can easily detect before doing much of anything. It should be a non-fatal error in the same way as IT_ERR_LMR_BUSY.</p>	3

Global No.	Company No.	Title	Filename	Problem Description	Suggested Severity
49	SUN15	<i>it_lmr_create</i> : missing flags	<i>it_lmr_create</i>	“An RMR Context allowing remote access to the memory region will be created if the <i>privs</i> argument includes either IT_PRIV_REMOTE_READ or IT_PRIV_REMOTE_WRITE.” (96-97) Strictly speaking this list should include IT_PRIV_REMOTE and IT_PRIV_ALL.	2
50	HP10	<i>it_address_handle_free</i> : bad advice regarding freed handles	<i>it_address_handle_free</i>	<i>it_address_handle_free</i> says two conflicting things about what happens to a Consumer who uses a freed handle. The current spec text is: “...Once <i>it_address_handle_free</i> returns, <i>addr_handle</i> can no longer be used in DTO operations. If an Address Handle is freed while there is still a Send DTO outstanding that references the Address Handle, whether or not that Send completes successfully is Implementation-dependent.” It should be modified to state that once the handle has been freed, the handle may no longer be used.	3
51	HP11	<i>it_listen_free</i> : nothing said about freed handles	<i>it_listen_free</i>	All of our other IT-Object destructor routines make a statement (vague though it may be) about the use of freed handles. This routine says nothing, which is a superfluous inconsistency that could potentially lead the Consumers to believe that the disposition of a handle freed by this routine is different than the disposition of a handle freed by any of the other IT-Object destructor routines. This routine should carry the standard boilerplate regarding the use of a freed handle, namely that once the handle has been freed, the handle may no longer be used.	2
52	HP12	<i>it_ep_free</i> : bad advice regarding freed handles	<i>it_ep_free</i>	<i>it_ep_free</i> has an incorrect statement about what happens if the Consumer attempts to use a freed handle. Currently the spec states: “Use of the handle <i>ep_handle</i> of the destroyed Endpoint in any subsequent operation fails.” That’s not necessarily true. It should be modified to state that once the handle has been freed, the handle may no longer be used.	3

Global No.	Company No.	Title	Filename	Problem Description	Suggested Severity
53	HP13	<i>it_evd_free</i> : bad advice regarding freed handles	<i>it_evd_free</i>	<i>it_evd_free</i> has an incorrect statement about what happens if the Consumer attempts to use a freed handle. Currently the spec states: "Use of the handle <i>evd_handle</i> in any subsequent operation fails." That's not necessarily true. It should be modified to state that once the handle has been freed, the handle may no longer be used.	3
54	HP14	<i>it_ep_free</i> : disconnect sequence is too terse	<i>it_ep_free</i>	<i>it_ep_free.pdf</i> recommends doing <i>it_ep_disconnect</i> followed by posting/waiting for the marker DTO. It should instead recommend the following sequence: <ol style="list-style-type: none"> 1. First, <i>it_ep_disconnect</i> 2. Wait for DISCONNECT event or CONN_BROKEN or REJECT event 3. Post a marker <i>send/rmr_bind</i> DTO if there have been <i>send/rmr_bind</i> DTOs with COMPLETION_FLAG off 4. Wait for the last posted <i>recv</i> DTO and wait for the last posted <i>send/rmr_bind</i> DTO or marker DTO 	2
55	HP15	<i>it_ep_free</i> : shouldn't allow arbitrary events to be lost	<i>it_ep_free</i>	Lines 21-24 of the <i>it_ep_free</i> reference page currently state: "Freeing an Endpoint potentially means Events might be lost on the <i>recv_sevd_handle</i> or <i>request_sevd_handle</i> SEVDs associated with the Endpoint. There is also potential to lose Events on the <i>connect_sevd_handle</i> SEVD associated with the Endpoint. The Consumer should first drain these EVDs before calling <i>it_ep_free</i> ." The above doesn't specify which Events might be lost. In particular, it doesn't rule out the possibility that arbitrary Events enqueued in the EVD that are completely unrelated to the Endpoint that is being freed might be lost. We didn't intend to give the Implementation that much leeway in punishing the Consumer for their bad behavior; we intended to only allow Events associated with the EP that is being freed to be lost. This should be clarified.	2

Global No.	Company No.	Title	Filename	Problem Description	Suggested Severity
56	HP16	<i>it_handoff</i> : add clarifying sentence	<i>it_handoff</i>	In the <i>it_handoff</i> reference page Description section, add a line after the sentence: “ <i>it_handoff</i> forwards a Connection Request to the specified Spigot and Connection Qualifier of the IA on which the Connection Request originally arrived.” as follows: “Specifying a <i>conn_qual</i> or <i>spigot_id</i> on any IA other than that on which the Connection Request originally arrived will yield IT_ERR_INVALID_CONN_QUAL or IT_ERR_INVALID_SPIGOT errors respectively.”	2
57	SUN16	<i>it_evd_create</i> : unblocking behavior language unclear	<i>it_evd_create</i>	<p>Re <i>it_evd_create</i>(Page 77), what do we mean to say here? “If arriving event causes SEVD to reach notification criteria then SEVD waiter will be unblocked if one exists and the SEVD is disabled and not associated with AEVD with IT_EVD_DEQUEUE_NOTIFICATIONS <i>evd_flag</i> bit cleared” This is very confusing and quite possibly incorrect: It implies that waiters will be unblocked only if the SEVD is disabled, which is not true. This sentence really covers two separate topics. The first concerns thread wakeup, and it should be edited as follows: “If an arriving event causes an SEVD to reach notification criteria, then an SEVD waiter will be unblocked, if one exists.” The rest of the sentence applies to whether you can successfully wait on the SEVD in the first place. These are “wait-time” checks rather than “wake-time” checks, and are already covered in the bottom half of Page 73:</p> <p>“For a Simple EVD that does not have an associated AEVD, the Consumer can wait on and dequeue from the SEVD. If the SEVD has an associated AEVD with the IT_EVD_DEQUEUE_NOTIFICATIONS <i>evd_flag</i> cleared, then it is an error for the Consumer to wait on or dequeue from the SEVD. Attempting to wait on or dequeue from the SEVD will return IT_ERR_INVALID_EVD_STATE. If the SEVD has an associated AEVD with the IT_EVD_DEQUEUE_NOTIFICATIONS <i>evd_flag</i> set, then the Consumer can always dequeue from the SEVD, and the Consumer can wait on the SEVD but only if they disable the SEVD first (see <i>it_evd_modify</i>). Attempting to wait on the</p>	3

Global No.	Company No.	Title	Filename	Problem Description	Suggested Severity
				<p>SEVD when disallowed will return IT_ERR_INVALID_EVD_STATE.”</p> <p>Thus the second part of this sentence says nothing new and it should be deleted. Here is the current paragraph for context, and the proposed edits (for the record):</p> <p>“If arriving event causes SEVD to reach notification criteria then SEVD waiter will be unblocked if one exists and the SEVD is disabled and not associated with AEVD with IT_EVD_DEQUEUE_NOTIFICATIONS <i>evd_flag</i> bit cleared. As many waiters as there are events available on SEVD can be unblocked. If arriving notification event causes SEVD to reach notification criteria and SEVD is enabled then notification will be generated for associated AEVD or <i>fd</i>. As many notifications can be generated as there are events available on all SEVDs of the AEVD. If an arriving event causes an SEVD to reach notification criteria, then an SEVD waiter will be unblocked, if one exists. If there are multiple waiters on the SEVD, as many waiters as there are events available on the SEVD may be unblocked. If the SEVD is enabled and associated with an AEVD or <i>fd</i>, then notification will be generated for that AEVD or <i>fd</i>. In the AEVD case, as many notifications may be generated as there are events available on all SEVDs of the AEVD.”</p>	

Global No.	Company No.	Title	Filename	Problem Description	Suggested Severity
58	SUN17	<i>it_rmr_bind</i> : ordering rules have gone missing	<i>it_post_send</i> <i>it_rmr_bind</i> <i>it_rmr_unbind</i>	<p>The reference page for <i>it_post_send()</i> discusses the ordering of various operations, but the discussion is incomplete. The following contains the related text: “The Implementation ensures that a RDMA Write DTO preceding the Send has fully delivered its payload prior to the completion of the remote Receive corresponding to the Send. The Implementation ensures that all Sends start and complete in the order posted. Send and RDMA DTOs following an RDMA Read DTO may start during execution of the RDMA Read DTO and complete before the RDMA Read completes. To ensure deterministically that subsequent Sends and RDMA DTOs following an RDMA Read DTO do start after the RDMA Read completes, specify the IT_BARRIER_FENCE_FLAG on the DTOs following the RDMA Read.”</p> <p>There is no discussion of the following IBTA ordering rule:</p> <p>C10-64: Any Work Request posted to a Send Queue subsequent to a Bind Work Request shall not begin execution until the Bind operation completes.</p> <p>Worse, the reference pages for <i>it_rmr_[un]bind()</i> do not contain any text regarding the ordering of various operations.</p>	3

Global No.	Company No.	Title	Filename	Problem Description	Suggested Severity
59	SUN18	<i>it_rmr_bind</i> : completion suppression advice is confusing	<i>it_rmr_bind</i>	<p>The following text in <i>it_rmr_bind()</i> is not clear. “For reasons already described, the Bind Completion Event marks an important change in the status of an RMR that some Consumers may need to monitor. It is inadvisable for such Consumers to suppress this Completion Event by omitting IT_COMPLETION_FLAG, although the completion status of the Bind operation may be inferred by other means. For example, successful completion of a subsequently posted operation of any type indicates that the Bind operation has completed successfully. If the Bind operation fails, a Bind Completion Event is generated regardless.”</p> <p>In addition to being unclear, the text is not complete. There is a third way a Consumer can use an RMR context while ignoring completions, and that is to only reference the new RMR context in DTOs posted to the same EP after the bind operation. Those DTOs will only begin processing after the bind completes successfully.</p>	2
60	HP17	<i>it_cm_msg_events</i> : broken cross-reference	<i>it_cm_msg_events</i>	Broken cross-references appear in the <i>it_cm_msg_events</i> reference page. Search for “Error! Reference source not found.” in the PDF file.	1
61	HP18	<i>it_ep_disconnect</i> : bad grammar	<i>it_ep_disconnect</i>	Replace “ <i>cn_est_id</i> then the Implementation generate an IT_ERR_INVALID_CN_EST_ID error,” with: “ <i>cn_est_id</i> then the Implementation may generate an IT_ERR_INVALID_CN_EST_ID error,”.	1

Global No.	Company No.	Title	Filename	Problem Description	Suggested Severity
62	HP19	it_cm_msg_events: no ordering guarantees	<i>it_cm_msg_events</i> <i>it_dto_events</i>	When an Endpoint connection is disconnected, any pending Receive DTOs are completed with a flushed status, and a disconnect Event is enqueued on the connection message Event Stream. The IT-API doesn't say anything about the relative order of these two actions. At least one IT-API application made the assumption that the disconnect Event would be enqueued before the flushed event. It seems to me that we need to clarify the behavior of the IT-API in this area, because failing to do so can create portability problems between different Implementations. As far as what clarification we should make, we should state explicitly that there is no order guarantee here.	2
63	IBM1	Erratum: logical OR		IT-API 1 uses the term "logical OR" on several reference pages, where in fact "bitwise OR" should have been used. For example, it doesn't make sense to logically OR access privileges; e.g., for <i>it_lmr_create</i> or <i>it_rmr_bind_calls</i> , but it would make sense to bitwise OR them. To resolve the problem for the access privileges, it would be further necessary to redefine IT_PRIV_REMOTE as the bitwise OR of IT_PRIV_REMOTE_READ and IT_PRIV_REMOTE_WRITE (currently not the case).	2

11249

11250 **F** Header Files

11251 **F.1** **it_api.h**

```
11252     #include "it_api_os_specific.h"
11253
11254     #define IN
11255     #define OUT
11256
11257     /* define IT-API 1.0 variable name mappings for
11258        it_rc_only_attributes_t */
11259     #define rdma_read_inflight_incoming rdma_read_ird
11260     #define rdma_read_inflight_outgoing rdma_read_ord
11261
11262     /* define IT-API 1.0 variable name mappings for it_ia_info_t */
11263     #define ird_support ird_ord_ia_support
11264     #define ord_support ird_ord_ia_support
11265
11266     /* define IT-API 1.0 name mappings for it_make_rdma_addr */
11267     #define it_make_rdma_addr it_make_rdma_addr_absolute
11268
11269     /* typedefs */
11270     typedef enum {
11271         IT_SUCCESS = 0,
11272         IT_ERR_ABORT,
11273         IT_ERR_ACCESS,
11274         IT_ERR_ADDRESS,
11275         IT_ERR_AEVD_NOT_ALLOWED,
11276         IT_ERR_ASYNC_AFF_EVD_EXISTS,
11277         IT_ERR_ASYNC_UNAFF_EVD_EXISTS,
11278         IT_ERR_CANNOT_RESET,
11279         IT_ERR_CONN_QUAL_BUSY,
11280         IT_ERR_EP_TIMEOUT,
11281         IT_ERR_EVD_BUSY,
11282         IT_ERR_EVD_QUEUE_FULL,
11283         IT_ERR_FAULT,
11284         IT_ERR_IA_CATASTROPHE,
11285         IT_ERR_INTERRUPT,
11286         IT_ERR_INVALID_ADDRESS,
11287         IT_ERR_INVALID_AEVD,
11288         IT_ERR_INVALID_AH,
11289         IT_ERR_INVALID_ATIMEOUT,
11290         IT_ERR_INVALID_CM_RETRY,
11291         IT_ERR_INVALID_CN_EST_FLAGS,
11292         IT_ERR_INVALID_CN_EST_ID,
11293         IT_ERR_INVALID_CONN_EVD,
11294         IT_ERR_INVALID_CONN_QUAL,
11295         IT_ERR_INVALID_CONVERSION,
```

11296	IT_ERR_INVALID_DTO_FLAGS,
11297	IT_ERR_INVALID_EP,
11298	IT_ERR_INVALID_EP_ATTR,
11299	IT_ERR_INVALID_EP_KEY,
11300	IT_ERR_INVALID_EP_STATE,
11301	IT_ERR_INVALID_EP_TYPE,
11302	IT_ERR_INVALID_EVD,
11303	IT_ERR_INVALID_EVD_STATE,
11304	IT_ERR_INVALID_EVD_TYPE,
11305	IT_ERR_INVALID_FLAGS,
11306	IT_ERR_INVALID_HANDLE,
11307	IT_ERR_INVALID_IA,
11308	IT_ERR_INVALID_LENGTH,
11309	IT_ERR_INVALID_LISTEN,
11310	IT_ERR_INVALID_LMR,
11311	IT_ERR_INVALID_LTIMEOUT,
11312	IT_ERR_INVALID_MAJOR_VERSION,
11313	IT_ERR_INVALID_MASK,
11314	IT_ERR_INVALID_MINOR_VERSION,
11315	IT_ERR_INVALID_NAME,
11316	IT_ERR_INVALID_NETADDR,
11317	IT_ERR_INVALID_NUM_SEGMENTS,
11318	IT_ERR_INVALID_PDATA_LENGTH,
11319	IT_ERR_INVALID_PRIVS,
11320	IT_ERR_INVALID_PZ,
11321	IT_ERR_INVALID_QUEUE_SIZE,
11322	IT_ERR_INVALID_RECV_EVD,
11323	IT_ERR_INVALID_RECV_EVD_STATE,
11324	IT_ERR_INVALID_REQ_EVD,
11325	IT_ERR_INVALID_REQ_EVD_STATE,
11326	IT_ERR_INVALID_RETRY,
11327	IT_ERR_INVALID_RMR,
11328	IT_ERR_INVALID_RNR_RETRY,
11329	IT_ERR_INVALID_RTIMEOUT,
11330	IT_ERR_INVALID_SOFT_EVD,
11331	IT_ERR_INVALID_SOURCE_PATH,
11332	IT_ERR_INVALID_SPIGOT,
11333	IT_ERR_INVALID_THRESHOLD,
11334	IT_ERR_INVALID_UD_STATUS,
11335	IT_ERR_INVALID_UD_SVC,
11336	IT_ERR_INVALID_UD_SVC_REQ_ID,
11337	IT_ERR_LMR_BUSY,
11338	IT_ERR_MISMATCH_FD,
11339	IT_ERR_NO_CONTEXT,
11340	IT_ERR_NO_PERMISSION,
11341	IT_ERR_PAYLOAD_SIZE,
11342	IT_ERR_PDATA_NOT_SUPPORTED,
11343	IT_ERR_PZ_BUSY,
11344	IT_ERR_QUEUE_EMPTY,
11345	IT_ERR_RANGE,
11346	IT_ERR_RESOURCES,
11347	IT_ERR_RESOURCE_IRD,
11348	IT_ERR_RESOURCE_LMR_LENGTH,
11349	IT_ERR_RESOURCE_ORD,

```

11350         IT_ERR_RESOURCE_QUEUE_SIZE,
11351         IT_ERR_RESOURCE_RECV_DTO,
11352         IT_ERR_RESOURCE_REQ_DTO,
11353         IT_ERR_RESOURCE_RRSEG,
11354         IT_ERR_RESOURCE_RSEG,
11355         IT_ERR_RESOURCE_RWSEG,
11356         IT_ERR_RESOURCE_SSEG,
11357         IT_ERR_TIMEOUT_EXPIRED,
11358         IT_ERR_TOO_MANY_POSTS,
11359         IT_ERR_WAITER_LIMIT,
11360         IT_ERR_INVALID_SRQ,
11361         IT_ERR_SOFT_HI_WATERMARK,
11362         IT_ERR_HARD_HI_WATERMARK,
11363         IT_ERR_INVALID_WATERMARK,
11364         IT_ERR_INVALID_RECV_DTO,
11365         IT_ERR_INVALID_SRQ_SIZE,
11366         IT_ERR_SRQ_LOW_WATERMARK,
11367         IT_ERR_SRQ_BUSY,
11368         IT_ERR_SRQ_NOT_SUPPORTED,
11369         IT_ERR_INVALID_ADDR_MODE,
11370         IT_ERR_INVALID_RMR_TYPE,
11371         IT_ERR_OP_NOT_SUPPORTED,
11372         IT_ERR_EP_BUSY
11373     } it_status_t;
11374
11375     typedef uint32_t it_rmr_context_t;
11376
11377     #ifdef IT_32BIT
11378         typedef uint32_t it_length_t; /* a 32-bit platform */
11379     #else
11380         typedef uint64_t it_length_t; /* a 64-bit platform */
11381     #endif
11382
11383     typedef enum {
11384         IT_PRIV_NONE           = 0x0000, /* needed for IT-API 1.0 compat */
11385         IT_PRIV_LOCAL_READ    = 0x0001,
11386         IT_PRIV_LOCAL_WRITE   = 0x0002,
11387         IT_PRIV_LOCAL         = 0x0003,
11388         IT_PRIV_DEFAULT       = 0x0003, /* deprecated by IT_PRIV_LOCAL */
11389         IT_PRIV_REMOTE_READ   = 0x0004,
11390         IT_PRIV_REMOTE_WRITE  = 0x0008,
11391         IT_PRIV_REMOTE        = 0x000c,
11392         IT_PRIV_ALL           = 0x000f
11393     } it_mem_priv_t;
11394
11395     typedef enum {
11396         IT_LMR_FLAG_NONE      = 0x0001,
11397         IT_LMR_FLAG_SHARED    = 0x0002,
11398         IT_LMR_FLAG_NONCOHERENT = 0x0004
11399     } it_lmr_flag_t;
11400
11401     typedef enum {
11402         IT_RMR_TYPE_DEFAULT = 0,
11403         IT_RMR_TYPE_NARROW  = 1,

```

```

11404         IT_RMR_TYPE_WIDE      = 2
11405     } it_rmr_type_t;
11406
11407     typedef enum {
11408         IT_ADDR_MODE_ABSOLUTE = 0,
11409         IT_ADDR_MODE_RELATIVE = 1
11410     } it_addr_mode_t;
11411
11412     typedef uint64_t it_ud_svc_req_identififer_t;
11413
11414     typedef uint64_t it_cn_est_identififer_t;
11415
11416     typedef enum {
11417         IT_FALSE = 0,
11418         IT_TRUE  = 1
11419     } it_boolean_t;
11420
11421     typedef enum {
11422         IT_HANDLE_TYPE_ADDR,
11423         IT_HANDLE_TYPE_EP,
11424         IT_HANDLE_TYPE_EVD,
11425         IT_HANDLE_TYPE_IA,
11426         IT_HANDLE_TYPE_LISTEN,
11427         IT_HANDLE_TYPE_LMR,
11428         IT_HANDLE_TYPE_PZ,
11429         IT_HANDLE_TYPE_RMR,
11430         IT_HANDLE_TYPE_UD_SVC_REQ,
11431         IT_HANDLE_TYPE_SRQ
11432     } it_handle_type_enum_t;
11433
11434     typedef void *it_handle_t;
11435
11436     #define IT_NULL_HANDLE ((it_handle_t) NULL)
11437
11438     typedef struct it_addr_handle_s      *it_addr_handle_t;
11439     typedef struct it_ep_handle_s        *it_ep_handle_t;
11440     typedef struct it_evd_handle_s       *it_evd_handle_t;
11441     typedef struct it_ia_handle_s         *it_ia_handle_t;
11442     typedef struct it_listen_handle_s    *it_listen_handle_t;
11443     typedef struct it_lmr_handle_s        *it_lmr_handle_t;
11444     typedef struct it_pz_handle_s         *it_pz_handle_t;
11445     typedef struct it_rmr_handle_s        *it_rmr_handle_t;
11446     typedef struct it_ud_svc_req_handle_s *it_ud_svc_req_handle_t;
11447     typedef struct it_srq_handle_s        *it_srq_handle_t;
11448
11449     typedef enum {
11450
11451         /* IANA (TCP/UDP) Port Number */
11452         IT_IANA_PORT = 0x01,
11453
11454         /* InfiniBand Service ID, as described in section 12.7.3 of
11455         Volume 1 of the InfiniBand specification. */
11456         IT_IB_SERVICEID = 0x02,
11457

```

```

11458         /* VIA Connection Discriminator */
11459         IT_VIA_DISCRIMINATOR = 0x04,
11460
11461         /* iWARP local and remote IP (IANA) port object */
11462         IT_IANA_LR_PORT = 0x08
11463     } it_conn_qual_type_t;
11464
11465 #define IT_MAX_VIA_DISC_LEN 64
11466
11467 typedef struct {
11468
11469     /* The total number of bytes in the array below */
11470     /* that are significant */
11471     uint16_t len;
11472
11473     /* VIA connection discriminator, which is an array of bytes */
11474     unsigned char discriminator[IT_MAX_VIA_DISC_LEN];
11475
11476 } it_via_discriminator_t;
11477
11478 typedef uint64_t it_ib_serviceid_t;
11479
11480 typedef struct {
11481     uint16_t local;
11482     uint16_t remote;
11483 } it_iana_lr_port_t;
11484
11485 typedef struct {
11486
11487     /* The discriminator for the union below. */
11488     it_conn_qual_type_t type;
11489
11490     union {
11491
11492         /* IANA Port Number, in network byte order */
11493         uint16_t port;
11494
11495         /* InfiniBand Service ID, in network byte order */
11496         it_ib_serviceid_t serviceid;
11497
11498         /* VIA connection discriminator. */
11499         it_via_discriminator_t discriminator;
11500
11501         /* IANA local/remote Port numbers, in network byte order */
11502         it_iana_lr_port_t lr_port;
11503
11504     } conn_qual;
11505
11506 } it_conn_qual_t;
11507
11508 typedef union {
11509     void * ptr;
11510     uint64_t index;
11511

```

```

11512     } it_context_t;
11513
11514     typedef uint64_t it_dto_cookie_t;
11515
11516     typedef enum {
11517         IT_DTO_SUCCESS                = 0,
11518         IT_DTO_ERR_LOCAL_LENGTH      = 1,
11519         IT_DTO_ERR_LOCAL_EP         = 2,
11520         IT_DTO_ERR_LOCAL_PROTECTION = 3,
11521         IT_DTO_ERR_FLUSHED          = 4,
11522         IT_RMR_OPERATION_FAILED     = 5,
11523         IT_DTO_ERR_BAD_RESPONSE     = 6,
11524         IT_DTO_ERR_REMOTE_ACCESS    = 7,
11525         IT_DTO_ERR_REMOTE_RESPONDER = 8,
11526         IT_DTO_ERR_TRANSPORT        = 9,
11527         IT_DTO_ERR_RECEIVER_NOT_READY = 10,
11528         IT_DTO_ERR_PARTIAL_PACKET   = 11,
11529         IT_DTO_ERR_LOCAL_MM_OPERATION = 12
11530     } it_dto_status_t;
11531
11532     typedef enum
11533     {
11534         /* If flag set, completion generates a local event */
11535         IT_COMPLETION_FLAG = 0x01,
11536
11537         /* If flag set, completion causes local Notification */
11538         IT_NOTIFY_FLAG = 0x02,
11539
11540         /* If flag set, receipt of DTO at remote will cause
11541          Notification at remote */
11542         IT_SOLICITED_WAIT_FLAG = 0x04,
11543
11544         /* If flag set, DTO processing will not start if
11545          previously posted RDMA Reads are not complete. */
11546         IT_BARRIER_FENCE_FLAG = 0x08,
11547     } it_dto_flags_t;
11548
11549     typedef enum {
11550
11551         /* IPv4 address */
11552         IT_IPV4 = 0x1,
11553
11554         /* IPv6 address */
11555         IT_IPV6 = 0x2,
11556
11557         /* InfiniBand GID */
11558         IT_IB_GID = 0x3,
11559
11560         /* VIA Network Address */
11561         IT_VIA_HOSTADDR = 0x4
11562     } it_net_addr_type_t;
11563
11564     #define IT_MAX_VIA_ADDR_LEN 64

```

```

11566
11567     typedef struct {
11568
11569         /* The number of bytes in the array below that are
11570            significant */
11571         uint16_t len;
11572
11573         /* VIA host address, which is an array of bytes */
11574         unsigned char hostaddr[IT_MAX_VIA_ADDR_LEN];
11575
11576     } it_via_net_addr_t;
11577
11578     typedef struct in6_addr  it_ib_gid_t;
11579
11580     typedef struct {
11581
11582         /* The discriminator for the union below. */
11583         it_net_addr_type_t  addr_type;
11584
11585         union {
11586
11587             /* IPv4 address, in network byte order */
11588             struct in_addr  ipv4;
11589
11590             /* IPv6 address, in network byte order */
11591             struct in6_addr  ipv6;
11592
11593             /* InfiniBand GID, in network byte order */
11594             it_ib_gid_t  gid;
11595
11596             /* VIA Network Address */
11597             it_via_net_addr_t  via;
11598
11599         } addr;
11600
11601     } it_net_addr_t;
11602
11603     typedef enum {
11604
11605         /* InfiniBand Transport */
11606         IT_IB_TRANSPORT = 1,
11607
11608         /* VIA host Interface using IP transport, supporting
11609            only the Reliable Delivery reliability level */
11610         IT_VIA_IP_TRANSPORT = 2,
11611
11612         /* VIA host Interface, using Fibre Channel transport, supporting
11613            only the Reliable Delivery reliability level*/
11614         IT_VIA_FC_TRANSPORT = 3,
11615
11616         /* iWARP over TCP transport */
11617         IT_IWARP_TCP_TRANSPORT = 4,
11618
11619         /* Vendor-proprietary Transport */

```

```

11620         IT_VENDOR_TRANSPORT = 1000
11621
11622     } it_transport_type_t;
11623
11624     typedef enum {
11625
11626         /* Reliable Connected Transport Service Type */
11627         IT_RC_SERVICE = 0x1,
11628
11629         /* Unreliable Datagram Transport Service Type */
11630         IT_UD_SERVICE = 0x2,
11631
11632     } it_transport_service_type_t;
11633
11634     typedef struct {
11635
11636         /* Spigot identifier */
11637         size_t spigot_id;
11638
11639         /* Maximum sized Send operation for the RC service
11640          on this spigot. */
11641         size_t max_rc_send_len;
11642
11643         /* Maximum sized RDMA Read/Write operation for the RC service on
11644          this spigot. */
11645         size_t max_rc_rdma_len;
11646
11647         /* Maximum sized Send operation for the UD service
11648          on this spigot. */
11649         size_t max_ud_send_len;
11650
11651         /* Indicates whether the Spigot is online or offline.
11652          An IT_TRUE value means online. */
11653         it_boolean_t spigot_online;
11654
11655         /* A mask indicating which Connection Qualifier types this
11656          IA supports for input to it_ep_connect and
11657          it_ud_service_request_handle_create. The bits in the mask are
11658          an inclusive OR of the values for Connection Qualifier types
11659          that this IA supports. */
11660         it_conn_qual_type_t active_side_conn_qual;
11661
11662         /* A mask indicating which Connection Qualifier types this to
11663          it_listen_create. The bits in the mask are an inclusive OR of
11664          the values for Connection Qualifier types that this IA
11665          supports. */
11666         it_conn_qual_type_t passive_side_conn_qual;
11667
11668         /* The number of Network Addresses associated with spigot. */
11669         size_t num_net_addr;
11670
11671         /* Pointer to array of Network Address addresses. */
11672         it_net_addr_t* net_addr;
11673

```

```

11674     } it_spigot_info_t;
11675
11676     typedef struct {
11677
11678         /* The NodeInfo:VendorID as described in chapter 14 of the
11679            IB spec. */
11680         uint32_t vendor : 24;
11681
11682         /* The NodeInfo:DeviceID as described in chapter 14 of the
11683            IB spec. */
11684         uint16_t device;
11685
11686         /* The NodeInfo:Revision as described in chapter 14 of the
11687            IB spec. */
11688         uint32_t revision;
11689     } it_vendor_ib_t;
11690
11691     typedef struct {
11692         /* The "Name" member of the VIP_NIC_ATTRIBUTES structure, as
11693            described in the VIA spec. */
11694         char name[64];
11695
11696         /* The "HardwareVersion" member of the VIP_NIC_ATTRIBUTES
11697            structure, as described in the VIA spec. */
11698         unsigned long hardware;
11699
11700         /* The "ProviderVersion" member of the VIP_NIC_ATTRIBUTES
11701            structure, as described in the VIA spec. */
11702         unsigned long provider;
11703     } it_vendor_via_t;
11704
11705     typedef struct {
11706         /* Indicates whether or not vid field contains valid data */
11707         it_boolean_t valid_vid;
11708
11709         /* Vendor Identification field - strictly vendor-specific (only
11710            valid if valid_vid field is IT_TRUE */
11711         unsigned char vid[64];
11712     } it_vendor_iwarp_tcp_t;
11713
11714     typedef struct {
11715
11716         /* Interface Adapter name, as specified in it_ia_create */
11717         char* ia_name;
11718
11719         /* The major version number of the latest version of the IT-API
11720            that this IA supports. */
11721         uint32_t api_major_version;
11722
11723         /* The minor version number of the latest version of the IT-API
11724            that this IA supports. */
11725         uint32_t api_minor_version;
11726
11727         /* The major version number for the software being used to control

```

```

11728         this IA. The IT-API imposes no structure whatsoever on this
11729         number; its meaning is completely IA-dependent. */
11730     uint32_t  sw_major_version;
11731
11732     /* The minor version number for the software being used to control
11733     this IA. The IT-API imposes no structure whatsoever on this
11734     number; its meaning is completely IA-dependent. */
11735     uint32_t  sw_minor_version;
11736
11737     /* The vendor associated with the IA. This information is useful
11738     if the Consumer wishes to do device-specific programming. This
11739     union is discriminated by transport_type. No vendor
11740     identification is provided for transports not listed below. */
11741     union {
11742
11743         /* Used if transport_type is IT_IB_TRANSPORT */
11744         it_vendor_ib_t  ib;
11745
11746         /* Used if transport_type is IT_VIA_IP_TRANSPORT or
11747         IT_VIA_FC_TRANSPORT */
11748         it_vendor_via_t  via;
11749
11750         /* Used if transport_type is IT_IWARP_TCP_TRANSPORT */
11751         it_vendor_iwarp_tcp_t  iwarp;
11752     } vendor;
11753
11754     /* The Interface Adapter and platform provide a data alignment hint
11755     to the Consumer to help the Consumer align their data transfer
11756     buffers in a way that is optimal for the performance of the IA.
11757     For example, if the best throughput is obtained by aligning
11758     buffers to 128-byte boundaries, dto_alignment_hint will have the
11759     value 128. The Consumer may choose to ignore the alignment hint
11760     without any adverse functional impact. (There may be an adverse
11761     performance impact.) */
11762     uint32_t  dto_alignment_hint;
11763
11764     /* The transport type (e.g., InfiniBand) supported by an Interface
11765     Adapter. An Interface Adapter supports precisely one transport
11766     type. */
11767     it_transport_type_t  transport_type;
11768
11769     /* The Transport Service Types supported by this IA. This is
11770     constructed by doing an inclusive OR of the Transport Service
11771     Type values.*/
11772     it_transport_service_type_t  supported_service_types;
11773
11774     /* Indicates whether Work Queues are resizable */
11775     it_boolean_t  ep_work_queues_resizable;
11776
11777     /* Indicates whether the underlying transport used by this IA uses
11778     a three-way handshake for doing Connection establishment. Note
11779     that if the underlying transport supports a three-way handshake
11780     the Consumer can choose whether to use two handshakes or three
11781

```

```

11782         when establishing the Connection. If the underlying transport
11783         supports a two-way handshake for establishing a Connection, the
11784         Consumer can only use two handshakes when establishing the
11785         Connection. */
11786     it_boolean_t  three_way_handshake_support;
11787
11788     /* Indicates whether Private Data is supported on Connection
11789     establishment or UD service resolution operations. */
11790     it_boolean_t  private_data_support;
11791
11792     /* Indicates whether the max_message_size field in the
11793     IT_CM_REQ_CONN_REQUEST_EVENT is valid for this IA. */
11794     it_boolean_t  max_message_size_support;
11795
11796     /* Indicates whether or not the IA supports IRD/ORD. Affects
11797     whether the rdma_read_ird or rdma_read_ord fields in the
11798     IT_CM_REQ_CONN_REQUEST_EVENT or the
11799     IT_CM_MSG_CONN_ESTABLISHED_EVENT are valid for this IA.
11800     Also affects whether IRD/ORD suppression is an option.
11801     Deprecates IT-API 1.0 values ird_support and ord_support. */
11802     it_boolean_t  ird_ord_ia_support;
11803
11804     /* Indicates whether IRD/ORD suppression is supported
11805     for this IA. If this member has a value of IT_TRUE, the
11806     Consumer can control IRD/ORD suppression in it_ep_connect
11807     and it_listen_create. Otherwise they cannot. */
11808     it_boolean_t  ird_ord_suppressible;
11809
11810     /* Indicates whether the IA generates IT_ASYNC_UNAFF_SPIGOT_ONLINE
11811     Events. See it_unaffiliated_event_t for details. */
11812     it_boolean_t  spigot_online_support;
11813
11814     /* Indicates whether the IA generates IT_ASYNC_UNAFF_SPIGOT_OFFLINE
11815     Events. See it_unaffiliated_event_t for details. */
11816     it_boolean_t  spigot_offline_support;
11817
11818     /* The maximum number of bytes of Private Data supported for the
11819     it_ep_connect routine. This will be less than or equal to
11820     IT_MAX_PRIV_DATA. */
11821     size_t  connect_private_data_len;
11822
11823     /* The maximum number of bytes of Private Data supported for the
11824     it_ep_accept routine. This will be less than or equal to
11825     IT_MAX_PRIV_DATA. */
11826     size_t  accept_private_data_len;
11827
11828     /* The maximum number of bytes of Private Data supported for the
11829     it_reject routine. This will be less than or equal to
11830     IT_MAX_PRIV_DATA. */
11831     size_t  reject_private_data_len;
11832
11833     /* The maximum number of bytes of Private Data supported for the
11834     it_ep_disconnect routine. This will be less than or equal to
11835     IT_MAX_PRIV_DATA. */

```

```

11836     size_t  disconnect_private_data_len;
11837
11838     /* The maximum number of bytes of Private Data supported for the
11839        it_ud_service_request_handle_create routine. This will be
11840        less than or equal to IT_MAX_PRIV_DATA. */
11841     size_t  ud_req_private_data_len;
11842
11843     /* The maximum number of bytes of Private Data supported for the
11844        it_ud_service_reply routine. This will be less than or equal to
11845        IT_MAX_PRIV_DATA. */
11846     size_t  ud_rep_private_data_len;
11847
11848     /* Specifies the number of Spigots associated with this Interface
11849        Adapter */
11850     size_t  num_spigots;
11851
11852     /* An array of Spigot information data structures. The array
11853        contains num_spigots elements. */
11854     it_spigot_info_t*  spigot_info;
11855
11856     /* The Handle for the EVD that contains the affiliated async Event
11857        Stream. If no EVD contains the Affiliated Async Event Stream,
11858        this member will have the distinguished value IT_NULL_HANDLE */
11859     it_evd_handle_t  affiliated_err_evd;
11860
11861     /* The Handle for the EVD that contains the Unaffiliated Async
11862        Event Stream. If no EVD contains the Unaffiliated Async Event
11863        Stream, this member will have the distinguished value
11864        IT_NULL_HANDLE */
11865     it_evd_handle_t  unaffiliated_err_evd;
11866
11867     /* Indicates whether the IA supports the S-RQ feature */
11868     it_boolean_t  srq_support;
11869
11870     /* Indicates whether the IA supports the Endpoint Hard
11871        High Watermark mechanism for limiting the number of Receive
11872        DTOs that can be in progress on an Endpoint that has an
11873        associated S-RQ */
11874     it_boolean_t  hard_hi_watermark_support;
11875
11876     /* Indicates whether the IA supports the Endpoint Soft
11877        High Watermark mechanism for generating an Affiliated
11878        Asynchronous Event when the number of Receive DTOs in
11879        progress on an Endpoint that has an associated S-RQ exceeds
11880        the Endpoint Soft High Watermark. */
11881     it_boolean_t  soft_hi_watermark_support;
11882
11883     /* Indicates whether an S-RQ can be resized after it is created */
11884     it_boolean_t  srq_resizable;
11885
11886     /* Indicates that an iWARP V-RNIC supports a modified qp state
11887        diagram (outside the RDMAC verbs) */
11888     it_boolean_t  extended_iwarp_qp_states;
11889

```

```

11890      /* Indicates that Implementation supports socket conversion (TDI).
11891         This attribute is IT_TRUE if and only if transport_type is
11892         IT_IWARP_TCP_TRANSPORT. */
11893      it_boolean_t  socket_conversion_support;
11894
11895      /* Indicates that IA supports increasing ORD
11896         (decreasing ORD is mandatory for all RNICs) */
11897      it_boolean_t  rdma_read_ord_increasable;
11898
11899      /* Indicates that IA supports modifying IRD */
11900      it_boolean_t  rdma_read_ird_modifiable;
11901
11902      /* Bit set indicating which RMR types are supported.
11903         Possible values are IT_RMR_TYPE_NARROW, IT_RMR_TYPE_WIDE, and
11904         (IT_RMR_TYPE_NARROW|IT_RMR_TYPE_WIDE). See also it_rmr_type_t.*/
11905      it_rmr_type_t  rmr_types_supported;
11906
11907      /* Indicates whether the IA supports Relative Addressing. See also
11908         it_addr_mode_t. */
11909      it_boolean_t  addr_mode_relative_support;
11910
11911      /* Indicates whether the Destination buffer for an RDMA Read DTO
11912         must have remote or local write permission, and whether or not
11913         the Endpoint to which an RDMA Read DTO is posted must have RDMA
11914         Write access enabled. See also it_post_rdma_read. */
11915      it_boolean_t  rdma_read_requires_remote_write;
11916
11917      /* Indicates whether the IA supports changing the RDMA enables
11918         after EP creation. */
11919      it_boolean_t  ep_rdma_enables_modifiable;
11920
11921      /* Indicates whether the IA supports it_post_rdma_read_to_rmr. */
11922      it_boolean_t  rdma_read_local_extensions;
11923
11924      /* Indicates whether the IA supports DTO EVD overflow detection. */
11925      it_boolean_t  dto_evd_overflow_detection;
11926  } it_ia_info_t;
11927
11928  typedef struct {
11929      it_lmr_handle_t  lmr;
11930      union {
11931          void          *abs;
11932          it_length_t  rel;
11933      } addr;
11934      it_length_t  length;
11935  } it_lmr_triplet_t;
11936
11937  typedef struct {
11938      it_rmr_handle_t  rmr;
11939      union {
11940          void          *abs;
11941          it_length_t  rel;
11942      } addr;
11943      it_length_t  length;

```

```

11944     } it_rmr_triplet_t;
11945
11946     typedef struct {
11947
11948         /* Partition Key, as defined in the REQ message for the IB
11949            CM protocol */
11950         uint16_t  partition_key;
11951
11952         /* Path Packet Payload MTU, as defined in the REQ message
11953            for the IB CM protocol */
11954         uint8_t   path_mtu : 4;
11955
11956         /* PacketLifeTime, as defined in the PathRecord in IB
11957            specification. This field is useful for Consumers that
11958            wish to use timeout values other than the default ones
11959            for doing Connection establishment. */
11960         uint8_t   packet_lifetime : 6;
11961
11962         /* Local Port LID, as defined in the REQ message for the IB
11963            CM protocol. The low order bits of this value also
11964            constitute the Source Path Bits that are used to
11965            create an Address Handle. */
11966         uint16_t  local_port_lid;
11967
11968         /* Remote Port LID, as defined in the REQ message for the
11969            IB CM protocol. This is also the Destination LID used
11970            to create an Address Handle. */
11971         uint16_t  remote_port_lid;
11972
11973         /* Local Port GID in network byte order, as defined in the
11974            REQ message for the IB CM protocol. This is also used to
11975            determine the appropriate Source GID Index to be used
11976            when creating an Address Handle. */
11977         it_ib_gid_t  local_port_gid;
11978
11979         /* Remote Port GID in network byte order, as defined in the
11980            REQ message for the IB CM protocol. This is also the
11981            Destination GID or MGID used to create an Address
11982            Handle. */
11983         it_ib_gid_t  remote_port_gid;
11984
11985         /* Packet Rate, as defined in the REQ message for the IB CM
11986            protocol. This is also the Maximum Static Rate to be
11987            used when creating an Address Handle. */
11988         uint8_t   packet_rate : 6;
11989
11990         /* SL, as defined in the REQ message for the IB CM
11991            protocol. This is also the Service Level to be used
11992            when creating an Address Handle. */
11993         uint8_t   sl : 4;
11994
11995         /* Subnet Local, as defined in the REQ message for the IB
11996            CM protocol. When creating an Address Handle, setting
11997            this bit causes a GRH to be included as part of any

```

```

11998         Unreliable Datagram sent using the Address Handle. */
11999         uint8_t  subnet_local : 1;
12000
12001         /* Flow Label, as defined in the REQ message for the IB CM
12002            protocol. This is also the Flow Label to be used when
12003            creating an Address Handle. This is only valid if
12004            subnet_local is clear. */
12005         uint32_t  flow_label : 20;
12006
12007         /* Traffic Class, as defined in the REQ message for the IB
12008            CM protocol. This is also the Traffic Class to be
12009            used when creating an Address Handle. This is only
12010            valid if subnet_local is clear. */
12011         uint8_t  traffic_class;
12012
12013         /* Hop Limit, as defined in the REQ message for the IB CM
12014            protocol. This is also the Hop Limit to be used when
12015            creating an Address Handle. This is only valid if
12016            subnet_local is clear. */
12017         uint8_t  hop_limit;
12018
12019     } it_ib_net_endpoint_t;
12020
12021     typedef it_via_net_addr_t it_via_net_endpoint_t;
12022
12023     typedef enum {
12024         IT_IP_VERS_IPV4 = 0x1,
12025         IT_IP_VERS_IPV6 = 0x2
12026     } it_ip_vers_t;
12027
12028     typedef struct {
12029         /* Designates the type of IP address that is found in
12030            both the laddr and raddr unions below */
12031         it_ip_vers_t  ip_vers;
12032
12033         /* Local path element */
12034         union {
12035             struct in_addr  ipv4;
12036             struct in6_addr ipv6;
12037         } laddr;
12038
12039         /* Remote path element */
12040         union {
12041             struct in_addr  ipv4;
12042             struct in6_addr ipv6;
12043         } raddr;
12044     } it_iwarp_net_endpoint_t;
12045
12046     typedef struct {
12047
12048         /* Identifier for the Spigot to be used on the local IA.
12049            Note that this data structure is always used in a
12050            Context where the IA associated with the Spigot can be
12051            deduced. */

```

```

12052     size_t  spigot_id;
12053
12054     /* The transport-independent timeout parameter for how long
12055        to wait, in microseconds, before timing out a Connection
12056        establishment attempt using this Path. The timeout
12057        period for establishing a Connection
12058        can only be specified on the Active side; the timeout
12059        period cannot be changed on the Passive side. */
12060     uint64_t  timeout;
12061
12062     /* The remote component of the Path */
12063     union {
12064
12065         /* For use with InfiniBand */
12066         it_ib_net_endpoint_t  ib;
12067
12068         /* For use with VIA */
12069         it_via_net_endpoint_t  via;
12070
12071         /* For use with iWARP */
12072         it_iwarp_net_endpoint_t  iwarp;
12073
12074     } remote;
12075
12076 } it_path_t;
12077
12078 typedef uint32_t  it_ud_ep_id_t;
12079 typedef uint32_t  it_ud_ep_key_t;
12080
12081 typedef enum {
12082     IT_EP_PARAM_ALL                = 0x00000001,
12083     IT_EP_PARAM_IA                 = 0x00000002,
12084     IT_EP_PARAM_SPIGOT             = 0x00000004,
12085     IT_EP_PARAM_STATE              = 0x00000008,
12086     IT_EP_PARAM_SERV_TYPE          = 0x00000010,
12087     IT_EP_PARAM_PATH               = 0x00000020,
12088     IT_EP_PARAM_PZ                 = 0x00000040,
12089     IT_EP_PARAM_REQ_SEVD           = 0x00000080,
12090     IT_EP_PARAM_RECV_SEVD          = 0x00000100,
12091     IT_EP_PARAM_CONN_SEVD          = 0x00000200,
12092     IT_EP_PARAM_RDMA_RD_ENABLE     = 0x00000400,
12093     IT_EP_PARAM_RDMA_WR_ENABLE     = 0x00000800,
12094     IT_EP_PARAM_MAX_RDMA_READ_SEG  = 0x00001000,
12095     IT_EP_PARAM_MAX_RDMA_WRITE_SEG = 0x00002000,
12096     IT_EP_PARAM_MAX_IRD            = 0x00004000,
12097     IT_EP_PARAM_MAX_ORD            = 0x00008000,
12098     IT_EP_PARAM_EP_ID              = 0x00010000,
12099     IT_EP_PARAM_EP_KEY             = 0x00020000,
12100     IT_EP_PARAM_MAX_PAYLOAD        = 0x00040000,
12101     IT_EP_PARAM_MAX_REQ_DTO        = 0x00080000,
12102     IT_EP_PARAM_MAX_RECV_DTO       = 0x00100000,
12103     IT_EP_PARAM_MAX_SEND_SEG       = 0x00200000,
12104     IT_EP_PARAM_MAX_RECV_SEG       = 0x00400000,
12105     IT_EP_PARAM_SRQ                = 0x00800000,

```

```

12106         IT_EP_PARAM_SOFT_HI_WATERMARK = 0x01000000,
12107         IT_EP_PARAM_HARD_HI_WATERMARK = 0x02000000
12108     } it_ep_param_mask_t;
12109
12110     typedef struct {
12111         it_boolean_t rdma_read_enable;
12112         /* IT_EP_PARAM_RDMA_RD_ENABLE */
12113         it_boolean_t rdma_write_enable;
12114         /* IT_EP_PARAM_RDMA_WR_ENABLE */
12115         size_t max_rdma_read_segments;
12116         /* IT_EP_PARAM_MAX_RDMA_READ_SEG */
12117         size_t max_rdma_write_segments;
12118         /* IT_EP_PARAM_MAX_RDMA_WRITE_SEG */
12119         uint32_t rdma_read_ird;
12120         /* IT_EP_PARAM_MAX_IRD */
12121         uint32_t rdma_read_ord;
12122         /* IT_EP_PARAM_MAX_ORD */
12123         it_srq_handle_t srq;
12124         /* IT_EP_PARAM_SRQ */
12125         size_t soft_hi_watermark;
12126         /* IT_EP_PARAM_SOFT_HI_WATERMARK */
12127         size_t hard_hi_watermark;
12128         /* IT_EP_PARAM_HARD_HI_WATERMARK */
12129     } it_rc_only_attributes_t;
12130
12131     #define IT_HARD_HI_WATERMARK_DISABLE ((size_t) -1)
12132
12133     typedef struct {
12134         it_ud_ep_id_t ud_ep_id; /* IT_EP_PARAM_EP_ID */
12135         it_ud_ep_key_t ud_ep_key; /* IT_EP_PARAM_EP_KEY */
12136     } it_remote_ep_info_t;
12137
12138     typedef struct {
12139         it_remote_ep_info_t ep_info;
12140     } it_ud_only_attributes_t;
12141
12142     typedef union {
12143         it_rc_only_attributes_t rc;
12144         it_ud_only_attributes_t ud;
12145     } it_service_attributes_t;
12146
12147     typedef struct {
12148         size_t max_dto_payload_size; /* IT_EP_PARAM_MAX_PAYLOAD */
12149         size_t max_request_dtos; /* IT_EP_PARAM_MAX_REQ_DTO */
12150         size_t max_recv_dtos; /* IT_EP_PARAM_MAX_RECV_DTO */
12151         size_t max_send_segments; /* IT_EP_PARAM_MAX_SEND_SEG */
12152         size_t max_recv_segments; /* IT_EP_PARAM_MAX_RECV_SEG */
12153     } it_service_attributes_t srv;
12154
12155     } it_ep_attributes_t;
12156
12157     #define IT_EVENT_STREAM_MASK 0xff000
12158
12159

```

```

12160 #define IT_TIMEOUT_INFINITE ((uint64_t)(-1))
12161
12162 typedef enum
12163 {
12164     /*
12165      * Event Stream for WR/DTO completions
12166      */
12167     IT_DTO_EVENT_STREAM           = 0x00000,
12168     IT_DTO_SEND_CMPL_EVENT       = 0x00001,
12169     IT_DTO_RC_RECV_CMPL_EVENT    = 0x00002,
12170     IT_DTO_UD_RECV_CMPL_EVENT    = 0x00003,
12171     IT_DTO_RDMA_WRITE_CMPL_EVENT = 0x00004,
12172     IT_DTO_RDMA_READ_CMPL_EVENT  = 0x00005,
12173     IT_RMR_BIND_CMPL_EVENT       = 0x00006,
12174     IT_RMR_LINK_CMPL_EVENT       = 0x00006,
12175
12176     /*
12177      * Event Stream for Communication Management Request Events
12178      */
12179     IT_CM_REQ_EVENT_STREAM        = 0x01000,
12180     IT_CM_REQ_CONN_REQUEST_EVENT  = 0x01001,
12181     IT_CM_REQ_UD_SERVICE_REQUEST_EVENT = 0x01002,
12182
12183     /*
12184      * Event Stream for Communication Management Message Events
12185      */
12186     IT_CM_MSG_EVENT_STREAM        = 0x02000,
12187     IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT = 0x02001,
12188     IT_CM_MSG_CONN_ESTABLISHED_EVENT  = 0x02002,
12189     IT_CM_MSG_CONN_DISCONNECT_EVENT   = 0x02003,
12190     IT_CM_MSG_CONN_PEER_REJECT_EVENT   = 0x02004,
12191     IT_CM_MSG_CONN_NONPEER_REJECT_EVENT = 0x02005,
12192     IT_CM_MSG_CONN_BROKEN_EVENT        = 0x02006,
12193     IT_CM_MSG_UD_SERVICE_REPLY_EVENT   = 0x02007,
12194
12195     /* Event Stream for Affiliated Asynchronous Events */
12196     IT_ASYNC_AFF_EVENT_STREAM          = 0x04000,
12197     IT_ASYNC_AFF_SEVD_ENQUEUE_FAILURE  = 0x04001,
12198     IT_ASYNC_AFF_EP_FAILURE            = 0x04002,
12199     IT_ASYNC_AFF_EP_BAD_TRANSPORT_OPCODE = 0x04003,
12200     IT_ASYNC_AFF_EP_REQ_DROPPED        = 0x04005,
12201     IT_ASYNC_AFF_EP_RDMAW_ACCESS_VIOLATION = 0x04006,
12202     IT_ASYNC_AFF_EP_RDMAW_CORRUPT_DATA  = 0x04007,
12203     IT_ASYNC_AFF_EP_RDMAR_ACCESS_VIOLATION = 0x04008,
12204     IT_ASYNC_AFF_EP_LOCAL_ACCESS_VIOLATION = 0x04020,
12205     IT_ASYNC_AFF_EP_L_ACCESS_VIOLATION  = 0x04020,
12206     IT_ASYNC_AFF_EP_L_RECV_ACCESS_VIOLATION = 0x04021,
12207     IT_ASYNC_AFF_EP_L_IRRQ_ACCESS_VIOLATION = 0x04022,
12208     IT_ASYNC_AFF_EP_L_TRANSPORT_ERROR   = 0x04023,
12209     IT_ASYNC_AFF_EP_L_LLP_ERROR         = 0x04024,
12210     IT_ASYNC_AFF_EP_R_ERROR             = 0x04040,
12211     IT_ASYNC_AFF_EP_R_ACCESS_VIOLATION  = 0x04041,
12212     IT_ASYNC_AFF_EP_R_RECV_ACCESS_VIOLATION = 0x04042,
12213     IT_ASYNC_AFF_EP_R_RECV_LENGTH_ERROR  = 0x04043,

```

```

12214     IT_ASYNC_AFF_EP_SOFT_HI_WATERMARK           = 0x04060,
12215     IT_ASYNC_AFF_SRQ_LOW_WATERMARK             = 0x04100,
12216
12217     /* Event Stream for Unaffiliated Asynchronous Events */
12218     IT_ASYNC_UNAFF_EVENT_STREAM                 = 0x08000,
12219     /* 0x08001 is deprecated */
12220     IT_ASYNC_UNAFF_SPIGOT_ONLINE                = 0x08002,
12221     IT_ASYNC_UNAFF_SPIGOT_OFFLINE              = 0x08003,
12222     IT_ASYNC_UNAFF_SEVD_ENQUEUE_FAILURE        = 0x08004,
12223
12224     /* Event Stream for Software Events */
12225     IT_SOFTWARE_EVENT_STREAM                   = 0x10000,
12226     IT_SOFTWARE_EVENT                         = 0x10001,
12227
12228     /* Event Stream for AEVD Notifications */
12229     IT_AEVD_NOTIFICATION_EVENT_STREAM          = 0x20000,
12230     IT_AEVD_NOTIFICATION_EVENT                = 0x20001
12231 } it_event_type_t;
12232
12233 typedef struct {
12234     it_event_type_t  event_number;
12235     it_evd_handle_t  aevd;
12236     it_evd_handle_t  sevd;
12237 } it_aevd_notification_event_t;
12238
12239 typedef struct {
12240     it_event_type_t  event_number;
12241     it_evd_handle_t  evd;
12242
12243     union {
12244         it_evd_handle_t  sevd;
12245         it_ep_handle_t   ep;
12246         it_srq_handle_t  srq;
12247     } cause;
12248 } it_affiliated_event_t;
12249
12250 #define IT_MAX_PRIV_DATA 256
12251
12252 typedef enum {
12253     IT_CN_REJ_OTHER           = 0,
12254     IT_CN_REJ_TIMEOUT        = 1,
12255     IT_CN_REJ_BAD_PATH       = 2,
12256     IT_CN_REJ_STALE_CONN     = 3,
12257     IT_CN_REJ_BAD_ORD        = 4,
12258     IT_CN_REJ_RESOURCES      = 5,
12259     IT_CN_REJ_BAD_CONN_PARMS = 6,
12260 } it_conn_reject_code_t;
12261
12262 typedef struct {
12263     it_event_type_t          event_number;
12264     it_evd_handle_t          evd;
12265     it_cn_est_identifier_t   cn_est_id;
12266     it_ep_handle_t           ep;
12267     uint32_t                 rdma_read_ird;

```

```

12268         uint32_t             rdma_read_ord;
12269         it_path_t           dst_path;
12270         it_conn_reject_code_t reject_reason_code;
12271         unsigned char       private_data[IT_MAX_PRIV_DATA];
12272         it_boolean_t        private_data_present;
12273     } it_connection_event_t;
12274
12275     typedef enum {
12276         IT_UD_SVC_EP_INFO_VALID           = 0,
12277         IT_UD_SVC_ID_NOT_SUPPORTED       = 1,
12278         IT_UD_SVC_REQ_REJECTED           = 2,
12279         IT_UD_NO_EP_AVAILABLE           = 3,
12280         IT_UD_REQ_REDIRECTED             = 4
12281     } it_ud_svc_req_status_t;
12282
12283     typedef struct {
12284         it_event_type_t             event_number;
12285         it_evd_handle_t            evd;
12286         it_ud_svc_req_handle_t     ud_svc;
12287         it_ud_svc_req_status_t     status;
12288         it_remote_ep_info_t        ep_info;
12289         it_path_t                  dst_path;
12290         unsigned char              private_data[IT_MAX_PRIV_DATA];
12291         it_boolean_t               private_data_present;
12292     } it_ud_svc_reply_event_t;
12293
12294     typedef struct {
12295         it_event_type_t             event_number;
12296         it_evd_handle_t            evd;
12297         it_cn_est_identifier_t     cn_est_id;
12298         it_conn_qual_t            conn_qual;
12299         it_net_addr_t             source_addr;
12300         size_t                     spigot_id;
12301         uint32_t                   max_message_size;
12302         uint32_t                   rdma_read_ird;
12303         uint32_t                   rdma_read_ord;
12304         unsigned char              private_data[IT_MAX_PRIV_DATA];
12305         it_boolean_t               private_data_present;
12306     } it_conn_request_event_t;
12307
12308     typedef struct {
12309         it_event_type_t             event_number;
12310         it_evd_handle_t            evd;
12311         it_ud_svc_req_identifier_t ud_svc_req_id;
12312         it_conn_qual_t            conn_qual;
12313         it_net_addr_t             source_addr;
12314         size_t                     spigot_id;
12315         unsigned char              private_data[IT_MAX_PRIV_DATA];
12316         it_boolean_t               private_data_present;
12317     } it_ud_svc_request_event_t;
12318
12319     typedef enum {
12320         IT_UD_IB_GRP_PRESENT = 0x01
12321     } it_dto_ud_flags_t;

```

```

12322
12323     typedef struct {
12324         it_event_type_t  event_number;
12325         it_evd_handle_t  evd;
12326         it_ep_handle_t   ep;
12327         it_dto_cookie_t  cookie;
12328         it_dto_status_t  dto_status;
12329         uint32_t         transferred_length;
12330     } it_dto_cmpl_event_t;
12331
12332     typedef struct {
12333         it_event_type_t  event_number;
12334         it_evd_handle_t  evd;
12335         it_ep_handle_t   ep;
12336         it_dto_cookie_t  cookie;
12337         it_dto_status_t  dto_status;
12338         uint32_t         transferred_length;
12339         it_dto_ud_flags_t flags;
12340         it_ud_ep_id_t    ud_ep_id;
12341         it_path_t        src_path;
12342     } it_all_dto_cmpl_event_t;
12343
12344     typedef struct {
12345         it_event_type_t  event_number;
12346         it_evd_handle_t  evd;
12347         void              *data;
12348     } it_software_event_t;
12349
12350     typedef struct {
12351         it_event_type_t  event_number;
12352         it_evd_handle_t  evd;
12353         it_ia_handle_t   ia;
12354
12355         size_t           spigot_id;
12356     } it_unaffiliated_event_t;
12357
12358     typedef struct {
12359         it_event_type_t  event_number;
12360         it_evd_handle_t  evd;
12361     } it_any_event_t;
12362
12363     typedef union
12364     {
12365         /*
12366          * The following two union elements are
12367          * available for programming convenience.
12368          *
12369          * The event_number may be used to determine the
12370          * it_event_type_t of any Event. it_any_event_t
12371          * allows the EVD to be determined as well.
12372          */
12373         it_event_type_t  event_number;
12374         it_any_event_t  any;
12375

```

```

12376      /*
12377      * The remaining union elements correspond to
12378      * the various it_event_type_t types.
12379      */
12380
12381      /*
12382      * The following two Event structures
12383      * support the IT.DTO.EVENT.STREAM Event Stream.
12384      *
12385      * it_dto_cmpl_event_t supports
12386      * only the following events:
12387      *     IT.DTO.SEND.CMPL.EVENT
12388      *     IT.DTO.RC.RECV.CMPL.EVENT
12389      *     IT.DTO.RDMA.WRITE.CMPL.EVENT
12390      *     IT.DTO.RDMA.READ.CMPL.EVENT
12391      *     IT.RMR.BIND.CMPL.EVENT = IT.RMR.LINK.CMPL.EVENT
12392      *
12393      * it_all_dto_cmpl_event_t supports all
12394      * possible DTO and RMR events:
12395      *     IT.DTO.SEND.CMPL.EVENT
12396      *     IT.DTO.RC.RECV.CMPL.EVENT
12397      *     IT.DTO.UD.RECV.CMPL.EVENT
12398      *     IT.DTO.RDMA.WRITE.CMPL.EVENT
12399      *     IT.DTO.RDMA.READ.CMPL.EVENT
12400      *     IT.RMR.BIND.CMPL.EVENT = IT.RMR.LINK.CMPL.EVENT
12401      */
12402      it_dto_cmpl_event_t      dto_cmpl;
12403      it_all_dto_cmpl_event_t  all_dto_cmpl;
12404
12405      /*
12406      * The following two Event structures
12407      * support the IT.CM.REQ.EVENT.STREAM Event
12408      * stream:
12409      *
12410      * it_conn_request_event_t supports:
12411      *     IT.CM.REQ.CONN.REQUEST.EVENT
12412      *
12413      * it_ud_svc_request_event_t supports:
12414      *     IT.CM.REQ.UD.SERVICE.REQUEST.EVENT
12415      */
12416      it_conn_request_event_t  conn_req;
12417      it_ud_svc_request_event_t  ud_svc_request;
12418
12419      /*
12420      * The following two Event structures
12421      * support the IT.CM.MSG.EVENT.STREAM Event
12422      * stream:
12423      *
12424      * it_connection_event_t supports:
12425      *     IT.CM.MSG.CONN.ACCEPT.ARRIVAL.EVENT
12426      *     IT.CM.MSG.CONN.ESTABLISHED.EVENT
12427      *     IT.CM.MSG.CONN.PEER.REJECT.EVENT
12428      *     IT.CM.MSG.CONN.NONPEER.REJECT.EVENT
12429      *     IT.CM.MSG.CONN.DISCONNECT.EVENT

```

```

12430     *      IT_CM_MSG_CONN_BROKEN_EVENT
12431     *
12432     * it_ud_svc_reply_event_t supports:
12433     *      IT_CM_MSG_UD_SERVICE_REPLY_EVENT
12434     */
12435     it_connection_event_t    conn;
12436     it_ud_svc_reply_event_t  ud_svc_reply;
12437
12438     /*
12439     * it_affiliated_event_t supports
12440     * the following Event Stream:
12441     *      IT_ASYNC_AFF_EVENT_STREAM
12442     */
12443     it_affiliated_event_t    aff_async;
12444
12445     /*
12446     * it_unaffiliated_event_t supports
12447     * the following Event Stream:
12448     *      IT_ASYNC_UNAFF_EVENT_STREAM
12449     */
12450     it_unaffiliated_event_t  unaff_async;
12451
12452     /*
12453     * it_software_event_t supports
12454     * the following Event Stream:
12455     *      IT_SOFTWARE_EVENT_STREAM
12456     */
12457     it_software_event_t     sw;
12458
12459     /*
12460     * it_aevd_notification_event_t supports
12461     * the following Event Stream:
12462     *      IT_AEVD_NOTIFICATION_EVENT_STREAM
12463     */
12464     it_aevd_notification_event_t  aevd_notify;
12465 } it_event_t;
12466
12467 typedef enum
12468 {
12469     IT_EP_STATE_UNCONNECTED           = 0,
12470     IT_EP_STATE_ACTIVE1_CONNECTION_PENDING = 1,
12471     IT_EP_STATE_ACTIVE2_CONNECTION_PENDING = 2,
12472     IT_EP_STATE_PASSIVE_CONNECTION_PENDING = 3,
12473     IT_EP_STATE_CONNECTED             = 4,
12474     IT_EP_STATE_NONOPERATIONAL        = 5,
12475     IT_EP_STATE_PASSIVE_WAIT_RDMA_TRANS_REQ = 6
12476 } it_ep_state_rc_t;
12477
12478 typedef enum
12479 {
12480     IT_EP_STATE_UD_NONOPERATIONAL = 0,
12481     IT_EP_STATE_UD_OPERATIONAL   = 1
12482 } it_ep_state_ud_t;
12483

```

```

12484 typedef union
12485 {
12486     it_ep_state_rc_t rc;
12487     it_ep_state_ud_t ud;
12488 } it_ep_state_t;
12489
12490 typedef struct
12491 {
12492     it_addr_handle_t addr;
12493     it_remote_ep_info_t ep_info;
12494 } it_ib_ud_addr_t;
12495
12496 typedef enum
12497 {
12498     IT_DG_TYPE_IB_UD
12499 } it_dg_type_t;
12500
12501 typedef struct
12502 {
12503     it_dg_type_t type; /* IT_DG_TYPE_IB_UD */
12504     union {
12505         it_ib_ud_addr_t ud;
12506     } addr;
12507 } it_dg_remote_ep_addr_t;
12508
12509 typedef enum {
12510     IT_AH_PATH_COMPLETE = 0x1
12511 } it_ah_flags_t;
12512
12513 typedef enum {
12514     IT_ADDR_PARAM_ALL = 0x0001,
12515     IT_ADDR_PARAM_IA = 0x0002,
12516     IT_ADDR_PARAM_PZ = 0x0004,
12517     IT_ADDR_PARAM_PATH = 0x0008
12518 } it_addr_param_mask_t;
12519
12520 typedef struct {
12521     it_ia_handle_t ia; /* IT_ADDR_PARAM_IA */
12522     it_pz_handle_t pz; /* IT_ADDR_PARAM_PZ */
12523     it_path_t path; /* IT_ADDR_PARAM_PATH */
12524 } it_addr_param_t;
12525
12526 typedef struct {
12527
12528     /* Remote CM Response Timeout, as defined in the REQ
12529     message for the IB CM protocol */
12530     uint8_t remote_cm_timeout : 5;
12531
12532     /* Local CM Response Timeout, as defined in the REQ
12533     message for the IB CM protocol */
12534     uint8_t local_cm_timeout : 5;
12535
12536     /* Retry Count, as defined in the REQ message for the
12537     IB CM protocol */

```

```

12538         uint8_t  retry_count : 3;
12539
12540         /* RNR Retry Count, as defined in the REQ message for
12541            the IB CM protocol */
12542         uint8_t  rnr_retry_count : 3;
12543
12544         /* Max CM retries, as defined in the REQ message for
12545            the IB CM protocol */
12546         uint8_t  max_cm_retries : 4;
12547
12548         /* Local ACK Timeout, as defined in the REQ message
12549            for the IB CM protocol */
12550         uint8_t  local_ack_timeout : 5;
12551
12552     } it_ib_conn_attributes_t;
12553
12554     typedef struct {
12555
12556         /* VIA currently has no transport-specific connection
12557            attributes. A dummy entry is defined to allow ANSI
12558            compilation. */
12559         void  *unused;
12560
12561     } it_via_conn_attributes_t;
12562
12563     typedef struct {
12564
12565         /* iWARP currently has no transport-specific connection
12566            attributes. A dummy entry is defined to allow ANSI
12567            compilation. */
12568         void  *unused;
12569
12570     } it_iwarp_conn_attributes_t;
12571
12572     typedef union {
12573         it_ib_conn_attributes_t    ib;
12574         it_via_conn_attributes_t    via;
12575         it_iwarp_conn_attributes_t  iwarp;
12576     } it_conn_attributes_t;
12577
12578     typedef enum {
12579         IT_CONNECT_FLAG_TWO_WAY      = 0x0001,
12580         IT_CONNECT_FLAG_THREE_WAY    = 0x0002,
12581         IT_CONNECT_SUPPRESS_IRD_ORD  = 0x0004
12582     } it_cn_est_flags_t;
12583
12584     typedef struct {
12585         it_ia_handle_t    ia;          /* IT_EP_PARAM_IA */
12586         size_t            spigot_id;   /* IT_EP_PARAM_SPIGOT */
12587         it_ep_state_t     ep_state;    /* IT_EP_PARAM_STATE */
12588         it_transport_service_type_t service_type;
12589                                     /* IT_EP_PARAM_SERV_TYPE */
12590         it_path_t         dst_path;    /* IT_EP_PARAM_PATH */
12591         it_pz_handle_t    pz;          /* IT_EP_PARAM_PZ */

```

```

12592         it_evd_handle_t         request_sevd;
12593                                 /* IT_EP_PARAM_REQ_SEVD */
12594         it_evd_handle_t         recv_sevd; /* IT_EP_PARAM_RECV_SEVD */
12595         it_evd_handle_t         connect_sevd;
12596                                 /* IT_EP_PARAM_CONN_SEVD */
12597         it_ep_attributes_t       attr;     /* see it_ep_attributes_t
12598                                           for mask flags for
12599                                           attr */
12600     } it_ep_param_t;
12601
12602     typedef enum {
12603         IT_EP_NO_FLAG           = 0x00,
12604         IT_EP_REUSEADDR        = 0x01,
12605         IT_EP_SRQ               = 0x02
12606     } it_ep_rc_creation_flags_t;
12607
12608     #define IT_THRESHOLD_DISABLE 0
12609
12610     typedef enum {
12611         IT_EVD_DEQUEUE_NOTIFICATIONS = 0x01,
12612         IT_EVD_CREATE_FD              = 0x02,
12613         IT_EVD_OVERFLOW_DEFAULT       = 0x04,
12614         IT_EVD_OVERFLOW_NOTIFY        = 0x08,
12615         IT_EVD_OVERFLOW_AUTO_RESET    = 0x10
12616     } it_evd_flags_t;
12617
12618     typedef enum {
12619         IT_EVD_PARAM_ALL              = 0x000001,
12620         IT_EVD_PARAM_IA               = 0x000002,
12621         IT_EVD_PARAM_EVENT_NUMBER     = 0x000004,
12622         IT_EVD_PARAM_FLAG             = 0x000008,
12623         IT_EVD_PARAM_QUEUE_SIZE       = 0x000010,
12624         IT_EVD_PARAM_THRESHOLD        = 0x000020,
12625         IT_EVD_PARAM_AEVD_HANDLE      = 0x000040,
12626         IT_EVD_PARAM_FD               = 0x000080,
12627         IT_EVD_PARAM_BOUND            = 0x000100,
12628         IT_EVD_PARAM_ENABLED          = 0x000200,
12629         IT_EVD_PARAM_OVERFLOWED       = 0x000400
12630     } it_evd_param_mask_t;
12631
12632     typedef struct {
12633         it_ia_handle_t   ia;           /* IT_EVD_PARAM_IA */
12634         it_event_type_t  event_number; /* IT_EVD_PARAM_EVENT_NUMBER */
12635         it_evd_flags_t   evd_flag;    /* IT_EVD_PARAM_FLAG */
12636         size_t           sev_queue_size; /* IT_EVD_PARAM_QUEUE_SIZE */
12637         size_t           sev_threshold; /* IT_EVD_PARAM_THRESHOLD */
12638         it_evd_handle_t  aevd;        /* IT_EVD_PARAM_AEVD_HANDLE */
12639         int              fd;          /* IT_EVD_PARAM_FD */
12640         it_boolean_t     evd_bound;   /* IT_EVD_PARAM_BOUND */
12641         it_boolean_t     evd_enabled; /* IT_EVD_PARAM_ENABLED */
12642         it_boolean_t     evd_overflowed; /* IT_EVD_PARAM_OVERFLOWED */
12643     } it_evd_param_t;
12644
12645     typedef struct {

```

```

12646
12647      /* Most recent major version number of the IT-API supported by the
12648         Interface */
12649      uint32_t major_version;
12650
12651      /* Most recent minor version number of the IT-API supported by the
12652         Interface */
12653      uint32_t minor_version;
12654
12655      /* The transport that the Interface uses, as defined in
12656         it_ia_info_t. */
12657      it_transport_type_t transport_type;
12658
12659      /* The name of the Interface, suitable for input to it_ia_create.
12660         The name is a string of maximum length IT_INTERFACE_NAME_SIZE,
12661         including the terminating NULL character. */
12662      char name[IT_INTERFACE_NAME_SIZE];
12663
12664 } it_interface_t;
12665
12666 typedef enum {
12667     IT_LISTEN_NO_FLAG           = 0x0000,
12668     IT_LISTEN_CONN_QUAL_INPUT  = 0x0001,
12669     IT_LISTEN_SUPPRESS_IRD_ORD = 0x0002
12670 } it_listen_flags_t;
12671
12672 typedef enum {
12673     IT_LISTEN_PARAM_ALL           = 0x0001,
12674     IT_LISTEN_PARAM_IA_HANDLE    = 0x0002,
12675     IT_LISTEN_PARAM_SPIGOT_ID    = 0x0004,
12676     IT_LISTEN_PARAM_CONNECT_EVD  = 0x0008,
12677     IT_LISTEN_PARAM_CONN_QUAL    = 0x0010
12678 } it_listen_param_mask_t;
12679
12680 typedef struct {
12681     it_ia_handle_t   ia_handle;      /* IT_LISTEN_PARAM_IA_HANDLE */
12682     size_t           spigot_id;     /* IT_LISTEN_PARAM_SPIGOT_ID */
12683     it_evd_handle_t connect_evd;    /* IT_LISTEN_PARAM_CONNECT_EVD */
12684     it_conn_qual_t  connect_qual;   /* IT_LISTEN_PARAM_CONN_QUAL */
12685 } it_listen_param_t;
12686
12687 typedef enum {
12688     IT_LMR_PARAM_ALL           = 0x000001,
12689     IT_LMR_PARAM_IA           = 0x000002,
12690     IT_LMR_PARAM_PZ           = 0x000004,
12691     IT_LMR_PARAM_ADDR         = 0x000008,
12692     IT_LMR_PARAM_LENGTH       = 0x000010,
12693     IT_LMR_PARAM_MEM_PRIV     = 0x000020,
12694     IT_LMR_PARAM_FLAG         = 0x000040,
12695     IT_LMR_PARAM_SHARED_ID    = 0x000080,
12696     IT_LMR_PARAM_RMR_CONTEXT  = 0x000100,
12697     IT_LMR_PARAM_ACTUAL_ADDR  = 0x000200,
12698     IT_LMR_PARAM_ACTUAL_LENGTH = 0x000400,
12699     IT_LMR_PARAM_ADDR_MODE    = 0x000800

```

```

12700     } it_lmr_param_mask_t;
12701
12702     typedef struct {
12703         it_ia_handle_t    ia;           /* IT_LMR_PARAM_IA */
12704         it_pz_handle_t    pz;           /* IT_LMR_PARAM_PZ */
12705         void               *addr;       /* IT_LMR_PARAM_ADDR */
12706         it_length_t       length;       /* IT_LMR_PARAM_LENGTH */
12707         it_mem_priv_t     privs;        /* IT_LMR_PARAM_MEM_PRIV */
12708         it_lmr_flag_t     flags;        /* IT_LMR_PARAM_FLAG */
12709         uint32_t          shared_id;    /* IT_LMR_PARAM_SHARED_ID */
12710         it_rmr_context_t  rmr_context;  /* IT_LMR_PARAM_RMR_CONTEXT */
12711         void               *actual_addr; /* IT_LMR_PARAM_ACTUAL_ADDR */
12712         it_length_t       actual_length; /* IT_LMR_PARAM_ACTUAL_LENGTH */
12713         it_addr_mode_t    addr_mode;    /* IT_LMR_PARAM_ADDR_MODE */
12714     } it_lmr_param_t;
12715
12716     typedef uint64_t it_rdma_addr_t;
12717
12718     typedef enum {
12719         IT_PZ_PARAM_ALL    = 0x01,
12720         IT_PZ_PARAM_IA    = 0x02
12721     } it_pz_param_mask_t;
12722
12723     typedef struct {
12724         it_ia_handle_t ia; /* IT_PZ_PARAM_IA */
12725     } it_pz_param_t;
12726
12727     typedef enum {
12728         IT_RMR_PARAM_ALL          = 0x000001,
12729         IT_RMR_PARAM_IA          = 0x000002,
12730         IT_RMR_PARAM_PZ          = 0x000004,
12731         IT_RMR_PARAM_LINKED      = 0x000008,
12732         IT_RMR_PARAM_BOUND       = 0x000008,
12733                                     /* deprecated by IT_RMR_PARAM_LINKED */
12734         IT_RMR_PARAM_LMR         = 0x000010,
12735         IT_RMR_PARAM_ADDR        = 0x000020,
12736         IT_RMR_PARAM_LENGTH      = 0x000040,
12737         IT_RMR_PARAM_MEM_PRIV    = 0x000080,
12738         IT_RMR_PARAM_RMR_CONTEXT = 0x000100,
12739         IT_RMR_PARAM_TYPE        = 0x000200,
12740         IT_RMR_PARAM_ADDR_MODE   = 0x000400
12741     } it_rmr_param_mask_t;
12742
12743     /* Need to use "bound" rather than "linked" in IT-API 1.0 */
12744     #ifndef ITAPI_ENABLE_V20_BINDINGS
12745
12746     typedef struct {
12747         it_ia_handle_t    ia;           /* IT_RMR_PARAM_IA */
12748         it_pz_handle_t    pz;           /* IT_RMR_PARAM_PZ */
12749         it_boolean_t      bound;        /* IT_RMR_PARAM_LINKED */
12750         it_lmr_handle_t   lmr;          /* IT_RMR_PARAM_LMR */
12751         void *            addr;         /* IT_RMR_PARAM_ADDR */
12752         it_length_t       length;       /* IT_RMR_PARAM_LENGTH */
12753         it_mem_priv_t     privs;        /* IT_RMR_PARAM_MEM_PRIV */

```

```

12754         it_rmr_context_t  rmr_context; /* IT_RMR_PARAM_RMR_CONTEXT */
12755         it_rmr_type_t      type;        /* IT_RMR_PARAM_TYPE */
12756         it_addr_mode_t     addr_mode;   /* IT_RMR_PARAM_ADDR_MODE */
12757     } it_rmr_param_t;
12758
12759     #else
12760
12761     typedef struct {
12762         it_ia_handle_t      ia;          /* IT_RMR_PARAM_IA */
12763         it_pz_handle_t      pz;          /* IT_RMR_PARAM_PZ */
12764         it_boolean_t        linked;     /* IT_RMR_PARAM_LINKED */
12765         it_lmr_handle_t     lmr;        /* IT_RMR_PARAM_LMR */
12766         void *               addr;       /* IT_RMR_PARAM_ADDR */
12767         it_length_t         length;     /* IT_RMR_PARAM_LENGTH */
12768         it_mem_priv_t        privs;      /* IT_RMR_PARAM_MEM_PRIV */
12769         it_rmr_context_t    rmr_context; /* IT_RMR_PARAM_RMR_CONTEXT */
12770         it_rmr_type_t        type;       /* IT_RMR_PARAM_TYPE */
12771         it_addr_mode_t      addr_mode;   /* IT_RMR_PARAM_ADDR_MODE */
12772     } it_rmr_param_t;
12773
12774     #endif /* #ifndef ITAPI_ENABLE_V20_BINDINGS */
12775
12776     typedef enum {
12777         IT_SC_DEFAULT        = 0x0000,
12778         IT_SC_NO_REQ_REP    = 0x0001,
12779     } it_sc_flags_t;
12780
12781     typedef enum {
12782         IT_SRQ_PARAM_ALL     = 0x000001,
12783         IT_SRQ_PARAM_IA     = 0x000002,
12784         IT_SRQ_PARAM_PZ     = 0x000004,
12785         IT_SRQ_PARAM_MAX_RECV_DTO = 0x000008,
12786         IT_SRQ_PARAM_MAX_RECV_SEG = 0x000010,
12787         IT_SRQ_PARAM_LOW_WATERMARK = 0x000020
12788     } it_srq_param_mask_t;
12789
12790     typedef struct {
12791         it_ia_handle_t      ia;          /* IT_SRQ_PARAM_IA */
12792         it_pz_handle_t      pz;          /* IT_SRQ_PARAM_PZ */
12793         size_t               max_recv_dtos; /* IT_SRQ_PARAM_MAX_RECV_DTO */
12794         size_t               max_recv_segs; /* IT_SRQ_PARAM_MAX_RECV_SEG */
12795         size_t               low_watermark; /* IT_SRQ_PARAM_LOW_WATERMARK */
12796     } it_srq_param_t;
12797
12798     typedef enum {
12799         IT_UD_PARAM_ALL     = 0x00000001,
12800         IT_UD_PARAM_IA_HANDLE = 0x00000002,
12801         IT_UD_PARAM_REQ_ID   = 0x00000004,
12802         IT_UD_PARAM_REPLY_EVD = 0x00000008,
12803         IT_UD_PARAM_CONN_QUAL = 0x00000010,
12804         IT_UD_PARAM_DEST_PATH = 0x00000020,
12805         IT_UD_PARAM_PRIV_DATA = 0x00000040,
12806         IT_UD_PARAM_PRIV_DATA_LENGTH = 0x00000080
12807     } it_ud_svc_req_param_mask_t;

```

```

12808
12809     typedef struct {
12810         it_ia_handle_t    ia;                /* IT_UD_PARAM_IA_HANDLE */
12811         uint32_t          request_id;        /* IT_UD_PARAM_REQ_ID */
12812         it_evd_handle_t   reply_evd;        /* IT_UD_PARAM_REPLY_EVD */
12813         it_conn_qual_t    conn_qual;        /* IT_UD_PARAM_CONN_QUAL */
12814         it_path_t         destination_path; /* IT_UD_PARAM_DEST_PATH */
12815         unsigned char     private_data[IT_MAX_PRIV_DATA];
12816                                     /* IT_UD_PARAM_PRIV_DATA */
12817         size_t            private_data_length;
12818                                     /* IT_UD_PARAM_PRIV_DATA_LENGTH */
12819     } it_ud_svc_req_param_t;
12820
12821     /* prototypes */
12822     #ifndef ITAPI_ENABLE_V20_BINDINGS
12823     /*
12824      * Backwards compatibility mode:
12825      * For functions whose signature changed from v1.0 to v2.0,
12826      * <it_api.h> converts v1.0 function names to explicit v1.0
12827      * function names. Functions such as it_lmr_create continue
12828      * to use v1.0 signatures.
12829      */
12830     #define it_lmr_create    it_lmr_create10
12831     #define it_rmr_create    it_rmr_create10
12832     #define it_rmr_bind     it_rmr_bind10
12833
12834     #else
12835     /*
12836      * Full v2.0 functionality:
12837      * For functions whose signature changed from v1.0 to v2.0,
12838      * <it_api.h> converts v1.0 function names to explicit v2.0
12839      * function names. Functions such as it_lmr_create use the
12840      * new v2.0 signatures.
12841      */
12842     #define it_lmr_create    it_lmr_create20
12843     #define it_rmr_create    it_rmr_create20
12844     #define it_rmr_bind     it_rmr_link
12845
12846     #endif
12847
12848     it_status_t it_address_handle_create(
12849         IN          it_pz_handle_t    pz_handle,
12850         IN  const   it_path_t         *destination_path,
12851         IN          it_ah_flags_t     ah_flags,
12852         OUT         it_addr_handle_t  *addr_handle
12853     );
12854     it_status_t it_address_handle_free(
12855         IN          it_addr_handle_t  addr_handle
12856     );
12857     it_status_t it_address_handle_modify(
12858         IN          it_addr_handle_t  addr_handle,
12859         IN          it_addr_param_mask_t mask,
12860         IN  const   it_addr_param_t  *params
12861

```

```

12862 );
12863 it_status_t it_address_handle_query(
12864     IN          it_addr_handle_t      addr_handle,
12865     IN          it_addr_param_mask_t  mask,
12866     OUT         it_addr_param_t       *params
12867 );
12868 it_status_t it_convert_net_addr(
12869     IN  const  it_net_addr_t      *source_addr,
12870     IN          it_net_addr_type_t  addr_type,
12871     OUT         it_net_addr_t      *destination_addr
12872 );
12873 it_status_t it_ep_accept(
12874     IN          it_ep_handle_t      ep_handle,
12875     IN          it_cn_est_identifier_t  cn_est_id,
12876     IN  const  unsigned char       *private_data,
12877     IN  size_t   private_data_length
12878 );
12879 it_status_t it_ep_connect(
12880     IN          it_ep_handle_t      ep_handle,
12881     IN  const  it_path_t*          path,
12882     IN  const  it_conn_attributes_t*  conn_attr,
12883     IN  const  it_conn_qual_t*       connect_qual,
12884     IN          it_cn_est_flags_t    cn_est_flags,
12885     IN  const  unsigned char*       private_data,
12886     IN          size_t               private_data_length
12887 );
12888 it_status_t it_ep_disconnect (
12889     IN          it_ep_handle_t      ep_handle,
12890     IN  const  unsigned char       *private_data,
12891     IN          size_t               private_data_length
12892 );
12893 it_status_t it_ep_free(
12894     IN          it_ep_handle_t      ep_handle
12895 );
12896 it_status_t it_ep_modify(
12897     IN          it_ep_handle_t      ep_handle,
12898     IN          it_ep_param_mask_t  mask,
12899     IN  const  it_ep_attributes_t  *ep_attr
12900 );
12901 it_status_t it_ep_query(
12902     IN          it_ep_handle_t      ep_handle,
12903     IN          it_ep_param_mask_t  mask,
12904     OUT         it_ep_param_t       *params
12905 );
12906 it_status_t it_ep_rc_create (
12907     IN          it_pz_handle_t      pz_handle,
12908     IN          it_evd_handle_t      request_sevd_handle,
12909     IN          it_evd_handle_t      recv_sevd_handle,
12910     IN          it_evd_handle_t      connect_sevd_handle,
12911     IN          it_ep_rc_creation_flags_t  flags,
12912     IN  const  it_ep_attributes_t  *ep_attr,
12913     OUT         it_ep_handle_t      *ep_handle
12914 );
12915 it_status_t it_ep_reset(

```

```

12916         IN          it_ep_handle_t  ep_handle
12917     );
12918     it_status_t it_ep_ud_create (
12919         IN          it_pz_handle_t    pz_handle,
12920         IN          it_evd_handle_t   request_sevd_handle,
12921         IN          it_evd_handle_t   recv_sevd_handle,
12922         IN  const   it_ep_attributes_t *ep_attr,
12923         IN          size_t             spigot_id,
12924         OUT         it_ep_handle_t    *ep_handle
12925     );
12926     it_status_t evd_create (
12927         IN  it_ia_handle_t   ia_handle,
12928         IN  it_event_type_t  event_number,
12929         IN  it_evd_flags_t   evd_flag,
12930         IN  size_t           sevd_queue_size,
12931         IN  size_t           sevd_threshold,
12932         IN  it_evd_handle_t  aevd_handle,
12933         OUT it_evd_handle_t  *evd_handle,
12934         OUT int              *fd
12935     );
12936     it_status_t it_evd_dequeue(
12937         IN  it_evd_handle_t  evd_handle,
12938         OUT it_event_t       *event
12939     );
12940     it_status_t it_evd_free(
12941         IN  it_evd_handle_t  evd_handle
12942     );
12943     it_status_t it_evd_modify(
12944         IN          it_evd_handle_t    evd_handle,
12945         IN          it_evd_param_mask_t mask,
12946         IN  const   it_evd_param_t    *params
12947     );
12948     it_status_t it_evd_post_se(
12949         IN          it_evd_handle_t  evd_handle,
12950         IN  const   void             *event
12951     );
12952     it_status_t it_evd_query(
12953         IN  it_evd_handle_t    evd_handle,
12954         IN  it_evd_param_mask_t mask,
12955         OUT it_evd_param_t     *params
12956     );
12957     it_status_t it_evd_wait(
12958         IN  it_evd_handle_t  evd_handle,
12959         IN  uint64_t         timeout,
12960         OUT it_event_t       *event,
12961         OUT size_t           *nmore
12962     );
12963     it_status_t it_get_consumer_context(
12964         IN  it_handle_t   handle,
12965         OUT it_context_t  *context
12966     );
12967     it_status_t it_get_handle_type(
12968         IN  it_handle_t   handle,
12969         OUT it_handle_type_enum_t *type_of_handle

```

```

12970     );
12971     it_status_t it_get_pathinfo(
12972         IN         it_ia_handle_t   ia_handle,
12973         IN         size_t           spigot_id,
12974         IN const   it_net_addr_t    *net_addr,
12975         IN OUT    size_t           *num_paths,
12976         OUT       size_t           *total_paths,
12977         OUT       it_path_t        *paths
12978     );
12979     it_status_t it_handoff(
12980         IN const   it_conn_qual_t    *conn_qual,
12981         IN         size_t           spigot_id,
12982         IN         it_cn_est_identifier_t  cn_est_id
12983     );
12984     uint64_t it_hton64(
12985         uint64_t hostint
12986     );
12987     uint64_t it_ntoh64(
12988         uint64_t hostint
12989     );
12990     it_status_t it_ia_create(
12991         IN const   char             *name,
12992         IN         uint32_t         major_version,
12993         IN         uint32_t         minor_version,
12994         OUT       it_ia_handle_t    *ia_handle
12995     );
12996     it_status_t it_ia_free(
12997         IN         it_ia_handle_t    ia_handle
12998     );
12999     void it_ia_info_free(
13000         IN         it_ia_info_t      *ia_info
13001     );
13002     it_status_t it_ia_query(
13003         IN   it_ia_handle_t   ia_handle,
13004         OUT  it_ia_info_t     **ia_info
13005     );
13006     void it_interface_list(
13007         OUT   it_interface_t   *interfaces,
13008         IN OUT size_t         *num_interfaces,
13009         IN OUT size_t         *total_interfaces
13010     );
13011     it_status_t it_listen_create(
13012         IN   it_ia_handle_t     ia_handle,
13013         IN   size_t             spigot_id,
13014         IN   it_evd_handle_t    connect_evd,
13015         IN   it_listen_flags_t  flags,
13016         IN OUT it_conn_qual_t    *conn_qual,
13017         OUT   it_listen_handle_t *listen_handle
13018     );
13019     it_status_t it_listen_free(
13020         IN   it_listen_handle_t listen_handle
13021     );
13022     it_status_t it_listen_query(
13023         IN   it_listen_handle_t listen_handle,

```

```

13024     IN   it_listen_param_mask_t  mask,
13025     OUT  it_listen_param_t       *params
13026 );
13027 /*
13028     it_lmr_create10 is provided for backwards-compatibility
13029     and may be dropped in a future IT-API version.
13030
13031     Calls with a suffix "10" can be typically implemented through
13032     direct inlining to the corresponding calls with suffix "20",
13033     providing default arguments as necessary.
13034 */
13035 it_status_t it_lmr_create10(
13036     IN   it_pz_handle_t   pz_handle,
13037     IN   void              *addr,
13038     IN   it_length_t      length,
13039     IN   it_mem_priv_t    privs,
13040     IN   it_lmr_flag_t    flags,
13041     IN   uint32_t         shared_id,
13042     OUT  it_lmr_handle_t  *lmr_handle,
13043     IN OUT it_rmr_context_t *rmr_context
13044 );
13045
13046 /*
13047     it_lmr_create20 provides the v2.0 functionality
13048     and may be renamed to it_lmr_create in a future IT-API version.
13049 */
13050 it_status_t it_lmr_create20(
13051     IN   it_pz_handle_t   pz_handle,
13052     IN   void              *addr,
13053     IN   it_length_t      length,
13054     IN   it_addr_mode_t   addr_mode,
13055     IN   it_mem_priv_t    privs,
13056     IN   it_lmr_flag_t    flags,
13057     IN   uint32_t         shared_id,
13058     OUT  it_lmr_handle_t  *lmr_handle,
13059     IN OUT it_rmr_context_t *rmr_context
13060 );
13061 it_status_t it_lmr_free(
13062     IN   it_lmr_handle_t  lmr_handle
13063 );
13064 it_status_t it_lmr_modify(
13065     IN   it_lmr_handle_t  lmr_handle,
13066     IN   it_lmr_param_mask_t mask,
13067     IN   const it_lmr_param_t *params
13068 );
13069 it_status_t it_lmr_query(
13070     IN   it_lmr_handle_t  lmr_handle,
13071     IN   it_lmr_param_mask_t mask,
13072     OUT  it_lmr_param_t   *params
13073 );
13074 it_status_t it_lmr_sync_rdma_read(
13075     IN   const it_lmr_triplet_t *local_segments,
13076     IN   size_t                 num_segments
13077 );

```

```

13078 it_status_t it_lmr_sync_rdma_write(
13079     IN const it_lmr_triplet_t *local_segments,
13080     IN     size_t             num_segments
13081 );
13082 it_rdma_addr_t it_make_rdma_addr_absolute(
13083     void *addr
13084 );
13085 it_rdma_addr_t it_make_rdma_addr_relative(
13086     it_length_t offset
13087 );
13088 it_status_t it_post_rdma_read (
13089     IN     it_ep_handle_t     ep_handle,
13090     IN const it_lmr_triplet_t *local_segments,
13091     IN     size_t             num_segments,
13092     IN     it_dto_cookie_t    cookie,
13093     IN     it_dto_flags_t     dto_flags,
13094     IN     it_rdma_addr_t     rdma_addr,
13095     IN     it_rmr_context_t    rmr_context
13096 );
13097 it_status_t it_post_rdma_read_to_rmr (
13098     IN     it_ep_handle_t     ep_handle,
13099     IN const it_rmr_triplet_t *local_segments,
13100     IN     size_t             num_segments,
13101     IN     it_dto_cookie_t    cookie,
13102     IN     it_dto_flags_t     dto_flags,
13103     IN     it_rdma_addr_t     rdma_addr,
13104     IN     it_rmr_context_t    rmr_context
13105 );
13106 it_status_t it_post_rdma_write (
13107     IN     it_ep_handle_t     ep_handle,
13108     IN const it_lmr_triplet_t *local_segments,
13109     IN     size_t             num_segments,
13110     IN     it_dto_cookie_t    cookie,
13111     IN     it_dto_flags_t     dto_flags,
13112     IN     it_rdma_addr_t     rdma_addr,
13113     IN     it_rmr_context_t    rmr_context
13114 );
13115 it_status_t it_post_recv(
13116     IN     it_handle_t        handle,
13117     IN const it_lmr_triplet_t *local_segments,
13118     IN     size_t             num_segments,
13119     IN     it_dto_cookie_t    cookie,
13120     IN     it_dto_flags_t     dto_flags
13121 );
13122 it_status_t it_post_recvfrom(
13123     IN     it_ep_handle_t     ep_handle,
13124     IN const it_lmr_triplet_t *local_segments,
13125     IN     size_t             num_segments,
13126     IN     it_dto_cookie_t    cookie,
13127     IN     it_dto_flags_t     dto_flags
13128 );
13129 it_status_t it_post_send(
13130     IN     it_ep_handle_t     ep_handle,
13131     IN const it_lmr_triplet_t *local_segments,

```

```

13132         IN          size_t          num_segments,
13133         IN          it_dto_cookie_t  cookie,
13134         IN          it_dto_flags_t   dto_flags
13135     );
13136     it_status_t it_post_sendto(
13137         IN          it_ep_handle_t    ep_handle,
13138         IN  const   it_lmr_triplet_t   *local_segments,
13139         IN          size_t            num_segments,
13140         IN          it_dto_cookie_t    cookie,
13141         IN          it_dto_flags_t     dto_flags,
13142         IN  const   it_dg_remote_ep_addr_t *remote_ep_addr
13143     );
13144     it_status_t it_pz_create(
13145         IN  it_ia_handle_t  ia_handle,
13146         OUT it_pz_handle_t  *pz_handle
13147     );
13148     it_status_t it_pz_free(
13149         IN  it_pz_handle_t  pz_handle
13150     );
13151     it_status_t it_pz_query(
13152         IN  it_pz_handle_t    pz_handle,
13153         IN  it_pz_param_mask_t mask,
13154         OUT it_pz_param_t     *params
13155     );
13156     it_status_t it_reject(
13157         IN          it_cn_est_identifier_t  cn_est_id,
13158         IN  const   unsigned char          *private_data,
13159         IN          size_t                  private_data_length
13160     );
13161
13162     /* IT-API 1.0 prototype for B/W compatibility */
13163     it_status_t it_rmr_bind10(
13164         IN  it_rmr_handle_t    rmr_handle,
13165         IN  it_lmr_handle_t    lmr_handle,
13166         IN  void                *addr,
13167         IN  it_length_t        length,
13168         IN  it_mem_priv_t      privs,
13169         IN  it_ep_handle_t     ep_handle,
13170         IN  it_dto_cookie_t    cookie,
13171         IN  it_dto_flags_t     dto_flags,
13172         OUT it_rmr_context_t   *rmr_context
13173     );
13174
13175     /*
13176        it_rmr_create10 is provided for backwards compatibility
13177        and may be dropped in a future IT-API version.
13178
13179        Calls with a suffix "10" can be typically implemented through
13180        direct inlining to the corresponding calls with suffix "20",
13181        providing default arguments as necessary.
13182     */
13183
13184     it_status_t it_rmr_create10(
13185         IN  it_pz_handle_t    pz_handle,

```

```

13186         OUT  it_rmr_handle_t  *rmr_handle
13187     );
13188
13189     /*
13190         it_rmr_create20 provides the v2.0 functionality
13191         and may be renamed to it_rmr_create in a future IT-API version.
13192     */
13193     it_status_t it_rmr_create20(
13194         IN    it_pz_handle_t    pz_handle,
13195         IN    it_rmr_type_t     rmr_type,
13196         OUT   it_rmr_handle_t   *rmr_handle
13197     );
13198
13199     it_status_t it_rmr_free(
13200         IN    it_rmr_handle_t   rmr_handle
13201     );
13202     it_status_t it_rmr_link(
13203         IN    it_rmr_handle_t   rmr_handle,
13204         IN    it_lmr_handle_t   lmr_handle,
13205         IN    void              *addr,
13206         IN    it_length_t       length,
13207         IN    it_addr_mode_t    addr_mode,
13208         IN    it_mem_priv_t     privs,
13209         IN    it_ep_handle_t     ep_handle,
13210         IN    it_dto_cookie_t   cookie,
13211         IN    it_dto_flags_t    dto_flags,
13212         OUT   it_rmr_context_t  *rmr_context
13213     );
13214     it_status_t it_rmr_query(
13215         IN    it_rmr_handle_t    rmr_handle,
13216         IN    it_rmr_param_mask_t mask,
13217         OUT   it_rmr_param_t     *params
13218     );
13219     it_status_t it_rmr_unlink(
13220         IN    it_rmr_handle_t   rmr_handle,
13221         IN    it_ep_handle_t     ep_handle,
13222         IN    it_dto_cookie_t   cookie,
13223         IN    it_dto_flags_t    dto_flags
13224     );
13225     it_status_t it_set_consumer_context(
13226         IN    it_handle_t       handle,
13227         IN    it_context_t      context
13228     );
13229     it_status_t it_socket_convert(
13230         IN    int                sd,
13231         IN    it_ep_handle_t     ep_handle,
13232         IN    it_sc_flags_t      flags,
13233         IN    const unsigned char* msg,
13234         IN    size_t             len
13235     );
13236     it_status_t it_srq_create(
13237         IN    it_pz_handle_t     pz_handle,
13238         IN    size_t             max_recv_segments,
13239         IN    size_t             max_recv_dtos,

```

```

13240         OUT it_srq_handle_t *srq_handle
13241     );
13242     it_status_t it_srq_free(
13243         IN it_srq_handle_t srq_handle
13244     );
13245     it_status_t it_srq_modify(
13246         IN it_srq_handle_t srq_handle,
13247         IN it_srq_param_mask_t mask,
13248         IN const it_srq_param_t *params
13249     );
13250     it_status_t it_srq_query(
13251         IN it_srq_handle_t srq_handle,
13252         IN it_srq_param_mask_t mask,
13253         OUT it_srq_param_t *params
13254     );
13255     it_status_t it_ud_service_reply (
13256         IN it_ud_svc_req_identifier_t ud_svc_req_id,
13257         IN it_ud_svc_req_status_t status,
13258         IN it_remote_ep_info_t ep_info,
13259         IN const unsigned char *private_data,
13260         IN size_t private_data_length
13261     );
13262     it_status_t it_ud_service_request (
13263         IN it_ud_svc_req_handle_t ud_svc_handle
13264     );
13265     it_status_t it_ud_service_request_handle_create (
13266         IN const it_conn_qual_t *conn_qual,
13267         IN it_evd_handle_t reply_evd,
13268         IN const it_path_t *destination_path,
13269         IN const unsigned char *private_data,
13270         IN size_t private_data_length,
13271         OUT it_ud_svc_req_handle_t *ud_svc_handle
13272     );
13273     it_status_t it_ud_service_request_handle_free (
13274         IN it_ud_svc_req_handle_t ud_svc_handle
13275     );
13276     it_status_t it_ud_service_request_handle_query (
13277         IN it_ud_svc_req_handle_t ud_svc_handle,
13278         IN it_ud_svc_req_param_mask_t mask,
13279         OUT it_ud_svc_req_param_t *ud_svc_handle_info
13280     );

```

13281 **F.2 it_api_os_specific.h**

```

13282     #include "/usr/include/netinet/in.h"
13283     #include "/usr/include/sys/types.h"
13284
13285     #define IT_NO_ADDR NULL
13286     #define IT_INTERFACE_NAME_SIZE 128

```