

1 *Consortium Specification*

2 **Interconnect Transport API (IT-API)**
3 **Version 2.1**

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 2.1**

34 ISBN: 1-931624-59-3

35 Document Number: C055

36

37 Published by The Open Group, November, 2005.

38

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

40 ogspecs@opengroup.org

41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77

Contents

1	Introduction.....	1
1.1	Interface Adapters.....	1
1.2	Memory Management.....	2
1.3	Communication Endpoints	3
1.4	Work Requests and Data Transfer Operations	4
1.5	Events	5
2	Referenced Documents	7
3	Definitions.....	9
4	Global Behavior	30
4.1	Asynchronous versus Synchronous APIs	30
4.2	Thread Safety.....	30
4.3	Signal Handlers.....	35
4.4	Fork Semantics	35
4.5	Exec Semantics	35
4.6	Exit Semantics	35
4.7	Error Handling.....	35
4.8	IT Handle Management	36
4.9	Output Parameters	36
4.10	Privileged Consumer	36
4.11	Memory Management.....	37
5	Connection Management	38
5.1	Overview	38
5.1.1	IRD/ORD Negotiation.....	39
5.2	Transport-Independent Interface.....	40
5.2.1	Overview	40
5.2.2	ULP Constraints for iWARP	41
5.2.3	API-Supported IRD/ORD Negotiation.....	41
5.2.4	Transport-Dependent Attributes.....	42
5.3	Transport-Dependent Interface (iWARP-Only)	43
5.3.1	Overview	43
5.3.2	IRD/ORD Negotiation.....	45
6	API Reference Pages.....	46
	it_address_handle_create().....	49
	it_address_handle_free().....	52
	it_address_handle_modify().....	53
	it_address_handle_query().....	55

78	it_convert_net_addr().....	57
79	it_ep_accept().....	59
80	it_ep_connect()	62
81	it_ep_disconnect().....	68
82	it_ep_free().....	70
83	it_ep_modify()	72
84	it_ep_query().....	74
85	it_ep_rc_create()	76
86	it_ep_reset()	81
87	it_ep_ud_create()	82
88	it_evd_callback_attach()	85
89	it_evd_callback_detach()	88
90	it_evd_create()	89
91	it_evd_dequeue().....	100
92	it_evd_free().....	102
93	it_evd_modify()	104
94	it_evd_post_se().....	107
95	it_evd_query().....	109
96	it_evd_wait().....	111
97	it_get_consumer_context().....	115
98	it_get_handle_type()	116
99	it_get_pathinfo()	117
100	it_handoff()	120
101	it_hton64().....	122
102	it_ia_create()	123
103	it_ia_free().....	125
104	it_ia_info_free()	126
105	it_ia_query().....	127
106	it_interface_list().....	128
107	it_listen_create()	131
108	it_listen_free().....	133
109	it_listen_query().....	134
110	it_lmr_create().....	136
111	it_lmr_create_unlinked().....	142
112	it_lmr_flush_to_mem()	145
113	it_lmr_free().....	147
114	it_lmr_link().....	148
115	it_lmr_modify().....	154
116	it_lmr_query()	156
117	it_lmr_refresh_from_mem()	159
118	it_lmr_unlink()	161
119	it_make_rdma_addr_absolute()	164
120	it_make_rdma_addr_relative().....	165
121	it_mem_map().....	166
122	it_mem_unmap().....	168
123	it_post_atomic()	169
124	it_post_rdma_read().....	175
125	it_post_rdma_read_to_rmr()	181
126	it_post_rdma_write().....	187

127	it_post_recv()	192
128	it_post_recvfrom()	197
129	it_post_send()	201
130	it_post_send_and_unlink()	206
131	it_post_sendto()	212
132	it_pz_create()	216
133	it_pz_free()	217
134	it_pz_query()	218
135	it_reject()	219
136	it_rmr_create()	221
137	it_rmr_free()	223
138	it_rmr_link()	224
139	it_rmr_query()	230
140	it_rmr_unlink()	232
141	it_set_consumer_context()	236
142	it_socket_convert()	237
143	it_srq_create()	245
144	it_srq_free()	248
145	it_srq_modify()	249
146	it_srq_query()	251
147	it_ud_service_reply()	253
148	it_ud_service_request()	256
149	it_ud_service_request_handle_create()	259
150	it_ud_service_request_handle_free()	262
151	it_ud_service_request_handle_query()	263
152	7 Data Type Reference Pages	265
153	it_addr_mode_t	266
154	it_aevd_notification_event_t	268
155	it_affiliated_event_t	269
156	it_boolean_t	277
157	it_cm_msg_events	278
158	it_cm_req_events	289
159	it_conn_qual_t	292
160	it_context_t	294
161	it_dg_remote_ep_addr_t	295
162	it_dto_cookie_t	297
163	it_dto_events	298
164	it_dto_flags_t	302
165	it_dto_status_t	309
166	it_ep_attributes_t	318
167	it_ep_state_t	326
168	it_event_t	338
169	it_handle_t	343
170	it_ia_info_t	345
171	it_io_addr_t	353
172	it_iobl_t	354
173	it_lmr_triplet_t	357

174		it_mem_priv_t.....	358
175		it_net_addr_t.....	360
176		it_path_t.....	362
177		it_rmr_triplet_t.....	366
178		it_rmr_type_t.....	367
179		it_software_event_t.....	368
180		it_status_t.....	369
181		it_unaffiliated_event_t.....	372
182	A	Implementer's Guide	374
183	B	Implementer's Guide to Connection Management for iWARP	391
184	B.1	Overview	391
185	B.2	Miscellaneous	392
186	B.3	Interoperability Considerations	392
187	B.4	MPA Marker Control.....	392
188	B.5	Transport-Independent Interface.....	393
189	B.6	Transport-Dependent Interface	400
190	C	Backwards Compatibility with Earlier IT-API Versions	407
191	D	Functional Changes (IT-API Version 2.1 Relative to IT-API Version 2.0)	411
192	E	Functional Changes (IT-API Version 2.0 Relative to IT-API Version 1.0)	416
193	F	IT-API 1.0 Errata	432
194	G	Header Files	449
195	G.1	it_api.h	449
196	G.2	it_api_os_specific.h	493
197			

List of Figures

199	Figure 1: Connection Establishment with the TII	40
200	Figure 2: Conversion Process with MPA Startup (TDI)	44
201	Figure 3: Conversion Process with MPA Startup Suppression (TDI)	45
202	Figure 4: Conversion Initiator with <i>flags</i> set to IT_SC_DEFAULT	240
203	Figure 5: Conversion Responder with <i>flags</i> set to IT_SC_DEFAULT	241
204	Figure 6: Conversion Initiator with the IT_SC_NO_REQ_REP bit set in <i>flags</i>	242
205	Figure 7: Conversion Responder with the IT_SC_NO_REQ_REP bit set in <i>flags</i>	243
206	Figure 8: Three Way Passive RC Endpoint State Diagram	333
207	Figure 9: Three Way Active RC Endpoint State Diagram	334
208	Figure 10: Two Way Active RC Endpoint State Diagram	335
209	Figure 11: Two Way Passive RC Endpoint State Diagram	336
210	Figure 12: Unreliable Datagram Endpoint State Diagram	336
211	Figure 13: Connection Establishment with MPA Startup (TII): RNIC without RTR State	393
212	Figure 14: Connection Establishment with MPA Startup (TII): RNIC with RTR State	395
213	Figure 15: IRD/ORD Header	396
214	Figure 16: RDMAC Initiator to Permissive IETF Responder	397
215	Figure 17: Permissive IETF Initiator to RDMAC Responder	398
216	Figure 18: RDMAC Initiator to Non-Permissive IETF Responder	399
217	Figure 19: Non-Permissive IETF Initiator to RDMAC Responder	400
218	Figure 20: Conversion Process with MPA Startup (TDI) - RNIC without RTR State	401
219	Figure 21: Conversion Process with MPA Startup Suppression (TDI): RNIC without RTR State	402
220	Figure 22: Permissive IETF Conversion Initiator to RDMAC Conversion Responder	403
221	Figure 23: RDMAC Conversion Initiator to Permissive IETF Conversion Responder	404
222	Figure 24: Non-Permissive IETF Conversion Initiator to RDMAC Conversion Responder	405
223	Figure 25: RDMAC Conversion Initiator to Non-Permissive IETF Conversion Responder	406
224		

List of Tables

226	Table 1: Thread Safety Models	31
227	Table 2: Thread-Safety Models Applied to IT-APIs	34
228	Table 3: Connection Management Event Definitions	282
229	Table 4: Event Management Event Fields	282
230	Table 5: reject_reason_code Descriptions	283
231	Table 6: UD Service Resolution Reply Event Definitions	284
232	Table 7: Service Resolution Reply Status	284
233	Table 8: InfiniBand reject_reason_code Mapping	285
234	Table 9: VIA reject_reason_code Mapping	285
235	Table 10: Communication Management Request Event Definitions	290
236		

237

Preface

238

The Interconnect Software Consortium

239
240
241

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.

242

The Open Group

243
244
245
246
247
248
249
250
251

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.

252

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

253
254
255

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.

256

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

257
258
259
260

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.

261
262

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

263

This Document

264
265
266

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.

267
268

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

269 2.1 of the IT-API. The first publication – also known as IT-API Issue 1.0 – corresponds to
270 Version 1.0 of the IT-API.

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

- 274 • An unchanged major version number (*m*) and a new minor version number (*n*) indicate
275 there is no change to the definitive information contained in the previous version of the
276 specification, but additions/extensions are included. The new version of the specification
277 is source code-compatible with the previous one and replaces the previous specification.
- 278 • A new major version number (*m*) and a minor version number (*n*) set to zero indicate
279 there is substantive change to the definitive information contained in the previous version
280 of the specification, and there may also be additions/extensions. The new version of the
281 specification is not source code-compatible with the previous one. The specifications
282 corresponding to both versions are maintained as current publications.

283 **Typographical Conventions**

284 The following typographical conventions are used throughout this document:

- 285 • Bold font is used in text for filenames and type names.
- 286 • Italic strings are used for emphasis. Italics in text also denote variable names, functions,
287 and data structures.
- 288 • Normal font is used for the names of constants and literals.
- 289 • Syntax and code examples are shown in fixed width font.
- 290 • Bold Italic is used for all terms defined in the Definitions section when they first appear.
291 IT-API objects are capitalized throughout the document (e.g., Interface Adapter,
292 Endpoint, etc.).

293 **Reference Page Conventions**

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

- 295 • The SYNOPSIS section summarizes the syntax for an item.
- 296 • An APPLICABILITY section is provided for any API routine or data type which either
297 cannot be used for all service types supported by the IT-API or which is supported only
298 optionally.
- 299 • The DESCRIPTION section provides a summary description of an item.
- 300 • An EXTENDED DESCRIPTION section gives more detailed information about the use
301 of the item being documented, and about its behavior and features. It expands on subjects
302 discussed in the DESCRIPTION section, but with more detail and background
303 information. It also provides information on transport dependencies.

- 304 • A BACKWARDS COMPATIBILITY section is provided for API routines whose
305 signature has changed from the previous IT-API version. For API routines that were
306 renamed or removed, see Appendix D.
- 307 • The RETURN VALUES section (for API routines only) lists the returned values
308 including immediate errors for API routines, along with a description for each.
- 309 • The ASYNCHRONOUS ERRORS section describes errors that cannot be reported as
310 immediate errors via return values.
- 311 • The APPLICATION USAGE section provides additional guidance to application writers
312 on intended or typical usage of IT-API routines or objects, on using an IT-API routine or
313 object jointly with other routines or objects, and performance hints.

314

Trademarks

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

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

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

319

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

323

324

Acknowledgements

325

The Interconnect Software Consortium gratefully acknowledges the contribution of:

Caitlin Bestler	Fredy Neeser
Cathy Fox	Matthew Pearson
Jim Hamrick	Jay Rosser
Martin Kirk	Ramesh VelurEunni
Bernard Metzler	Fred Worley

326

in the development of the IT-API Specification, Version 2.1, and of:

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

327

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 Delegates	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

328

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

329 1 Introduction

330 The IT-API defines interfaces for direct interaction with Remote Direct Memory Access
331 (RDMA)-capable transports. This IT-API Specification Version 2.1 covers the *Reliable*
332 *Connection* and *Unreliable Datagram* services of the *InfiniBand Transport* [IB-R1.1] [IB-
333 R1.2], the *iWARP Transport* (which also provides a Reliable Connection service) [MPA-IETF]
334 [MPA-RDMAC], [DDP-IETF] [DDP-RDMAC] [RDMAP-IETF] [RDMAP-RDMAC], and *VIA*
335 networks [VIA-V1.0]. The specification includes:

- 336 • An Introduction (this section) (Chapter 1)
- 337 • A list of Referenced Documents (Chapter 2)
- 338 • A Glossary (Chapter 3)
- 339 • A section on Global Behavior (Chapter 4)
- 340 • A section on Connection Management (Chapter 5)
- 341 • Reference pages for 68 APIs and their supporting data type definitions (Chapters 6 and 7)
- 342 • Implementer’s Guides (Appendix A and B)
- 343 • A section on Backwards Compability with earlier versions of IT-API (Appendix C)
- 344 • A list of Functional Changes from IT-API Version 1.0 and the detailed Errata (Appendix
345 D and F)
- 346 • Two sample header files (Appendix G)

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

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

355 1.1 Interface Adapters

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

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

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

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

379 1.2 Memory Management

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

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

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

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

408 A Privileged Consumer (see Section 4.10) may also create an unlinked LMR using the
409 *it_lmr_create_unlinked* call. The unlinked LMR is not usable for I/O operations by the IA until it
410 is linked. A Privileged Consumer may subsequently link the LMR to an **I/O Buffer List (IOBL)**
411 using the *it_lmr_link* call, also referred to as an **LMR Link** operation, yielding a usable LMR.
412 Access privileges for the LMR are specified in the *it_lmr_link* call. The *it_lmr_unlink* call
413 disassociates the LMR from the IOBL.

414 A Privileged Consumer can avoid the need to create LMRs by using the **Direct LMR Handle**.
415 The Direct LMR Handle can be used by a Privileged Consumer to give the IA direct access to
416 local memory mapped via the *it_mem_map* call. The Direct LMR Handle gives no remote access
417 rights.

418 The Consumer need not create an RMR to expose message buffers for incoming RDMA Read or
419 RDMA Write operations. The Consumer may simply advertize the *it_rmr_context_t* identifier
420 implicitly created when remote access privileges are set in the call to *it_lmr_create* or the call to
421 *it_lmr_link* (Privileged Consumers only). The *it_rmr_context_t* identifier is returned by
422 *it_lmr_create* and *it_lmr_link* , and may also be retrieved by using the *it_lmr_query* call.

423 Repeatedly creating or modifying LMRs is a far less efficient operation than linking or unlinking
424 RMRs, if the RMRs are linked to underlying LMRs and the underlying LMRs do not require
425 frequent *it_lmr_create* or *it_lmr_modify* operations. Similarly, linking or unlinking LMRs is also
426 more efficient than creating or modifying LMRs but is restricted to use by Privileged
427 Consumers. The Direct LMR Handle offers Privileged Consumers a method of accessing a local
428 buffer that is even more efficient than any of the previous methods mentioned but has
429 restrictions on the operations that may be performed with the LMR.

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

436 1.3 Communication Endpoints

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

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

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

452 For Reliable Connection communications, a Consumer may issue a request to connect a local
453 Endpoint to a remote Endpoint using the *it_ep_connect* call. In order to receive a **Connection**
454 **Request**, a Consumer creates an IT Listen Point object that is used to await Connection
455 Requests. The Listen Point is created by calling *it_listen_create*, which returns an
456 *it_listen_handle_t*. Attributes of the Listen Point can be queried by using the *it_listen_query* and
457 the Listen Point can be released with the *it_listen_free* call. A Consumer can accept or reject a
458 Connection Request using the *it_ep_accept* and *it_reject* calls, respectively.

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

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

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

467 For Unreliable Datagram communications, an IT **Address Handle** object can be created for use
468 in defining and targeting specific remote Endpoints. An Address Handle is created through the
469 *it_address_handle_create* call, which returns an *it_address_handle_t*. Attributes of the Address
470 Handle can be queried and modified by using the *it_address_handle_query* and
471 *it_address_handle_modify* calls. The Address Handle can be released with the
472 *it_address_handle_free* call.

473 For Unreliable Datagram communications, the Consumer can create an IT Service Request
474 Handle that is used to store Destination address information. A Service Request Handle is
475 created through the *it_ud_service_request_handle_create* call, which returns an
476 *it_ud_svc_req_handle_t*. Attributes of the Service Request Handle can be queried through the
477 *it_ud_service_request_handle_query* call and the Service Request Handle can be released with
478 the *it_ud_service_request_handle_free* call. A Service Request Handle is used in the
479 *it_ud_service_request* call to provide addressing information for use in sending the reply
480 message sent by the *it_ud_service_reply* call.

481 1.4 Work Requests and Data Transfer Operations

482 The Consumer can queue different kinds of **Work Requests** to an Endpoint. Work Requests
483 include **Send**, **Receive**, **RDMA Read**, **RDMA Write**, and **Atomic** Data Transfer Operations, as
484 well as **RMR Link**, **RMR Unlink**, **LMR Link**, and **LMR Unlink** memory management
485 operations. Work Requests are posted by Data Transfer Operations to a **Work Queue**.

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

489 The Consumer may issue requests to send messages using the *it_post_send* or
490 *it_post_send_and_unlink* interfaces for RC type Endpoints or the *it_post_sendto* interface for
491 UD type Endpoints. The Consumer issues requests to receive messages using either the
492 *it_post_recv* or *it_post_recvfrom* calls. RDMA operations are initiated through the
493 *it_post_rdma_read*, *it_post_rdma_read_to_rmr*, and *it_post_rdma_write* calls. Atomic
494 operations, if supported, are initiated through the *it_post_atomic* call.

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

497 1.5 Events

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

505 Common Event types include **Completion Events** for Work Requests (see *it_dto_status_t*;
506 includes any Completion Errors), **Communication Management Request Events** (see
507 *it_cm_req_events*), **Communication Management Message Events** (see *it_cm_msg_events*),
508 **Affiliated Asynchronous Events** or **Errors** (see *it_affiliated_event_t*), and **Unaffiliated**
509 **Asynchronous Events** or **Errors** (see *it_unaffiliated_event_t*).

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

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

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

523 The IT-API supports two types of EVDs. A **Simple EVD**, also called an **SEVD**, enqueues only
524 Events of a single type. This simplifies the implementation of an SEVD and enhances its
525 performance characteristics. For many communication scenarios, this provides a Consumer with

526 the best performance option. An **Aggregate EVD**, also called an **AEVD**, can be used to collect
527 Events from a set of SEVDs. This allows the Consumer to create a single **Notification**
528 mechanism that will enqueue many different types of Events. In addition to these IT-API
529 Notification mechanisms, *it_evd_create* allows an Implementation-defined file descriptor to be
530 associated with each EVD. This allows the Consumer to use a file descriptor-based Notification
531 mechanism provided by the Implementation (e.g., POSIX *poll*) to collect both non-IT-API
532 Events and IT-API Events from multiple IAs. A Privileged Consumer may also use the
533 *it_evd_callback_attach* call to ask for Notification to be given via a Consumer-supplied callback
534 routine. When the Consumer no longer wishes to receive Notification via a callback routine they
535 can call *it_evd_callback_detach*.

536 Most Events are associated with Endpoints. These Events include Work Request completions
537 and communication management Events related to the Endpoint. When an Endpoint is created,
538 the Consumer associates EVDs with it. These EVDs collect a specific type of Event. One EVD
539 enqueues Events associated with the completion of Consumer-initiated Work Requests,
540 including the completion of RMR Link and Unlink operations, and outgoing Send or RDMA
541 DTOs. A second EVD enqueues Events that result when an incoming Send message that was
542 directed to an Endpoint matches with a corresponding Receive DTO. For Endpoints using
543 connected communications, a third EVD enqueues Events related to Connection management.
544 EVDs may be shared across multiple Endpoints.

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

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

551 The IT-API gives the Consumer control over a number of aspects of Event handling. IT-API
552 interfaces used to initiate Work Requests include flags for enabling or suppressing the
553 generation of an Event object (also known as per-WR **Completion Suppression**), for enabling or
554 suppressing the associated Consumer Notification (also known as per-WR **Notification**
555 **Suppression**), and for requesting remote-side Endpoint Notification of message delivery. These
556 features are further described in *it_dto_flags_t*.

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

559 2 Referenced Documents

- 560 The following documents are referenced in this specification:
- 561 [DDP-IETF] Direct Data Placement over Reliable Transports, H. Shah et al,
562 IETF Internet Draft, February 2005.
563 (www.ietf.org/internet-drafts/draft-ietf-rddp-ddp-04.txt)
- 564 [DDP-RDMAC] Direct Data Placement over Reliable Transports (Version 1.0), H. Shah et al,
565 RDMA Consortium, October 2002.
566 (www.rdmaconsortium.org/home/draft-shah-iwarp-ddp-v1.0.pdf)
- 567 [IB-R1.1] InfiniBand Architecture Specification Volume 1, Release 1.1,
568 InfiniBand Trade Association, November 2002.
- 569 [IB-R1.2] InfiniBand Architecture Specification Volume 1, Release 1.2,
570 InfiniBand Trade Association, October 2004.
- 571 [INTEROP-IETF] RNIC Interoperability, J. Carrier & J. Pinkerton,
572 IETF Internet Draft, November 2004.
573 (www.ietf.org/internet-drafts/draft-carrier-rddp-rnic-interop-00.txt)
- 574 [IT-API-V1.0] Interconnect Transport API (IT-API), Issue 1.0,
575 The Interconnect Software Consortium, February 2004.
- 576 [IT-API-V2.0] Interconnect Transport API (IT-API), Version 2.0,
577 The Interconnect Software Consortium, March 2005.
- 578 [MPA-IETF] Marker PDU Aligned Framing for TCP Specification, P. Culley et al,
579 IETF Internet Draft, February 2005.
580 (www.ietf.org/internet-drafts/draft-ietf-rddp-mpa-02.pdf)
- 581 [MPA-RDMAC] Marker PDU Aligned Framing for TCP Specification (Version 1.0),
582 P. Culley et al, RDMA Consortium, October 2002.
583 (www.rdmaconsortium.org/home/draft-culley-iwarp-mpa-v1.0.pdf)
- 584 [RDMAP-IETF] An RDMA Protocol Specification, R. Recio et al,
585 IETF Internet Draft, February 2005.
586 (www.ietf.org/internet-drafts/draft-ietf-rddp-rdmap-03.txt)
- 587 [RDMAP-RDMAC] An RDMA Protocol Specification (Version 1.0), R. Recio et al,
588 RDMA Consortium, October 2002.
589 (www.rdmaconsortium.org/home/draft-recio-iwarp-rdmap-v1.0.pdf)
- 590 [VERBS-RDMAC] RDMA Protocol Verbs Specification (Version 1.0), J. Hilland et al,
591 RDMA Consortium, April 2003.
592 (www.rdmaconsortium.org/home/draft-hilland-iwarp-verbs-v1.0-RDMAC.pdf)

593 [VIA-V1.0] Virtual Interface Architecture Specification Version 1.0,
594 Compaq Computer Corporation, Intel Corporation, & Microsoft Corporation,
595 December 1997.

596

3 Definitions

597

Absolute Addressing

598

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

599

600

601

602

The *addr* 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 *addr.abs* member of an LMR Triplet or the *rdma_addr* parameter passed to an RDMA DTO is interpreted as the Base Address of the LMR plus a byte offset.

603

604

605

606

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

607

608

609

610

The *addr* 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 *addr.abs* member of an RMR Triplet or the *rdma_addr* parameter passed to an RDMA DTO is interpreted as the Base Address of the RMR plus a byte offset.

611

See also **Relative Addressing**.

612

Address Handle

613

614

615

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

616

AEVD

617

See **Aggregation Event Dispatcher**.

618

Affiliated Asynchronous Error

619

620

621

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

622

Affiliated Asynchronous Event

623

624

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

- 625 **Affiliated Event**
- 626 See **Affiliated Asynchronous Event**.
- 627 **Aggregation Event Dispatcher (AEVD)**
- 628 An IT Object that conceptually merges Event completion Notifications from one or more Simple
629 Event Dispatchers. This provides the Consumer with a single point to receive Notification of
630 Event completions across multiple Event Streams.
- 631 **Asynchronous Error**
- 632 An error that could not be reported as an immediate error. Asynchronous Errors include
633 Completion Errors, Affiliated Asynchronous Errors, and Unaffiliated Asynchronous Errors.¹
- 634 **Atomic**
- 635 The Data Transfer Operation (DTO) that is initiated by the *it_post_atomic* routine.
- 636 **AV-RNIC**
- 637 API +Verbs-enabled RNIC. See **V-RNIC**. An implementation of an RDMA API (such as IT-
638 API) over a V-RNIC.
- 639 **AV-RNIC/IETF**
- 640 AV-RNIC based on a V-RNIC/IETF with API support for handling the MPA Startup. An AV-
641 RNIC/IETF based on a Permissive (Non-permissive) V-RNIC/IETF is called Permissive (Non-
642 permissive).
- 643 **AV-RNIC/IETF with MPA Startup Suppression**
- 644 An AV-RNIC/IETF, whose API is instructed to suppress MPA Startup for transitioning a
645 particular streaming-mode connection to RDMA mode. API support for suppressing the MPA
646 Startup is outside of [MPA-IETF] but is expected to be supported for interoperability with the
647 device class “AV-RNIC/RDMAC without MPA Startup”. The IT-API provides this capability
648 for the TDI (Conversion process), but not for the TII.
- 649 **AV-RNIC/RDMAC with MPA Startup**
- 650 AV-RNIC based on a V-RNIC/RDMAC and with API support for handling the MPA Startup.
651 according to [INTEROP-IETF], wherein such an AV-RNIC is designated a (Software) Upgraded
652 RDMAC RNIC. An AV-RNIC/RDMAC with MPA Startup has MPA 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].

653	AV-RNIC/RDMAC without MPA Startup
654	AV-RNIC based on a V-RNIC/RDMAC without API support for handling the MPA Startup or
655	with suppressed MPA Startup. An AV-RNIC/RDMAC without MPA Startup also has MPA
656	Marker/CRC restrictions.
657	Base Address
658	The address corresponding to the first byte of an LMR or RMR in a given linear address space.
659	The Base Address of a linked RMR is within the address range of the underlying LMR.
660	Bind
661	See RMR Link .
662	Block List
663	A list of memory blocks with arbitrary alignment and constant block size.
664	Bus Address
665	An address exposed on an I/O interconnect bus. On some system architectures, bus addresses
666	and physical addresses are identical.
667	Communication Management Message Events
668	The set of Event types related to the sequence of messages involved in RC Connection
669	establishment, normal disconnect, Connection error conditions, and Unreliable Datagram
670	Service Resolution Replies.
671	Communication Management Request Events
672	The set of Event types that result from messages received requesting RC Connection
673	establishment or Unreliable Datagram service. Normally these Events trigger state changes at the
674	receiving Endpoint.
675	Completion Error
676	A processing error that could not be reported through an immediate error and which is associated
677	with a specific Work Request. Completion Errors are reported through an Event Stream for
678	WR/DTO completions.
679	Completion Event
680	An Event indicating that a previously posted Work Request has completed.

681	Completion Suppression
682	An optional Work Request behavior specifying that no Event is to be generated upon successful
683	completion of the requested operation.
684	Connection
685	An association between a pair of Endpoints such that data posted via Data Transfer Operations
686	of either Endpoint arrives at the other Endpoint of the Connection.
687	Connection Qualifier
688	A value that allows an incoming Connection Request or Unreliable Datagram Service Resolution
689	Request to be associated with an entity that can provide that service.
690	Connection Reply
691	A message in response to a Connection Request message.
692	Connection Request
693	A message that requests RC Connection establishment.
694	Consumer
695	An application that utilizes the IT-API.
696	Context
697	A Consumer-supplied value that can be associated with an instance of an IT Object.
698	Conversion
699	The actions performed by <i>it_socket_convert</i> for converting an unconnected Endpoint of the RC
700	type and a connected socket to a connected, RDMA-enabled Endpoint. For the Conversion
701	Initiator, the Conversion includes the transmission of the Last ULP Streaming-Mode Message.
702	Conversion Initiator
703	A Consumer calling <i>it_socket_convert</i> with a Last ULP Streaming-Mode Message of length
704	greater than zero.
705	Conversion Responder
706	A Consumer calling <i>it_socket_convert</i> with a Last ULP Streaming-Mode Message of length
707	zero.

708	Data Transfer Operation (DTO)
709 710	A request submitted by the Consumer to the Implementation to move data between two Endpoints. See also Work Request .
711	Deferred RDMA Transition
712 713	The enablement of RDMAP/DDP/MPA on top of a TCP connection after the connection has been established and used by an ULP for some data exchange in streaming mode.
714	Destination
715	The Endpoint where a message is received.
716	Direct LMR Handle
717 718 719	An LMR Handle that supports bypass of RNIC memory protection and translation tables. The Direct LMR Handle may only be used by a Privileged Consumer. The Direct LMR Handle corresponds to the iWARP “Stag of Zero” and to the InfiniBand v1r1.2 “Reserved L_Key”.
720	DTO
721	See Data Transfer Operation .
722	DTO Cookie
723 724	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.
725	Endpoint (EP)
726 727 728	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.
729	Endpoint Hard High Watermark
730 731	The maximum number of Receive DTOs that an Endpoint with an associated Shared Receive Queue can have in progress.
732	Endpoint ID
733 734	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.

735	Endpoint Key
736	A construct that some transports require to be associated with an outgoing datagram to allow the
737	Receiver to validate that the sender of the datagram has permission to access the Receiver's
738	Endpoint.
739	Endpoint Protection Zone
740	The Protection Zone associated with an Endpoint.
741	Endpoint Soft High Watermark
742	The maximum number of Receive DTOs that an Endpoint with an associated Shared Receive
743	Queue can have in progress without causing an Endpoint Soft High Watermark Event to be
744	generated.
745	Endpoint Soft High Watermark Event
746	An Affiliated Asynchronous Event that is generated when the Consumer has armed the Endpoint
747	Soft High Watermark mechanism for an Endpoint and the total number of Receive DTOs in
748	progress on that Endpoint exceeds the Endpoint Soft High Watermark.
749	EP
750	See Endpoint .
751	EVD
752	See Event Dispatcher .
753	Event
754	A structure or record that is delivered to the Consumer through an Event Dispatcher to provide
755	notice of some kind. Types of Events include DTO completions, Connection state changes,
756	Asynchronous Errors, and information passed through the <i>it_evd_post_se</i> interface that is
757	generated by the Consumer.
758	Event Dispatcher (EVD)
759	An IT Object that conceptually merges Event completion Notifications for the Consumer. The
760	IT-API defines two types of EVDs: a Simple Event Dispatcher and an Aggregation Event
761	Dispatcher.
762	Event Stream
763	A type of Event for an Event Dispatcher such as WR/DTO completions, Communication
764	Management Request Events (e.g., Connection Requests), Communication Management
765	Message Events (e.g., Connection reject Notifications, Connection establishment completion

766 Notifications, disconnect Notifications, Connection errors, Connection Request timeouts),
767 Affiliated Asynchronous Errors, Unaffiliated Asynchronous Errors, Consumer-generated
768 Software Events, or AEVD Notifications. The Event Stream of an Event Dispatcher determines
769 the type of Events that can be enqueued onto the Event Dispatcher by IT Objects.

770 **FBO**

771 See **First-Byte-Offset**.

772 **First-Byte-Offset**

773 The offset into the first block or page in an **IOBL**.

774 **Handle**

775 An opaque data type used to reference an object.

776 **IA**

777 See **Interface Adapter**.

778 **IANA**

779 See **Internet Address Naming Authority**.

780 **IANA Port Number**

781 A specific port address as defined by IANA.

782 **IB**

783 See **InfiniBand**.

784 **ICSC**

785 See **Interconnect Software Consortium**.

786 **IETF**

787 See **Internet Engineering Task Force**.

788 **Immediate RDMA Transition**

789 The enablement of RDMAP/DDP/MPA on top of a TCP connection immediately after the
790 connection has been established; i.e., without any prior ULP use of the connection for data
791 exchange in streaming mode.

792	Implementation
793	The collection of software and hardware that combine to provide the service exported by the IT-
794	API.
795	Implementer's Guide
796	A non-normative section of the IT-API documentation set that contains information provided to
797	assist implementers of the IT-API.
798	Inbound RDMA Read Queue
799	An internal queue of an Endpoint that handles incoming RDMA Read Requests. The processing
800	of an incoming RDMA Read Request involves generating an RDMA Read Response.
801	InfiniBand (IB)
802	One of the transports that the IT-API supports. The host interface portion of InfiniBand is
803	defined in [IB-R1.1] and [IB-R1.2] for Releases 1.1 and 1.2, respectively.
804	InfiniBand Global Routing Header
805	A routing header that may be present in the first 40 bytes of a completed Unreliable Datagram
806	Receive operation. See the InfiniBand specification for a description of the format of this routing
807	header.
808	InfiniBand Transport
809	Transport services defined by the InfiniBand Architecture.
810	Interconnect Software Consortium (ICSC)
811	Standards organization that includes the ITWG. The <i>Interconnect Software Consortium</i> is
812	affiliated with <i>The Open Group</i> .
813	Interconnect Transport Working Group (ITWG)
814	The <i>ICSC Working Group</i> that created the IT-API.
815	Internet Engineering Task Force (IETF)
816	<i>Internet Engineering Task Force</i> .
817	Internet Address Naming Authority (IANA)
818	IETF Network Address naming authority.

819	Interface
820	A host resident device that transfers data to and from the host memory to which it is attached.
821	Interface Adapter (IA)
822	An instance of an Interface that is created by the <i>it_ia_create</i> call. An Interface Adapter may
823	contain one or more Spigots.
824	I/O Address
825	An address accessible by an IA. Depending on the system architecture, this may be a Physical
826	Address or a Bus Address . In IT-API, the I/O Address object is an opaque, Implementation-
827	dependent object.
828	IOBL
829	See I/O Buffer List .
830	I/O Buffer List (IOBL)
831	An IT-API object that specifies a list of buffers using I/O Addresses . Elements of an I/O Buffer
832	List can have constant length or variable length.
833	IP
834	The IETF Internet Protocol.
835	IPv4
836	The IETF Internet Protocol Version 4.
837	IPv6
838	The IETF Internet Protocol Version 6.
839	IRD
840	Inbound RDMA Read Queue Depth (IRD). The maximum number of inbound RDMA Read
841	requests that can be outstanding on an Endpoint.
842	IRRQ
843	See Inbound RDMA Read Queue .

844	IT-API
845	The data structures and routines that make up the Interconnect Transport Application
846	Programming Interface.
847	IT Handle
848	An opaque reference to an IT Object. An IT Handle is returned to the Consumer whenever an IT
849	Object is created for the Consumer's use. The IT Handle can be used to reference the IT Object
850	in subsequent calls into the IT-API.
851	IT Object
852	A software object created by the IT-API Implementation as a result of a Consumer call into the
853	IT-API, used to satisfy subsequent Consumer requests. When the IT Object is created, an opaque
854	reference to the object, called a Handle, is returned to the Consumer for use in subsequent calls
855	into the IT-API.
856	ITWG
857	See Interconnect Transport Working Group .
858	iWARP
859	The protocol suite enabling RDMA on top of TCP/IP or SCTP/IP, comprising MPA [MPA-
860	IETF] [MPA-RDMAC], DDP [DDP-IETF] [DDP-RDMAC], and RDMAP [RDMAP-IETF
861	RDMAP-RDMAC].
862	iWARP Transport
863	The iWARP protocol suite on top of TCP/IP or SCTP/IP.
864	Last Streaming-Mode Message
865	The last message sent in streaming mode prior to enabling RDMA mode on a connection. In the
866	presence of an MPA Startup, this message is the MPA Reply message.
867	Last ULP Streaming-Mode Message
868	The last ULP message sent in TCP streaming mode prior to enabling RDMA mode on a
869	connection, sent by an IT-API Consumer acting as the Conversion Initiator. Need not be the Last
870	Streaming-Mode Message since the Conversion of a TCP connection typically involves an MPA
871	Startup handshake that is invisible to the Consumers.
872	Link
873	See LMR Link and RMR Link .

- 874 **Listen Point**
- 875 An object capable of listening for incoming Connection Requests and of passing such requests to
876 a Simple Event Dispatcher. A Listen Point is associated with a single Spigot.
- 877 **LLP**
- 878 See **Lower Layer Protocol**.
- 879 **LMR**
- 880 See **Local Memory Region**.
- 881 **LMR Link**
- 882 The memory management operation initiated by *it_lmr_link*, which associates an LMR with an
883 IOBL and thereby enables local and optional remote access to that LMR.
- 884 **LMR Triplet**
- 885 A type used to specify a section of a Local Memory Region. An LMR Triplet specifies an LMR
886 Handle, an address, and a length.
- 887 **LMR Unlink**
- 888 The memory management operation initiated by *it_lmr_unlink*, which destroys the association of
889 an LMR with an IOBL.
- 890 **Local Memory Region (LMR)**
- 891 A contiguously addressable area of arbitrary size within a linear address space enabling local
892 access and optional remote access.
- 893 **Lower Layer Protocol (LLP)**
- 894 A protocol used in a layer of the underlying RDMA transport.
- 895 **Marker PDU Aligned Framing for TCP (MPA)**
- 896 The framing protocol used to support messaging over TCP streams. See [\[MPA-IETF\]](#) and
897 [\[MPA-RDMAC\]](#).
- 898 **MPA**
- 899 See **Marker PDU Aligned Framing for TCP**.

900	MPA Startup
901 902	Exchange of MPA Request/Reply messages according to [MPA-IETF] , preferably with additional support for interoperability according to [INTEROP-IETF] .
903	Narrow RMR
904 905 906	An RMR that can be referenced or accessed only via the associated Endpoint if the RMR is in the linked state, where the associated Endpoint is the one through which the RMR was linked. See also Wide RMR .
907	Network Address
908	An identifier that can be used to reach a particular Spigot attached to a network.
909	Non-Permissive AV-RNIC/IETF
910 911	An AV-RNIC/IETF that cannot support downgrading versions of RDMAP, DDP, and MPA. See [INTEROP-IETF] .
912	Notification
913 914	An asynchronous mechanism for providing the Consumer with information about the completion of a previously posted operation.
915	Notification Event
916 917	An Event in an Event Stream whose arrival triggers the Notification of the Event to a waiting Consumer via either a wakeup from <i>it_evd_wait</i> , or via a higher-level Notification mechanism.
918	Notification Suppression
919 920 921 922	A Consumer-specified option for Data Transfer Operations that informs the Implementation that no Notification Event should be created if the DTO completes successfully. Notification Suppression has no effect on operations that complete in error – in this case the completion will generate an error Event.
923	ORD
924 925	Outbound RDMA Read Queue Depth (ORD). The maximum number of outbound RDMA Read Work Requests a Consumer can have outstanding on an Endpoint.
926	Organization Unique Identifier (OUI)
927 928 929	An OUI is a 24-bit globally unique number assigned by the Institute of Electrical and Electronics Engineers (IEEE). The IT-API uses an OUI to map IETF IANA Port Numbers into the IB Service ID space for use within the IT-API.

930	OUI
931	See Organization Unique Identifier .
932	Outstanding Operation
933	An operation is “Outstanding” until the Event for the operation completes, or for an operation
934	whose completion has been suppressed, until an operation posted subsequent to it completes.
935	Page List
936	A list of memory pages with page alignment and constant or variable page sizes.
937	Path
938	The collection of links, switches, and routers a message traverses from a Source Spigot to a
939	Destination Spigot. This is represented in the IT-API by the <i>it_path_t</i> structure.
940	Permissive AV-RNIC/IETF
941	An AV-RNIC/IETF that supports downgrading versions of RDMAP, DDP, and MPA. See
942	[INTEROP-IETF] .
943	Physical Address
944	An address exposed on the CPU/memory bus. Contrasts with Bus Address .
945	Port Number
946	See IANA Port Number .
947	Private Data
948	Consumer data that is opaque to the Implementation and is passed between the local and remote
949	Consumers by the Implementation's Connection establishment and UD service resolution
950	routines.
951	Privilege
952	The right of an IT-API object to support operations denied to normal (non-Privileged)
953	Consumers. Such operations include use of the Direct LMR Handle and ability to perform LMR
954	Link operations.
955	Privileged Consumer
956	An application that utilizes the IT-API and that is granted permission by the O/S to perform
957	operations denied to normal Consumers. An example Privileged Consumer application may be

958 an O/S kernel application. The definition of Privileged Consumer application does not preclude
959 user-mode applications. Such distinctions are O/S-specific.

960 **Privileged Mode Endpoint**

961 An Endpoint created with Privilege.

962 **Protection Zone (PZ)**

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

965 **PZ**

966 See **Protection Zone**.

967 **RC**

968 See **Reliable Connected**.

969 **RDMA**

970 See **Remote Direct Memory Access**.

971 **RDMA Initiator**

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

975 **RDMA Responder**

976 A Consumer calling *it_listen_create* followed by *it_ep_accept* or *it_reject*, or a Consumer
977 calling *it_socket_convert* with a Last ULP Streaming-Mode Message of length greater than zero.
978 Also used to identify the corresponding side of the connection. For the RDMA Responder, the
979 passive Endpoint states of the Endpoint finite-state machine are used.

980 **RDMA Read**

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

982 **RDMA Read Request**

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

986 **RDMA Read Response**

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

988 **RDMA Write**

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

990 **RDMA Write Request**

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

994 **Receive**

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

997 **Receive Queue (RQ)**

998 An internal queue associated with an Endpoint on which Receive Work Requests are posted.
999 Alternatively, an Endpoint may be associated with a Shared Receive Queue, in which case
1000 Receive Work Requests are posted to the Shared Receive Queue.

1001 **Relative Addressing**

1002 Addressing data within an LMR or RMR using byte offsets relative to the beginning of the LMR
1003 or RMR, respectively.

1004 The *addr* attribute (requested starting address) of an LMR with Relative Addressing is
1005 interpreted as the Base Address of the LMR, unless it equals `IT_NO_ADDR`. This attribute can
1006 be used for specifying or registering an LMR but is ignored in case of Relative Addressing when
1007 the LMR is accessed by a DTO. When an LMR with Relative Addressing is accessed by a DTO,
1008 the *addr.rel* member of an LMR Triplet or the *rdma_addr* parameter passed to an RDMA DTO
1009 is interpreted as a byte offset relative to the first byte of the LMR.

1010 When an RMR is linked to an LMR, with Relative Addressing selected for the RMR, the *addr*
1011 parameter of the RMR Link operation and the *addr* attribute of the LMR (which must use
1012 Absolute Addressing) are used for identifying the offset of the first byte of the RMR from the
1013 first byte of the LMR.

1014 The *addr* attribute (starting address) of a linked RMR with Relative Addressing is interpreted as
1015 the Base Address of the RMR. When a linked RMR with Relative Addressing is accessed by a
1016 DTO, the *addr.rel* member of an RMR Triplet or the *rdma_addr* parameter passed to an RDMA
1017 DTO is interpreted as a byte offset relative to the first byte of the RMR. The total offset relative
1018 to the first byte of the underlying LMR is obtained by adding to this byte offset the difference
1019 between the *addr* attributes of the RMR and the LMR established at RMR Link time.

- 1020 See also **Absolute Addressing**.
- 1021 **Reliable Connected (RC)**
- 1022 A Transport Service Type in which an Endpoint is associated with only one other Endpoint, such
1023 that messages transmitted from one Endpoint are reliably delivered to the other Endpoint,
1024 uncorrupted in the absence of errors and in the order defined by the Reliable Connection
1025 ordering rules. As such, each Endpoint is said to be “connected” to the opposite Endpoint.
- 1026 **Reliable Connection**
- 1027 A Connection type such that data of posted DTOs of either Endpoint of the Connection reliably
1028 arrives at the other Endpoint of the Connection uncorrupted in the absence of errors and in the
1029 order defined by the Reliable Connection ordering rules.
- 1030 **Remote Direct Memory Access (RDMA)**
- 1031 A method of accessing memory on a remote system without interrupting the processing of the
1032 CPU(s) on that system.
- 1033 **Remote Memory Region (RMR)**
- 1034 A contiguously addressable area of arbitrary size within a linear address space enabling remote
1035 access, or a placeholder for such an area. An RMR in unlinked state can be linked to a section of
1036 an LMR through an RMR Link operation. An RMR in linked state can be unlinked through an
1037 RMR Unlink operation.
- 1038 **RMR**
- 1039 See **Remote Memory Region**.
- 1040 **RMR Context**
- 1041 An opaque identifier generated by the Implementation to represent a contiguous memory region.
1042 Used by remote Consumers in RDMA operations that target this region.
- 1043 **RMR Link**
- 1044 The memory management operation initiated by *it_rmr_link*, which associates an RMR with a
1045 section of an LMR and thereby enables remote access to that section.
- 1046 **RMR Unlink**
- 1047 The memory management operation initiated by *it_rmr_unlink*, which destroys the association of
1048 an RMR with a section of an LMR.

1049	RMR Triplet
1050 1051	A type used to specify a section of a Remote Memory Region. An RMR Triplet specifies an RMR Handle, an address, and a length.
1052	RNIC
1053 1054	Hardware or software implementation of RDMAP/DDP over MPA/TCP/IP or RDMAP/DDP over SCTP/IP.
1055	RQ
1056	See Receive Queue .
1057	SE
1058	See Software Event .
1059	Send
1060 1061	The Data Transfer Operation (DTO) that is initiated by the <i>it_post_send</i> or <i>it_post_sendto</i> routine.
1062	Send Queue (SQ)
1063 1064	An internal queue of an Endpoint on which Send, RDMA Write, RDMA Read, and RMR (Un)Link Work Requests are posted.
1065	Service Reply
1066	See UD Service Reply .
1067	Service Request
1068	See UD Service Request .
1069	Service Request Handle
1070 1071 1072	An IT Object that identifies a request to translate a Connection Qualifier associated with a provider of a service into a Path, Endpoint ID, and Endpoint Key that can be used to communicate with that provider via the Unreliable Datagram Transport Service Type.
1073	Service Type
1074 1075	A class of transport service defining basic attributes of the communication; e.g., connected or unconnected, reliable or unreliable.

1076	SEVD
1077	See Simple Event Dispatcher .
1078	Shared Receive Queue (S-RQ)
1079	A Work Queue that can be shared by multiple Endpoints and to which Receive DTOs (Work
1080	Requests) can be posted.
1081	Simple Event Dispatcher (SEVD)
1082	An IT Object that conceptually merges Events from one or more Event Streams. These Events
1083	can be dequeued by the Consumer directly. The Consumer is notified that Events are available
1084	through the <i>it_evd_wait</i> interface, or through higher-level Notification mechanisms, such as the
1085	Aggregation Event Dispatcher. The Simple Event Dispatcher is responsible for completion of
1086	transport-specific fetching and handshaking for the Events it collects. Each Event is delivered to
1087	the Consumer exactly once.
1088	Socket Conversion
1089	See Conversion .
1090	Software Event (SE)
1091	An Event generated for a Simple Event Dispatcher by the Consumer, as opposed to those
1092	generated by the Interface Adapter.
1093	Solicited Wait
1094	A modifier for Send DTOs submitted to an Endpoint of the Connection. It specifies that the
1095	completion of matching Receive DTOs on the remote side of the Connection generate
1096	Notification Receive DTO Completion Events.
1097	Source
1098	The Endpoint where a message originates.
1099	Spigot
1100	A host resident device that transfers data to and from the host memory to which it is attached. A
1101	Spigot is associated with a single Interface. One or more Spigots may be associated with the
1102	same Interface.
1103	SQ
1104	See Send Queue .

- 1105 **S-RQ**
- 1106 See **Shared Receive Queue**.
- 1107 **S-RQ Low Watermark**
- 1108 The minimum number of Receive DTOs that can be sitting idle in a Shared Receive Queue
1109 without causing an S-RQ Low Watermark Event to be generated.
- 1110 **S-RQ Low Watermark Event**
- 1111 An Affiliated Asynchronous Event that is generated when the Consumer has armed the S-RQ
1112 Low Watermark mechanism for an S-RQ and the total number of Receive DTOs sitting idle in
1113 the S-RQ drops below the S-RQ Low Watermark.
- 1114 **TCP**
- 1115 See **Transmission Control Protocol**.
- 1116 **Transmission Control Protocol (TCP)**
- 1117 The IP-based, reliable, connection-oriented transport protocol.
- 1118 **The Open Group**
- 1119 *The Open Group* is a vendor-neutral and technology-neutral consortium, whose vision of
1120 Boundaryless Information Flow will enable access to integrated information within and between
1121 enterprises based on open standards and global interoperability.
- 1122 **Transport-Dependent Interface (TDI)**
- 1123 RDMA connection management service for iWARP only, allowing the Conversion of an
1124 unconnected Endpoint of the RC type and a connected socket to a connected Endpoint. The TDI
1125 includes the *it_socket_convert* and *it_ep_disconnect* calls.
- 1126 **Transport-Independent Interface (TII)**
- 1127 Unified RDMA connection management service abstraction for any RC transport supported by
1128 the IT-API. The TII includes the *it_ep_connect*, *it_listen_create*, *it_ep_accept*, *it_reject*, and
1129 *it_ep_disconnect* calls.
- 1130 **Transport Service Type**
- 1131 See **Service Type**.
- 1132 **UD**
- 1133 See **Unreliable Datagram**.

- 1134 **UD Service Reply**
- 1135 A reply message sent via the Unreliable Datagram service in response to a UD Service Request.
- 1136 **UD Service Request**
- 1137 A request message sent via the Unreliable Datagram service requesting service resolution.
- 1138 **ULP**
- 1139 See **Upper Layer Protocol**.
- 1140 **Unaffiliated Asynchronous Error**
- 1141 An error which is associated only with an Interface Adapter, rather than a specific Endpoint,
1142 EVD, or S-RQ, and which could not be reported through an immediate error or Completion
1143 Error. Unaffiliated Asynchronous Errors are reported through an Event Stream for Unaffiliated
1144 Asynchronous Events.
- 1145 **Unaffiliated Asynchronous Event**
- 1146 An Event that is associated only with an Interface Adapter, rather than a specific Endpoint,
1147 EVD, S-RQ, or Work Request.
- 1148 **Unaffiliated Event**
- 1149 See **Unaffiliated Asynchronous Event**.
- 1150 **Unbind**
- 1151 See **RMR Unlink**.
- 1152 **Unlink**
- 1153 See **LMR Unlink** and **RMR Unlink**.
- 1154 **Unreliable Datagram (UD)**
- 1155 A Transport Service Type in which an Endpoint may transmit and Receive single-packet
1156 messages to/from any other Endpoint that supports that Service Type. Ordering and delivery are
1157 not guaranteed, and the Receiver may drop delivered packets.
- 1158 **Upper Layer Protocol (ULP)**
- 1159 A protocol taking advantage of RDMA services as provided by the IT-API.

- 1160 **VIA**
- 1161 See **Virtual Interface Architecture**.
- 1162 **Virtual Interface Architecture (VIA)**
- 1163 One of the transports that the IT-API supports [[VIA-V1.0](#)].
- 1164 **V-RNIC**
- 1165 Verbs-enabled RNIC (equivalent to RI defined in [[VERBS-RDMAC](#)]).
- 1166 **V-RNIC/IETF**
- 1167 V-RNIC compliant with [[MPA-IETF](#)] and exposing Verbs that are slightly extended relative to
1168 [[VERBS-RDMAC](#)] for MPA Marker/CRC control according to [[MPA-IETF](#)]. A V-RNIC/IETF
1169 can be Permissive or Non-permissive [[INTEROP-IETF](#)].
- 1170 **V-RNIC/RDMAC**
- 1171 V-RNIC compliant with [[MPA-RDMAC](#)] and [[VERBS-RDMAC](#)].
- 1172 **Wide RMR**
- 1173 An RMR that can be referenced or accessed via all Endpoints in the Protection Zone of the
1174 RMR. See also **Narrow RMR**.
- 1175 **Work Queue (WQ)**
- 1176 A queue on which Work Requests are posted. Endpoints commonly have two internal Work
1177 Queues: a Send Queue and a Receive Queue. Alternatively, an Endpoint may be associated with
1178 a Shared Receive Queue, in which case the Endpoint has no internal Receive Queue. Send,
1179 RDMA Read, RDMA Write, RMR Link, and RMR Unlink Work Requests are posted to an
1180 Endpoint's Send Queue. Receive Work Requests are posted either to an Endpoint's Receive
1181 Queue or to a Shared Receive Queue.
- 1182 **Work Request (WR)**
- 1183 A request to perform an operation, posted by the Consumer to an Endpoint. Work Requests
1184 include DTOs, RMR Link, and RMR Unlink operations.
- 1185 **WQ**
- 1186 See **Work Queue**.
- 1187 **WR**
- 1188 See **Work Request**.

1189 4 Global Behavior

1190 This section describes certain general aspects of the behavior of the IT-API. Behavior described
1191 in this section is applicable to all IT-API interfaces except where noted explicitly in individual
1192 reference pages.

1193 4.1 Asynchronous versus Synchronous APIs

1194 Many IT-API operations are made available through *asynchronous APIs* comprising an
1195 *asynchronous API request call* for requesting an operation and a *Notification mechanism* for
1196 asynchronously notifying the Consumer of the operation's completion. An asynchronous API
1197 request call may return success or failure, with call success indicating only that the request has
1198 been accepted for later execution. The Consumer can be notified subsequently through an Event
1199 whether the operation completed successfully or in error. Each Event surfaced by the IT-API has
1200 an associated Event object that contains information about the disposition and status of the
1201 Event. The calls for posting Work Requests to an Endpoint form one particularly important set
1202 of asynchronous API request calls.

1203 A few IT-API operations are made available through *synchronous APIs* comprising a single API
1204 call. Such a *synchronous API call* may return success or failure, with call success indicating that
1205 the operation was completed successfully.

1206 API calls for posting Work Requests to an Endpoint (e.g., *it_post_send*) or for reaping Events
1207 (e.g., *it_evd_dequeue*) typically return without blocking. These calls are often implemented
1208 through a *fast path* to or from underlying hardware or firmware.

1209 Some synchronous API calls as well as some asynchronous API request calls (such as
1210 *it_ep_connect*) may block the caller's execution waiting for a locally generated event or
1211 condition. A few synchronous API calls (such as *it_evd_wait* and *it_get_pathinfo*) may block the
1212 caller's execution waiting for a remotely generated event. All routines that can block waiting for
1213 a remotely generated event are explicitly noted in the appropriate reference pages.

1214 4.2 Thread Safety

1215 The IT-API supports multi-threaded applications through a variety of different thread safety
1216 models. The basic issue in thread-safety is to provide mutually exclusive access to a shared
1217 resource being accessed by multiple threads executing in parallel. Within a multi-threaded
1218 application, it is common to share data resources. A common solution to provide mutual
1219 exclusion is to serialize potentially conflicting accesses into a well-ordered succession of
1220 executions. For example, if two callers make a call at the same time the results of the two
1221 executions are as if the two calls were serialized in arbitrary order. Normally ensuring mutual
1222 exclusion introduces some performance reduction, and so is only desirable when needed.

1223 To support multi-threaded applications, the IT-API defines three models of thread safety.
 1224 Briefly, these three models are described as:

- 1225 • Strongly Thread-Safe
- 1226 • Efficiently Thread-Safe
- 1227 • Not Thread-Safe

1228 While none of the three models is completely thread-safe, each provides a different degree of
 1229 thread-safety. Each of the models is appropriate for a different Consumer programming model.
 1230 The thread safety models are described in more detail below.

1231 Implementations of the IT-API may support one or more thread safety models. Which model or
 1232 models a particular Implementation supports, and how that thread safety support is
 1233 communicated to the Consumer, is beyond the scope of the IT-API. One potential mechanism
 1234 would have the Implementation vendor associate a specific thread safety model with a particular
 1235 library Implementation of the IT-API. In this scheme, different libraries would support different
 1236 thread safety models. The IT-API defines the thread safety models described in Table 1.

Model	Description
Strongly Thread Safe	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. 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.
Efficiently Thread Safe	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. 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. 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.
Not Thread-Safe	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.

1237 **Table 1: Thread Safety Models**

1238 For the Strongly Thread-Safe and Efficiently Thread-Safe models, the IT-API defines thread
 1239 safety on a routine-by-routine basis, and applies thread safety to IT Objects according to specific
 1240 rules.

1241 Thread-safety means that the routine:

1242 1. Provides well-defined results without imposing any restrictions on other IT-API
1243 routines called by other threads in the system

1244 2. Provides well-defined results without regard to which other IT-API routines
1245 currently have threads of execution within them

1246 Not thread-safe means that the results of the routine can possibly be *not* well-defined if another
1247 in-progress not thread-safe routine is called with the same primary (i.e., first) call argument, and
1248 none of the calls is an object destructor routine. (For *it_rmr_link* the *rmr_handle*, *lmr_handle*,
1249 and *ep_handle* arguments must be treated as primary call arguments for thread-safety purposes.
1250 For *it_rmr_unlink* the *rmr_handle* and *ep_handle* arguments, and the *lmr_handle* of the LMR to
1251 which the RMR is linked must be treated as primary call arguments for thread-safety purposes.)

1252 This definition of thread-safe and not thread-safe routines allows simultaneous execution of not
1253 thread-safe calls involving different instances of an IT Object (e.g., two different EPs) or
1254 involving IT Objects that are related (e.g., an EVD and an EP), so long as the primary call
1255 argument is not the same for the two routines.

1256 For the not-thread-safe model, the Implementation does not provide thread-safety for any data
1257 structures whatsoever.

1258 The IT-API applies these thread safety models on a routine-by-routine basis. IT-API routines can
1259 be classified into five groups according to their basic function. These groups determine their
1260 thread-safety under each thread safety model:

- 1261 • Non-performance critical routines: These routines create objects and manage and query
1262 the state of objects that do interact with the I/O code paths.
- 1263 • Event harvesting and posting routines: These routines retrieve Events from the
1264 Implementation, and invoke software Events.
- 1265 • Performance critical routines: These routines invoke Data Transfer Operations and RMR
1266 Operations.
- 1267 • Object management routines: These routines modify and query the state of objects that
1268 interact with the I/O code paths.
- 1269 • Object destructor routines: These routines free and/or destroy IT Objects.

1270 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_evd_callback_attach</i> <i>it_evd_callback_detach</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_lmr_create_unlinked</i> <i>it_lmr_link</i> <i>it_make_rdma_addr_absolute</i> <i>it_make_rdma_addr_relative</i> <i>it_mem_map</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_atomic</i> <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_send_and_unlink</i> <i>it_post_sendto</i> <i>it_rmr_link</i>	Thread-safe	Not thread-safe	

<i>it_rmr_unlink</i> <i>it_ud_service_reply</i> <i>it_ud_service_request</i>			
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_flush_to_mem</i> <i>it_lmr_refresh_from_mem</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_lmr_unlink</i> <i>it_mem_unmap</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

1273 **4.3 Signal Handlers**

1274 IT-API interfaces are not required to be safely executable from within a signal handler
1275 invocation.

1276 **4.4 Fork Semantics**

1277 Use of the POSIX *fork* family of calls is supported within the IT-API with the following
1278 semantics. After a fork call, the parent process' references to IT Objects are unchanged, and it
1279 may continue to use the references as it had before the *fork* call.

1280 The child process' IT Object references are invalid following the *fork* call (with one exception
1281 for file descriptors discussed below).

1282 The one exception to this behavior for the child is for file descriptors that were associated with
1283 EVDs before the *fork* call occurred. The Implementation supports the child's use of the *close* call
1284 to close the file descriptor.

1285 **4.5 Exec Semantics**

1286 The process' IT Object references are invalid following the POSIX *exec* family of calls.

1287 **4.6 Exit Semantics**

1288 Following an implicit or explicit call to the POSIX *exit*, all IT Objects associated with the
1289 process are destroyed and all references to them are invalid.

1290 **4.7 Error Handling**

1291 Error Notification is provided to Consumers of the IT-API in two ways:

- 1292 1. As error return values to interface calls
1293 2. As Events containing error status information for the earlier request

1294 In general, interface calls return an immediate error when a call argument is invalid or
1295 incompatible with a condition relevant to the request. However, some errors of this type are
1296 determined by the transport layer stack executing below the IT-API, and in this case the
1297 Consumer is notified of call parameter-related errors through an Event.

1298 For Work Requests posted to an RC-type Endpoint in the IT_EP_STATE_CONNECTED state,
1299 a completion status other than IT_DTO_SUCCESS will break the Connection by moving the
1300 Endpoint to the IT_EP_STATE_NONOPERATIONAL state and deliver an
1301 IT_CM_MSG_CONN_BROKEN_EVENT Event to the Connect EVD of the Endpoint.

1302 An Affiliated Asynchronous Error that is associated with an Endpoint will move that Endpoint to
1303 the IT_EP_STATE_NONOPERATIONAL state; if this state transition was from the

1304 IT_EP_STATE_CONNECTED state, an IT_CM_MSG_CONN_BROKEN_EVENT Event will
1305 be delivered to the Connect EVD of *ep_handle*.

1306 Once the Connection is broken, all outstanding and in-progress operations on the Connection
1307 will complete with an error status.

1308 Any posting to an Endpoint that is in the IT_EP_STATE_NONOPERATIONAL state will be
1309 flushed with completion status (*it_dto_status_t*) set to IT_DTO_ERR_FLUSHED.

1310 **4.8 IT Handle Management**

1311 IT-API interfaces that create an IT Object return an opaque type reference Handle that the
1312 Consumer can use in subsequent IT-API calls. It is the Consumer's responsibility to track these
1313 Handles, and use them appropriately. The Implementation will make its best effort to detect
1314 improper use of Handles by the Consumer and will return an invalid Handle immediate error
1315 whenever possible. However, it may not always be possible for the Implementation to detect
1316 improper use of a Handle, and improper use may manifest for the Consumer in Completion
1317 Errors for Work Requests, a broken Connection, data corruption, or other more severe errors.

1318 **4.9 Output Parameters**

1319 Many of the routines in the IT-API take an output parameter pointer that is used by the
1320 Implementation to copy information into an address specified by the Consumer. An example of
1321 such a routine is *it_evd_query*, which copies the state associated with an EVD into the data
1322 structure pointed to by the *params* pointer passed by the Consumer. The Implementation will not
1323 attempt to detect invalid pointers for output parameters, and passing such a pointer may result in
1324 severe errors such as data corruption or program termination.

1325 **4.10 Privileged Consumer**

1326 IT-API 2.1 adds the concept of a Privileged Consumer to the API. A Privileged Consumer is
1327 entitled to execute operations not permitted to a normal Consumer. Such operations include use
1328 of the Direct LMR Handle and use of the LMR Link capabilities.

1329 The definition of Privilege is O/S-dependent and an IT-API 2.1 implementation on a specific
1330 O/S will enforce Privilege as required by the O/S. When a non-Privileged Consumer attempts a
1331 Privileged operation, the IT-API calls will either return an immediate error or will generate a
1332 completion error.

1333 Only a Privileged Consumer can create a Privileged Mode Endpoint. A Privileged Consumer
1334 does not need to use Privileged Mode Endpoints unless the Privileged Consumer wishes to
1335 initiate Privileged operations.

1336 **4.11 Memory Management**

1337 IT-API 2.1 adds the concept of memory address space support. Such address spaces include:
1338 user virtual address space, kernel virtual address space, physical address space, and bus address
1339 space. The absolute address range associated with an LMR or RMR is considered a virtual
1340 address space, regardless of whether this virtual address space is known to the OS.

1341 In order to guarantee 8-byte alignment of data accessed remotely by Atomic operations, IT-API
1342 2.1 requires that the 3 least significant address bits of two addresses (in any address space)
1343 referring to the same byte of memory be identical. For a number of OSes, the number of
1344 matching least significant address bits between two such addresses will in fact be 12 or greater.

1345 5 Connection Management

1346 5.1 Overview

1347 The IT-API provides RDMA services for several underlying RDMA transports, including
1348 InfiniBand and iWARP². The specifics of these RDMA transports as well as different
1349 application requirements led to the design of two different Connection management interfaces
1350 for Reliable Connected (RC) RDMA transports, as introduced below.

1351 The Transport-Independent Interface (TII) provides a unified RDMA Connection management
1352 service abstraction for any RC transport supported by the IT-API and is the legacy Connection
1353 management API from IT-API Version 1.0. The TII includes the *it_ep_connect*, *it_listen_create*,
1354 *it_ep_accept*, *it_reject*, and *it_ep_disconnect* calls³. The TII allows Consumers to negotiate the
1355 *Inbound RDMA Read Queue Depth* (IRD) and *Outbound RDMA Read Queue Depth* (ORD)
1356 parameters and to exchange Private Data during Connection establishment. Consumers are
1357 expected to preconfigure IRD and ORD prior to calling *it_ep_connect* or *it_ep_accept* by setting
1358 the corresponding Endpoint attributes *rdma_read_ird* and *rdma_read_ord*, respectively. From
1359 the Consumer's viewpoint, the RDMA service is available immediately after Connection
1360 establishment. No Lower Layer Protocol (LLP) handle is exposed to the Consumer. The usage of
1361 the TII is described in *Transport-Independent Interface*. Implementation aspects for the TII on
1362 iWARP are described in Appendix B.

1363 The Transport-Dependent Interface (TDI) is available for iWARP only and allows the
1364 Conversion of an unconnected Endpoint of the RC type and a connected socket into a connected
1365 Endpoint. The TDI includes the *it_socket_convert* and *it_ep_disconnect* calls. It enables
1366 Consumers to exchange an arbitrary (but non-zero) amount of streaming-mode data via a TCP
1367 socket (and possibly via an SCTP socket in the future) prior to performing a Conversion, which
1368 is a requirement for several iWARP-specific ULPs. This prior exchange of streaming-mode data
1369 occurs outside the IT-API. In contrast to Connection establishment with the TII, the
1370 *it_socket_convert* call neither supports the negotiation of IRD and ORD parameters, nor the
1371 exchange of Private Data. The main reason for this simplification is that the peers have an
1372 opportunity to negotiate IRD and ORD, to select the corresponding Endpoint attributes
1373 *rdma_read_ird* and *rdma_read_ord*, and to exchange Private Data prior to Conversion;
1374 moreover, they have an opportunity to modify ORD after the Conversion⁴; i.e., while already in
1375 RDMA mode. The TDI also allows the Consumer more detailed control of the Connection
1376 establishment process, including whether or not the MPA Request/Reply startup sequence is
1377 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].

1378 the IT-API, interoperating with RNICs that have dissimilar feature sets, etc. The usage of the
1379 TDI is described in *Transport-Dependent Interface (iWARP-only)*. Implementation aspects for
1380 the TDI are described in Appendix B.

1381 **5.1.1 IRD/ORD Negotiation**

1382 Negotiation of IRD/ORD between RDMA Initiator and RDMA Responder should be done by a
1383 Consumer intending to use RDMA Read operations in their programming model. The Inbound
1384 RDMA Read Queue Depth, IRD, represents the maximum number of outstanding inbound
1385 RDMA Read operations supported for an Endpoint. The Outbound RDMA Read Queue Depth,
1386 ORD, represents the maximum number of emitted outbound RDMA Read operations the
1387 Endpoint may have outstanding.

1388 IRD and ORD must be negotiated such that each side of a Connection uses compatible values. If
1389 ORD on one side of a Connection exceeds IRD on the other, there is a potential for the
1390 Connection to be terminated if more RDMA Reads are issued than the advertised IRD can
1391 tolerate⁵.

1392 The IT-API supports two models of IRD/ORD negotiation: API-supported and Consumer-
1393 defined (i.e., via their own ULP in Private Data). The former, API-supported model is supported
1394 only by the TII. The latter approach, Consumer-defined, is needed for the TDI and optionally
1395 used by the Consumer for the TII (by disabling the API-supported mechanism).⁶ For the TII with
1396 IRD/ORD use suppressed, the Consumer must define their own scheme to convey the values.

1397 With either model, the RDMA Initiator should create and configure an Endpoint with desired
1398 IRD/ORD values prior to use. At the Responder, IRD/ORD values are also chosen per Endpoint
1399 with potentially differing criteria than at the Initiator.

1400 If the local ORD exceeds the remote IRD, then the local ORD must be reduced. If the local IRD
1401 exceeds the remote ORD, then resources can be saved by reducing the local IRD; whether IRD
1402 reduction is possible depends on the capabilities of the underlying device (see *it_ia_info_t*).

1403 The RDMA Initiator's IRD/ORD values (IRD_i/ORD_i) must be conveyed to and examined by the
1404 RDMA Responder.

1405 The RDMA Responder's IRD (IRD_r) can be reduced from its preset value to a value not
1406 exceeding ORD_i.

1407 The RDMA Responder's ORD (ORD_r) should be set to the minimum of its preset value and
1408 IRD_i.

1409 Finally, the RDMA Initiator adjusts ORD_i and optionally IRD_i after receiving the RDMA
1410 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.

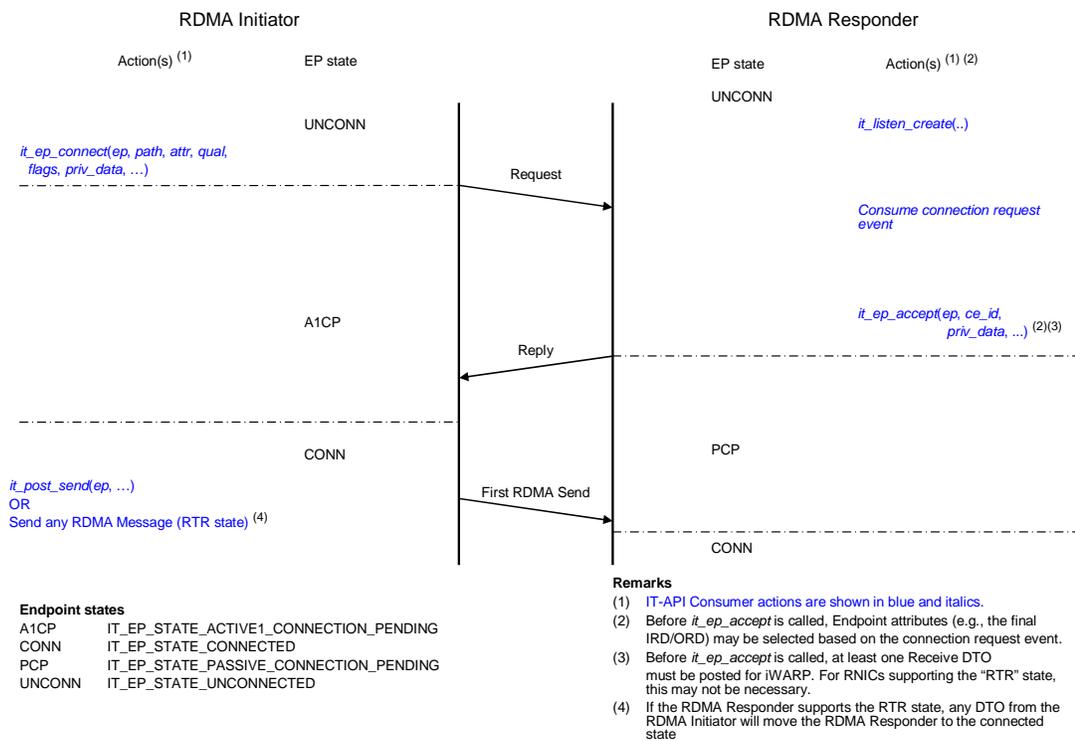
1411 **5.2 Transport-Independent Interface**

1412 **5.2.1 Overview**

1413 For the Transport-Independent Interface (TII), the Consumer calling *it_ep_connect* and the
 1414 corresponding side of the Connection is referred to as the *RDMA Initiator*, while the Consumer
 1415 calling *it_listen_create* followed by *it_ep_accept* (or *it_reject*) and the corresponding side of the
 1416 Connection is referred to as the *RDMA Responder*.

1417 The Endpoint finite-state machine uses the active (passive) Endpoint states for the RDMA
 1418 Initiator (Responder). See the *it_ep_state_t* reference page.

1419 Connection establishment for the TII is illustrated in Figure 1.



1420
 1421 **Figure 1: Connection Establishment with the TII**

1422 The RDMA Responder uses *it_listen_create* to create a Listen Point, which causes the (API)
 1423 Implementation to prepare for receiving an RDMA Connection Request. The RDMA Initiating
 1424 Consumer uses *it_ep_connect* to connect an Endpoint of the RC type, which causes the
 1425 Implementation to send an RDMA Connection Request.

1426 The Implementation of the RDMA Initiator generates and sends an RDMA Connection Request
1427 containing the local Endpoint IRD and ORD parameters⁷ and, amongst other things, the Private
1428 Data provided by the Consumer with *it_ep_connect*. After receiving a valid RDMA Connection
1429 reply from the Responding side, the Implementation generates a Connection established event
1430 for the RDMA Initiator providing the remote Consumer's Private Data.

1431 The Responding side Listen Point expects an incoming RDMA Connection Request. Upon
1432 receiving a valid request, it generates a Connection Request event for the Consumer containing
1433 the remote Endpoint's IRD and ORD parameters⁸ as well as the remote Consumer's Private
1434 Data. The Connection Request event is identified by the Connection Establishment ID
1435 (*cn_est_id*). The Consumer on the RDMA Responder side now invokes either *it_ep_accept* or
1436 *it_reject*.

1437 Calling *it_ep_accept* causes the Implementation to associate the Connection Establishment ID
1438 with an Endpoint. It then causes the Implementation to convey an RDMA Connection reply
1439 including the Private Data provided by the Consumer to the Initiating side. Finally, on receipt of
1440 a first RDMA message (see next paragraph), the Implementation generates a Connection
1441 established event for the Consumer and transitions the Endpoint into the
1442 IT_EP_STATE_CONNECTED state.

1443 5.2.2 ULP Constraints for iWARP

1444 Consumers using iWARP AV-RNICs must satisfy the following additional ULP constraints:

- 1445 • The RDMA Responder must post at least one Receive DTO prior to the calling
1446 *it_ep_accept*.
- 1447 • After reaping the Connection established event, the RDMA Initiator must post a Send
1448 DTO so that a Connection established Event will be generated for the RDMA Responder.

1449 In order to minimize such transport-dependent ULP constraints, the IT-API offers an alternative
1450 for RNICs that implement an extended QP state machine (i.e., they define an "RTR" state as
1451 footnoted in the above figure – see Appendix A for details). The Consumer may determine
1452 whether the underlying RNIC supports the extended QP state machine via the
1453 *extended_iwarp_qp_states* attribute found in the *it_ia_info_t* structure retrieved from
1454 *it_ia_query*.

1455 Consumers using iWARP AV-RNICs with "RTR" state support must satisfy the following ULP
1456 constraint only:

- 1457 • After reaping the Connection established event, the RDMA Initiator must post an RDMA
1458 or Send DTO so that a Connection established Event will be generated for the RDMA
1459 Responder.

1460 5.2.3 API-Supported IRD/ORD Negotiation

1461 Prior to calling *it_ep_connect*, the RDMA Initiator preconfigures the Endpoint to be used with
1462 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.

1463 When *it_ep_connect* is called, IRDi/ORDi are filled into the Connection Request message (for
1464 InfiniBand, the CM REQ; for iWARP the MPA Request's IRD/ORD Header found in Private
1465 Data).

1466 When the Connection Request is received by the IT-API Listen Point (RDMA Responder side),
1467 a Connection Request event is generated for the ULP reporting the peer's IRDi/ORDi in the
1468 *rdma_read_ird* and *rdma_read_ord* fields respectively (see *it_cm_req_events*).

1469 The RDMA Responder now updates its IRDr/ORDr values, preferably through the algorithm:

```
1470 IRDr := MIN(IRDr, ORDi)  
1471 ORDr := MIN(ORDr, IRDi)
```

1472 and updates its Endpoint (which is still in the IT_EP_STATE_UNCONNECTED state) with the
1473 new IRDr/ORDr via *it_ep_modify*. Modification of IRD/ORD depends on the capabilities of the
1474 underlying device. All devices will support decrease of ORD. The *it_ia_info_t* attributes,
1475 *rdma_read_ird_modifiable* and *rdma_read_ord_increasable*, define any additional permitted
1476 operations on IRD/ORD for the RNIC.

1477 When the RDMA Responder calls *it_ep_accept* for the previous Connection Request,
1478 IRDr/ORDr are filled into the Connection Reply (for InfiniBand, the CM REP; for iWARP, the
1479 MPA Reply's IRD/ORD Header).

1480 When the Connection Reply is received by the RDMA Initiator side implementation, a
1481 Connection established event is generated reporting the peer's IRDr/ORDr.

1482 The RDMA Initiator may now update its IRDi/ORDi values, preferably through the algorithm:

```
1483 IRDi := MIN(IRDi, ORDr)  
1484 ORDi := MIN(ORDi, IRDr)
```

1485 and update its EP attributes *rdma_read_ird* and *rdma_read_ord* accordingly, as permitted by the
1486 underlying device.

1487 The Connection established event generated for the RDMA Responder contains *rdma_read_ird*
1488 and *rdma_read_ord* fields that represent the initial IRD/ORD sent by the Initiator. Since the
1489 Initiator optionally may have subsequently altered the values, the Consumer cannot rely on their
1490 contents.

1491 **5.2.4 Transport-Dependent Attributes**

1492 *5.2.4.1 iWARP*

1493 **5.2.4.1.1 Connection Qualifiers**

1494 Only the IT_IANA_PORT or IT_IANA_LR_PORT Connection Qualifiers are supported for the
1495 iWARP Transport.

1496 **5.2.4.1.2 Service Types**

1497 Only the IT_RC_SERVICE service type is supported for the iWARP Transport.

1498 **5.2.4.1.3 Handshake**
1499 The IT-API supports only two-way Connection establishment for iWARP.

1500 **5.3 Transport-Dependent Interface (iWARP-Only)**

1501 **5.3.1 Overview**

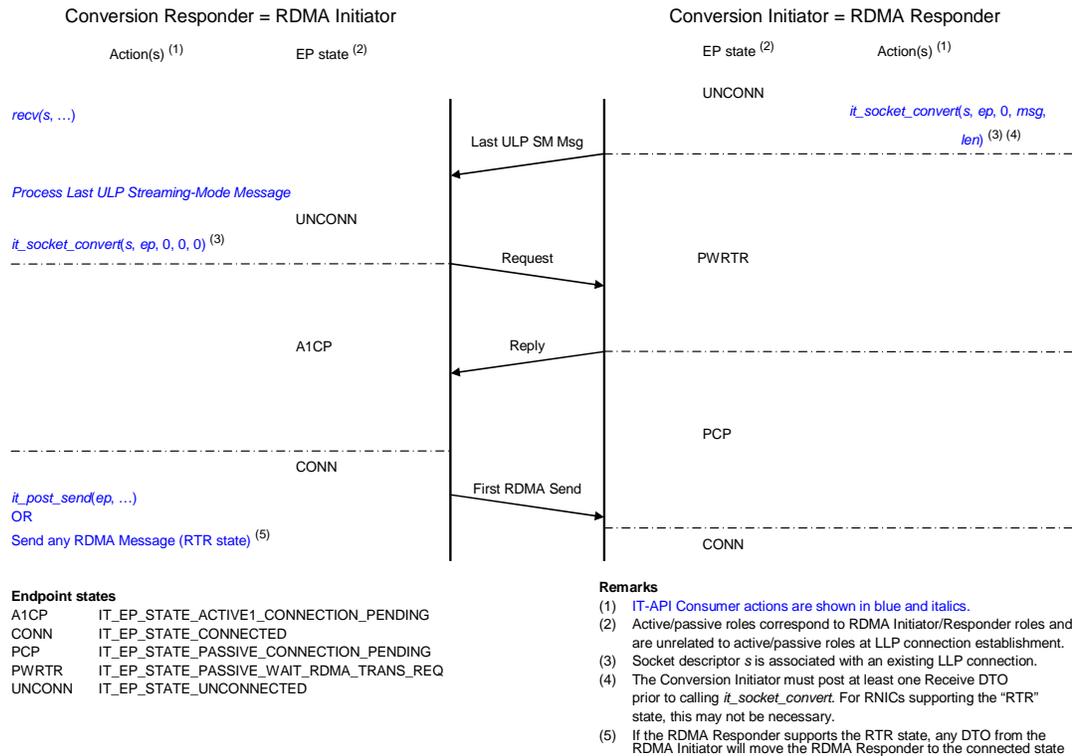
1502 The *Transport-Dependent Interface* (TDI) is defined to support iWARP-specific ULPs.

1503 For the *TDI*, the Consumer calling *it_socket_convert* with a Last ULP Streaming-Mode Message
1504 of length greater than zero is referred to as the *Conversion Initiator*, while the Consumer calling
1505 *it_socket_convert* with a Streaming Mode Message length of zero is referred to as the
1506 *Conversion Responder*.

1507 As for the *TII*, the Endpoint finite-state machine uses the active (passive) Endpoint states for the
1508 RDMA Initiator (Responder). In order to avoid an inconsistency in the meaning of Endpoint
1509 states between *TII* and *TDI*, the *TDI* uses the convention that the Conversion Initiator and the
1510 corresponding side of the Connection is in the RDMA Responder role, while the Conversion
1511 Responder and the corresponding side of the Connection is in the RDMA Initiator role. See the
1512 *it_socket_convert* reference page for the finite-state machine diagrams.

1513

The Conversion process is illustrated below.



1514

1515

Figure 2: Conversion Process with MPA Startup (TDI)

1516

1517

1518

1519

1520

1521

1522

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.

1523

1524

1525

1526

1527

1528

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.

1529

1530

1531

1532

1533

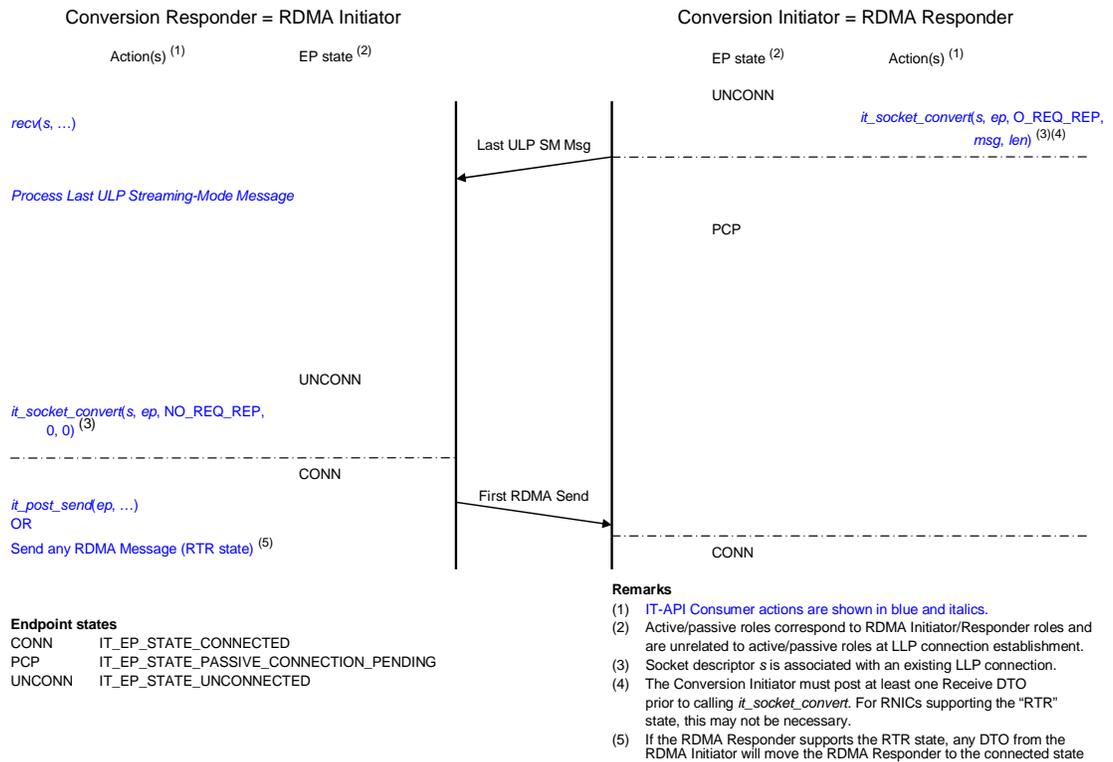
The Conversion Responder expects to receive the Last ULP Streaming-Mode Message. Upon receiving that message, it invokes *it_socket_convert*, passing an unconnected Endpoint of the RC type, a socket handle corresponding to an established LLP Connection, a *flags* argument of zero, and a Streaming Mode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

1534
1535
1536
1537
1538

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 generates a Connection established event for the Consumer with IRD/ORD set to zero and the Private Data present bit clear, and transitions the Endpoint into the IT_EP_STATE_CONNECTED state.

1539
1540
1541
1542

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 IT_SC_NO_REQ_REP in the *it_socket_convert* call. The Conversion process with MPA Startup suppression in case of an RNIC without RTR state is shown below.



1543
1544

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

1545 **5.3.2 IRD/ORD Negotiation**

1546 For the TDI (*it_socket_convert*), IRD/ORD must be negotiated either before or after Socket
1547 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_callback_attach</i>	Attach a callback routine to an EVD.
<i>it_evd_callback_detach</i>	Detach a callback routine from an EVD.
<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_create_unlinked</i>	Create an unlinked Local Memory Region.
<i>it_lmr_flush_to_mem</i>	Make memory changes visible to an incoming RDMA Read or Atomic operation.
<i>it_lmr_free</i>	Destroy a Local Memory Region.
<i>it_lmr_link</i>	Link an unlinked 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_refresh_from_mem</i>	Make effects of an incoming RDMA Write or Atomic operation visible.
<i>it_lmr_unlink</i>	Unlink a Local Memory Region.
<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_mem_map</i>	Map a Virtual Address to I/O Address Space.
<i>it_mem_unmap</i>	Undo a Virtual to I/O Address mapping.
<i>it_post_atomic</i>	Post an Atomic Operation DTO to an RC-type Endpoint.
<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_send_and_unlink</i>	Post a Send DTO to an RC-type Endpoint and Unlink a remote RMR.
<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_create</i>	Create a Remote Memory Region (RMR).
<i>it_rmr_free</i>	Destroy a Remote Memory Region.
<i>it_rmr_link</i>	Post request to Link a Remote Memory Region to a memory range.
<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.

1549

1550

it_address_handle_create()

1551

1552 NAME

1553 `it_address_handle_create` – create an Address Handle

1554 SYNOPSIS

```
1555 #include <it_api.h>
1556
1557 it_status_t it_address_handle_create(
1558     IN          it_pz_handle_t      pz_handle,
1559     IN const    it_path_t           *destination_path,
1560     IN          it_ah_flags_t       ah_flags,
1561     OUT         it_addr_handle_t     *addr_handle
1562 );
1563
1564 typedef enum {
1565     IT_AH_PATH_COMPLETE = 0x1
1566 } it_ah_flags_t;
```

1567 APPLICABILITY

1568 `it_address_handle_create` is applicable only to the UD service type.

1569 DESCRIPTION

1570 *pz_handle* Handle for the Protection Zone to be associated with the created Address
1571 Handle. This implicitly identifies the Interface Adapter that the Address
1572 Handle will be associated with.

1573 *destination_path* The Path to use to create the Address Handle.

1574 *ah_flags* The bitwise OR of the set of operation modifier flags specified below.

1575 *addr_handle* Returned datagram Address Handle.

1576 `it_address_handle_create` creates an Address Handle, which is used when performing a Send
1577 DTO on an Unreliable Datagram Endpoint.

1578 The Protection Zone to associate with the newly created Address Handle is specified by
1579 *pz_handle*. An Address Handle can only be used to post a Send DTO on an Unreliable Datagram
1580 Endpoint that has a matching Protection Zone.

1581 The Source and Destination address information necessary to create the Address Handle are
1582 specified in the *destination_path* parameter. The Path can either be completely or incompletely
1583 specified. A completely specified Path is one that contains all the necessary information to create
1584 the Address Handle without the Implementation needing to consult a database of Path records.
1585 An incompletely specified Path does not contain enough information to create the Address
1586 Handle directly, but does contain enough information that the Implementation can determine the
1587 rest of the information needed by consulting a database of Path records. The Consumer should
1588 set the IT_AH_PATH_COMPLETE bit in *ah_flags* if the Path is completely specified, or clear it
1589 if it is incompletely specified.

1590 A completely specified Path that the Consumer can use to access a given remote network
1591 Endpoint can be obtained using the *it_get_pathinfo* routine. A Path returned from

1592 *it_get_pathinfo* can be used without modification by *it_address_handle_create*. If the Consumer
 1593 wishes to have full control over the Path that datagrams sent using the created Address Handle
 1594 will take, they should furnish a completely specified Path.

1595 An incompletely specified Path is obtained from the Completion Event for a Receive operation
 1596 on a datagram Endpoint. (See *it_dto_events* for details.) If an incompletely specified Path is
 1597 supplied to *it_address_handle_create*, the routine will automatically choose the unspecified
 1598 components of the Path required in order to reach the intended Destination.

1599 The Consumer may also directly format the *destination_path* if they so desire. The
 1600 *destination_path* actually contains more information than is necessary to create an Address
 1601 Handle. The members of the *it_path_t* structure that are pertinent for creating an Address Handle
 1602 using a completely specified Path are listed in the table below. For each member, whether the
 1603 member is needed for an incompletely specified Path and the input modifier for the Infiniband
 1604 “Create Address Handle” verb that the member corresponds to are also identified. For a detailed
 1605 explanation of the semantics associated with each input modifier, see “Create Address Handle”
 1606 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.

1607
 1608 If the Consumer chooses to directly format the Path, it is possible that the Implementation will
 1609 decide that the resulting Path is one that the Consumer should not have access to. If so, a
 1610 permission violation error will be returned. The Implementation will generally not return such a
 1611 permission violation error if the Consumer instead uses a Path returned by *it_get_pathinfo* or
 1612 from the Completion Event for a Receive operation. (It is still possible that a permission
 1613 violation error could be returned if the network were reconfigured after the Path was returned
 1614 but before *it_address_handle_create* was furnished with that Path.)

1615 **RETURN VALUE**

1616 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

1617	IT_ERR_RESOURCES	The requested operation failed due to insufficient resources.
1618	IT_ERR_INVALID_PZ	The Protection Zone Handle (<i>pz_handle</i>) was invalid.
1619	IT_ERR_INVALID_FLAGS	The flags (<i>ah_flags</i>) value was invalid.
1620 1621	IT_ERR_NO_PERMISSION	The Consumer did not have the proper permissions to perform the requested operation.
1622 1623	IT_ERR_INVALID_SOURCE_PATH	One of the components of the Source portion of the supplied Path was invalid.
1624	IT_ERR_INVALID_SPIGOT	An invalid Spigot ID was specified.
1625 1626 1627 1628	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.
1629	SEE ALSO	
1630 1631		<i>it_get_pathinfo()</i> , <i>it_path_t</i> , <i>it_address_handle_query()</i> , <i>it_address_handle_modify()</i> , <i>it_address_handle_free()</i>

it_address_handle_free()

1632

1633 NAME

1634 `it_address_handle_free` – free an Address Handle

1635 SYNOPSIS

```
1636 #include <it_api.h>
1637
1638 it_status_t it_address_handle_free(
1639     IN it_addr_handle_t addr_handle
1640 );
```

1641 APPLICABILITY

1642 `it_address_handle_free` is applicable only to the UD service type.

1643 DESCRIPTION

1644 `addr_handle` Address Handle to free.

1645 `it_address_handle_free` removes an existing Address Handle and frees all associated underlying
1646 resources. Once `it_address_handle_free` returns, `addr_handle` can no longer be used. Consumers
1647 that wish to write code that is independent of the Implementation are therefore advised to allow
1648 all outstanding Send operations that reference an Address Handle to complete before freeing the
1649 Address Handle.

1650 RETURN VALUE

1651 A successful call returns `IT_SUCCESS`. Otherwise, an error code is returned as described below:

1652 `IT_ERR_INVALID_AH` The Address Handle (`addr_handle`) was invalid.

1653 `IT_ERR_IA_CATASTROPHE` The Interface Adapter has experienced a catastrophic error and is
1654 in the disabled state. None of the output parameters from this
1655 routine are valid. See `it_ia_info_t` for a description of the disabled
1656 state.

1657 SEE ALSO

1658 `it_address_handle_create()`, `it_address_handle_query()`, `it_address_handle_modify()`

it_address_handle_modify()

1659

1660 **NAME**

1661 `it_address_handle_modify` – modify an Address Handle

1662 **SYNOPSIS**

```

1663 #include <it_api.h>
1664
1665 it_status_t it_address_handle_modify(
1666     IN          it_addr_handle_t      addr_handle,
1667     IN          it_addr_param_mask_t  mask,
1668     IN  const   it_addr_param_t      *params
1669 );

```

1670 **APPLICABILITY**

1671 `it_address_handle_modify` is applicable only to the UD service type.

1672 **DESCRIPTION**

1673 *addr_handle* Address Handle to modify.

1674 *mask* Bitwise OR of flags for desired parameters to be modified.

1675 *params* Structure whose members contain the new parameter values.

1676 `it_address_handle_modify` changes selected attributes of the Address Handle *addr_handle*. If
 1677 this routine returns success, all requested attributes are modified. If it does not return success,
 1678 none of the requested attributes are modified.

1679 The Consumer should avoid calling this routine while a DTO that references this Address
 1680 Handle is in progress. If the Consumer fails to abide by this restriction, the Destination that the
 1681 DTO is sent to is undefined.

1682 The attributes to be modified are specified by the flags in *mask*. New values for the attributes are
 1683 specified by the corresponding fields in the structure pointed to by *params*. Each field and the
 1684 corresponding flag name that must appear in *mask* to modify the given field are shown below.
 1685 (The flag name appears in a comment to the right of the field.) Note that attributes represented
 1686 by fields of *it_addr_param_t* that are not shown below cannot be modified.

```

1687 typedef struct {
1688     ...
1689     it_path_t  path; /* IT_ADDR_PARAM_PATH */
1690     ...
1691 } it_addr_param_t;

```

1692 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 <code>it_address_handle_query</code> 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.

1694
1695
1696
1697

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.

1698
1699

RETURN VALUE

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

1700

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

1701

IT_ERR_INVALID_MASK The *mask* contained invalid flag values.

1702
1703

IT_ERR_NO_PERMISSION The Consumer did not have the proper permissions to perform the requested operation.

1704
1705

IT_ERR_INVALID_SOURCE_PATH One of the components of the Source portion of the supplied Path was invalid.

1706

IT_ERR_INVALID_SPIGOT An invalid Spigot ID was specified.

1707
1708
1709
1710

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 *it_ia_info_t* for a description of the disabled state.

1711 **SEE ALSO**

1712 *it_address_handle_create()*, *it_address_handle_query()*, *it_address_handle_free()*

it_address_handle_query()

1713

1714 NAME

1715 `it_address_handle_query` – query an Address Handle

1716 SYNOPSIS

```
1717 #include <it_api.h>
1718
1719 it_status_t it_address_handle_query(
1720     IN  it_addr_handle_t      addr_handle,
1721     IN  it_addr_param_mask_t mask,
1722     OUT it_addr_param_t      *params
1723 );
1724
1725 typedef enum {
1726     IT_ADDR_PARAM_ALL    = 0x0001,
1727     IT_ADDR_PARAM_IA    = 0x0002,
1728     IT_ADDR_PARAM_PZ    = 0x0004,
1729     IT_ADDR_PARAM_PATH  = 0x0008
1730 } it_addr_param_mask_t;
1731
1732 typedef struct {
1733     it_ia_handle_t  ia;    /* IT_ADDR_PARAM_IA */
1734     it_pz_handle_t  pz;    /* IT_ADDR_PARAM_PZ */
1735     it_path_t       path;  /* IT_ADDR_PARAM_PATH */
1736 } it_addr_param_t;
```

1737 APPLICABILITY

1738 `it_address_handle_query` is applicable only to the UD service type.

1739 DESCRIPTION

1740 `addr_handle` Address Handle to query.

1741 `mask` Bitwise OR of flags for desired parameters.

1742 `params`: Structure whose members are written with the desired parameters.

1743 `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.

1747 The table below defines the meaning of each member of the `it_addr_param_t` structure.

it_addr_param_t Member	Meaning
<code>ia</code>	The Handle for the IA with which this Address Handle is associated.
<code>pz</code>	The Handle for the Protection Zone with which this Address Handle is associated.
<code>path</code>	The Path that is associated with this Address Handle. Not all fields in the Path are relevant for an Address Handle; see the

it_addr_param_t Member	Meaning
	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 .

1748 **RETURN VALUE**

1749 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

1750	IT_ERR_INVALID_AH	The Address Handle (<i>addr_handle</i>) was invalid.
1751	IT_ERR_INVALID_MASK	The <i>mask</i> contained invalid flag values.
1752	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.
1753		
1754		
1755		

1756 **SEE ALSO**

1757 [it_address_handle_create\(\)](#), [it_address_handle_modify\(\)](#), [it_address_handle_free\(\)](#)

it_convert_net_addr()

1758

1759 NAME

1760 it_convert_net_addr – convert a Network Address from one format to another

1761 SYNOPSIS

```
1762 #include <it_api.h>
1763
1764 it_status_t it_convert_net_addr(
1765     IN  const  it_net_addr_t      *source_addr,
1766     IN  it_net_addr_type_t      addr_type,
1767     OUT  it_net_addr_t      *destination_addr
1768 );
```

1769 DESCRIPTION

1770 *source_addr* The input Network Address that is to be converted.

1771 *addr_type* The new type of address to convert the *source_addr* address to.

1772 *destination_addr* The returned Network Address.

1773 The *it_convert_net_addr* routine is used to convert one form of Network Address into another.
1774 The type of Network Address desired is specified by *addr_type*, and upon successful return from
1775 this routine *destination_addr* will contain an address of that type. If this routine does not return
1776 success, the contents of *destination_addr* are undefined.

1777 The Implementation might not support the requested Network Address conversion. If it does not,
1778 an error will be returned.

1779 The set of Network Addresses that are associated with a given Spigot is dynamic, and can
1780 change over time. (For example, a link on a switch or router could become inoperative, thus
1781 decreasing the set of Network Addresses by which a given Spigot can be reached.) There is
1782 therefore no guarantee that given the same input parameters two different invocations of
1783 *it_convert_net_addr* will return the same results. The Network Address returned by
1784 *it_convert_net_addr* is chosen from amongst the Network Addresses that match the selection
1785 criteria at the time of the call. In addition, since multiple Network Addresses of a given type can
1786 be associated with the same Spigot, the Implementation may return a different Network Address
1787 for two different invocations of *it_convert_net_addr* regardless of the state of the network.

1788 This call may block the caller's execution waiting for a remotely generated event.

1789 RETURN VALUE

1790 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

1791 IT_ERR_INVALID_ADDRESS The Network Address specified in *source_addr* was invalid.

1792 IT_ERR_INVALID_NETADDR The type of the Network Address specified in *source_addr*
1793 was not recognized.

1794 IT_ERR_INVALID_CONVERSION The requested Network Address conversion either was not
1795 supported by the Implementation, or was supported but could
1796 not be performed by the Implementation for the particular
1797 *source_addr* that was input.

1798	IT_ERR_INTERRUPT	The call was unblocked by a signal.
1799	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.

1803 **APPLICATION USAGE**

1804 When a Consumer receives an incoming Connection establishment attempt Event, the
1805 *source_addr_info* field in that Event will contain the Network Address of the initiator of the
1806 Connection establishment attempt. The type of Network Address contained within
1807 *source_addr_info* is Implementation-specific, and therefore may not be one that the Consumer
1808 wishes to deal with. The Consumer can use *it_convert_net_addr* to convert the Network Address
1809 supplied in *source_addr_info* into a preferred type.

1810 **SEE ALSO**

1811 *it_listen_create()*, *it_net_addr_t*

it_ep_accept()

1812

1813 NAME

1814 `it_ep_accept` – accept an incoming Connection Request or Connection Reply

1815 SYNOPSIS

```
1816 #include <it_api.h>
1817
1818 it_status_t it_ep_accept(
1819     IN          it_ep_handle_t          ep_handle,
1820     IN          it_cn_est_identifier_t  cn_est_id,
1821     IN const unsigned char             *private_data,
1822     IN          size_t                  private_data_length
1823 );
1824
1825 typedef uint64_t it_cn_est_identifier_t;
```

1826 APPLICABILITY

1827 `it_ep_accept` is applicable only to Endpoints created for the RC service type.

1828 DESCRIPTION

1829 `ep_handle` Local Endpoint to be bound to the Connection Request being accepted.

1830 `cn_est_id` Connection Establishment Identifier associated with the Connection
1831 Request being accepted. The `cn_est_id` is obtained from the data delivered
1832 in the Connection Request Event (IT_CM_REQ_CONN_
1833 REQUEST_EVENT). See the [it_cm_req_events](#) reference page for details.

1834 `private_data` Opaque Private Data provided by the IT_CM_MSG_CONN_
1835 PEER_REJECT_EVENT Event delivered to the Remote Consumer that
1836 will be sent to the Remote Endpoint. If the Interface Adapter does not
1837 support Private Data, `private_data_length` must be zero. When `it_ep_accept`
1838 is called on the active side of the Connection (i.e., in three-way Connection
1839 Establishment), delivery of Private Data is unreliable.

1840 `private_data_length` Length of `private_data`. This field must be 0 if the IA does not support
1841 Private Data.

1842 `it_ep_accept` accepts an incoming Connection Request Event (IT_CM_REQ_CONN_
1843 REQUEST_EVENT) or Connection accept arrival Event (IT_CM_MSG_CONN_ACCEPT_
1844 ARRIVAL_EVENT). Calling `it_ep_accept` is the last Local Consumer step in establishing an
1845 Endpoint-to-Endpoint Connection for a three-way Connection Establishment. The Consumer is
1846 notified of an established Connection by an IT_CM_MSG_CONN_ESTABLISHED_EVENT
1847 Event being delivered on the connect EVD of the Endpoint. The Event is generated on both the
1848 active and passive side of the Connection establishment. See the Communication Management
1849 Message Event ([it_cm_msg_events](#)) reference page for details.

1850 If the initial `it_ep_connect` specified two-way Connection Establishment, then `it_ep_accept` is
1851 called only on the Passive side of the Endpoint-to-Endpoint Connection.

1852 If the initial `it_ep_connect` specified three-way Connection Establishment, then `it_ep_accept` is
1853 called on both the Active and the Passive sides of the Endpoint-to-Endpoint Connection.

1854 For two-way Connection Establishment, on the Passive side the Endpoint will transition into the
1855 IT_EP_STATE_CONNECTED state when the Consumer calls *it_ep_accept*.

1856 For three-way Connection Establishment, on the Passive side the Endpoint will transition into
1857 the IT_EP_STATE_PASSIVE_CONNECTION_PENDING state when the Consumer calls
1858 *it_ep_accept*. An IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT Event will be delivered
1859 to the Active-side Consumer after the Passive side calls *it_ep_accept* and will cause the Active
1860 Endpoint eventually to transition into the IT_EP_STATE_ACTIVE2_
1861 CONNECTION_PENDING state. Subsequently, the Passive side will transition into the
1862 IT_EP_STATE_CONNECTED state after the Active side consumer calls *it_ep_accept*.

1863 *it_ep_accept* destroys the Connection Establishment Identifier, *cn_est_id*. After *it_ep_accept*
1864 returns, *cn_est_id* is no longer valid and cannot be used.

1865 The Connection Establishment process cannot be successfully completed unless the attributes of
1866 the Local and Remote Endpoints are compatible; see *it_cm_msg_events* for details. The
1867 Consumer can call *it_ep_modify* to make the Local Endpoint attributes compatible before calling
1868 *it_ep_accept*.

1869 RETURN VALUE

1870 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

1871	IT_ERR_PDATA_NOT_SUPPORTED	Private Data was supplied by the Consumer but this
1872		Interface Adapter does not support Private Data.
1873	IT_ERR_INVALID_PDATA_LENGTH	The Interface Adapter supports Private Data, but the
1874		length specified exceeded the Interface Adapter's
1875		capabilities.
1876	IT_ERR_INVALID_EP	The Endpoint Handle (<i>ep_handle</i>) was invalid.
1877	IT_ERR_INVALID_EP_STATE	The Endpoint was not in the proper state for the
1878		attempted operation. See <i>it_ep_state_t</i> reference page.
1879	IT_ERR_INVALID_EP_TYPE	The attempted operation was invalid for the Service
1880		Type of the Endpoint.
1881	IT_ERR_INVALID_CN_EST_ID	The Connection Establishment Identifier (<i>cn_est_id</i>) was
1882		invalid.
1883	IT_ERR_INVALID_EP_ATTR	The Local and Remote Endpoint attributes conflicted.
1884		Either the <i>max_message_size</i> , the number of
1885		<i>rdma_read_ird</i> , or the number of <i>rdma_read_ord</i>
1886		conflicted between the two Endpoints. This error will not
1887		be reported on the Passive-side accept of a three-way
1888		Connection Establishment.
1889	IT_ERR_EP_TIMEWAIT	The Endpoint provided to <i>it_ep_accept</i> was in the
1890		TimeWait condition, therefore the Connection could not
1891		be established. See <i>it_ep_rc_create</i> for details of the
1892		TimeWait condition.
1893	IT_ERR_IA_CATASTROPHE	The IA has experienced a catastrophic error and is in the
1894		disabled state. None of the output parameters from this

1895 routine are valid. See *it_ia_info_t* for a description of the
1896 disabled state.

1897 **ASYNCHRONOUS ERRORS**

1898 For the iWARP Transport, if the local side (Passive side, caller of *it_ep_accept*) uses an
1899 RDMAC AV-RNIC and the remote (Active side) uses a Non-permissive AV-RNIC/IETF, then
1900 the local Implementation for the side calling *it_ep_accept* will generate an IT_CM_MSG_
1901 NONPEER_REJECT_EVENT during MPA Startup, with an IT_CN_REJ_BAD_
1902 CONN_PARMS reject reason code indicating incompatible protocol versions (see [INTEROP-
1903 IETF]).

1904 **APPLICATION USAGE**

1905 Calls to routines such as *it_ep_accept*, *it_reject*, and *it_ep_disconnect* that pertain to the same
1906 Endpoint or Connection Establishment Identifier should be serialized by the Consumer. Failure
1907 to abide by this restriction may result in a segmentation violation or other error.

1908 If *it_ep_accept* returns the IT_ERR_EP_TIMEWAIT error, the Consumer can recover either by
1909 retrying the Connection Establishment after the TimeWait interval has elapsed, or by retrying the
1910 Connection Establishment using a different Endpoint that is not under a TimeWait condition.

1911 The Consumer should post at least one Receive buffer using the *it_post_recv* routine before
1912 calling *it_ep_accept*. Failure to do so can prevent a Connection from being established under
1913 certain circumstances on some transports.

1914 **SEE ALSO**

1915 *it_ep_connect()*, *it_reject()*, *it_ep_disconnect()*, *it_handoff()*, *it_ep_state_t*, *it_cm_req_events*,
1916 *it_cm_msg_events*

it_ep_connect()

1917

1918 **NAME**

1919

it_ep_connect – initiate an Endpoint Connection establishment request

1920 **SYNOPSIS**

1921

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

1922

```
it_status_t it_ep_connect(  
    IN          it_ep_handle_t          ep_handle,  
    IN const    it_path_t*              path,  
    IN const    it_conn_attributes_t*   conn_attr,  
    IN const    it_conn_qual_t*        connect_qual,  
    IN          it_cn_est_flags_t       cn_est_flags,  
    IN const    unsigned char*          private_data,  
    IN          size_t                   private_data_length  
);
```

1931

```
/* Transport-specific connection attributes for InfiniBand */  
typedef struct {
```

1932

1933

1934

1935

```
    /* Remote CM Response Timeout, as defined in the REQ  
       message for the IB CM protocol */  
    uint8_t remote_cm_timeout : 5;
```

1938

1939

```
    /* Local CM Response Timeout, as defined in the REQ  
       message for the IB CM protocol */  
    uint8_t local_cm_timeout : 5;
```

1940

1941

1942

```
    /* Retry Count, as defined in the REQ message for the  
       IB CM protocol */  
    uint8_t retry_count : 3;
```

1943

1944

1945

1946

```
    /* RNR Retry Count, as defined in the REQ message for  
       the IB CM protocol */  
    uint8_t rnr_retry_count : 3;
```

1947

1948

1949

```
    /* Max CM retries, as defined in the REQ message for  
       the IB CM protocol */  
    uint8_t max_cm_retries : 4;
```

1950

1951

1952

1953

```
    /* Local ACK Timeout, as defined in the REQ message  
       for the IB CM protocol */  
    uint8_t local_ack_timeout : 5;
```

1954

1955

1956

1957

1958

```
} it_ib_conn_attributes_t;
```

1959

1960

1961

```
/* Transport-specific connection attributes for VIA */  
typedef struct {
```

1962

1963

1964

```
    /* VIA currently has no transport-specific connection  
       attributes. A dummy entry is defined to allow ANSI  
       compilation. */
```

1965

1966

1967

```

1968         void *unused;
1969
1970     } it_via_conn_attributes_t;
1971
1972     /* Transport-specific connection attributes for iWARP */
1973     typedef struct {
1974
1975         /* iWARP currently has no transport-specific connection
1976            attributes. A dummy entry is defined to allow ANSI
1977            compilation. */
1978         void *unused;
1979
1980     } it_iwarp_conn_attributes_t;
1981
1982     /* Transport-specific connection attributes. This union is
1983        discriminated by the transport type being used to form the
1984        connection. This can be determined by examining
1985        the transport_type member in the it_ia_info_t that is
1986        associated with the IA that contains ep_handle. */
1987
1988     typedef union {
1989         it_ib_conn_attributes_t    ib;
1990         it_via_conn_attributes_t   via;
1991         it_iwarp_conn_attributes_t iwarp;
1992     } it_conn_attributes_t;
1993
1994     typedef enum {
1995         IT_CONNECT_FLAG_TWO_WAY      = 0x0001,
1996         IT_CONNECT_FLAG_THREE_WAY   = 0x0002,
1997         IT_CONNECT_SUPPRESS_IRD_ORD  = 0x0004
1998     } it_cn_est_flags_t;

```

1999 **APPLICABILITY**

2000 *it_ep_connect* is applicable only to Endpoints created for the RC service type.

2001 **DESCRIPTION**

2002	<i>ep_handle</i>	Handle of the local Endpoint.
2003	<i>path</i>	Path for Connection establishment request.
2004	<i>conn_attr</i>	The transport-specific attributes for the Connection establishment attempt.
2005	<i>connect_qual:</i>	The Connection Qualifier for which the Consumer is initiating a
2006		Connection establishment request. See <i>it_conn_qual_t</i> for more information
2007		on Connection Qualifiers.
2008	<i>cn_est_flags</i>	Bitwise OR of flags for the Connection establishment request.
2009		

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.

2010

2011
2012
2013

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

2014
2015

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

2016
2017
2018
2019
2020
2021

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

2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034

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

2035 The flags `IT_CONNECT_FLAG_THREE_WAY` and `IT_CONNECT_FLAG_TWO_WAY` of
2036 `it_cn_est_flags_t` select three-way and two-way Connection establishment, respectively, and
2037 their use in `cn_est_flags` is mutually exclusive. If the Interface Adapter attribute
2038 `three_way_handshake_support` equals `IT_TRUE`, the Consumer can select three-way or two-
2039 way Connection establishment; otherwise, two-way Connection establishment must be selected.

2040 The default behavior of the IT-API is to attempt to negotiate IRD/ORD between the active and
2041 passive sides as described in Chapter 5 where supported by the IA. For an IA where the
2042 `it_ia_info_t` values `ird_ord_ia_support` and `ird_ord_suppressible` are both `IT_TRUE`, the
2043 `IT_CONNECT_SUPPRESS_IRD_ORD` bit may be set by the Consumer to cause the IA not to
2044 perform IRD/ORD negotiation. For such an IA, if the `IT_CONNECT_SUPPRESS_IRD_ORD`
2045 bit is cleared in `cn_est_flags`, then the IRD/ORD Endpoint attributes will be negotiated. For an
2046 IA where `ird_ord_ia_support` is `IT_FALSE`, the `IT_CONNECT_SUPPRESS_IRD_ORD` bit is
2047 ignored. For an IA where `ird_ord_ia_support` is `IT_TRUE` but `ird_ord_suppressible` is
2048 `IT_FALSE`, attempting to set `IT_CONNECT_SUPPRESS_IRD_ORD` will yield an immediate
2049 error.

2050 The passive side Consumer can choose either to accept or reject the Connection Request. If the
2051 passive side chooses to reject the Connection by calling `it_reject`, then an
2052 `IT_CM_MSG_CONN_PEER_REJECT_EVENT` Event is generated on the active side and the
2053 active side Endpoint transitions into the `IT_EP_STATE_NONOPERATIONAL` state. If it
2054 chooses to accept the Connection by calling `it_ep_accept`, then the behavior is dependent on the
2055 type of Connection setup specified by the `cn_est_flags`.

2056 For three-way Connection establishments, an `IT_CM_MSG_CONN_ACCEPT_ARRIVAL_`
2057 `EVENT` Event is generated on the active side if the passive side Consumer accepts the
2058 Connection establishment request by calling `it_ep_accept`. The active Consumer can choose to
2059 either accept or reject the Connection by calling `it_ep_accept` or `it_reject` respectively. For a two-
2060 way Connection establishment, an `IT_CM_MSG_CONN_ESTABLISHED_EVENT` Event is
2061 generated on the active side after the passive side Consumer accepts the Connection and the
2062 Endpoint on the active side transitions into the (`IT_EP_STATE_CONNECTED`) connected
2063 state. See the `it_ep_state_t` reference page for a complete description on the Endpoint state
2064 diagram for both the three-way and two-way Connection establishment.

2065 Whenever an Endpoint transitions to the connected (`IT_EP_STATE_CONNECTED`) state the
2066 Consumer will Receive an `IT_CM_MSG_CONN_ESTABLISHED_EVENT` Event on the
2067 simple Event Dispatcher to which the Communication Management Message Event Stream is
2068 routed after the Endpoint transitions into the `IT_EP_STATE_CONNECTED` state. This Event is
2069 generated on both the active and passive side of the Connection establishment after the Endpoint
2070 state transition takes place.

2071 For a complete definition of Endpoint state and a more complete description of the state
2072 transitions see the `it_ep_state_t` reference page. If for any reason an Endpoint Connection fails to
2073 be established, the Endpoint will transition into the `IT_EP_STATE_NONOPERATIONAL` state
2074 and any Receive DTO operations that were successfully posted to the Endpoint will be
2075 completed with an `IT_DTO_ERR_FLUSHED` status.

2076 **EXTENDED DESCRIPTION**

2077 An Endpoint can only be connected to a different Endpoint. An Endpoint cannot be connected to
2078 itself.

2079	RETURN VALUE	
2080		A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:
2081	IT_ERR_NO_PERMISSION	The Consumer did not have the proper permissions to perform the requested operation.
2082		
2083	IT_ERR_RESOURCES	The operation failed due to resource limitations.
2084	IT_ERR_INVALID_CONN_QUAL	The Connection Qualifier is invalid.
2085	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.
2086		
2087		
2088	IT_ERR_INVALID_PDATA_LENGTH	The IA supports Private Data, but the length specified exceeds the IA's capabilities.
2089		
2090	IT_ERR_INVALID_SOURCE_PATH	The Source Path specified in the <i>it_path_t</i> was invalid.
2091	IT_ERR_INVALID_SPIGOT	The Spigot specified in the <i>it_path_t</i> was invalid.
2092	IT_ERR_INVALID_EP	The <i>ep_handle</i> was invalid.
2093	IT_ERR_INVALID_EP_STATE	The Endpoint is not in the proper state to be connected.
2094	IT_ERR_INVALID_EP_TYPE	The Endpoint Service Type does not support this operation.
2095		
2096	IT_ERR_INVALID_CN_EST_FLAGS	The Connection establishment flags are invalid.
2097	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.
2098		
2099		
2100	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.
2101		
2102		
2103	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.
2104		
2105		
2106	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.
2107		
2108		
2109	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.
2110		
2111		
2112	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.
2113		
2114		
2115	IT_ERR_INVALID_IANA_LR_PORT	The local port in <i>connect_qual</i> is already in use.

2116 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in
2117 the disabled state. None of the output parameters from
2118 this routine are valid. See *it_ia_info_t* for a description
2119 of the disabled state.

2120 **ASYNCHRONOUS ERRORS**

2121 For the iWARP Transport, if one side (local or remote) uses a Non-permissive AV-RNIC/IETF
2122 and the other side (remote or local) uses an RDMAC AV-RNIC, then the local Implementation
2123 for the caller of *it_ep_connect* will generate an IT_CM_MSG_NONPEER_REJECT_EVENT
2124 during MPA Startup, with an IT_CN_REJ_BAD_CONN_PARMS reject reason code indicating
2125 incompatible protocol versions (see [INTEROP-IETF]).

2126 **APPLICATION USAGE**

2127 It is possible that between the time the Consumer calls *it_get_pathinfo* to retrieve a valid Path
2128 and the time the Consumer passes that Path as the *path* parameter to this routine, the set of
2129 available Paths through the network for forming the Connection may change, rendering some
2130 portion of the given *path* invalid. If IT_ERR_INVALID_SOURCE_PATH or IT_ERR_
2131 INVALID_SPIGOT is returned from this routine and the *path* the Consumer provided is one that
2132 was returned from *it_get_pathinfo* and was not subsequently modified by the Consumer, the
2133 Consumer can attempt to recover from the error by calling *it_get_pathinfo* again to retrieve an
2134 up-to-date Path to form the Connection and retrying the call to *it_ep_connect* with the new Path.

2135 On some transports, the passive side will not get an IT_CM_MSG_CONN_
2136 ESTABLISHED_EVENT Event until the active side first posts a Send operation.

2137 **SEE ALSO**

2138 *it_ep_accept()*, *it_reject()*, *it_ep_disconnect()*, *it_handoff()*, *it_ep_state_t*, *it_ia_query()*,
2139 *it_conn_qual_t*

it_ep_disconnect()

2140

2141 NAME

2142 `it_ep_disconnect` – disconnect an existing Endpoint-to-Endpoint Connection

2143 SYNOPSIS

```
2144 #include <it_api.h>
2145
2146 it_status_t it_ep_disconnect (
2147     IN          it_ep_handle_t  ep_handle,
2148     IN  const   unsigned char  *private_data,
2149     IN          size_t          private_data_length
2150 );
```

2151 APPLICABILITY

2152 `it_ep_disconnect` is applicable only to the RC service type.

2153 DESCRIPTION

2154 `ep_handle` Endpoint to be disconnected.

2155 `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.

2158 `private_data_length`: Length of `private_data`. This field must be 0 if the IA does not support Private Data.

2160 `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.

2170 `it_ep_disconnect` may be successfully called in all states except IT_EP_STATE_UNCONNECTED.

2172 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.

2175 See the [it_ep_state_t](#) reference page for a complete description of the Endpoint state behavior and transitions.

2177 RETURN VALUE

2178 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

2179	IT_ERR_INVALID_EP_STATE	The Endpoint was not in the proper state for the attempted operation.
2180		
2181	IT_ERR_INVALID_EP	The Endpoint Handle (<i>ep_handle</i>) was invalid.
2182	IT_ERR_INVALID_EP_TYPE	The attempted operation was invalid for the Service Type of the Endpoint.
2183		
2184	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.
2185		
2186		
2187		
2188	IT_ERR_INVALID_PDATA_LENGTH	The Interface Adapter supports Private Data, but the length specified exceeded the Interface Adapter's capabilities.
2189		
2190		
2191	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.
2192		
2193		
2194		

2195 **APPLICATION USAGE**

2196 The Consumer is responsible for coordinating the use of functions that free a Connection
 2197 Establishment Identifier (*cn_est_id*) such as [it_ep_accept](#), [it_reject](#), [it_ep_disconnect](#), and
 2198 [it_handoff](#). The behavior of functions that are passed as invalid Connection Establishment
 2199 Identifiers is indeterminate.

2200 The Consumer should be aware that successfully returning from this routine does not guarantee
 2201 that any interaction whatsoever will take place with the Remote Endpoint. If the Local
 2202 Consumer wishes to ensure that the Remote Consumer takes some action, an explicit message
 2203 should be sent to initiate that action before calling [it_ep_disconnect](#).

2204 With the three-way handshake Connection establishment method, there is also a potential race
 2205 condition between the Implementation generating the IT_CM_MSG_CONN_
 2206 ACCEPT_ARRIVAL_EVENT Event and the Consumer calling [it_ep_free](#). The Consumer
 2207 should not use the *cn_est_id* if the IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT Event
 2208 arrives after [it_ep_free](#) was called, regardless of whether the call returned yet, and regardless of
 2209 whether the Event was dequeued before or after the call was made. If the Consumer does use the
 2210 *cn_est_id*, then the Implementation may generate an IT_ERR_INVALID_CN_EST_ID error, or
 2211 it may generate a segmentation fault or other error.

2212 **SEE ALSO**

2213 [it_ep_accept\(\)](#), [it_reject\(\)](#), [it_ep_connect\(\)](#), [it_ep_state_t](#), [it_cm_msg_events](#), [it_ep_reset\(\)](#),
 2214 [it_ia_query\(\)](#)

it_ep_free()

2215

2216 NAME

2217 it_ep_free – destroy an RC or UD Endpoint

2218 SYNOPSIS

```
2219 #include <it_api.h>
2220
2221 it_status_t it_ep_free(
2222     IN it_ep_handle_t ep_handle
2223 );
```

2224 DESCRIPTION

2225 *ep_handle* Endpoint.

2226 *it_ep_free* destroys an Endpoint.

2227 An Endpoint cannot be destroyed if it has RMR(s) of type IT_RMR_TYPE_NARROW still
2228 bound to it. Otherwise, the Endpoint can be destroyed (freed) in any state. The freeing of an
2229 Endpoint also terminates the generation of Events to any of the EVDs associated with the
2230 Endpoint.

2231 Once *it_ep_free* returns, *ep_handle* may no longer be used.

2232 Freeing an Endpoint potentially means Events pertaining to that Endpoint might be lost on the
2233 *recv_sevd_handle* or *request_sevd_handle* SEVDs associated with the Endpoint. There is also
2234 potential to lose Events pertaining to that Endpoint on the *connect_sevd_handle* SEVD
2235 associated with the Endpoint. The Consumer should first drain these EVDs before calling
2236 *it_ep_free*.

2237 Freeing an RC-type Endpoint in the IT_EP_STATE_CONNECTED state while DTOs are in
2238 progress causes incoming DTOs to be ignored. All entries on the Endpoint *request_sevd_handle*,
2239 *recv_sevd_handle*, and *connect_sevd_handle* may or may not be on the EVDs after the Endpoint
2240 is destroyed.

2241 Freeing an RC-type Endpoint in the IT_EP_STATE_ACTIVE1_CONNECTION_PENDING,
2242 IT_EP_STATE_ACTIVE2_CONNECTION_PENDING, IT_EP_STATE_PASSIVE_WAIT_
2243 RDMA_TRANS_REQ, or IT_EP_STATE_PASSIVE_CONNECTION_PENDING state may
2244 cause a Connection establishment timeout or non-peer reject to be sent to the remote side.

2245 RETURN VALUE

2246 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

2247 IT_ERR_INVALID_EP The Endpoint Handle (*ep_handle*) was invalid.

2248 IT_ERR_EP_BUSY An RMR of type IT_RMR_TYPE_NARROW is still bound to
2249 the Endpoint.

2250 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the
2251 disabled state. None of the output parameters from this routine
2252 are valid. See *it_ia_info_t* for a description of the disabled state.

2253 **ASYNCHRONOUS ERRORS**

2254 Freeing an RC-type Endpoint in the IT_EP_STATE_CONNECTED state while Work Requests
2255 are still active on the Send Queue or Receive Queue of the Endpoint may cause these Work
2256 Requests to complete with the IT_DTO_ERR_FLUSHED status.

2257 **APPLICATION USAGE**

2258 Since the Implementation may not immediately free underlying resources, the user must not rely
2259 upon being immediately able to reallocate an Endpoint that has been freed.

2260 The following method can be used to ensure that all Completion Events for Work Requests
2261 posted to a Send Queue or Receive Queue are dequeued prior to calling *it_ep_free* for an RC-
2262 type Endpoint:

- 2263 1. The Consumer should call *it_ep_disconnect* first.
- 2264 2. Then the Consumer should dequeue a disconnect, Connection broken, or Connection
2265 reject Event as appropriate from the Connection EVD associated with the Endpoint.
- 2266 3. If the last Work Request posted to the Send Queue of the Endpoint did not request a
2267 Completion Event be generated, the Consumer should post a Work Request set up as a
2268 “marker” that is flushed by the Implementation to *recv_evd_handle* or
2269 *request_evd_handle*. The Work Request is made a “marker” operation by setting the
2270 IT_COMPLETION_FLAG on the operation. (If the last Work Request posted to the Send
2271 Queue did request a Completion Event, that Completion Event can serve as the marker.)
- 2272 4. When the Consumer dequeues the completion of the marker from the EVD associated
2273 with the Send Queue of the Endpoint, it is guaranteed that all previously posted Work
2274 Request completions (including those posted with IT_COMPLETION_FLAG cleared) for
2275 the Endpoint have also been dequeued from that EVD.
- 2276 5. When the Consumer dequeues the completion for the last Receive Work Request posted to
2277 the Receive Queue of the Endpoint from the EVD associated with the Endpoint Receive
2278 Queue, it is guaranteed that all previously posted Receive completions for the Endpoint
2279 have also been dequeued from that EVD.
- 2280 6. After all of the previous steps, it is safe to destroy or reset the Endpoint without losing any
2281 completions or Connection Events.

2282 With the three-way handshake Connection establishment method, there is also a potential race
2283 condition between the Implementation generating the IT_CM_MSG_CONN_ACCEPT_
2284 ARRIVAL_EVENT Event and the Consumer calling *it_ep_disconnect*. The Consumer should
2285 not use *cn_est_id* if the IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT Event arrives
2286 after *it_ep_disconnect* was called, regardless of whether the call returned yet, and regardless of
2287 whether the Event was dequeued before or after the call was made. If the Consumer does use the
2288 *cn_est_id*, then the Implementation may generate an IT_ERR_INVALID_CN_EST_ID error, or
2289 it may generate a segmentation fault or other error.

2290 **SEE ALSO**

2291 *it_ep_rc_create()*, *it_ep_ud_create()*, *it_ep_modify()*, *it_ep_query()*, *it_ep_reset()*,
2292 *it_ep_disconnect()*, *it_ia_info_t*, *it_ep_state_t*, *it_dto_flags_t*, *it_post_atomic()*, *it_post_send()*,
2293 *it_post_sendto()*, *it_post_recv()*, *it_post_recvfrom()*, *it_post_rdma_read()*, *it_post_rdma_write()*

it_ep_modify()

2294

2295 NAME

2296

it_ep_modify – modify attributes of an existing Endpoint

2297 SYNOPSIS

2298

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

2299

2300

```
it_status_t it_ep_modify(  
2301
```

2302

```
    IN          it_ep_handle_t      ep_handle,  
2303
```

2304

```
    IN          it_ep_param_mask_t  mask,  
2305
```

2306

```
    IN const    it_ep_attributes_t  *ep_attr  
2307
```

```
);
```

2305 DESCRIPTION

2306

ep_handle Endpoint.

2307

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

2308

ep_attr Pointer to Consumer-allocated structure that contains new Consumer-

2309

requested Endpoint attributes.

2310

it_ep_modify changes selected attributes of the Endpoint *ep_handle*.

2311

Attributes to be modified are specified by flags in *mask*. New values for the attributes are

2312

specified by the corresponding fields in the structure pointed to by *ep_attr*. See

2313

[it_ep_attributes_t](#) for the definition of the structure.

2314

Flag values for the *mask* parameter are shown below. Note that attributes represented by fields of

2315

ep_attr for which no flag value is shown below cannot be modified. The requested attribute

2316

changes only affect the local Endpoint and have no effect on attributes of any remote Endpoint.

2317

Flag values for attributes that may be potentially modified:

2318

IT_EP_PARAM_MAX_PAYLOAD

2319

IT_EP_PARAM_MAX_REQ_DTO

2320

IT_EP_PARAM_MAX_RECV_DTO

2321

IT_EP_PARAM_RDMA_RD_ENABLE

2322

IT_EP_PARAM_RDMA_WR_ENABLE

2323

IT_EP_PARAM_MAX_IRD

2324

IT_EP_PARAM_MAX_ORD

2325

IT_EP_PARAM_EP_KEY

2326

IT_EP_PARAM_SOFT_HI_WATERMARK

2327

IT_EP_PARAM_HARD_HI_WATERMARK

2328

See [it_ep_attributes_t](#) for the definition of valid states in which each of the above attributes may

2329

be modified.

2330

Values for *mask* must be created as the bitwise OR of the Endpoint attributes flag values (above)

2331

that the Consumer desires to change. *it_ep_modify* must succeed in modifying all the requested

2332

attributes atomically; if the attempt to modify any of the requested attributes generates an error,

2333

none of the other attributes supplied to the call will be applied.

2334	RETURN VALUE	
2335		A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:
2336	IT_ERR_INVALID_EP	The Endpoint Handle (<i>ep_handle</i>) was invalid.
2337	IT_ERR_INVALID_MASK	The mask contained invalid flag values.
2338	IT_ERR_INVALID_EP_STATE	The Endpoint was not in the proper state for the attempted operation. See it_ep_attributes_t .
2339		
2340	IT_ERR_INVALID_EP_TYPE	The attempted operation was invalid for the Service Type of the Endpoint.
2341		
2342	IT_ERR_PAYLOAD_SIZE	The requested <i>max_dto_payload_size</i> exceeds the maximum payload size supported by the underlying transport.
2343		
2344		
2345	IT_ERR_RESOURCE_REQ_DTO	The underlying transport could not allocate the requested <i>max_req_dtos</i> resources at this time.
2346		
2347	IT_ERR_RESOURCE_RECV_DTO	The underlying transport could not allocate the requested <i>max_recv_dtos</i> resources at this time.
2348		
2349	IT_ERR_RESOURCE_IRD	The underlying transport could not allocate the requested <i>rdma_read_ird</i> resources at this time.
2350		
2351	IT_ERR_RESOURCE_ORD	The underlying transport could not allocate the requested <i>rdma_read_ord</i> resources at this time.
2352		
2353	IT_ERR_INVALID_EP_KEY	Invalid Endpoint Key value. The Consumer does not have local permissions to use the specified Endpoint Key.
2354		
2355	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.
2356		
2357		
2358	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.
2359		
2360		
2361	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.
2362		
2363		
2364	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.
2365		
2366	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.
2367		
2368		
2369		
2370	SEE ALSO	
2371		it_ep_attributes_t , it_ep_rc_create() , it_ep_ud_create() , it_ep_query() , it_ep_free() , it_ia_info_t

it_ep_query()

2372

2373 NAME

2374 `it_ep_query` – query an existing Endpoint

2375 SYNOPSIS

```
2376 #include <it_api.h>
2377
2378 it_status_t it_ep_query(
2379     IN    it_ep_handle_t    ep_handle,
2380     IN    it_ep_param_mask_t mask,
2381     OUT   it_ep_param_t    *params
2382 );
2383
2384 typedef struct {
2385     it_ia_handle_t    ia;                /* IT_EP_PARAM_IA */
2386     size_t            spigot_id;        /* IT_EP_PARAM_SPIGOT */
2387     it_ep_state_t    ep_state;         /* IT_EP_PARAM_STATE */
2388     it_transport_service_type_t service_type; /* IT_EP_PARAM_SERV_TYPE */
2389     it_path_t        dst_path;         /* IT_EP_PARAM_PATH */
2390     it_pz_handle_t    pz;              /* IT_EP_PARAM_PZ */
2391     it_evd_handle_t    request_sevd;   /* IT_EP_PARAM_REQ_SEVD */
2392     it_evd_handle_t    recv_sevd;     /* IT_EP_PARAM_RECV_SEVD */
2393     it_evd_handle_t    connect_sevd;  /* IT_EP_PARAM_CONN_SEVD */
2394     it_ep_attributes_t attr;           /* See it_ep_attributes_t
2395                                         for mask flags for attr */
2396 } it_ep_param_t;
```

2397 DESCRIPTION

2398 *ep_handle* Endpoint.

2399 *mask* Bitwise OR of flags for desired parameters and attributes.

2400 *params* Pointer to Consumer-allocated structure whose members are written with
2401 the desired Endpoint parameters and attributes.

2402 *it_ep_query* returns the desired parameters and attributes of the Endpoint *ep_handle* in the
2403 structure pointed to by *params*. On return, each field of *params* is only valid if the corresponding
2404 flag as shown below each *it_ep_param_t* member is set in the *mask* argument. The *mask* value
2405 `IT_EP_PARAM_ALL` causes all fields to be returned. The *it_ep_param_mask_t* enum is defined
2406 in [it_ep_attributes_t](#).

2407 The definition of each field follows:

2408 *ia* Handle for the Interface Adapter specified to create the EP.

2409 *spigot_id* Spigot identifier. For RC Endpoints, this field is valid only if the Endpoint
2410 is not in the `IT_EP_STATE_UNCONNECTED` state. If the Endpoint is in
2411 the `IT_EP_STATE_UNCONNECTED` state, the value of this field is
2412 undefined.

2413 *ep_state* State of the Endpoint.

2414	<i>service_type</i>	Endpoint Service Type.
2415	<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.
2416		
2417		
2418		
2419		
2420		
2421		
2422	<i>pz</i>	Handle for the Protection Zone specified while creating the EP.
2423	<i>request_sevd</i>	Handle for the IT_DTO_EVENT_STREAM Simple Event Dispatcher for DTO request Completion Events of the created Endpoint.
2424		
2425	<i>recv_sevd</i>	Handle for the IT_DTO_EVENT_STREAM Simple Event Dispatcher for DTO Receive Completion Events of the created Endpoint.
2426		
2427	<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.
2428		
2429	<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> .
2430		
2431		
2432		

RETURN VALUE

2433 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

2434	IT_ERR_INVALID_EP	The Endpoint Handle (<i>ep_handle</i>) was invalid.
2435	IT_ERR_INVALID_MASK	The mask contained invalid flag values.
2436	IT_ERR_INVALID_EP_TYPE	The attempted operation was invalid for the Service Type of the Endpoint.
2437		
2438	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.
2439		
2440		
2441		

SEE ALSO

2443 *it_ep_attributes_t*, *it_ep_rc_create()*, *it_ep_ud_create()*, *it_ep_modify()*, *it_ep_free()*,
2444 *it_ia_info_t*

it_ep_rc_create()

2445

2446 NAME

2447

it_ep_rc_create – create an Endpoint for Reliable Connection

2448 SYNOPSIS

2449

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

2450

```
it_status_t it_ep_rc_create (  
    IN          it_pz_handle_t          pz_handle,  
    IN          it_evd_handle_t        request_sevd_handle,  
    IN          it_evd_handle_t        recv_sevd_handle,  
    IN          it_evd_handle_t        connect_sevd_handle,  
    IN          it_ep_rc_creation_flags_t flags,  
    IN const    it_ep_attributes_t      *ep_attr,  
    OUT         it_ep_handle_t          *ep_handle  
);
```

2459

```
typedef enum {  
    IT_EP_NO_FLAG      = 0x00,  
    IT_EP_REUSEADDR    = 0x01,  
    IT_EP_SRQ          = 0x02  
} it_ep_rc_creation_flags_t;
```

2466 APPLICABILITY

2467

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

2468 DESCRIPTION

2469

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

2470

2471

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

2472

2473

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

2474

2475

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

2476

2477

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

2478

2479

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

2480

ep_handle Handle for the created Endpoint.

2481

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

2482

2483

2484

The Connection Endpoint is created in the IT_EP_STATE_UNCONNECTED state. See *it_ep_state_t* for details.

2485

2486 The created Endpoint is not associated with an IA Spigot. An Endpoint is associated with a
2487 Spigot as part of Connection setup.

2488 The Protection Zone *pz_handle* allows Consumers to control what local memory the Endpoint
2489 can access for DTOs and what memory remote RDMA operations can access through the newly
2490 created Endpoint. Only memory referred to by LMRs and RMRs that match the Endpoint
2491 Protection Zone can be accessed through the Endpoint.

2492 *recv_sevd_handle* and *request_sevd_handle* are Event Dispatcher instances where the Consumer
2493 collects completion Notifications of DTOs and RMR operations. Completions of Receive DTOs
2494 are reported in *recv_sevd_handle* Event Dispatcher, and completions of Send, RDMA Read,
2495 RDMA Write DTOs, RMR Link, and RMR Unlink are reported in *request_sevd_handle*. It is
2496 permissible for *recv_sevd_handle* and *request_sevd_handle* to reference the same EVD. DTO
2497 and RMR operation Completion Events are defined in *it_dto_events*.

2498 The Consumer should not specify an SEVD in *recv_sevd_handle* or *request_sevd_handle* that is
2499 in overflowed state for use in the Endpoint creation call (see *it_evd_create* for more details on
2500 overflow). If the Consumer attempts to do so, the operation will fail with
2501 IT_ERR_INVALID_RECV_EVD_STATE or IT_ERR_INVALID_REQ_EVD_STATE.

2502 All Connection Events for the Endpoint are reported to the Consumer through the SEVD
2503 specified in *connect_sevd_handle*. For a complete list of Endpoint Connection Events, see
2504 *it_cm_msg_events*.

2505 The *flags* parameter allows the Consumer to control the behavior of the Implementation on
2506 Endpoint creation. Setting the IT_EP_REUSEADDR bit in *flags* allows the Consumer to specify
2507 that they allow the Implementation to return an Endpoint on creation that is possibly in the
2508 TimeWait state. Normally, the Implementation will only return Endpoints that are not in the
2509 TimeWait state. Setting the IT_EP_SRQ bit in *flags* allows the Consumer to specify that they
2510 wish to associate an S-RQ with the Endpoint that they are creating. If the Consumer sets this bit,
2511 then the *srq*, *soft_hi_watermark*, and *hard_hi_watermark* members of the input *ep_attr* structure
2512 are assumed to be valid; if this bit is cleared, these members of the input *ep_attr* structure shall
2513 be ignored by the Implementation.

2514 The TimeWait state exists for the purpose of preventing packets that were transmitted over one
2515 Connection from being inadvertently received in another subsequently established Connection.
2516 The TimeWait state is not a state of the Endpoint *per se*, but rather a state associated with a
2517 Connection the Endpoint had previously established. A Connection enters the TimeWait state
2518 when a disconnect is performed, and exits the TimeWait state after a TimeWait interval has
2519 elapsed. The duration of the TimeWait interval is transport-dependent, and for some transports it
2520 is also dependent upon network configuration parameters. This interval can be in the order of a
2521 minute or two in length.

2522 An Endpoint that is “in the TimeWait state” still has at least one Connection that it had
2523 previously established for which the TimeWait interval has not elapsed. (It is possible for an
2524 Endpoint to be in the TimeWait state with respect to multiple Connections it had previously
2525 established.) If an Endpoint attempts to establish a Connection that will use the same pair of
2526 Spigots that were involved in a previous Connection involving the Endpoint, and if that previous
2527 Connection is currently in the TimeWait state, the Connection establishment attempt may fail
2528 with an IT_ERR_EP_TIMEWAIT error; see *it_ep_accept*. This error will never be returned
2529 unless the Consumer sets the IT_EP_REUSEADDR bit in *flags* for the *it_ep_rc_create* routine.

2530 If the Consumer wishes to associate an S-RQ with the Endpoint being created, the following
2531 attributes in the *ep_attr* structure must be initialized appropriately:

- 2532 • *srq*
- 2533 • *soft_hi_watermark*
- 2534 • *hard_hi_watermark*

2535 See *it_ep_attributes* for details of how to initialize these attributes.

2536 The *ep_attr* parameter specifies the Consumer-requested attributes of the created Endpoint. The
2537 Implementation is required to satisfy all requested attributes or fail the operation. Hence, the
2538 Implementation must allocate all necessary resources to satisfy Consumer-requested attributes.
2539 The Implementation is allowed to allocate more resources than the Consumer requested in
2540 *ep_attr*. The Consumer can find the actual allocated resources by using *it_ep_query*. For detailed
2541 Endpoint attributes see the reference page for *it_ep_attributes*.

2542 RETURN VALUE

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

2544	IT_ERR_INVALID_PZ	The Protection Zone Handle (<i>pz_handle</i>) was
2545		invalid.
2546	IT_ERR_INVALID_REQ_EVD	The Simple Event Dispatcher Handle for Data
2547		Transfer Operation request completions
2548		(<i>request_sevd_handle</i>) was invalid.
2549	IT_ERR_INVALID_RECV_EVD	The Simple Event Dispatcher Handle for Data
2550		Transfer Operation Receive completions
2551		(<i>recv_sevd_handle</i>) was invalid.
2552	IT_ERR_INVALID_CONN_EVD	The Connection Simple Event Dispatcher Handle
2553		was invalid.
2554	IT_ERR_INVALID_EVD_TYPE	The Event Stream type for the Event Dispatcher
2555		was invalid.
2556	IT_ERR_INVALID_REQ_EVD_STATE	The Simple Event Dispatcher for Data Transfer
2557		Operation request completions was in an unusable
2558		state.
2559	IT_ERR_INVALID_RECV_EVD_STATE	The Simple Event Dispatcher for Data Transfer
2560		Operation Receive completions was in an unusable
2561		state.
2562	IT_ERR_INVALID_FLAGS	The <i>flags</i> value was invalid.
2563	IT_ERR_RESOURCES	The requested operation failed due to insufficient
2564		resources.
2565	IT_ERR_PAYLOAD_SIZE	The requested <i>max_dto_payload_size</i> exceeds the
2566		maximum payload size supported by the
2567		underlying transport.

2568 2569	IT_ERR_RESOURCE_REQ_DTO	The underlying transport could not allocate the requested <i>max_req_dtos</i> resources at this time.
2570 2571	IT_ERR_RESOURCE_RECV_DTO	The underlying transport could not allocate the requested <i>max_recv_dtos</i> resources at this time.
2572 2573 2574	IT_ERR_RESOURCE_SSEG	The underlying transport could not allocate the requested <i>max_send_segments</i> resources at this time.
2575 2576 2577	IT_ERR_RESOURCE_RSEG	The underlying transport could not allocate the requested <i>max_recv_segments</i> resources at this time.
2578 2579 2580	IT_ERR_RESOURCE_RRSEG	The underlying transport could not allocate the requested <i>max_rdma_read_segments</i> resources at this time.
2581 2582 2583	IT_ERR_RESOURCE_RWSEG	The underlying transport could not allocate the requested <i>max_rdma_write_segments</i> resources at this time.
2584 2585	IT_ERR_RESOURCE_IRD	The underlying transport could not allocate the requested <i>rdma_read_ird</i> resources at this time.
2586 2587	IT_ERR_RESOURCE_ORD	The underlying transport could not allocate the requested <i>rdma_read_ord</i> resources at this time.
2588	IT_ERR_INVALID_SRQ	The S-RQ handle was invalid.
2589 2590 2591 2592	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.
2593 2594 2595	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.
2596 2597 2598 2599	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.
2600 2601	IT_ERR_PRIV_OPS_UNAVAILABLE	The IA does not support Privileged Mode operations.
2602	IT_ERR_NO_PERMISSION	The Consumer is not Privileged.
2603 2604 2605 2606	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.

2607 **APPLICATION USAGE**

2608 Use of `IT_EP_REUSEADDR` requires the Consumer to handle a potential
2609 `IT_ERR_EP_TIMEWAIT` error from *it_ep_accept* if the Endpoint and an incoming Connection
2610 Request are in the TimeWait state with respect to each other.

2611 Sometimes the required attribute values for an Endpoint depend on parameters in an incoming
2612 Connection Request and are not known at Endpoint creation time. The Consumer should specify
2613 these attributes at a later time using *it_ep_modify*; for example, before accepting an incoming
2614 Connection Request.

2615 Specifying an overflowed SEVD in *connect_sevd_handle* is recoverable but may result in
2616 connect Events being lost.

2617 **SEE ALSO**

2618 *it_ep_attributes_t*, *it_ep_ud_create()*, *it_ep_query()*, *it_ep_modify()*, *it_ep_free()*, *it_ep_accept()*,
2619 *it_cm_msg_events()*, *it_dto_events()*, *it_ia_info_t*, *it_srq_create()*

it_ep_reset()

2620

2621 NAME

2622 `it_ep_reset` – reset a Reliable Connected Endpoint to the initial state

2623 SYNOPSIS

```
2624 #include <it_api.h>
2625
2626 it_status_t it_ep_reset(
2627     IN it_ep_handle_t ep_handle
2628 );
```

2629 APPLICABILITY

2630 `it_ep_reset` is applicable only to Endpoints created for the RC Service Type.

2631 DESCRIPTION

2632 `ep_handle` Reliable Connected Endpoint.

2633 `it_ep_reset` resets a Reliable Connected Endpoint into the `IT_EP_STATE_UNCONNECTED`
2634 state it had at original creation while maintaining the other attributes of the Endpoint in their
2635 current settings. `it_ep_reset` may only be applied to Reliable Connected Endpoints in the
2636 `IT_EP_STATE_NONOPERATIONAL` state. An Endpoint in the `IT_EP_STATE_`
2637 `NONOPERATIONAL` due to overflow of a DTO completion EVD cannot be reset.

2638 Upon return of this operation any Completion Events for the Endpoint not yet harvested by the
2639 Consumer may be dropped or not delivered to the EVD(s) associated with the Request or
2640 Receive Queue for the Endpoint. This operation is only needed if Consumers would like to re-
2641 use the Endpoint. Otherwise, they can just free the Endpoint using `it_ep_free`.

2642 RETURN VALUE

2643 A successful call returns `IT_SUCCESS`. Otherwise, an error code is returned as described below:

2644	<code>IT_ERR_INVALID_EP</code>	The Endpoint Handle (<code>ep_handle</code>) was invalid.
2645	<code>IT_ERR_INVALID_EP_STATE</code>	The Endpoint was not in the proper state for the 2646 attempted operation.
2647	<code>IT_ERR_INVALID_EP_TYPE</code>	The attempted operation was invalid for the Service Type 2648 of the Endpoint.
2649	<code>IT_ERR_CANNOT_RESET</code>	The Endpoint could not be reset due to an overflow of 2650 one of its Data Transfer Operation Event Stream Event 2651 Dispatchers.
2652	<code>IT_ERR_IA_CATASTROPHE</code>	The IA has experienced a catastrophic error and is in the 2653 disabled state. None of the output parameters from this 2654 routine are valid. See <code>it_ia_info_t</code> for a description of the 2655 disabled state.

2656 SEE ALSO

2657 `it_ep_rc_create()`, `it_ep_disconnect()`, `it_ep_free()`, `it_ia_info_t`

it_ep_ud_create()

2658

2659 NAME

2660 it_ep_ud_create – create an Endpoint for Unreliable Datagram

2661 SYNOPSIS

```
2662 #include <it_api.h>
2663
2664 it_status_t it_ep_ud_create (
2665     IN          it_pz_handle_t          pz_handle,
2666     IN          it_evd_handle_t        request_sevd_handle,
2667     IN          it_evd_handle_t        recv_sevd_handle,
2668     IN const    it_ep_attributes_t     *ep_attr,
2669     IN          size_t                  spigot_id,
2670     OUT         it_ep_handle_t         *ep_handle
2671 );
```

2672 APPLICABILITY

2673 *it_ep_ud_create* is applicable only to create Endpoints of the UD Service Type.

2674 DESCRIPTION

2675 *pz_handle* Handle for the Protection Zone of the created Endpoint. Implicitly
2676 identifies the Interface Adapter.

2677 *request_sevd_handle:* Handle for the IT_DTO_EVENT_STREAM Simple Event Dispatcher for
2678 DTO request Completion Events of the created Endpoint.

2679 *recv_sevd_handle:* Handle for the IT_DTO_EVENT_STREAM Simple Event Dispatcher for
2680 DTO Receive Completion Events of the created Endpoint.

2681 *ep_attr* Pointer to a structure that contains Consumer-requested Endpoint
2682 Attributes.

2683 *spigot_id* Interface Adapter Spigot identifier to use when creating Endpoint.

2684 *ep_handle* Handle for the created Endpoint.

2685 *it_ep_ud_create* creates, on the requested *spigot_id* of the Interface Adapter implicitly identified
2686 by *pz_handle*, an Unreliable Datagram Endpoint that is provided to the Consumer as *ep_handle*.
2687 The value of *ep_handle* is only defined if the return value is IT_SUCCESS.

2688 The Unreliable Datagram Endpoint is created in the IT_EP_STATE_UD_OPERATIONAL
2689 state. See *it_ep_state_t* for details.

2690 Protection Zone *pz_handle* allows Consumers to control what local memory the Endpoint can
2691 access for DTOs. Only memory referred to by LMRs that match the Endpoint Protection Zone
2692 can be accessed by the Endpoint.

2693 *recv_sevd_handle* and *request_sevd_handle* are Event Dispatcher instances where the Consumer
2694 collects completion Notifications of DTOs. Completions of Receive DTOs are reported in the
2695 *recv_sevd_handle* Event Dispatcher, and completions of Send DTOs are reported in
2696 *request_sevd_handle*. It is permissible for *recv_sevd_handle* and *request_sevd_handle* to
2697 reference the same EVD. DTO Completion Events are defined in *it_dto_events*.

2698 The Consumer should not specify an SEVD in *recv_sevd_handle* or *request_sevd_handle* that is
 2699 in overflowed state for use in the Endpoint creation call (see *it_evd_create* for more details on
 2700 overflow). If the Consumer attempts to do so, the operation will fail with
 2701 IT_ERR_INVALID_RECV_EVD_STATE or IT_ERR_INVALID_REQ_EVD_STATE.

2702 The *ep_attr* parameter specifies the Consumer-requested attributes of the created Endpoint. The
 2703 Implementation is required to satisfy all requested attributes or fail the operation. Hence, the
 2704 Implementation must allocate all necessary resources to satisfy Consumer-requested attributes.
 2705 The Implementation is allowed to allocate more resources than the Consumer requested in
 2706 *ep_attr*. The Consumer can find the actual allocated resources by using *it_ep_query*. For detailed
 2707 Endpoint attributes see the reference page for *it_ep_attributes_t*.

2708 **RETURN VALUE**

2709 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

2710	IT_ERR_INVALID_PZ	The Protection Zone Handle (<i>pz_handle</i>) was
2711		invalid.
2712	IT_ERR_INVALID_REQ_EVD	The Simple Event Dispatcher Handle for Data
2713		Transfer Operation request completions
2714		(<i>request_sevd_handle</i>) was invalid.
2715	IT_ERR_INVALID_RECV_EVD	The Simple Event Dispatcher Handle for Data
2716		Transfer Operation Receive completions
2717		(<i>recv_sevd_handle</i>) was invalid.
2718	IT_ERR_INVALID_EVD_TYPE -	The Event Stream Type for the Event Dispatcher was
2719		invalid.
2720	IT_ERR_INVALID_REQ_EVD_STATE	The Simple Event Dispatcher for Data Transfer
2721		Operation request completions was in an unusable
2722		state.
2723	IT_ERR_INVALID_RECV_EVD_STATE	The Simple Event Dispatcher for Data Transfer
2724		Operation Receive completions was in an unusable
2725		state.
2726	IT_ERR_INVALID_SPIGOT	An invalid Spigot ID was specified.
2727	IT_ERR_RESOURCES	The requested operation failed due to insufficient
2728		resources.
2729	IT_ERR_PAYLOAD_SIZE	The requested <i>max_dto_payload_size</i> exceeds the
2730		maximum payload size supported by the underlying
2731		transport.
2732	IT_ERR_RESOURCE_REQ_DTO	The underlying transport could not allocate the
2733		requested <i>max_req_dtos</i> resources at this time.
2734	IT_ERR_RESOURCE_RECV_DTO	The underlying transport could not allocate the
2735		requested <i>max_recv_dtos</i> resources at this time.
2736	IT_ERR_RESOURCE_SSEG	The underlying transport could not allocate the
2737		requested <i>max_send_segments</i> resources at this time.

2738	IT_ERR_RESOURCE_RSEG	The underlying transport could not allocate the requested <i>max_rcv_segments</i> resources at this time.
2739		
2740	IT_ERR_INVALID_EP_KEY	Invalid Endpoint Key value. The Consumer doesn't have local permissions to use the specified Endpoint Key.
2741		
2742		
2743	IT_ERR_PRIV_OPS_UNAVAILABLE	The IA does not support Privileged Mode operations.
2744	IT_ERR_NO_PERMISSION	The Consumer is not Privileged.
2745	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.
2746		
2747		
2748		
2749	SEE ALSO	
2750		<i>it_ep_attributes_t</i> , <i>it_ep_rc_create()</i> , <i>it_ep_query()</i> , <i>it_ep_modify()</i> , <i>it_ep_free()</i> , <i>it_ep_state_t</i> ,
2751		<i>it_dto_events</i> , <i>it_ia_info_t</i>

2752

it_evd_callback_attach()

2753 **NAME**

2754 `it_evd_callback_attach` – attach a callback routine to an Event Dispatcher

2755 **SYNOPSIS**

```
2756 #include <it_api.h>
2757
2758 typedef void (*it_evd_callback_rtn_t)(
2759     IN it_evd_handle_t  evd,
2760     IN void             *arg
2761 );
2762
2763 it_status_t it_evd_callback_attach(
2764     IN it_evd_handle_t  evd_handle,
2765     IN it_evd_callback_rtn_t callback,
2766     IN void             *arg
2767 );
```

2768 **DESCRIPTION**

2769 *evd_handle*: Handle for simple or aggregate Event Dispatcher.

2770 *callback*: The routine to be invoked when Notification occurs for *evd_handle*.

2771 *arg*: A Consumer-supplied parameter to pass to *callback* when it is invoked.

2772 *it_evd_callback_attach* can be used by a Privileged Consumer of the IT-API to attach a callback routine to an input EVD. The EVD can be either an SEVD or an AEVD. If a non-Privileged Consumer attempts to attach a callback routine to an EVD, an error will be returned.

2775 An EVD can only have a single callback routine attached to it. Attempting to attach more than one callback routine to an EVD will cause *it_evd_callback_attach* to return an error.

2777 A Consumer is not required to attach a callback routine to an EVD in order to get Notification from that EVD; the other mechanisms for getting Notification documented in the IT-API (e.g., calling *it_evd_wait*) can be used instead. The Consumer is not allowed to use multiple Notification mechanisms concurrently, however, and an attempt to do so shall return an error. Specifically, the following Consumer actions are disallowed:

- 2782 • Having a call to *it_evd_wait* in progress for the EVD and then attempting to use *it_evd_callback_attach* to attach a callback routine to the EVD.
- 2784 • Calling *it_evd_wait* for an EVD that has a callback routine attached.
- 2785 • Attempting to use *it_evd_callback_attach* to attach a callback routine to an EVD that has an associated file descriptor (i.e., the IT_EVD_CREATE_FD flag was set when the EVD was created).
- 2788 • Attempting to use *it_evd_callback_attach* to attach a callback routine to an SEVD that is associated with an AEVD.
- 2790 • Attempting to associate an AEVD with an SEVD that has a callback routine attached.

2791 Any Privileged Consumer can attach a callback routine to an EVD that was created by any other
2792 Privileged Consumer, provided the EVD does not currently have a callback routine attached.
2793 Privileged Consumers therefore need to cooperate to avoid inadvertently attaching callback
2794 routines to one another's EVDs.

2795 Once a Consumer has successfully attached a callback routine to an EVD, the callback routine
2796 shall not be invoked until the Consumer has first dequeued all Events from the EVD using
2797 *it_evd_dequeue*. After the Consumer has dequeued all Events, the callback routine will be
2798 invoked the first time a transition from Notification criteria absent to Notification criteria present
2799 takes place for that EVD. For example, if an SEVD with a callback routine attached has a
2800 threshold value of two, the callback routine will be invoked when the second Event is enqueued
2801 within the SEVD. For a definition of Notification criteria, see *it_evd_create*. Once a callback
2802 routine has been invoked, it shall not be invoked again until all Events have been dequeued from
2803 the EVD using *it_evd_dequeue*.

2804 The IT-API guarantees that only one invocation of *callback* shall be in progress at a time for a
2805 given EVD. The Consumer does not need to guard against multiple invocations of *callback*
2806 happening concurrently, even on a multi-processor platform. Multiple concurrent invocations of
2807 *callback* can only occur if *callback* has been attached to more than one EVD.

2808 A callback routine will be invoked in the same memory context that was present when the
2809 *it_evd_callback_attach* call that attached the callback routine to the EVD was performed.

2810 A callback routine may safely invoke any IT-API routine with the exception of the following:

- 2811 • *it_address_handle_create*
- 2812 • *it_address_handle_modify*
- 2813 • *it_convert_net_addr*
- 2814 • *it_ep_connect*
- 2815 • *it_ep_modify*
- 2816 • *it_ep_rc_create*
- 2817 • *it_ep_ud_create*
- 2818 • *it_evd_create*
- 2819 • *it_evd_modify*
- 2820 • *it_evd_wait*
- 2821 • *it_get_pathinfo*
- 2822 • *it_ia_create*
- 2823 • *it_ia_query*
- 2824 • *it_listen_create*
- 2825 • *it_lmr_create*
- 2826 • *it_lmr_modify*
- 2827 • *it_pz_create*
- 2828 • *it_rmr_create*
- 2829 • *it_socket_convert*
- 2830 • *it_srq_create*
- 2831 • *it_srq_modify*
- 2832 • *it_ud_service_request_handle_create*

2833 If a Consumer calls one of the above-listed routines from within a callback routine, the results
2834 are Implementation-dependent.

2835 The specification of which routines external to the IT-API that a Consumer may safely invoke
2836 from within a callback routine is outside the scope of the IT-API.

2837 **RETURN VALUE**

2838 A successful call returns `IT_SUCCESS`. Otherwise, an error code is returned as described below:

2839 `IT_ERR_INVALID_EVD` The Event Dispatcher Handle (*evd_handle*) was invalid.

2840 `IT_ERR_CALLBACK_EXISTS` A callback routine has already been attached to the input
2841 EVD.

2842 `IT_ERR_EVD_WAIT` A callback routine could not be attached to the EVD
2843 because a call to *it_evd_wait* is currently in progress for
2844 the EVD.

2845 `IT_ERR_NO_PERMISSION` The caller was not a Privileged Consumer.

2846 `IT_ERR_INVALID_EVD_STATE` A callback routine could not be attached to the EVD
2847 because another notification mechanism is already
2848 enabled.

2849 `IT_ERR_IA_CATASTROPHE` The IA has experienced a catastrophic error and is in the
2850 disabled state. None of the output parameters from this
2851 routine are valid. See *it_ia_info_t* for a description of the
2852 disabled state.

2853 **SEE ALSO**

2854 *it_evd_callback_detach()*, *it_evd_create()*, *it_evd_query()*, *it_evd_wait()*

2855

it_evd_callback_detach()

2856 **NAME**

2857 `it_evd_callback_detach` – detach a callback routine from an Event Dispatcher

2858 **SYNOPSIS**

```
2859 #include <it_api.h>
2860
2861 it_status_t it_evd_callback_detach(
2862     IN it_evd_handle_t evd_handle
2863 );
```

2864 **DESCRIPTION**

2865 *evd_handle*: Handle for simple or aggregate Event Dispatcher whose callback routine is
2866 to be detached.

2867 *it_evd_callback_detach* can be used by a Privileged Consumer of the IT-API to detach a
2868 callback routine from an input EVD. The EVD can be either an SEVD or an AEVD. If a non-
2869 Privileged Consumer attempts to detach a callback routine from an EVD an error will be
2870 returned.

2871 If this routine is called for an EVD that does not have a callback routine attached, an
2872 `IT_ERR_INVALID_EVD_STATE` error shall be returned.

2873 The Consumer is allowed to detach a callback routine from an EVD whenever they wish. In
2874 particular, a Consumer is allowed to detach a callback routine from an EVD from within an
2875 invocation of the callback routine itself.

2876 A Consumer is not required to detach a callback routine from an EVD before calling *it_evd_free*
2877 to free the EVD. However, if a Consumer attempts to free an EVD while there is an invocation
2878 of the callback routine for that EVD outstanding, an error will be returned. See *it_evd_free* for
2879 details.

2880 **RETURN VALUE**

2881 A successful call returns `IT_SUCCESS`. Otherwise, an error code is returned as described below:

2882 `IT_ERR_INVALID_EVD` The Event Dispatcher Handle (*evd_handle*) was invalid.

2883 `IT_ERR_INVALID_EVD_STATE` The attempted operation was invalid for the current state
2884 of the Event Dispatcher.

2885 `IT_ERR_NO_PERMISSION` The caller was not a Privileged Consumer.

2886 `IT_ERR_IA_CATASTROPHE` The IA has experienced a catastrophic error and is in the
2887 disabled state. None of the output parameters from this
2888 routine are valid. See *it_ia_info_t* for a description of the
2889 disabled state.

2890 **SEE ALSO**

2891 *it_evd_create()*, *it_evd_callback_attach()*, *it_evd_free()*, *it_evd_query()*

it_evd_create()

2892

2893 NAME

2894

it_evd_create – create Simple or Aggregate Event Dispatcher

2895 SYNOPSIS

2896

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

2897

```
it_status_t it_evd_create (  
    IN          it_ia_handle_t    ia_handle,  
    IN          it_event_type_t   event_number,  
    IN          it_evd_flags_t    evd_flag,  
    IN          size_t            sevd_queue_size,  
    IN          size_t            sevd_threshold,  
    IN          it_evd_handle_t   aevd_handle,  
    OUT         it_evd_handle_t   *evd_handle,  
    OUT         int               *fd  
);
```

2908

```
#define IT_THRESHOLD_DISABLE 0
```

2909

```
typedef enum {  
    IT_EVD_DEQUEUE_NOTIFICATIONS = 0x01,  
    IT_EVD_CREATE_FD             = 0x02,  
    IT_EVD_OVERFLOW_DEFAULT      = 0x04,  
    IT_EVD_OVERFLOW_NOTIFY       = 0x08,  
    IT_EVD_OVERFLOW_AUTO_RESET   = 0x10  
} it_evd_flags_t;
```

2917

2918 DESCRIPTION

2919

ia_handle Handle for the Interface Adapter to which the created Event Dispatcher belongs.

2920

2921

event_number Identifier for Event Stream type that can be enqueued to the EVD.

2922

evd_flag Bitwise OR of flag values for creation operation.

2923

sevd_queue_size Minimum size of the Simple EVD Event queue. This parameter is ignored for an Aggregate EVD.

2924

2925

sevd_threshold Number of Events on the Simple EVD queue required for Notification of the associated AEVD or *fd* and for SEVD waiters unblocking. This parameter is ignored for Aggregate EVD.

2926

2927

2928

aevd_handle Optional Handle to associate an Aggregate EVD with the Simple EVD. This parameter must be IT_NULL_HANDLE when the IT_EVD_CREATE_FD *evd_flag* is set. This parameter must also be IT_NULL_HANDLE when using *it_evd_create* to create an Aggregate EVD.

2929

2930

2931

2932

2933

evd_handle Handle for the created Event Dispatcher.

2934 *fd* Pointer to an optional file descriptor corresponding to the Event Dispatcher.
 2935 Only valid if return value is IT_SUCCESS and the IT_EVD_CREATE_FD
 2936 *evd_flag* was set.

2937 *it_evd_create* creates an instance of an Event Dispatcher (EVD) that is provided to the
 2938 Consumer as *evd_handle*. Two different types of EVDs are supported by the Implementation:
 2939 Simple EVDs (SEVD) and Aggregate EVDs (AEVD). An SEVD is an EVD for a single Event
 2940 Stream. An AEVD is an aggregation of SEVDs and thus can potentially return Events for more
 2941 than one Event Stream type. *it_evd_create* can also optionally return a file descriptor (*fd*)
 2942 associated with an EVD.

2943 The values of *evd_handle* and *fd* are only defined if the return value is IT_SUCCESS.

2944 The scope of an EVD is a single Interface Adapter identified by *ia_handle*.

2945 *event_number* identifies the type of Event Stream that the created EVD will handle. Multiple
 2946 Event Streams of the same Event Stream type (such as DTO Completion Event Streams) can
 2947 feed the EVD. Event Stream types are defined in *it_event_t*.

2948 To create an Aggregate EVD, the *event_number* must be set to IT_AEVD_NOTIFICATION_
 2949 EVENT_STREAM; a Simple EVD (SEVD) is created otherwise. To create a Simple EVD the
 2950 *event_number* can be any one of IT.DTO_EVENT_STREAM, IT_CM_REQ_
 2951 EVENT_STREAM, IT_CM_MSG_EVENT_STREAM, IT_ASYNC_AFF_EVENT_STREAM,
 2952 IT_ASYNC_UNAFF_EVENT_STREAM, or IT_SOFTWARE_EVENT_STREAM.

2953 A Simple EVD may feed only one Aggregate EVD. An Aggregate EVD may be fed by many
 2954 Simple EVDs. The Consumer may create multiple AEVDs and SEVDs with the following two
 2955 exceptions:

- 2956 1. Only one IT_ASYNC_AFF_EVENT_STREAM Simple EVD may be created per
 2957 Interface Adapter instance. Subsequent calls to *it_evd_create* for the
 2958 IT_ASYNC_AFF_EVENT_STREAM Event Stream, without intervening calls to
 2959 *it_evd_free* the EVD, will fail with the error return IT_ERR_ASYNC_AFF_
 2960 EVD_EXISTS.
- 2961 2. Only one IT_ASYNC_UNAFF_EVENT_STREAM Simple EVD may be created per
 2962 Interface Adapter instance. Subsequent calls to *it_evd_create* for the
 2963 IT_ASYNC_UNAFF_EVENT_STREAM Event Stream, without intervening calls to
 2964 *it_evd_free* the EVD, will fail with the error return IT_ERR_ASYNC_UNAFF_
 2965 EVD_EXISTS.

2966 For all Event Stream types except IT_SOFTWARE_EVENT_STREAM, IT_ASYNC_
 2967 AFF_EVENT_STREAM, and IT_ASYNC_UNAFF_EVENT_STREAM, upon creation there is
 2968 no Event Stream of *event_number* feeding Events to the created EVD. For an Aggregate EVD
 2969 this means that there are no Simple EVDs associated with *evd_handle*. No Events are fed to
 2970 *evd_handle* until *evd_handle* is associated with an object that feeds Events to it. For an
 2971 Aggregate EVD this means that no Events are fed to *evd_handle* until *evd_handle* is associated
 2972 with a Simple EVD. For a Simple EVD this means that no Events are fed to *evd_handle* until
 2973 *evd_handle* is associated with an Endpoint, Listen Point, or UD Service Request Handle
 2974 depending on the stream type.

2975 For the IT_ASYNC_AFF_EVENT_STREAM Event Stream type, the Simple EVD receives the
 2976 Async Affiliated Events for the *ia_handle*. For the IT_ASYNC_UNAFF_EVENT_STREAM
 2977 Event Stream type, the Simple EVD receives the Async Unaffiliated Events for the *ia_handle*.

2978 Multiple Event Streams of the same Event Stream type can be associated with the same EVD,
2979 with the exception of IT_ASYNC_AFF_EVENT_STREAM, IT_ASYNC_UNAFF_EVENT_
2980 STREAM, and IT_SOFTWARE_EVENT_STREAM Event Stream types. For the IT_AEVD_
2981 NOTIFICATION_EVENT_STREAM Event Stream type, multiple SEVDs can be associated
2982 with the same AEVD. For IT_DTO_EVENT_STREAM, multiple EPs can be associated with the
2983 same SEVD. For IT_CM_REQ_EVENT_STREAM, multiple Listens can be associated with the
2984 same SEVD. For IT_CM_MSG_EVENT_STREAM, multiple RC EPs and/or UD Service
2985 Requests can be associated with the same SEVD. For the IT_ASYNC_AFF_EVENT_STREAM,
2986 IT_ASYNC_UNAFF_EVENT_STREAM, and IT_SOFTWARE_EVENT_STREAM Event
2987 Stream types, only a single Event Stream feeds each EVD, respectively, and the corresponding
2988 Event Stream is created upon EVD creation. For IT_SOFTWARE_EVENT_STREAM, the
2989 Events are generated explicitly by the Consumer calling *it_evd_post_se*.

2990 When the Implementation attempts to enqueue more Events on an SEVD than the queue size of
2991 the SEVD will permit, the SEVD is said to overflow. An AEVD cannot overflow.

2992 Once an SEVD overflows, subsequent Events from the Event Stream will be dropped. For all
2993 Event Streams, with the exception of IT_DTO_EVENT_STREAM, Events will no longer be
2994 dropped once the Consumer makes more space available in the SEVD's Event queue. The
2995 Consumer can make room in an SEVD either by dequeuing an Event, or by using
2996 *it_evd_modify* to increase the queue size of the SEVD. For IT_DTO_EVENT_STREAM,
2997 however, Events will continue to be dropped; the overflow cannot be corrected.

2998 The behavior of an SEVD after an overflow depends upon the Event Stream associated with the
2999 SEVD, and upon whether the default overflow behavior has been configured for the SEVD. The
3000 reference page associated with each Event Stream type provides details of the default overflow
3001 behavior. The Consumer specifies default overflow behavior by setting the
3002 IT_EVD_OVERFLOW_DEFAULT *evd_flag* value.

3003 If default overflow behavior is not configured (IT_EVD_OVERFLOW_DEFAULT is cleared in
3004 *evd_flag*), then the Consumer can control two possible parameters: Whether the overflow
3005 occurrence causes generation (IT_EVD_OVERFLOW_NOTIFY flag value) of an overflow
3006 Event on the Affiliated or Unaffiliated SEVD, and, if configured, how the generation of the
3007 overflow Event is controlled (IT_EVD_OVERFLOW_AUTO_RESET flag value). Each
3008 subsequent SEVD Event that arrives after overflow of the SEVD initially occurs can potentially
3009 generate an overflow Event.

3010 The Consumer can request that an overflow Event be generated when an overflow occurs by
3011 setting IT_EVD_OVERFLOW_NOTIFY in *evd_flag*. For SEVDs associated with any Event
3012 Streams other than the IT_ASYNC_AFF_EVENT_STREAM or the IT_ASYNC_UNAFF_
3013 EVENT_STREAM, an IT_ASYNC_AFF_SEVD_ENQUEUE_FAILURE Event is enqueued on
3014 the affiliated asynchronous error Event Stream of *ia_handle*. For an SEVD associated with the
3015 IT_ASYNC_AFF_EVENT_STREAM, an IT_ASYNC_UNAFF_SEVD_ENQUEUE_FAILURE
3016 Event is enqueued on the unaffiliated asynchronous error Event Stream of *ia_handle*. The Event
3017 identifies the SEVD that overflowed. Overflow of the IT_ASYNC_UNAFF_EVENT_
3018 STREAM is never detected and no indication of such overflow is ever generated; however, no
3019 adverse consequences occur other than the dropping of some Unaffiliated Events.

3020 If an SEVD overflow has occurred, the *evd_overflowed* member of the *it_evd_param_t* structure
3021 (as returned by the *it_evd_query* routine) will have an IT_TRUE value until the condition is
3022 corrected or manually changed. When the Consumer creates an SEVD to hold Events of an
3023 Event Stream and has enabled generation of overflow Events on the SEVD

3024 (IT_EVD_OVERFLOW_NOTIFY flag value), the Consumer must chose one of two modes for
3025 generation of overflow Events using the IT_EVD_OVERFLOW_AUTO_RESET flag:
3026 automatic, or Consumer-controlled. In automatic mode, overflow Events may again be enqueued
3027 on the Affiliated or Unaffiliated SEVD as soon as the Consumer makes more space available in
3028 the EVD's Event queue. In Consumer-controlled generation, overflow Events are only again
3029 generated after the Consumer calls *it_evd_modify* to clear the *evd_overflowed* field. See
3030 *it_evd_modify* for more details.

3031 Note that even if overflow generation is disabled, the Consumer may still clear *evd_overflowed*
3032 using *it_evd_modify*. A subsequent overflow will again set the *evd_overflowed* member of the
3033 *it_evd_param_t* structure.

3034 For a newly created SEVD, the *evd_overflowed* member of the *it_evd_param_t* structure is not
3035 set.

3036 The *evd_flag* value of IT_EVD_DEQUEUE_NOTIFICATIONS applies only to AEVDs.

3037 When the IT_EVD_DEQUEUE_NOTIFICATIONS bit is set in *evd_flag*, then wait and dequeue
3038 operations on the AEVD will dequeue IT_AEVD_NOTIFICATION_EVENT_STREAM
3039 Events; such Events provide the SEVD Handle of the underlying SEVD that caused the
3040 Notification. To retrieve the underlying Event, the Consumer must call *it_evd_dequeue* on the
3041 SEVD Handle provided in the IT_AEVD_NOTIFICATION_EVENT_STREAM Event from the
3042 AEVD.

3043 When the IT_EVD_DEQUEUE_NOTIFICATIONS bit is cleared in *evd_flag*, then calling
3044 *it_evd_wait* on the AEVD directly returns the first Event from a notifying underlying SEVD
3045 (such as IT_DTO_EVENT_STREAM Events, etc.). The dequeue operation on the AEVD
3046 directly returns the first Event from an underlying SEVD. These Events will be of whatever
3047 Event Stream types that feed each of these associated SEVDs. The associated SEVD can be
3048 determined from the *evd_handle* found in every Event.

3049 If an underlying SEVD of an AEVD has been disabled, then the SEVD will no longer generate
3050 Notification Events for the AEVD until the SEVD is enabled (see *it_evd_modify*). Previously
3051 generated SEVD Notifications for the AEVD are unaffected by the enabling and disabling of
3052 SEVD.

3053 For a Simple EVD that does not have an associated AEVD, the Consumer can wait on and
3054 dequeue from the SEVD.

3055 If the SEVD has an associated AEVD with the IT_EVD_DEQUEUE_NOTIFICATIONS
3056 *evd_flag* cleared, then it is an error for the Consumer to wait on or dequeue from the SEVD.
3057 Attempting to wait on or dequeue from the SEVD will return IT_ERR_INVALID_EVD_
3058 STATE.

3059 If the SEVD has an associated AEVD with the IT_EVD_DEQUEUE_NOTIFICATIONS
3060 *evd_flag* set, then the Consumer can always dequeue from the SEVD, and the Consumer can
3061 wait on the SEVD, but only if they disable the SEVD first (see *it_evd_modify*). Attempting to
3062 wait on the SEVD when disallowed will return IT_ERR_INVALID_EVD_STATE.

3063 The *evd_flag* bit value of IT_EVD_CREATE_FD set indicates that the Consumer requests
3064 creation of a File Descriptor associated with the EVD (either SEVD or AEVD). If the EVD has
3065 an associated *fd*, then the Consumer can wait on the EVD if they disable the EVD first (see
3066 *it_evd_modify*). Attempting to wait on the EVD when disallowed will return

3067
3068

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

3069
3070

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.

3071
3072
3073
3074
3075
3076

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.

3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088

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.

3089

A “Notification Event” is one of the following:

3090
3091
3092
3093

- 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

3094

- Any Event of Event Stream other than IT_DTO_EVENT_STREAM

3095
3096
3097

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*.

3098 For SEVD, the *sevd_threshold* must be set to either the value `IT_THRESHOLD_DISABLE` or
3099 to a value between at least one and at most the actual Implementation-allocated *sevd_queue_size*
3100 (can be determined by querying the SEVD). Setting *sevd_threshold* to
3101 `IT_THRESHOLD_DISABLE` will cause *it_evd_wait* to return only for Notification Events
3102 (specifically not for a threshold number of Events). For AEVD, the *sevd_threshold* is ignored.

3103 An *aevd_handle* specified on creation of a Simple EVD allows a Consumer to consolidate
3104 Notifications from multiple Simple Event Dispatchers (from the same Interface Adapter) to a
3105 single higher-level Aggregate Event Dispatcher. For an SEVD the *aevd_handle* value of
3106 `IT_NULL_HANDLE` means that no AEVD is associated with nor fed by the created SEVD. For
3107 Aggregate EVD creation this parameter must be `IT_NULL_HANDLE`; otherwise, *it_evd_create*
3108 will return `IT_ERR_AEVD_NOT_ALLOWED`.

3109 Alternatively, if the `IT_EVD_CREATE_FD` *evd_flag* bit value is set, then the Implementation
3110 will return a new unique file descriptor associated with the EVD. The file descriptor is placed
3111 into the contents of the *fd* pointer. The *fd* may be used in *select()* or *poll()* system calls and will
3112 be identified as ready to read when a Notification occurs on the underlying EVD. It is up to a
3113 Consumer then to go and dequeue Events from the EVD which is one-to-one associated with the
3114 particular *fd*. It is the Consumer's responsibility to keep track of the one-to-one association of *fd*
3115 and EVD.

3116 For Simple EVD the use of a value other than `IT_NULL_HANDLE` for *aevd_handle* is mutually
3117 exclusive with use of the `IT_EVD_CREATE_FD` *evd_flag*. That is, specifying both an
3118 *aevd_handle* not equal to `IT_NULL_HANDLE` and the `IT_EVD_CREATE_FD` bit set in
3119 *evd_flag* in a call to *it_evd_create* will fail and return a value of `IT_ERR_MISMATCH_FD`.

3120 IT-API supports the following configurations: Simple EVD, Simple EVD with associated *fd*,
3121 Simple EVD feeding Aggregate EVD, and Simple EVD feeding Aggregate EVD that is
3122 associated with *fd*.

3123 The Aggregate EVD specified by *aevd_handle* or the *fd* will be notified by the Implementation
3124 when a Notification Event arrives or *sevd_threshold* value is reached when the EVD is enabled.

3125 When an SEVD feeds an AEVD or *fd*, control of the capability of the feeding EVD to notify the
3126 fed AEVD or *fd* is done by enabling or disabling the feeding SEVD (see *it_evd_modify*).

3127 By default, the created EVD is enabled. An enabled SEVD will cause *aevd_handle* or *fd* (if
3128 applicable) to be notified when an Event arrival causes Notification criteria to be reached on that
3129 SEVD. An enabled AEVD will cause *fd* (if applicable) to be notified when an Event arrival
3130 causes Notification criteria to be reached. Notification is done on *aevd_handle* by generating an
3131 `IT_AEVD_NOTIFICATION_EVENT` for the AEVD if the `IT_EVD_DEQUEUE_`
3132 `NOTIFICATIONS` *evd_flag* bit is set on AEVD creation. When Notification is necessary for an
3133 AEVD with the `IT_EVD_DEQUEUE_NOTIFICATIONS` *evd_flag* bit cleared, no `IT_AEVD_`
3134 `NOTIFICATION_EVENT` will be enqueued; rather, the AEVD Consumer will be unblocked
3135 with the underlying SEVD Event delivered to it. Notification is done on the *fd* by marking it as
3136 ready to read.

3137 Consumers cannot wait on an enabled SEVD that feeds an AEVD or *fd*.

3138 A disabled feeding EVD will not generate Notification to the fed AEVD or *fd*. Consumers can
3139 wait on a disabled SEVD, unless it is associated with an AEVD with the `IT_EVD_`
3140 `DEQUEUE_NOTIFICATIONS` *evd_flag* bit cleared.

3141 An SEVD preserves the order of Events within each individual Event Stream as provided by the
3142 underlying Transport. No order is defined between Events of different Event Streams, even
3143 when they are of the same Event Stream type. For IT_DTO_EVENT_STREAM Event Stream
3144 type, the order of the Event completions is defined for each DTO and RMR post-operation on
3145 the Endpoint. No order is defined between Events of Event Streams coming from different
3146 SEVDs for an AEVD with the IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* bit cleared.
3147 The order of Events of the IT_AEVD_NOTIFICATION_EVENT Event Stream is
3148 Implementation-dependent.

3149 If the IT_EVD_DEQUEUE_NOTIFICATIONS bit is cleared in *evd_flag* on AEVD creation, the
3150 Consumer, when blocked in *it_evd_wait* and an SEVD Notification occurs, is unblocked and
3151 dequeues a lower-level Event from the same SEVD that caused Notification.

3152 Multiple SEVDs can feed the same AEVD. An SEVD generates a Notification for an AEVD
3153 when an Event arriving on the SEVD causes the SEVD to reach Notification criteria (but only if
3154 the SEVD is enabled).

3155 SEVD and AEVD can support multiple waiters. For an SEVD the *sevd_threshold* value must be
3156 one for multiple waiters to be supported.

3157 An SEVD waiter will block when the SEVD queue is empty. An AEVD waiter will block when
3158 all associated SEVDs are empty. An SEVD waiter may block when the SEVD is not in the
3159 Notification criteria. An AEVD waiter may block when all associated SEVDs are not in the
3160 Notification criteria. An SEVD waiter will return if there is a Notification Event on the queue or
3161 if the number of Events on the SEVD is equal or larger then *threshold*. An AEVD waiter will
3162 return if there is a Notification Event on any of the associated SEVDs or any of the associated
3163 SEVDs has a number of Events larger or equal to its *sevd_threshold*.

3164 If an arriving Event causes an SEVD to reach Notification criteria, then an SEVD waiter will be
3165 unblocked, if one exists. If there are multiple waiters on the SEVD, as many waiters as there are
3166 Events available on the SEVD may be unblocked. If the SEVD is enabled and associated with an
3167 AEVD or *fd*, then Notification will be generated for that AEVD or *fd*. In the AEVD case, as
3168 many Notifications may be generated as there are Events available on all SEVDs of the AEVD.

3169 *it_evd_dequeue* from an SEVD will return an Event, if one exists, from the SEVD queue
3170 regardless of whether there are waiters, except when the SEVD is associated with an AEVD with
3171 the IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* bit cleared. In the latter case, dequeue
3172 from the SEVD is not allowed. For an AEVD with the IT_EVD_DEQUEUE_NOTIFICATIONS
3173 *evd_flag* bit cleared, dequeue from the AEVD will return an Event, if one exists, from any of its
3174 associated SEVDs. For an AEVD with the IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag*
3175 bit set, dequeue from the AEVD will return an IT_AEVD_NOTIFICATION_EVENT if any of
3176 the associated enabled SEVDs is in Notification criteria or may return an
3177 IT_AEVD_NOTIFICATION_EVENT if any of the associated enabled SEVD simply has an
3178 Event.

3179 RETURN VALUE

3180 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

3181 IT_ERR_INVALID_IA The Interface Adapter Handle (*ia_handle*) was invalid.

3182 IT_ERR_INVALID_EVD_TYPE The Event Stream Type for the Event Dispatcher was
3183 invalid.

3184	IT_ERR_INVALID_FLAGS	The flags value was invalid.
3185 3186	IT_ERR_RESOURCE_QUEUE_SIZE	The underlying transport could not allocate the requested <i>sevd_queue_size</i> resources at this time.
3187 3188	IT_ERR_INVALID_THRESHOLD	An invalid value for the Simple Event Dispatcher threshold was specified.
3189 3190	IT_ERR_INVALID_AEVD	The Aggregation Event Dispatcher Handle (<i>aevd_handle</i>) was invalid.
3191 3192	IT_ERR_RESOURCES	The requested operation failed due to insufficient resources.
3193 3194	IT_ERR_MISMATCH_FD	An illegal request was made for both the File Descriptor and the Aggregation Event Dispatcher.
3195 3196 3197	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.
3198 3199	IT_ERR_ASYNC_AFF_EVD_EXISTS	The Asynchronous Affiliated Event Dispatcher already exists.
3200 3201	IT_ERR_ASYNC_UNAFF_EVD_EXISTS	The Asynchronous Unaffiliated Event Dispatcher already exists.
3202 3203 3204 3205	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.

3206 **APPLICATION USAGE**

3207 Consumers may use SEVDs with a pure polling model. Consumers create SEVDs and dequeue
3208 from them directly. The Consumer threads never wait on the SEVDs and just dequeue Events
3209 when they are ready to process them.

3210 Alternatively, Consumers may create SEVDs and wait on and dequeue from them directly. This
3211 also potentially requires many waiting threads, one per SEVD.

3212 For the “non-thread-safe” Implementation the Consumer cannot have multiple threads calling on
3213 the same EVD Handle simultaneously. When multiple threads retrieve Events concurrently from
3214 the same SEVD, each Event will be retrieved exactly once but it is unpredictable which thread
3215 will retrieve any particular Event.

3216 The use of an AEVD can reduce the number of distinct waiting threads required for an
3217 application. EVDs must be enabled to generate Notifications for the AEVD.

3218 Consumers can wait on an AEVD that had been created with the
3219 IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* bit set and all feeding SEVDs enabled. When
3220 wait returns, a returned IT_AEVD_NOTIFICATION_EVENT_STREAM Event identifies the
3221 SEVD that caused the unblocking. Consumer can then dequeue Events directly from that SEVD
3222 or any other SEVD that feeds the AEVD. Thus, the Consumer can choose to service the SEVDs
3223 feeding the AEVD in any order they wish.

3224 Consumers can wait on an AEVD that had been created with the
3225 IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* bit cleared and all feeding SEVDs enabled.
3226 When wait returns, it provides the first Event from an SEVD that is in Notification status.
3227 Consumer can then dequeue Events only from the AEVD. This dequeuing will provide Events
3228 from all SEVDs that feed the AEVD.

3229 The order of returned Events from the AEVD is Implementation-dependent. If Events are
3230 retrieved from a given AEVD strictly by a single thread, the order of each Event from its
3231 underlying SEVDs is maintained, but the order in which SEVDs are selected by the AEVD is
3232 Implementation-dependent. If Events are retrieved from a given AEVD by more than one thread,
3233 no order guarantees are made.

3234 The use of a file descriptor can also reduce the number of distinct waiting threads. File
3235 descriptors also can be used to wait for Notification Events across multiple Interface Adapters or
3236 Events not generated by this API. EVDs must be enabled to generate Notifications for the file
3237 descriptor.

3238 Consumers can *select* or *poll* on multiple *fds* that are associated with EVDs. The return for the
3239 *select* or *poll* call identifies the notifying *fd*. It is the Consumer's responsibility to keep track of
3240 which EVD is associated with each *fd*. The Consumer can dequeue Events from the EVD one-to-
3241 one associated with that *fd* using *it_evd_dequeue*.

3242 Typically, if the Consumer chooses to use an AEVD, they are then prohibited from waiting on
3243 the underlying SEVDs (see the Description section above for exceptions) and also may be
3244 prohibited from dequeuing from the underlying SEVDs (again see the Description section
3245 above for details). If the Consumer chooses to use an *fd*, then they are prohibited from waiting
3246 on the underlying AEVD(s) or SEVD(s).

3247 Overflow may occur and may not be reported to the Consumer via Events if there is no Simple
3248 EVD for IT_ASYNC_AFF_EVENT_STREAM or IT_ASYNC_UNAFF_EVENT_STREAM.
3249 Additionally, an IA can enter catastrophic state and not notify the Consumer about it if there is
3250 no Simple EVD for IT_ASYNC_UNAFF_EVENT_STREAM or if it has overflowed. For the
3251 effect of catastrophic error, see *it_unaffiliated_event_t* and *it_ia_create*.

3252 When an IA supports Spigot *online* or *offline* Events the number of Events that can be generated
3253 for the Unaffiliated Asynchronous Event Stream is potentially unbounded, but the queuing
3254 capacity of an EVD is finite. This can potentially lead to Events that are generated for the
3255 Unaffiliated Asynchronous Event Stream being silently discarded by the Implementation. Events
3256 that are generated for the Affiliated (or Unaffiliated) Asynchronous Event Stream will be silently
3257 discarded by the Implementation until such time as an EVD is created to hold the Affiliated (or
3258 Unaffiliated) Asynchronous Event Stream. If the Consumer needs to know with certainty the
3259 state of an entity that can generate an Unaffiliated Asynchronous Event (e.g., a Spigot), it should
3260 query for that state itself rather than relying upon getting a state change Notification via the
3261 Unaffiliated Asynchronous Event Stream.

3262 IT_ASYNC_AFF_EVENT_STREAM and IT_ASYNC_UNAFF_EVENT_STREAM SEVDs
3263 store Events that notify users of errors and other conditions that affect IA operation. These
3264 Events are usually unpredictable, which can make determining an appropriate size for these
3265 queues a challenge. Users should consider the size and type of the fabric, their resource usage,
3266 and their message patterns when setting the *sevd_queue_size* parameter for these EVDs.

3267 **FUTURE DIRECTIONS**
3268 IT-API support for a callback routine being invoked when an Event is enqueued on an SEVD
3269 may be added in the future.
3270 Aggregate EVD support for multiple IAs may be added in the future.

3271 **SEE ALSO**
3272 *it_evd_post_se()*, *it_ep_rc_create()*, *it_ep_ud_create()*, *it_listen_create()*,
3273 *it_ud_service_request_handle_create()*, *it_evd_query()*, *it_evd_modify()*, *it_evd_wait()*,
3274 *it_evd_dequeue()*, *it_evd_free()*, *it_event_t*, *it_dto_flags_t*, *it_unaffiliated_event_t*,
3275 *it_ia_create()*, *it_ia_info_t*

it_evd_dequeue()

3276

3277 NAME

3278

it_evd_dequeue – dequeue for Events from Event Dispatcher

3279 SYNOPSIS

3280

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

3281

```
it_status_t it_evd_dequeue(  
    IN  it_evd_handle_t  evd_handle,  
    OUT it_event_t       *event  
);
```

3282

3283

3284

3285

3286 DESCRIPTION

3287

evd_handle: Handle for simple or aggregate Event Dispatcher.

3288

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

3289

3290

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

3291

3292

3293

3294

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

3295

3296

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

3297

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

3298

3299

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

3300

3301

3302

3303

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

3304

3305

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

3306

3307

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

3308

3309

Attempting to dequeue from the SEVD when disallowed will return IT_ERR_INVALID_EVD_STATE.

3310

3311

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

3312

3313

3314

For IT_DTO_EVENT_STREAM Events when a Completion Event is returned for a given Send, RDMA Read, RDMA Write, RMR Link, or RMR Unlink operation that was posted to an

3315

3316 Endpoint, the Implementation guarantees that all Send, RDMA Read, RDMA Write, RMR Link,
3317 and RMR Unlink operations that were posted to the Endpoint prior to the one whose Completion
3318 Event was returned have also completed regardless of their *dto_flag* value for
3319 IT_COMPLETION_FLAG.

3320 The SEVD *sevd_threshold* value has no effect on this operation.

3321 **RETURN VALUE**

3322 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

3323 IT_ERR_QUEUE_EMPTY There were no entries on the Event Dispatcher queue.

3324 IT_ERR_INVALID_EVD The Event Dispatcher Handle (*evd_handle*) was invalid.

3325 IT_ERR_INVALID_EVD_STATE The attempted operation was invalid for the current state
3326 of the Event Dispatcher.

3327 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the
3328 disabled state. None of the output parameters from this
3329 routine are valid. See *it_ia_info_t* for a description of the
3330 disabled state.

3331 **APPLICATION USAGE**

3332 For an AEVD with IT_EVD_DEQUEUE_NOTIFICATIONS set, receipt of an
3333 IT_AEVD_NOTIFICATION_EVENT Event indicates that the SEVD (identified by *evd_handle*
3334 in the Event) reached Notification status or has Events available. By the time the Consumer calls
3335 *it_evd_dequeue* on the returned SEVD it may be empty or may not be in the Notification Criteria
3336 any longer if there are multiple dequeuers from the SEVD.

3337 For the “non-thread-safe” Implementation, the Consumer cannot have multiple threads calling
3338 *dequeue* on the same EVD Handle simultaneously.

3339 When multiple threads retrieve Events concurrently from the same SEVD, each Event will be
3340 retrieved exactly once but it is unpredictable which thread will retrieve any particular Event.

3341 **SEE ALSO**

3342 *it_evd_create()*, *it_evd_wait()*, *it_event_t*

it_evd_free()

3343

3344 NAME

3345 `it_evd_free` – destroy an Event Dispatcher

3346 SYNOPSIS

```
3347 #include <it_api.h>
3348
3349 it_status_t it_evd_free(
3350     IN it_evd_handle_t evd_handle
3351 );
```

3352 DESCRIPTION

3353 `evd_handle` Handle to Simple or Aggregate Event Dispatcher.

3354 `it_evd_free` Destroys an Event Dispatcher.

3355 On successful completion, all Events on the queue of the specified Event Dispatcher are lost.

3356 `it_evd_free` will return `IT_ERR_EVD_BUSY` if the EVD is still associated with an active Event
3357 Stream feeding it for all Event Streams except `IT_ASYNC_AFF_EVENT_STREAM`,
3358 `IT_ASYNC_UNAFF_EVENT_STREAM`, and `IT_SOFTWARE_EVENT_STREAM`.
3359 `it_evd_free` may be called at any time for `IT_ASYNC_AFF_EVENT_STREAM`,
3360 `IT_ASYNC_UNAFF_EVENT_STREAM`, and `IT_SOFTWARE_EVENT_STREAM` Event
3361 Streams but Events may be lost.

3362 An AEVD with `IT_EVD_DEQUEUE_NOTIFICATIONS` set may be dissociated from its
3363 SEVDs through use of `it_evd_modify` on each SEVD or through use of `it_evd_free` on each
3364 SEVD. An AEVD with the `IT_EVD_DEQUEUE_NOTIFICATIONS evd_flag` bit clear may be
3365 dissociated from SEVDs through use of `it_evd_free` on each SEVD. DTO SEVDs may be
3366 disassociated from their DTO Event Streams through use of `it_ep_free` on each associated
3367 Endpoint. Communication Management Request SEVDs may be disassociated from their Event
3368 Streams through use of `it_listen_free` on each associated listen Handle. Communication
3369 Management Message SEVDs may be disassociated from their Event Streams through use of
3370 `it_ep_free` on each associated Endpoint.

3371 An EVD that has an attached callback routine can be successfully freed while the callback
3372 routine is still attached. (See `it_evd_callback_attach` for a description of how a callback routine
3373 is attached to an EVD.) If an attempt is made to free an EVD while an invocation of a callback
3374 routine associated with the EVD is outstanding, however, `it_evd_free` will fail with an
3375 `IT_ERR_INVALID_EVD_STATE` error.

3376 Once `it_evd_free` returns, `evd_handle` may no longer be used.

3377 This operation is applicable to both AEVD and SEVD Handles.

3378 RETURN VALUE

3379 A successful call returns `IT_SUCCESS`. Otherwise, an error code is returned as described below:

3380 `IT_ERR_INVALID_EVD` The Event Dispatcher Handle (`evd_handle`) was invalid.

3381 `IT_ERR_EVD_BUSY` The Event Dispatcher was still associated with active Event
3382 Streams.

3383 IT_ERR_INVALID_EVD_STATE The attempted operation was invalid for the current state of the
3384 Event Dispatcher.

3385 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the
3386 disabled state. None of the output parameters from this routine
3387 are valid. See [it_ia_info_t](#) for a description of the disabled state.

3388 **APPLICATION USAGE**

3389 If an EVD has a callback routine attached, the Consumer should wait until there is no callback
3390 routine invocation outstanding for that EVD before freeing the EVD, since failing to do so will
3391 cause [it_evd_free](#) to return an error. One way to do that is to first call [it_evd_callback_detach](#) to
3392 ensure that no further callback routine invocations can take place, and then to wait for the
3393 current callback routine invocation to return.

3394 **SEE ALSO**

3395 [it_evd_create\(\)](#), [it_evd_modify\(\)](#), [it_evd_query\(\)](#), [it_ep_free\(\)](#), [it_listen_free\(\)](#)

it_evd_modify()

3396

3397 NAME

3398 `it_evd_modify` – modify an existing Event Dispatcher

3399 SYNOPSIS

```
3400 #include <it_api.h>
3401
3402 it_status_t it_evd_modify(
3403     IN          it_evd_handle_t      evd_handle,
3404     IN          it_evd_param_mask_t mask,
3405     IN const    it_evd_param_t      *params
3406 );
```

3407 DESCRIPTION

3408 *evd_handle* Simple or Aggregate Event Dispatcher.

3409 *mask* Bitwise OR of flags for requested EVD parameters.

3410 *params* Pointer to Consumer-allocated structure that contains new Consumer-
3411 requested Event Dispatcher parameters.

3412 *it_evd_modify* changes the desired parameters of the Simple or Aggregate Event Dispatcher
3413 *evd_handle*. Parameters to be modified are specified by flags in *mask*. New values for the
3414 parameters are specified by the corresponding fields in the structure pointed to by *params*. Fields
3415 and their flag values are shown below. Note that parameters represented by fields of
3416 *it_evd_param_t* that are not shown below cannot be modified. See *it_evd_query* for definition of
3417 *it_evd_param_t* and *it_evd_param_mask_t*.

```
3418 typedef struct {
3419     ...
3420     size_t      sevd_queue_size; /* IT_EVD_PARAM_QUEUE_SIZE */
3421     size_t      sevd_threshold; /* IT_EVD_PARAM_THRESHOLD */
3422     it_evd_handle_t aevid; /* IT_EVD_PARAM_AEVD_HANDLE */
3423     ...
3424     it_boolean_t evd_enabled; /* IT_EVD_PARAM_ENABLED */
3425     it_boolean_t evd_overflowed; /* IT_EVD_PARAM_OVERFLOWED */
3426 } it_evd_param_t;
```

3427
3428 The definition of each field follows:

3429 *sevd_queue_size* Minimum size of the Simple EVD Event queue. Attempting to modify this
3430 field for an AEVD will return an `IT_ERR_INVALID_MASK` error code.

3431 *sevd_threshold* For Simple EVD only. Number of Events on a single Event Dispatcher queue
3432 required for Notification of the associated AEVD or FD and for SEVD
3433 waiters unblocking. Attempting to modify this field for an AEVD will return
3434 an `IT_ERR_INVALID_MASK` error code.

3435 *aevid* For Simple EVD only. The Handle for the new associated Aggregate EVD.
3436 Attempting to modify this field for an AEVD will return an
3437 `IT_ERR_INVALID_MASK` error code if the above criteria are not met.

3438 Attempting to associate an Aggregate EVD with an SEVD that has a callback
3439 routine attached will return an IT_ERR_CALLBACK_EXISTS error code.

3440 *evd_enabled* Consumer may set this *it_boolean_t* to the value IT_TRUE to indicate that an
3441 EVD should notify an associated AEVD or *fd* when Notification criteria are
3442 reached. Clearing *evd_enabled* (making it equal to IT_FALSE) will disable
3443 this capability. May be done at any time.

3444 *evd_overflowed* Consumer may clear this *it_boolean_t* (make it equal to IT_FALSE) to reset
3445 an overflow condition on the EVD. See *it_evd_create* for more details.

3446 *it_evd_modify* cannot be used to associate an AEVD with an SEVD that has a callback routine
3447 attached; an attempt to do so will result in IT_ERR_CALLBACK_EXISTS being returned.
3448 *it_evd_modify* can be used to change the AEVD associated with an SEVD only if the SEVD
3449 (*evd_handle*) is disabled, there is no *fd* associated with the SEVD, and the
3450 IT_EVD_DEQUEUE_NOTIFICATIONS flag is set in the *evd_flag* for the provided AEVD
3451 (*params->aevd*). Otherwise, IT_ERR_INVALID_EVD_STATE is returned.

3452 The new AEVD may have IT_EVD_DEQUEUE_NOTIFICATIONS set or cleared.

3453 If the new AEVD has IT_EVD_DEQUEUE_NOTIFICATIONS cleared, the Consumer cannot
3454 subsequently disassociate the SEVD from the new AEVD.

3455 The Consumer may disassociate an SEVD from an AEVD by specifying the value of
3456 IT_NULL_HANDLE for *aevd* only if the SEVD is disabled and the AEVD has IT_EVD_
3457 DEQUEUE_NOTIFICATIONS set. Otherwise, IT_ERR_INVALID_EVD_STATE is returned.

3458 Consumer cannot use *it_evd_modify* to request *fd* to be associated with SEVD; instead, the
3459 Consumer can only do so at *it_evd_create* time.

3460 To disassociate the *fd*, the Consumer can simply close the *fd*. This clears the bit for IT_EVD_
3461 CREATE_FD in *evd_flag* for the SEVD or AEVD.

3462 If *sevd_queue_size* is requested to be changed for SEVD then the Implementation is required to
3463 provide a queue size of at least *sevd_queue_size*, but is free to provide a larger queue size (or
3464 provide dynamic queue enlargement when needed). The Consumer can determine the actual
3465 queue size by querying the modified Simple Event Dispatcher.

3466 Attempting to modify *sevd_queue_size* to be less than *sevd_threshold* returns
3467 IT_ERR_INVALID_QUEUE_SIZE. Attempting to modify *sevd_threshold* to be greater than
3468 *sevd_queue_size* returns IT_ERR_INVALID_THRESHOLD. In both error cases, the operation
3469 will not change the respective parameter from its current value.

3470 If the number of entries on the Event queue is greater than the requested *sevd_queue_size*, the
3471 operation will return IT_ERR_INVALID_QUEUE_SIZE and not change the Event queue size.

3472 The Consumer can enable the SEVD (set *evd_enabled*) so that the SEVD will generate
3473 Notifications for the current or future associated AEVD or *fd*, if either of them exists. Enabling
3474 an SEVD prohibits the Consumer from waiting on the SEVD if it has an associated AEVD or *fd*.
3475 If SEVD is in Notification criteria then SEVD generates the Notification for an associated
3476 existing AEVD or *fd*.

3477 The Consumer can enable AEVD so that the AEVD will generate Notification for an associated
3478 *fd*. Enabling the AEVD disallows the Consumer from waiting on the AEVD if it has an
3479 associated *fd*. For an AEVD with IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* cleared,
3480 the Consumer can still dequeue Events from the AEVD.

3481 Enabling the enabled EVD has no effect.

3482 The Consumer can disable the SEVD (clear *evd_enabled*) so that the SEVD will not generate
3483 Notifications for an associated AEVD, callback routine, or *fd*, if any of them exist. If an
3484 associated AEVD has the IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* cleared, the AEVD
3485 can dequeue Events from the SEVD. The Consumer cannot wait on or dequeue from the SEVD
3486 that is associated with the AEVD that has the IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag*
3487 cleared even when SEVD is disabled. The Consumer can wait on or dequeue from the SEVD
3488 that is associated with the AEVD that has the IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag*
3489 set when SEVD is disabled.

3490 The Consumer can disable an AEVD so that the AEVD will not generate Notification for an
3491 associated *fd*. Disabling the AEVD allows the Consumer to wait on the AEVD.

3492 Disabling the disabled EVD has no effect.

3493 **RETURN VALUE**

3494 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

3495	IT_ERR_INVALID_EVD	The Event Dispatcher Handle (<i>evd_handle</i>) was invalid.
3496	IT_ERR_INVALID_MASK	The mask contained invalid flag values.
3497 3498	IT_ERR_INVALID_EVD_STATE	The attempted operation was invalid for the current state of the Event Dispatcher.
3499 3500	IT_ERR_RESOURCE_QUEUE_SIZE	The underlying transport could not allocate the requested <i>sevd_queue_size</i> resources at this time.
3501 3502 3503	IT_ERR_INVALID_QUEUE_SIZE	The requested Simple Event Dispatcher queue size (<i>sevd_queue_size</i>) was less than the outstanding Events on the Event queue.
3504 3505	IT_ERR_INVALID_THRESHOLD	An invalid value for the Simple Event Dispatcher threshold was specified.
3506 3507	IT_ERR_INVALID_AEVD	The Aggregation Event Dispatcher Handle (<i>aevd</i>) was invalid.
3508 3509	IT_ERR_RESOURCES	The requested operation failed due to insufficient resources.
3510 3511	IT_ERR_CALLBACK_EXISTS	An attempt was made to associate an AEVD with an SEVD that has a callback routine attached.
3512 3513 3514 3515	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.

3516 **SEE ALSO**

3517 *it_evd_create()*, *it_evd_query()*, *it_evd_free()*, *it_evd_callback_attach()*

it_evd_post_se()

3518

3519 NAME

3520 it_evd_post_se – post software Event on Simple Event Dispatcher

3521 SYNOPSIS

```
3522 #include <it_api.h>
3523
3524 it_status_t it_evd_post_se(
3525     IN          it_evd_handle_t  evd_handle,
3526     IN  const void *event
3527 );
```

3528 DESCRIPTION

3529 *evd_handle* Simple Event Dispatcher of IT_SOFTWARE_EVENT_STREAM Event
3530 Stream type.

3531 *event* Pointer to the Consumer-created Software Event.

3532 *it_evd_post_se* posts a software Event to the IT_SOFTWARE_EVENT_STREAM simple Event
3533 Dispatcher Event queue. This causes an Event to arrive on the Event Dispatcher Software Event
3534 Stream. The *event* pointer is opaque to the Implementation and release of the memory referenced
3535 by the *event* pointer in a software Event is the Consumer's responsibility.

3536 If the Event queue is full, the operation is completed unsuccessfully and returns
3537 IT_ERR_EVD_QUEUE_FULL. The *event* is not queued. Since the Event queue for software
3538 Events can never overflow, the Affiliated Asynchronous Event Dispatcher is not affected.

3539 *it_evd_post_se* can only be used to post software Events within the same process since
3540 *evd_handle* has the scope of a single IA.

3541 RETURN VALUE

3542 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

3543 IT_ERR_EVD_QUEUE_FULL The Simple Event Dispatcher queue was full.

3544 IT_ERR_INVALID_EVD The Event Dispatcher Handle (*evd_handle*) was invalid.

3545 IT_ERR_INVALID_SOFT_EVD The Simple Event Dispatcher Handle (*evd_handle*) was
3546 not an IT_SOFTWARE_EVENT_STREAM Event
3547 Dispatcher.

3548 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the
3549 disabled state. None of the output parameters from this
3550 routine are valid. See *it_ia_info_t* for a description of the
3551 disabled state.

3552 APPLICATION USAGE

3553 The Consumer can use this operation to unblock an AEVD waiter as well as passing specific
3554 instruction for the unblocked waiter. The SEVD for the Software Event should be associated
3555 with the AEVD. A software Event is a Notification Event and will unblock the waiter.

3556 **SEE ALSO**
3557 *it_evd_create(), it_software_event_t, it_evd_wait()*

it_evd_query()

3558

3559 NAME

3560 `it_evd_query` – query an existing Simple or Aggregate Event Dispatcher

3561 SYNOPSIS

```
3562 #include <it_api.h>
3563
3564 it_status_t it_evd_query(
3565     IN    it_evd_handle_t    evd_handle,
3566     IN    it_evd_param_mask_t mask,
3567     OUT   it_evd_param_t     *params
3568 );
3569
3570 typedef enum {
3571     IT_EVD_PARAM_ALL           = 0x000001,
3572     IT_EVD_PARAM_IA           = 0x000002,
3573     IT_EVD_PARAM_EVENT_NUMBER = 0x000004,
3574     IT_EVD_PARAM_FLAG         = 0x000008,
3575     IT_EVD_PARAM_QUEUE_SIZE   = 0x000010,
3576     IT_EVD_PARAM_THRESHOLD    = 0x000020,
3577     IT_EVD_PARAM_AEVD_HANDLE  = 0x000040,
3578     IT_EVD_PARAM_FD           = 0x000080,
3579     IT_EVD_PARAM_BOUND        = 0x000100,
3580     IT_EVD_PARAM_ENABLED      = 0x000200,
3581     IT_EVD_PARAM_OVERFLOWED   = 0x000400,
3582     IT_EVD_PARAM_CALLBACK     = 0x000800
3583 } it_evd_param_mask_t;
3584
3585 typedef struct {
3586     it_ia_handle_t    ia;                /* IT_EVD_PARAM_IA */
3587     it_event_type_t  event_number;       /* IT_EVD_PARAM_EVENT_NUMBER*/
3588     it_evd_flags_t   evd_flag;          /* IT_EVD_PARAM_FLAG */
3589     size_t           sevd_queue_size;    /* IT_EVD_PARAM_QUEUE_SIZE */
3590     size_t           sevd_threshold;     /* IT_EVD_PARAM_THRESHOLD */
3591     it_evd_handle_t  aevid;             /* IT_EVD_PARAM_AEVD_HANDLE*/
3592     int              fd;                /* IT_EVD_PARAM_FD */
3593     it_boolean_t     evd_bound;         /* IT_EVD_PARAM_BOUND */
3594     it_boolean_t     evd_enabled;       /* IT_EVD_PARAM_ENABLED */
3595     it_boolean_t     evd_overflowed;    /* IT_EVD_PARAM_OVERFLOWED */
3596     it_boolean_t     evd_callback;      /* IT_EVD_PARAM_CALLBACK */
3597 } it_evd_param_t;
```

3598 DESCRIPTION

3599 *evd_handle* Event Dispatcher.

3600 *mask* Bitwise OR of flags for requested EVD parameters.

3601 *params* Pointer to Consumer-allocated structure that the Implementation fills with
3602 Consumer-requested Event Dispatcher parameters.

3603 *it_evd_query* returns the desired parameters of the Simple or Aggregate Event Dispatcher
3604 *evd_handle* in the structure pointed to by *params*. On return, each field of *params* is only valid if

3605 the corresponding flag as shown adjacent to each field is set in the *mask* argument. The *mask*
3606 value `IT_EVD_PARAM_ALL` causes all fields to be returned.

3607 The definition of each field follows:

3608	<i>ia</i>	Handle for the Interface Adapter.
3609	<i>event_number</i>	Identifier for Event Stream type that can be enqueued to the EVD.
3610	<i>evd_flag</i>	Flags for Event Dispatcher. See <i>it_evd_create</i> for definitions and use of
3611		<i>evd_flag</i> .
3612	<i>sevd_queue_size</i>	Minimum size of the SEVD Event queue or zero for an AEVD.
3613	<i>sevd_threshold</i>	The number of non-notification Events on the Simple Event Dispatcher
3614		queue for Notification, unblocking.
3615	<i>aevd</i>	Handle for Aggregate EVD associated with SEVD or <code>IT_NULL_HANDLE</code>
3616		if none.
3617	<i>fd</i>	<i>File descriptor</i> corresponding to Event Dispatcher or -1 if none.
3618	<i>evd_bound</i>	When it has the value <code>IT_TRUE</code> , indicates that the EVD is tied to an Event
3619		Stream so Events can be queued on EVD. For an AEVD, indicates that
3620		SEVDs are tied to the AEVD.
3621	<i>evd_enabled</i>	When it has the value <code>IT_TRUE</code> , indicates: for an SEVD that it has been
3622		configured to notify an associated AEVD or <i>fd</i> when Notification criteria is
3623		reached; for an AEVD that it has been configured to notify an associated <i>fd</i>
3624		when it is notified by one of its associated SEVDs. See <i>it_evd_modify</i> .
3625	<i>evd_overflowed</i>	When it has the value <code>IT_TRUE</code> , indicates that the EVD has overflowed.
3626		See <i>it_evd_create</i> for more details.
3627	<i>evd_callback</i>	When it has the value <code>IT_TRUE</code> , indicates that a callback routine has been
3628		attached to the EVD. See <i>it_evd_callback_attach</i> .

3629 **RETURN VALUE**

3630 A successful call returns `IT_SUCCESS`. Otherwise, an error code is returned as described below:

3631	<code>IT_ERR_INVALID_EVD</code>	The Event Dispatcher Handle (<i>evd_handle</i>) was invalid.
3632	<code>IT_ERR_INVALID_MASK</code>	The mask contained invalid flag values.
3633	<code>IT_ERR_IA_CATASTROPHE</code>	The IA has experienced a catastrophic error and is in the
3634		disabled state. None of the output parameters from this
3635		routine are valid. See <i>it_ia_info_t</i> for a description of the
3636		disabled state.

3637 **SEE ALSO**

3638 *it_evd_create()*, *it_evd_modify()*, *it_evd_free()*, *it_ia_info_t*

it_evd_wait()

3639

3640 NAME

3641 it_evd_wait – wait for Events on Event Dispatcher

3642 SYNOPSIS

```
3643 #include <it_api.h>
3644
3645 it_status_t it_evd_wait(
3646     IN    it_evd_handle_t    evd_handle,
3647     IN    uint64_t           timeout,
3648     OUT   it_event_t         *event,
3649     OUT   size_t             *nmore
3650 );
```

3651 DESCRIPTION

3652 *evd_handle* Handle for Simple or Aggregate Event Dispatcher.

3653 *timeout* The duration of time, in microseconds, that the Consumer is willing to wait
3654 for an Event.

3655 *event* Pointer to the Consumer-allocated structure that the Implementation fills
3656 with the Event information.

3657 *nmore* The snapshot of the number of Events queued on the EVD at the time of
3658 *it_evd_wait* return. Only applicable for SEVD.

3659 *it_evd_wait* removes the first Event from the Event Dispatcher Event queue and fills the
3660 Consumer-allocated *event* structure with Event information. For the Event information and *event*
3661 structure, see *it_event_t*. The Consumer should allocate an Event structure big enough to hold
3662 any Event that the Event Dispatcher can deliver.

3663 The return value for *event* is defined only if *it_evd_wait* returns IT_SUCCESS.

3664 The Consumer can wait on an EVD that is not associated with any higher-level object (AEVD or
3665 *fd*).

3666 The Consumer should not wait on an SEVD that has an associated AEVD with the
3667 IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* bit clear. An attempt by the Consumer to
3668 wait on *evd_handle* for that type of SEVD will result in routine failure with the return value of
3669 IT_ERR_INVALID_EVD_STATE.

3670 The Consumer should not wait on an EVD that is associated with and enabled for Notification to
3671 higher-level objects. An attempt by the Consumer to wait on *evd_handle* that is associated with
3672 and enabled for Notification to a higher-level object will result in routine failure with the return
3673 value of IT_ERR_INVALID_EVD_STATE. However, the Consumer can wait on the EVD
3674 associated with the higher-level object if the EVD is disabled for Notification (except if the
3675 object is an AEVD with the IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* bit clear, as
3676 stated above).

3677 The Consumer should not wait on an EVD that has a callback routine attached (see
3678 *it_evd_callback_attach*). An attempt to do so will result in routine failure with the return value
3679 of IT_ERR_CALLBACK_EXISTS.

3680 An Implementation can support one or more simultaneous waiters on the same EVD (for thread-
3681 safety models see [Global Behavior](#)) if *sevd_threshold* value of *evd_handle* (see *it_evd_create*) is
3682 greater than one, then only a single waiter is supported. An attempt for more than one waiter to
3683 wait on the EVD will result in an immediate error with IT_ERR_WAITER_LIMIT return value.
3684 If *sevd_threshold* value of *evd_handle* is 1, then one or more simultaneous waiters can be
3685 supported for the SEVD.

3686 A waiter can be blocked. An SEVD waiter will block when the SEVD queue is empty. An
3687 AEVD waiter will block when all associated SEVDs are empty. An SEVD waiter may block
3688 when the SEVD has not reached the Notification criteria (see *it_evd_create* for the definition of
3689 the Notification criteria). An AEVD waiter may block when all associated SEVDs have not
3690 reached their Notification criteria.

3691 An SEVD waiter will return immediately if there is a Notification Event (see *it_evd_create* for
3692 the definition of the Notification Event) on the queue or if the number of Events on the SEVD is
3693 larger than or equal to *sevd_threshold*. An AEVD waiter with the IT_EVD_DEQUEUE_
3694 NOTIFICATIONS *evd_flag* bit cleared will return immediately if there is a Notification Event
3695 on any of the associated SEVDs or any of the associated SEVDs has a number of Events larger
3696 than or equal to its *sevd_threshold*. An AEVD waiter with the IT_EVD_DEQUEUE_
3697 NOTIFICATIONS *evd_flag* bit set will return immediately if there is an IT_AEVD_
3698 NOTIFICATION_EVENT available.

3699 If an arriving Event causes SEVD to reach Notification criteria, then SEVD waiter will be
3700 unblocked if one exists and if the SEVD is disabled and not associated with AEVD with the
3701 IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* bit cleared. As many waiters as there are
3702 Events available on the SEVD can be unblocked. If arriving Event causes the SEVD to reach
3703 Notification criteria and the SEVD is enabled for Notification to higher-level objects, then
3704 Notification will be generated for the associated AEVD or *fd*. If the associated AEVD has a
3705 waiter, then the waiter will be unblocked. As many Notifications can be generated as there are
3706 Events available on all SEVDs of the AEVD. As many waiters as there are Notifications can be
3707 unblocked. Which waiters will be woken and in what order they will be woken is
3708 Implementation-dependent.

3709 The *timeout* allows the Consumer to restrict the amount of time it will be blocked waiting for an
3710 Event arrival. The value of IT_TIMEOUT_INFINITE indicates that the Consumer will wait
3711 indefinitely for an Event arrival. Consumers should use caution in using this value because wait
3712 may never return if Notification is not generated. Consumers can use *signal* to unblock the
3713 waiter in this case.

3714 For IT_DTO_EVENT_STREAM Events, when a Completion Event is returned for a given
3715 Send, RDMA Read, RDMA Write, RMR Link, or RMR Unlink operation that was posted to an
3716 Endpoint, the Implementation guarantees that all Send, RDMA Read, RDMA Write, RMR Link,
3717 and RMR Unlink operations that were posted to the Endpoint prior to the one whose Completion
3718 Event was returned have also completed regardless of their *dto_flag* value for
3719 IT_COMPLETION_FLAG.

3720 For an SEVD, if the return value is neither IT_SUCCESS nor IT_ERR_TIMEOUT_EXPIRED,
3721 then the returned values of *nmore* and *Event* are undefined. If the return value is
3722 IT_ERR_TIMEOUT_EXPIRED, then the return value of *event* is undefined, but the return value
3723 of *nmore* is defined. If the return value is IT_SUCCESS, then the return values of both *nmore*
3724 and *event* are defined.

3725 For an AEVD *nmore* is undefined for all returns. If the return value is not IT_SUCCESS, then
3726 returned value *event* is undefined.

3727 The routine returns with return value IT_ERR_INTERRUPT when the waiter is unblocked by an
3728 OS signal.

3729 This call may block the caller's execution waiting for a remotely generated event.

3730 RETURN VALUE

3731 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

3732 IT_ERR_WAITER_LIMIT No more waiters are permitted for the Event Dispatcher.

3733 IT_ERR_INVALID_EVD The Event Dispatcher Handle (*evd_handle*) was invalid.

3734 IT_ERR_INVALID_EVD_STATE The attempted operation was invalid for the current state
3735 of the Event Dispatcher.

3736 IT_ERR_ABORT The Event Dispatcher has been destroyed.

3737 IT_ERR_INTERRUPT The Event Dispatcher waiter was unblocked by a signal.

3738 IT_ERR_TIMEOUT_EXPIRED The operation timed out.

3739 IT_ERR_CALLBACK_EXISTS The wait for Notification was disallowed because the
3740 EVD has a callback routine attached.

3741 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the
3742 disabled state. None of the output parameters from this
3743 routine are valid. See *it_ia_info_t* for a description of the
3744 disabled state.

3745 APPLICATION USAGE

3746 The Consumer should allocate an Event structure big enough to hold any Event that the Event
3747 Dispatcher can deliver. The Implementation is not able to check that the *event* that the Consumer
3748 provides is sufficient to hold a returned Event. As a result, a segmentation fault or memory
3749 corruption may occur if the Implementation overruns the user-specified memory.

3750 For an AEVD with IT_EVD_DEQUEUE_NOTIFICATIONS set, any IT_AEVD_
3751 NOTIFICATION_EVENT Event only indicates that the SEVD (identified by *evd_handle* in the
3752 Event) reached Notification criteria. The order in which the associated SEVD's AEVD
3753 Notification Events are delivered is Implementation-dependent. No restriction is imposed by the
3754 Implementation on dequeuing Events from the underlying SEVD. If other Consumer threads
3755 are independently dequeuing Events from the SEVD, the thread receiving the
3756 IT_AEVD_NOTIFICATION_EVENT may find the SEVD to be empty when it dequeues from
3757 the SEVD.

3758 For an AEVD with IT_EVD_DEQUEUE_NOTIFICATIONS set, receipt of an IT_AEVD_
3759 NOTIFICATION_EVENT Event indicates that the SEVD (identified by *evd_handle* in the
3760 Event) reached Notification status. If the Consumer fails to dequeue Events from the SEVD
3761 sufficient to remove it from Notification status, then an additional
3762 IT_AEVD_NOTIFICATION_EVENT Event for the SEVD will appear at the AEVD when the
3763 Consumer next calls *it_evd_wait* or *it_evd_dequeue*.

3764 For an AEVD with IT_EVD_DEQUEUE_NOTIFICATIONS set, receipt of an IT_AEVD_
3765 NOTIFICATION_EVENT Event indicates that the SEVD (identified by *evd_handle* in the
3766 Event) reached Notification status. By the time the Consumer calls *it_evd_dequeue* on the
3767 returned SEVD it may be empty or may not be in the Notification Criteria any longer if there are
3768 multiple dequeuers from the SEVD.

3769 The Consumer must be prepared to handle return from *it_evd_wait* with fewer than the expected
3770 number of Events or without any Notification Events on an EVD. This can occur for the
3771 following reasons:

- 3772 • The underlying Implementation does not support thresholding.
- 3773 • The underlying Implementation does not support IT_NOTIFY_FLAG.

3774 For *sevd_threshold* value of 1, if an Event is on the SEVD, then *it_evd_wait* will return
3775 immediately with IT_SUCCESS for the SEVD or the AEVD fed by the SEVD.

3776 For the “non-thread-safe” Implementation the Consumer should not have multiple threads
3777 calling on the same EVD Handle simultaneously. The Consumer should choose an
3778 Implementation that supports multi-threaded applications if they want to have multiple waiters.
3779 The Consumer should set the *sevd_threshold* to 1 for an SEVD if they want to use multiple
3780 waiters on the SEVD.

3781 When multiple threads retrieve Events concurrently from the same SEVD, each Event will be
3782 retrieved exactly once, but it is unpredictable which thread will retrieve any particular Event.

3783 The Consumer is advised not to destroy an EVD on which it is currently waiting. If the
3784 Consumer does so, the *it_evd_wait* routine may return IT_ERR_ABORT, or a segmentation
3785 violation may take place. Which behavior occurs is Implementation-dependent.

3786 **SEE ALSO**

3787 *it_evd_create()*, *it_event_t*, *it_post_atomic()*, *it_post_send()*, *it_post_sendto()*,
3788 *it_post_rdma_read()*, *it_post_rdma_write()*, *it_rmr_link()*, *it_rmr_unlink()*, *it_dto_events*,
3789 *it_dto_flags_t*, *it_evd_callback_attach()*

it_get_consumer_context()

3790

3791 NAME

3792 `it_get_consumer_context` – return the Consumer Context associated with an IT Object Handle

3793 SYNOPSIS

```
3794 #include <it_api.h>
3795
3796 it_status_t it_get_consumer_context(
3797     IN  it_handle_t  handle,
3798     OUT it_context_t *context
3799 );
```

3800 DESCRIPTION

3801 *handle* Handle of the IT-API object associated with the Consumer Context to be
3802 retrieved.

3803 *context* The address of the location where the retrieved Consumer Context is
3804 returned.

3805 *it_get_consumer_context* retrieves the Consumer Context associated with the specified *handle*. If
3806 the Consumer Context was never set (by a call to *it_set_consumer_context*), then the value of the
3807 returned Consumer Context is 0 and the immediate error IT_ERR_NO_CONTEXT is returned.

3808 The *handle* must be one of the IT-API Handle types, cast as an *it_handle_t*. See *it_handle_t* for a
3809 description of the valid Handle types.

3810 RETURN VALUE

3811 A successful call returns SUCCESS. Otherwise, an error code is returned as described below:

3812 IT_ERR_INVALID_HANDLE The *handle* was invalid.

3813 IT_ERR_NO_CONTEXT The *handle* does not have an associated Context.

3814 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the disabled
3815 state. None of the output parameters from this routine are valid.
3816 See *it_ia_info_t* for a description of the disabled state.

3817 EXAMPLES

3818 The following code example demonstrates the use of a cast in the call to
3819 *it_get_consumer_context*. The *lmr* object is cast to the generic *it_handle_t* type for the call.

```
3820 it_lmr_handle_t lmr;
3821 it_context_t cxt;
3822 it_get_consumer_context( (it_handle_t) lmr, &cxt);
```

3823 SEE ALSO

3824 *it_set_consumer_context()*, *it_context_t*, *it_handle_t*

it_get_pathinfo()

3849

3850 NAME

3851 `it_get_pathinfo` – retrieve a set of Paths that can be used to communicate with a given remote
3852 Network Address

3853 SYNOPSIS

```
3854 #include <it_api.h>
3855
3856 it_status_t it_get_pathinfo(
3857     IN     it_ia_handle_t    ia_handle,
3858     IN     size_t            spigot_id,
3859     IN     const it_net_addr_t *net_addr,
3860     IN OUT size_t            *num_paths,
3861     OUT    size_t            *total_paths,
3862     OUT    it_path_t         *paths
3863 );
```

3864 DESCRIPTION

3865 *ia_handle* The Handle for the IA that the caller wishes to use for communicating with
3866 the remote Network Address.

3867 *spigot_id* The Spigot on the IA that the caller wishes to use for communicating with
3868 the remote Network Address.

3869 *net_addr* The remote Network Address with which to communicate.

3870 *num_paths* On input, points to the count of the maximum number of Paths that the
3871 Consumer wishes to have returned. On output, points to the count of the
3872 total number of Paths that were actually returned, which is guaranteed to be
3873 less than or equal to the number that the Consumer requested. This is only
3874 valid on output if the call returns IT_SUCCESS.

3875 *total_paths* The total number of Paths that were available to access the remote Network
3876 Address. This may be greater than the number of Paths returned via
3877 *num_paths* if there were more Paths available than the maximum the
3878 Consumer wished to have returned. This is only valid if the call returns
3879 IT_SUCCESS.

3880 *paths* An array allocated by the Consumer that holds the returned Path(s). This
3881 only contains valid information if the call returns IT_SUCCESS.

3882 *it_get_pathinfo* is used to retrieve a set of Paths that can be used to reach the specified remote
3883 Network Address. The local component of the Path is given by the combination of *ia_handle*
3884 (which identifies the local IA to use), and *spigot_id* (which identifies the Spigot to be used on
3885 that IA). The set of Paths that can be used is returned in *paths*.

3886 How the Consumer chooses which IA and Spigot to use for the local component of the Path is
3887 outside the scope of the API. The API does, however, provide the *it_interface_list* routine to
3888 enumerate all Interfaces that could be used to find a possible Path. The Consumer can use the
3889 names returned as input to the *it_ia_create* routine, which will return an IA Handle that can
3890 subsequently be fed to *it_ia_query* to determine what the valid Spigot identifiers are for that IA.

3891 Several different Network Address formats are supported; see the reference page for
3892 *it_net_addr_t* for details. The mechanism by which the Consumer determines the remote
3893 Network Address to target is outside the scope of the API. If the underlying transport is one that
3894 supports IP Network Addresses, existing APIs (such as *gethostbyname*) for translating host
3895 names into IP addresses can be used to convert a hostname into an IP address.

3896 The Consumer is responsible for allocating the storage necessary to hold the returned set of
3897 *paths*. Since the Consumer may not know how many Paths are available, it passes the number of
3898 Paths for which it has allocated storage in the *num_paths* parameter on input. This routine will
3899 return no more than that number of Paths to the Consumer. If more Paths are available than the
3900 Consumer has allocated space for, an arbitrary subset of the available Paths will be provided to
3901 the Consumer. A Consumer that does not wish to deal with Path selection can therefore avoid
3902 doing so by always specifying a value of 1 for the total number of Paths it wishes to have
3903 returned.

3904 The set of Paths that are available to reach a given remote Network Address is dynamic, and can
3905 change over time. (For example, a link on a switch or router could become inoperative, thus
3906 decreasing the set of available Paths.) There is therefore no guarantee that given the same input
3907 parameters two different invocations of *it_get_pathinfo* will return the same results. The
3908 information returned by *it_get_pathinfo* is a snapshot of the Paths available at the time of the
3909 call. In addition, if the Consumer asks for fewer Paths than are available, the API may return a
3910 different set of Paths for two different invocations of *it_get_pathinfo* regardless of the state of
3911 the network.

3912 It is possible that no Paths are available to reach the given remote Network Address. In that case,
3913 *it_get_pathinfo* will return IT_SUCCESS, but the total number of Paths available pointed to by
3914 *num_paths* will be zero.

3915 Once the Consumer has chosen one of the set of Paths returned, it can furnish that Path as input
3916 to the *it_ep_connect* routine. Consumers that wish to construct their own Path can also do so by
3917 populating the *it_path_t* data structure themselves, although this is inherently a transport-
3918 dependent programming practice. See the reference page for *it_path_t* for details on the internal
3919 structure of a Path.

3920 This call may block the caller's execution waiting for a remotely generated event.

3921 RETURN VALUE

3922 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

3923	IT_ERR_INVALID_IA	The Interface Adapter Handle (<i>ia_handle</i>) was invalid.
3924	IT_ERR_INVALID_SPIGOT	An invalid Spigot ID was specified.
3925	IT_ERR_INVALID_ADDRESS	The Network Address specified in <i>net_addr</i> was invalid.
3926	IT_ERR_INVALID_NETADDR	The type of the Network Address specified in <i>net_addr</i> was
3927		not recognized.
3928	IT_ERR_INTERRUPT	The call was unblocked by a signal.
3929	IT_ERR_IA_CATASTROPHE	The Interface Adapter has experienced a catastrophic error
3930		and is in the disabled state. None of the output parameters
3931		from this routine are valid. See <i>it_ia_info_t</i> for a description
3932		of the disabled state.

3933 **SEE ALSO**
3934 *it_interface_list(), it_ia_create(), it_ia_query(), it_ep_connect(), it_listen_create()*

it_handoff()

3935

3936 NAME

3937 it_handoff – forward an incoming Connection Request to another Spigot and Connection
3938 Qualifier

3939 SYNOPSIS

```
3940 #include <it_api.h>
3941
3942 it_status_t it_handoff(
3943     IN const it_conn_qual_t      *conn_qual,
3944     IN size_t                    spigot_id,
3945     IN it_cn_est_identifier_t    cn_est_id
3946 );
3947
3948 typedef uint64_t it_cn_est_identifier_t;
```

3949 DESCRIPTION

3950 *conn_qual* The Connection Qualifier to which the Connection Request should be
3951 forwarded.

3952 *spigot_id* Interface Adapter Spigot to which the Connection Request should be
3953 forwarded.

3954 *cn_est_id* Connection establishment identifier associated with the Connection
3955 Request to be forwarded.

3956 *it_handoff* forwards a Connection Request to the specified Spigot and Connection Qualifier of
3957 the IA on which the Connection Request originally arrived. Specifying a *conn_qual* or *spigot_id*
3958 on any IA other than that on which the Connection Request originally arrived will yield
3959 IT_ERR_INVALID_CONN_QUAL or IT_ERR_INVALID_SPIGOT errors respectively. The
3960 forwarded Connection Request generates an IT_CM_REQ_CONN_REQUEST_EVENT Event
3961 at the listen point to which the request was forwarded. Forwarded Connection Request Events
3962 look identical to the original Events, therefore the Consumer cannot distinguish them. The
3963 connection establishment identifier, *cn_est_id*, is destroyed by this function.

3964 RETURN VALUE

3965 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

3966 ERR_INVALID_CN_EST_ID The connection establishment identifier (*cn_est_id*) was
3967 invalid.

3968 IT_ERR_INVALID_CONN_QUAL The Connection Qualifier (*conn_qual*) was invalid.

3969 IT_ERR_INVALID_SPIGOT An invalid Spigot ID was specified.

3970 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the
3971 disabled state. None of the output parameters from this
3972 routine are valid. See *it_ia_info_t* for a description of the
3973 disabled state.

3974 **APPLICATION USAGE**
3975 Calls to *it_reject*, *it_ep_accept*, and *it_handoff* which pertain to the same Endpoint should be
3976 serialized by the Consumer. Failure to abide by this restriction may result in a segmentation
3977 violation or other error.

3978 **SEE ALSO**
3979 *it_ep_connect()*, *it_reject()*, *it_ep_accept()*, *it_cm_req_events*

it_hton64()

3980
3981 **NAME**
3982 `it_hton64`, `it_ntoh64` – convert 64-bit integers between host and network byte order

3983 **SYNOPSIS**
3984

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


3985
3986

```
uint64_t it_hton64(  
3987     uint64_t hostint  
3988 );
```


3989
3990

```
uint64_t it_ntoh64(  
3991     uint64_t netint  
3992 );
```

3993 **DESCRIPTION**

3994 *hostint* 64-bit integer stored in host byte order.

3995 *netint* 64-bit integer stored in network byte order.

3996 The *it_hton64* routine converts its input argument *hostint* from host byte order to network byte
3997 order and returns the result.

3998 *it_ntoh64* converts its input argument *netint* from network byte order to host byte order and
3999 returns the result.

4000 On some platforms, host byte order and network byte order are identical and these functions
4001 simply return their input argument.

4002 **RETURN VALUE**

4003 Both functions always succeed and return their converted input argument.

4004 **APPLICATION USAGE**

4005 The individual bytes of integer variables are stored in memory in an order that is platform-
4006 dependent, which is known as “host byte order”. To facilitate the exchange of integer variables
4007 between platforms having different host byte orders, a platform-independent byte order known
4008 as “network byte order” has been defined. To portably send an integer to a network peer, the
4009 Consumer should convert it from host to network byte order and send the network byte order
4010 value. The receiving peer then converts from network byte order to its own host byte order.

4011 Note that an integer must be stored in host byte order to be used correctly in normal arithmetic
4012 operations.

4013 **SEE ALSO**

4014 *htonl()* [UNIX], *ntohl()* [UNIX]

it_ia_create()

4015

4016 NAME

4017 `it_ia_create` – create an Interface Adapter

4018 SYNOPSIS

```
4019 #include <it_api.h>
4020
4021 it_status_t it_ia_create(
4022     IN  const char      *name,
4023     IN  uint32_t        major_version,
4024     IN  uint32_t        minor_version,
4025     OUT it_ia_handle_t  *ia_handle
4026 );
```

4027 DESCRIPTION

4028	<i>name</i>	The name of the Interface for which to create an Interface Adapter.
4029	<i>major_version</i>	The IT-API major version that the Consumer will use in subsequent calls to the IA.
4030		
4031	<i>minor_version</i>	The IT-API minor version that the Consumer will use in subsequent calls to the IA.
4032		
4033	<i>ia_handle</i>	Upon successful return, points to an Interface Adapter Handle for the created Interface Adapter.
4034		

4035 `it_ia_create` is used to create an Interface Adapter. The Consumer identifies the Interface
4036 Adapter to be created by its Interface name, major and minor version numbers for the most
4037 recent version of the IT-API supported. The Consumer may select these parameters from the list
4038 returned by the [it_interface_list](#) call.

4039 IT-API Version 1.0 – also known as IT-API Issue 1.0 – has a major version number of 1 and a
4040 minor version number of 0. IT-API Version 2.0 has a major version number of 2 and a minor
4041 version number of 0. When a new IT-API specification is released, a unique combination of
4042 major and minor version numbers is associated with it. If the new specification is source code-
4043 compatible with the previous one, the major version number of the new specification will be the
4044 same as that of the previous one, and the minor version number will be incremented by one. If
4045 the new specification is not source code-compatible with the previous one, the major version
4046 number of the new specification will be incremented by one, and the minor version number will
4047 be zero.

4048 The latest version of the IT-API that an Implementation supports is returned from the
4049 [it_interface_list](#) call. For the *major_version* returned from that call, the Implementation shall
4050 support all minor versions less than or equal to the *minor_version* returned from that call. The
4051 Implementation is not required to support major versions of the IT-API previous to the one
4052 returned from [it_interface_list](#). If the Implementation does not support conversing with the IA
4053 using the requested previous major version of the IT-API, an error will be returned from
4054 `it_ia_create`.

4055 If successful, this routine returns an Interface Adapter Handle. The returned Interface Adapter
4056 Handle may be passed to other IT-API routines that create and manage Interface Adapter objects
4057 such as Event Dispatchers and Local Memory Regions.

4058 **RETURN VALUE**

4059 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

4060 IT_ERR_RESOURCES The requested operation failed due to insufficient
4061 resources.

4062 IT_ERR_INVALID_NAME The specified *name* was invalid.

4063 IT_ERR_INVALID_MAJOR_VERSION The requested IT-API major version number was not
4064 supported for this Interface Adapter.

4065 IT_ERR_INVALID_MINOR_VERSION The requested IT-API minor version number was not
4066 supported for this Interface Adapter.

4067 **SEE ALSO**

4068 *it_interface_list(), it_ia_query(), it_ia_free()*

it_ia_free()

4069

4070 NAME

4071 `it_ia_free` – free Interface Adapter Handle

4072 SYNOPSIS

```
4073 #include <it_api.h>
4074
4075 it_status_t it_ia_free(
4076     IN it_ia_handle_t ia_handle
4077 );
```

4078 DESCRIPTION

4079 `ia_handle` Identifies the Interface Adapter Handle to be freed.

4080 `it_ia_free` is used to free an Interface Adapter Handle.

4081 All IT Objects associated with the specified Interface Adapter Handle are freed before this
4082 routine returns. The documented semantics associated with freeing the various IT Objects are
4083 observed when these objects are freed by the call to `it_ia_free`. Further use by the Consumer of
4084 Handles for those freed IT Objects after this routine returns successfully may have unpredictable
4085 effects. All `it_ia_info_t` structures that were returned to the Consumer by `it_ia_query` that have
4086 not already been freed by the Consumer (via `it_ia_info_free`) are freed. Examining an
4087 `it_ia_info_t` that was associated with `ia_handle` after this routine returns may have unpredictable
4088 effects.

4089 All pending operations associated with the specified `ia_handle`, with the exception of EVD
4090 callback routine invocations, will be terminated before this routine returns. Posted Data Transfer
4091 Operations that are currently in progress will be terminated before this routine returns. The
4092 completion status of such DTOs is indeterminate; if the Consumer wishes to know the
4093 completion status of the DTOs they have issued, they should dequeue the relevant Completion
4094 Events before freeing the IA. All callers blocked in `it_evd_wait` calls associated with the
4095 specified `ia_handle` will be unblocked. If an attempt is made to free an IA while any of the
4096 EVDs associated with that IA has a callback routine invocation outstanding, the call to `it_ia_free`
4097 will fail with an `IT_ERR_INVALID_EVD_STATE` error.

4098 All Connections and pending Connection Requests associated with the specified `ia_handle` are
4099 terminated before this routine returns.

4100 RETURN VALUE

4101 A successful call returns `IT_SUCCESS`. Otherwise, an error code is returned as described below:

4102	<code>IT_ERR_INVALID_IA</code>	The Interface Adapter Handle (<code>ia_handle</code>) was invalid.
4103	<code>IT_ERR_INVALID_EVD_STATE</code>	The attempted operation was invalid for the current state of the
4104		Event Dispatcher.

4105 SEE ALSO

4106 `it_ia_create()`, `it_ia_query()`

it_ia_info_free()

4107

4108 NAME

4109 `it_ia_info_free` – free an *it_ia_info_t* structure that was returned by *it_ia_query*

4110 SYNOPSIS

```
4111 #include <it_api.h>
4112
4113 void it_ia_info_free(
4114     IN it_ia_info_t *ia_info
4115 );
```

4116 DESCRIPTION

4117 `ia_info` Points to an *it_ia_info_t* data structure that was previously returned from a
4118 call to *it_ia_query*.

4119 *it_ia_info_free* is used to free the memory for the data structure allocated and returned by the
4120 *it_ia_query* routine. The Consumer should use this routine rather than the *free* routine to
4121 deallocate the data structure pointed to by *ia_info*; unpredictable behavior can result if *free* is
4122 used. Since this routine deallocates the input data structure, the Consumer should not attempt to
4123 access it after successfully returning from this routine.

4124 This routine does not free any of the resources that are associated with the *it_ia_info_t* data
4125 structure; it only frees the data structure itself. In particular, calling this routine does not cause
4126 the EVD Handle associated with the EVD that contains the Affiliated Asynchronous Event
4127 Stream (if present) or the EVD Handle associated with the EVD that contains the Unaffiliated
4128 Asynchronous Event Stream (if present) to be freed.

4129 When an IA is freed (by calling *it_ia_free*), any *it_ia_info_t* structures that were returned by
4130 *it_ia_query* for that IA will also be freed. The Consumer can call *it_ia_info_free* to free an
4131 *it_ia_info_t* structure before the IA is freed. After the IA has been freed, calling *it_ia_info_free*
4132 to free an *it_ia_info_t* associated with that IA will have undefined results, and may result in
4133 memory corruption.

4134 SEE ALSO

4135 *it_ia_info_t()*, *it_ia_query()*

it_ia_query()

4136

4137 NAME

4138 `it_ia_query` – retrieve attributes of given Interface Adapter and its Spigots

4139 SYNOPSIS

```
4140 #include <it_api.h>
4141
4142 it_status_t it_ia_query(
4143     IN  it_ia_handle_t  ia_handle,
4144     OUT it_ia_info_t   **ia_info
4145 );
```

4146 DESCRIPTION

4147 `ia_handle`

Identifies the Interface Adapter to be queried.

4148 `ia_info`

Points to a pointer to an `it_ia_info_t` structure upon successful return. The `it_ia_info_t` structure contains the attributes of the Interface Adapter and the identity of its Spigots.

4151 `it_ia_query` is used to retrieve the attributes of an Interface Adapter and its associated Spigots.
4152 See the reference page `it_ia_info_t` for details of the attributes structure.

4153 This routine allocates the storage necessary to hold the returned `it_ia_info_t` structure. The
4154 Consumer should free the allocated storage using the `it_ia_info_free` routine; if the Consumer
4155 fails to do so, the Implementation will free the storage when `it_ia_free` is called for `ia_handle`.

4156 If the query is unsuccessful, the return value will indicate failure, no `it_ia_info_t` structure will
4157 be allocated, and `ia_info` will point to a NULL (IT_NO_ADDR) pointer.

4158 RETURN VALUE

4159 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

4160 IT_ERR_INVALID_IA

The Interface Adapter Handle (`ia_handle`) was invalid.

4161 IT_ERR_RESOURCES

The requested operation failed due to insufficient resources.

4162 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.

4166 SEE ALSO

4167 `it_ia_create()`, `it_ia_free()`, `it_ia_info_t`, `it_ia_info_free()`

it_interface_list()

4168

4169 NAME

4170 `it_interface_list` – retrieve information about the available Interfaces

4171 SYNOPSIS

```
4172 #include <it_api.h>
4173
4174 void it_interface_list(
4175     OUT    it_interface_t  *interfaces,
4176     IN OUT size_t          *num_interfaces,
4177     IN OUT size_t          *total_interfaces
4178 );
4179
4180 typedef struct {
4181     /* Most recent major version number of the IT-API supported by the
4182     Interface. */
4183     uint32_t major_version;
4184
4185     /* Most recent minor version number of the IT-API supported by the
4186     Interface. */
4187     uint32_t minor_version;
4188
4189     /* The transport that the Interface uses, as defined in
4190     it_ia_info_t. */
4191     it_transport_type_t transport_type;
4192
4193     /* The name of the Interface, suitable for input to it_ia_create.
4194     The name is a string of maximum length IT_INTERFACE_NAME_SIZE,
4195     including the terminating NULL character. */
4196     char name[IT_INTERFACE_NAME_SIZE];
4197 } it_interface_t;
```

4199 DESCRIPTION

4200	<i>interfaces</i>	An array allocated by the Consumer that contains the information returned for the Interface(s).
4201		
4202	<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.
4203		
4204		
4205		
4206		
4207	<i>total_interfaces</i>	Upon return, points to the number of Interfaces potentially available for Consumer use. A Consumer may specify NULL (IT_NO_ADDR) for this parameter if it does not wish to know how many Interfaces are potentially available.
4208		
4209		
4210		
4211		<i>it_interface_list</i> 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 <i>it_ia_create</i> call.
4212		
4213		

4214 The Consumer is responsible for allocating the storage necessary to hold the information for the
4215 returned set of Interfaces. Since the local Consumer may not know how many Interfaces are
4216 available, it passes the number of Interfaces for which it has allocated storage in the
4217 *num_interfaces* parameter on input. This routine will return information for no more than that
4218 number of Interfaces to the Consumer. If more Interfaces are available than the Consumer has
4219 allocated space for, information will be provided to the Consumer for only *num_interfaces* such
4220 Interfaces; which Interfaces information will be returned for is arbitrary in this case. Upon
4221 return, the value pointed to by *total_interfaces* is the total number of available Interfaces.

4222 The set of Interfaces available to the Consumer is dynamic, and can change over time. (For
4223 example, an Interface can become inoperative, thus decreasing the set of available Interfaces.)
4224 There is therefore no guarantee that given the same input parameters two different invocations of
4225 *it_interface_list* will return the same results. The information returned by *it_interface_list* is a
4226 snapshot of the Interfaces available at the time of the call. In addition, if the Consumer asks for
4227 fewer Interfaces than are available, the API may return information for a different set of
4228 Interfaces for two different invocations of *it_interface_list* regardless of the state of the
4229 Interfaces.

4230 It is possible that no Interfaces are available. In that case the total number of Interfaces available
4231 pointed to by *num_interfaces* will be zero.

4232 **EXAMPLES**

4233 The following example illustrates how the Consumer can check after it has created the IA to
4234 ensure that the information it retrieved from the *it_interface_list* call is still valid.

```
4235 it_interface_t  interface;
4236 size_t         num_interfaces;
4237 it_ia_handle_t ia;
4238 it_ia_info_t   *infop;
4239
4240 num_interfaces = 1;
4241 it_interface_list(&interface, &num_interfaces, IT_NO_ADDR);
4242 if (num_interfaces != 1) {
4243
4244     /* Failed to find any IAs; */
4245 }
4246
4247 if (it_ia_create( interface.name, interface.major_version,
4248 interface.minor_version, &ia) != IT_SUCCESS) {
4249
4250     /* The IA wasn't found. Assuming sufficient resources were available,
4251        this can happen if the Interface that was retrieved by the
4252        it_interface_list call is no longer available. */
4253 }
4254
4255 if (it_ia_query(ia, &infop) != IT_SUCCESS) {
4256
4257     /* This can happen if the Implementation didn't have sufficient
4258        resources to return the information for the IA. */
4259 }
4260
4261 if ((infop->transport_type != interface.transport_type) {
4262
```

```
4263     /* This can happen if the Interface that was retrieved by the
4264     it_interface_list call is no longer available. (A different
4265     Interface with the same name as was retrieved by it_interface_list
4266     is available as it turns out. The most likely reason this happened
4267     is that a new Interface was added to the system between the time
4268     it_interface_list was called and the time that it_ia_create was
4269     called.) */
4270 }
4271
4272     /* Validation of the information returned by it_interface_list is
4273     complete. */
```

4274 **APPLICATION USAGE**

4275 The Interface Adapter associated with a given *name* can change between the time that the
4276 *it_interface_list* routine is called and the time that *it_ia_create* is called to actually create the IA.
4277 For that reason, the Consumer should check after it has created the IA to ensure that the
4278 information it retrieved from the *it_interface_list* call is still valid.

4279 **SEE ALSO**

4280 *it_ia_create()*, *it_ia_info_t*

it_listen_create()

4281

4282 NAME

4283 it_listen_create – create a Listen Point for incoming Connection Requests to a Connection
4284 Qualifier

4285 SYNOPSIS

```
4286 #include <it_api.h>
4287
4288 it_status_t it_listen_create(
4289     IN     it_ia_handle_t    ia_handle,
4290     IN     size_t            spigot_id,
4291     IN     it_evd_handle_t   connect_evd,
4292     IN     it_listen_flags_t flags,
4293     IN OUT it_conn_qual_t    *conn_qual,
4294     OUT    it_listen_handle_t *listen_handle
4295 );
4296
4297 typedef enum {
4298     IT_LISTEN_NO_FLAG           = 0x0000,
4299     IT_LISTEN_CONN_QUAL_INPUT  = 0x0001,
4300     IT_LISTEN_SUPPRESS_IRD_ORD = 0x0002
4301 } it_listen_flags_t;
```

4302 DESCRIPTION

4303 *ia_handle*: Interface Adapter Handle.

4304 *spigot_id*: Interface Adapter Spigot identifier.

4305 *connect_evd*: The Handle of the Simple Event Dispatcher where Connection Request
4306 Events for this Listen Point will be posted. The Event Stream Type of the
4307 Simple Event Dispatcher must be IT_CM_REQ_EVENT_STREAM.

4308 *flags*: Bitwise OR of flag values. Can specify whether the Connection Qualifer is
4309 an input or output parameter and whether or not IRD/ORD values are to be
4310 used (if applicable).

4311 *conn_qual*: The Connection Qualifier for which the Consumer wants to listen for
4312 Connection Requests.

4313 *listen_handle*: Upon successful return points to a Handle to the created Listen Point.

4314 *it_listen_create* establishes a Listen Point for incoming Connection Requests for a particular
4315 Connection Qualifier on the Spigot identified. Incoming Connection Request Events will be
4316 posted to the Simple Event Dispatcher specified until the Listen Point is destroyed. The
4317 *listen_handle* returned can be passed to *it_listen_free* when the Listen Point is no longer needed.

4318 When the IT_LISTEN_CONN_QUAL_INPUT bit is set in *flags*, *conn_qual* is an input
4319 parameter. When this bit is clear, *conn_qual* is an output parameter and an available Connection
4320 Qualifier is returned through that parameter.

4321 The default behavior of the IT-API is to attempt to negotiate IRD/ORD between the active and
4322 passive sides as described in Chapter 5 where supported by the IA. For an IA where the

4323 *it_ia_info_t* values *ird_ord_ia_support* and *ird_ord_suppressible* are both IT_TRUE, the
 4324 IT_LISTEN_SUPPRESS_IRD_ORD bit may be set by the Consumer in order to cause the IA
 4325 not to perform IRD/ORD negotiation. For such an IA, if the
 4326 IT_LISTEN_SUPPRESS_IRD_ORD bit is cleared in *flags*, then the IRD/ORD Endpoint
 4327 attributes will be negotiated. For an IA where *ird_ord_ia_support* is IT_FALSE, the
 4328 IT_LISTEN_SUPPRESS_IRD_ORD bit is ignored. For an IA where *ird_ord_ia_support* is
 4329 IT_TRUE but *ird_ord_suppressible* is IT_FALSE, attempting to set
 4330 IT_LISTEN_SUPPRESS_IRD_ORD will yield an immediate error.

4331 A backlog for the incoming Connection Request Events is provided by the size of the Simple
 4332 Event Dispatcher to which the Events are directed. If a Connection Request arrives while the
 4333 Simple Event Dispatcher is full, it is discarded and the Active side of the Connection
 4334 establishment attempt will receive an IT_CM_MSG_CONN_NONPEER_REJECT_EVENT
 4335 Event, with IT_CN_REJ_TIMEOUT as the reject reason code.

4336 **RETURN VALUE**

4337 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

4338	IT_ERR_INVALID_CONN_QUAL	The Connection Qualifier (<i>conn_qual</i>) was invalid.
4339	IT_ERR_CONN_QUAL_BUSY	The Connection Qualifier was already in use.
4340	IT_ERR_NO_PERMISSION	The Consumer did not have the proper permissions to perform the requested operation.
4341		
4342	IT_ERR_RESOURCES	The requested operation failed due to insufficient resources.
4343		
4344	IT_ERR_INVALID_CONN_EVD	The Connection Simple Event Dispatcher Handle was invalid.
4345		
4346	IT_ERR_INVALID_IA	The Interface Adapter Handle (<i>ia_handle</i>) was invalid.
4347	IT_ERR_INVALID_SPIGOT	An invalid Spigot ID was specified.
4348	IT_ERR_INVALID_FLAGS	The <i>flags</i> value was invalid.
4349	IT_ERR_INVALID_EVD_TYPE	The Event Stream Type for the Event Dispatcher was invalid.
4350		
4351	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.
4352		
4353		
4354		

4355 **SEE ALSO**

4356 *it_listen_free()*, *it_listen_query()*, *it_cm_msg_events*

it_listen_free()

4357

4358 NAME

4359 it_listen_free – free a Listen Point

4360 SYNOPSIS

4361 #include <it_api.h>

4362

4363 it_status_t it_listen_free(

4364 IN it_listen_handle_t listen_handle

4365);

4366 DESCRIPTION

4367 *listen_handle* Identifies the Listen Point to be destroyed.

4368 Frees a Listen Point associated with a Connection Qualifier. Upon return no more Connection
4369 Requests will be posted for the associated Connection Qualifier. Previously posted un-reaped
4370 Connection Requests, if any, will remain valid on the *connect_evd* and therefore can be used as
4371 input to either *it_ep_accept* or *it_reject*.

4372 Once *it_listen_free* returns, *listen_handle* may no longer be used.

4373 RETURN VALUE

4374 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

4375 IT_ERR_INVALID_LISTEN The Listen Point Handle (*listen_handle*) was invalid.

4376 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the
4377 disabled state. None of the output parameters from this routine
4378 are valid. See *it_ia_info_t* for a description of the disabled
4379 state.

4380 SEE ALSO

4381 *it_listen_create()*, *it_listen_query()*

it_listen_query()

4382

4383 NAME

4384 `it_listen_query` – query parameters associated with a Listen Point

4385 SYNOPSIS

```
4386 #include <it_api.h>
4387
4388 it_status_t it_listen_query(
4389     IN    it_listen_handle_t    listen_handle,
4390     IN    it_listen_param_mask_t mask,
4391     OUT   it_listen_param_t     *params
4392 );
4393
4394 typedef enum {
4395     IT_LISTEN_PARAM_ALL           = 0x0001,
4396     IT_LISTEN_PARAM_IA_HANDLE    = 0x0002,
4397     IT_LISTEN_PARAM_SPIGOT_ID    = 0x0004,
4398     IT_LISTEN_PARAM_CONNECT_EVD  = 0x0008,
4399     IT_LISTEN_PARAM_CONN_QUAL    = 0x0010
4400 } it_listen_param_mask_t;
4401
4402 typedef struct {
4403     it_ia_handle_t    ia_handle;    /* IT_LISTEN_PARAM_IA_HANDLE */
4404     size_t            spigot_id;    /* IT_LISTEN_PARAM_SPIGOT_ID */
4405     it_evd_handle_t  connect_evd;   /* IT_LISTEN_PARAM_CONNECT_EVD */
4406     it_conn_qual_t   connect_qual;  /* IT_LISTEN_PARAM_CONN_QUAL */
4407 } it_listen_param_t;
```

4408 DESCRIPTION

4409 *listen_handle* Handle associated with the Listen Point being queried.

4410 *mask* Bitwise OR of flags for desired parameters.

4411 *params* Pointer to Consumer-allocated structure whose members are written with
4412 the desired Listen Point parameters and attributes.

4413 *it_listen_query* queries the parameters associated with a Listen Point. On return, each field of
4414 *params* is only valid if the corresponding flag as shown in the Synopsis is set in the *mask*
4415 argument. The *mask* value `IT_LISTEN_PARAM_ALL` causes all fields to be returned.

4416 The definition of each field of *params* follows:

4417 *ia_handle*: Interface Adapter Handle.

4418 *spigot_id* Interface Adapter Spigot identifier.

4419 *connect_evd* The Handle of the Simple Event Dispatcher where incoming Connection
4420 Request Events for this Listen Point are posted.

4421 *connect_qual* The Connection Qualifier associated with the Listen Point.

4422 **RETURN VALUE**

4423 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

4424 IT_ERR_INVALID_LISTEN The Listen Point Handle (*listen_handle*) was invalid.

4425 IT_ERR_INVALID_MASK The mask contained invalid flag values.

4426 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the
4427 disabled state. None of the output parameters from this routine
4428 are valid. See *it_ia_info_t* for a description of the disabled
4429 state.

4430 **SEE ALSO**

4431 *it_listen_free()*, *it_listen_create()*

it_lmr_create()

4432

4433 NAME

4434 `it_lmr_create` – create a Local Memory Region (LMR) and register with an Interface Adapter

4435 SYNOPSIS

```
4436 #include <it_api.h>
4437
4438 it_status_t it_lmr_create(
4439     IN      it_pz_handle_t    pz_handle,
4440     IN      void              *addr,
4441     IN      it_iobl_t         *iobl,
4442     IN      it_length_t       length,
4443     IN      it_addr_mode_t    addr_mode,
4444     IN      it_mem_priv_t     privs,
4445     IN      it_lmr_flag_t     flags,
4446     IN      uint32_t          shared_id,
4447     OUT     it_lmr_handle_t   *lmr_handle,
4448     IN OUT  it_rmr_context_t  *rmr_context
4449 );
4450
4451 typedef uint32_t it_rmr_context_t;
4452
4453 #ifdef IT_32BIT
4454     typedef uint32_t it_length_t; /* a 32-bit platform */
4455 #else
4456     typedef uint64_t it_length_t; /* a 64-bit platform */
4457 #endif
4458
4459 typedef enum {
4460     IT_LMR_FLAG_NONE           = 0x0001,
4461     IT_LMR_FLAG_SHARED        = 0x0002,
4462     IT_LMR_FLAG_NONCOHERENT   = 0x0004,
4463     IT_LMR_FLAG_NON_SHAREABLE = 0x0008
4464 } it_lmr_flag_t;
```

4465 DESCRIPTION

4466	<i>pz_handle</i>	Protection Zone in which to create LMR.
4467	<i>addr</i>	Starting address of LMR. The <i>addr</i> argument is ignored if an IOBL is provided and <i>addr_mode</i> is <code>IT_ADDR_MODE_RELATIVE</code> .
4468		
4469	<i>iobl</i>	I/O Buffer List (IOBL, Privileged Consumers only).
4470	<i>length</i>	Length of LMR in bytes.
4471	<i>addr_mode</i>	Addressing mode of LMR to be created.
4472	<i>privs</i>	Bitwise OR of access privilege values for LMR, taken from <i>it_mem_priv_t</i> .
4473	<i>flags</i>	Bitwise OR of modifier flags.
4474	<i>shared_id</i>	Identifier for sharing Interface Adapter translation resources.

4475 *lmr_handle* Returned Handle for created LMR.

4476 *rmr_context* Optionally returned Context allowing remote access to this LMR.

4477 The *it_lmr_create* routine allows an Interface Adapter to access a contiguous Local Memory
4478 Region in a given linear address space known to the Consumer; it allows registering memory
4479 prior to using it as the Source or Destination of a DTO. The region starts at address *addr*, also
4480 known as the Base Address of the LMR, and extends for *length* bytes. The specified address
4481 range must already be valid in the Consumer's linear address space. The Interface Adapter is
4482 implicitly identified by the *pz_handle* argument. Registering a memory range that does not
4483 correspond to physically backed memory, such as the non-cacheable I/O address space, may
4484 work on some Implementations but not others. Applications that rely on this behavior will not be
4485 portable. The range can refer to memory that is exclusive to the calling process, or is being
4486 shared with other processes. Some Interface Adapters may require a memory region to be locked
4487 in physical memory. Such locking, if required, will be performed by the *it_lmr_create*
4488 implementation and is not the Consumer's responsibility.

4489 Non-Privileged Consumers must set *iobl* to IT_NO_ADDR.

4490 Privileged Consumers may pass an I/O Buffer List in the *iobl* argument (*it_iobl_t*). Jointly with
4491 the *length* argument, the IOBL specifies the I/O Address range of the region to be linked. The
4492 first byte of the region is identified by the First-Byte Offset (*fbo*) member of the IOBL. The
4493 length of the IOBL is the number of bytes from the first byte of the region to the last byte of the
4494 IOBL. The *length* argument specifies the total size of the LMR in bytes, which is upper-bounded
4495 by the length of the IOBL.

4496 If the *addr_mode* argument (*it_addr_mode_t*) selects Absolute Addressing, then DTOs will
4497 access the LMR through absolute addresses in the given linear address space. If the *addr_mode*
4498 argument selects Relative Addressing, then DTOs will access the LMR through offsets relative
4499 to the Base Address of the LMR. Consumers can check the IA attribute
4500 *addr_mode_relative_support* to determine whether Relative Addressing is supported.

4501 If an IOBL is provided and Absolute Addressing is used, the *addr* parameter specifies the Base
4502 Address of the LMR; moreover, if the IOBL is a Page List (IOBL element length is guaranteed
4503 to be a power of two), the absolute address (*addr*) modulo the length (*elt_len*) of the first IOBL
4504 element must equal the First-Byte Offset (*fbo*) in the IOBL. If an IOBL is provided and Relative
4505 Addressing is used, the *addr* parameter is ignored.

4506 The type of access granted is specified by the *privs* argument as the bitwise-inclusive OR of one
4507 or more of the bit values IT_PRIV_LOCAL_READ, IT_PRIV_LOCAL_WRITE,
4508 IT_PRIV_REMOTE_READ, and IT_PRIV_REMOTE_WRITE. Unless otherwise noted below,
4509 all bit combinations are allowed. See *it_mem_priv_t* for bit definitions and predefined bit
4510 combinations. It is invalid to set *privs* to 0 (no access privileges), or to include
4511 IT_PRIV_REMOTE_WRITE without also including IT_PRIV_LOCAL_WRITE. Consumers
4512 are advised to use IT_PRIV_LOCAL_READ or IT_PRIV_LOCAL (which equals
4513 IT_PRIV_LOCAL_READ | IT_PRIV_LOCAL_WRITE) as local access privileges for
4514 maximum portability between transports (see also Extended Description). The access privileges
4515 required for a DTO's Source or Destination buffer are specified on the corresponding DTO
4516 reference page.

4517 It is not possible to grant access privileges to which a process is not already entitled. If the
4518 calling process does not have read or write access privileges to the memory region, then any
4519 attempt to grant those privileges to the Interface Adapter will cause *it_lmr_create* to fail.

4520 The *flags* argument is a bitwise OR of zero or more of the following options. The value
4521 IT_LMR_FLAG_NONE or 0 may be used to specify no options.

4522 IT_LMR_FLAG_SHARED This flag may be used to conserve finite Interface Adapter
4523 translation resources by sharing resources between multiple
4524 LMRs. The LMRs may have been created in the caller's process
4525 or in a different process. If set, then the Implementation will
4526 create a new LMR that re-uses resources from a matching LMR
4527 that has the IT_LMR_FLAG_NON_SHAREABLE flag cleared.
4528 Two LMRs are defined as matching if they have the same
4529 *shared_id* attribute, correspond to the same list of physical
4530 memory pages, have the same First-Byte Offset and the same
4531 length, correspond to the same I/O Address range, and have the
4532 same coherency mode. If any of these conditions are not met, the
4533 LMRs do not match. If a matching LMR is not found among the
4534 LMRs with the IT_LMR_FLAG_NON_SHAREABLE flag
4535 cleared, then a new LMR is created.

4536 IT_LMR_FLAG_NON_SHAREABLE Controls whether an LMR may be shared. If set, then the LMR is
4537 not eligible to be shared. Otherwise, the LMR may be shared.

4538 IT_LMR_FLAG_NONCOHERENT Controls whether an LMR is created in coherent or non-coherent
4539 mode. Coherent mode is the default and is supported by all
4540 Implementations. Non-coherent mode is not supported by all
4541 Implementations, and IT_LMR_FLAG_NONCOHERENT is
4542 silently ignored on such Implementations.

4543 Set IT_LMR_FLAG_NONCOHERENT to create an LMR in non-coherent mode. Non-coherent
4544 mode may yield higher throughput for large DTOs, but may also increase latency for small
4545 DTOs. The downside of requesting non-coherent mode is that the Consumer must synchronize
4546 between local and remote access to the memory region using the *it_lmr_refresh_from_mem* and
4547 *it_lmr_flush_to_mem* calls.

4548 The coherency mode of an LMR is inherited by any RMR that is linked to it.

4549 An RMR Context allowing remote access to the memory region will be created if the *privs*
4550 argument includes either IT_PRIV_REMOTE_READ or IT_PRIV_REMOTE_WRITE. The
4551 Context will be returned in the location pointed to by *rmr_context* if the input value of
4552 *rmr_context* is not IT_NO_ADDR. The returned *rmr_context* is only valid if *privs* includes
4553 remote privileges and the call returns successfully; otherwise, it is undefined. The RMR Context
4554 may also be retrieved using *it_lmr_query*. The *rmr_context* is returned in network byte order,
4555 and may be passed by value to any remote process that wishes to use the Context in DTOs that
4556 target the corresponding LMR. Destroying the LMR using *it_lmr_free* revokes remote access
4557 using this RMR Context. Calling *it_lmr_modify* with remote access enabled results in a new
4558 RMR Context being associated with the LMR and revokes remote access using any previous
4559 RMR Contexts, as long as the new RMR Context is not a re-use of a previously generated RMR
4560 Context.

4561 The newly created LMR Handle is returned in the *lmr_handle* argument. A process may create
4562 multiple LMRs, and the address ranges of different LMRs may overlap.

4563 After a memory range has been registered with the IA using *it_lmr_create*, the Consumer should
4564 not call routines outside of the IT-API that would invalidate any part of the memory referred to
4565 by the LMR or revoke access privileges that were granted to the IA at registration time.

4566 Disallowed operations include but are not limited to unmapping part of the range using *munmap*,
4567 revoking privileges using *mprotect*, unlocking memory using *munlock*, truncating a file for file-
4568 backed regions, etc. Violation of this rule may result in DTO failures, data corruption in the
4569 Consumer's LMR, and/or program termination. These effects may extend to other Consumer
4570 processes if the LMR is in shared memory. However, the Implementation must prevent any
4571 adverse effect on unrelated processes that do not use this memory object.

4572 EXTENDED DESCRIPTION

4573 For the InfiniBand Transport, it is invalid not to include `IT_PRIV_LOCAL_READ` in the *privs*
4574 argument, or to include `IT_PRIV_REMOTE_WRITE` without also including
4575 `IT_PRIV_LOCAL_WRITE` when creating an LMR; such attempts will result in an
4576 `IT_ERR_INVALID_PRIVS` immediate error. Moreover, if the Consumer wishes to link an
4577 RMR with the `IT_PRIV_REMOTE_WRITE` access privilege to the LMR, then the LMR's
4578 access privileges must include `IT_PRIV_LOCAL_WRITE`.

4579 Also for InfiniBand, the Implementation may round the requested *addr* down and/or round the
4580 requested *length* up, and thus allow the Interface Adapter to access memory slightly outside the
4581 specified boundaries, but never beyond the IA pages that include the requested starting and
4582 ending addresses. The actual starting address and the actual length may be queried using
4583 *it_lmr_query*. Note that if the *privs* argument enables remote access, then remote Consumers
4584 may also access memory slightly outside the requested boundaries. If this is undesirable, the
4585 Consumer should not enable remote access in this routine, but should instead create and link an
4586 RMR using *it_rmr_create* and *it_rmr_link*, respectively, which guarantees byte-level registration
4587 granularity.

4588 BACKWARDS COMPATIBILITY

4589 *it_lmr_create* called with *iobl* set to `IT_NO_ADDR` behaves identically to *it_lmr_create* (v2.0).

4590 *it_lmr_create* called with *iobl* set to `IT_NO_ADDR` and *addr_mode* set to
4591 `IT_ADDR_MODE_ABSOLUTE` behaves identically to *it_lmr_create* (v1.0).

4592 See also Appendix C.

4593 RETURN VALUE

4594 A successful call returns `IT_SUCCESS`. Otherwise, an error code is returned as described below:

4595	<code>IT_ERR_ACCESS</code>	The Consumer was not allowed to have the requested memory privileges.
4596		
4597	<code>IT_ERR_FAULT</code>	Part or all of the supplied address range was invalid.
4598	<code>IT_ERR_INVALID_ADDR_MODE</code>	The addressing mode (<i>addr_mode</i>) was invalid or unsupported.
4599		
4600	<code>IT_ERR_INVALID_FLAGS</code>	The <i>flags</i> value was invalid.
4601	<code>IT_ERR_INVALID_IOBL</code>	The IOBL specified with <i>iobl</i> was invalid. An IOBL is invalid if it has an unsupported type, if the number of IOBL elements exceeds the maximum that the LMR can support (<i>iobl_num_elts</i> as reported via <i>it_lmr_query</i>), if it includes elements of an unsupported length, or if it has an invalid First-Byte Offset (<i>fbo</i>). An IOBL that is a Page List is invalid if an IOBL element has an invalid page
4602		
4603		
4604		
4605		
4606		
4607		

4608		size or is not page aligned or, for Absolute Addressing only, if the absolute address (<i>addr</i>) modulo the length (<i>elt_len</i>) of the first IOBL element does not equal the First-Byte Offset (<i>fbo</i>) in the IOBL.
4609		
4610		
4611		
4612	IT_ERR_INVALID_LENGTH	An IOBL was provided and the requested LMR length (<i>length</i>) exceeded the length of the IOBL.
4613		
4614	IT_ERR_INVALID_PRIVS	The requested memory privileges (<i>privs</i>) contained an invalid flag.
4615		
4616	IT_ERR_INVALID_PZ	The Protection Zone Handle (<i>pz_handle</i>) was invalid.
4617	IT_ERR_RESOURCES	The requested operation failed due to insufficient resources.
4618		
4619	IT_ERR_RESOURCE_LMR_LENGTH	The underlying transport could not allocate an LMR of the requested <i>length</i> at this time.
4620		
4621	IT_ERR_PRIV_OPS_UNAVAILABLE	The IA does not support Privileged Mode operations.
4622	IT_ERR_NO_PERMISSION	The Consumer is not Privileged.
4623	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.
4624		
4625		
4626		

4627 APPLICATION USAGE

4628 Memory that is to be the Source or Destination of a DTO must first be registered using
4629 *it_lmr_create*. The Consumer typically copies the returned *lmr_handle* to an *it_lmr_triplet_t*
4630 structure that is used in DTO calls such as *it_post_send*.

4631 If the LMR is created with flags that enable remote access, then the Consumer typically passes
4632 the returned RMR Context to a remote peer using a DTO. The remote peer uses the RMR
4633 Context in RDMA calls such as *it_post_rdma_write* that access memory within the range of the
4634 LMR.

4635 Consumers can enable remote access more selectively over any portion of an LMR by creating
4636 an RMR and linking the RMR to the desired region of the LMR using *it_rmr_link*. This
4637 operation returns an RMR Context.

4638 A Consumer wishing to prevent an LMR becoming shared as a side-effect of another Consumer
4639 creating a shared LMR should set the IT_LMR_NON_SHAREABLE flag.

4640 A Consumer interested in (eventually) sharing a group of LMRs should clear the
4641 IT_LMR_NON_SHAREABLE flag, optionally set the IT_LMR_FLAG_SHARED flag, and
4642 specify a distinct value of *shared_id* for that group when creating these LMRs. The use of
4643 distinct values of *shared_id* for unrelated groups of shared LMRs improves the efficiency of the
4644 matching process, which searches only among LMRs having the same *shared_id* value.
4645 However, using distinct values of *shared_id* for unrelated LMRs is not necessary. For example,
4646 if uncoordinated Consumers supply the same value for *shared_id*, matches for an LMR will still
4647 be found if they exist, with no false matches, but the search may take longer on some

4648 Implementations. It is recommended that uncoordinated Consumers wishing to take advantage of
4649 potential sharing opportunities should adopt the convention of setting *shared_id* to zero.

4650 **SEE ALSO**

4651 *it_lmr_free()*, *it_lmr_query()*, *it_lmr_modify()*, *it_lmr_flush_to_mem()*,
4652 *it_lmr_refresh_from_mem()*, *it_addr_mode_t*, *it_iobl_t*, *it_mem_priv_t*

it_lmr_create_unlinked()

4653

4654 NAME

4655 `it_lmr_create_unlinked` – create an unlinked Local Memory Region (LMR)

4656 SYNOPSIS

```
4657 #include <it_api.h>
4658
4659 it_status_t it_lmr_create_unlinked(
4660     IN  it_pz_handle_t    pz_handle,
4661     IN  size_t            iobl_num_elts,
4662     IN  it_lmr_flag_t    flags,
4663     IN  uint32_t          shared_id,
4664     OUT it_lmr_handle_t  *lmr_handle
4665 );
```

4666 APPLICABILITY

4667 `it_lmr_create_unlinked` is supported only if the Interface Adapter attribute `lmr_link_support` (see
4668 [it_ia_info_t](#)) is `IT_TRUE`.

4669 DESCRIPTION

4670 `pz_handle` Protection Zone in which to create LMR.

4671 `iobl_num_elts` Number of entries allowed in corresponding IOBL.

4672 `flags` Bitwise OR of modifier flags.

4673 `shared_id` Identifier for sharing Interface Adapter translation resources.

4674 `lmr_handle` Returned Handle for created LMR.

4675 The `it_lmr_create_unlinked` routine allows a Privileged Consumer to request that an Interface
4676 Adapter allocate a set of Local Memory Region (LMR) resources without associating them with
4677 a range of local memory addresses. The LMR is returned in the unlinked state and is not fully
4678 usable until linked successfully using the [it_lmr_link](#) routine.

4679 The Interface Adapter is implicitly identified by the `pz_handle` argument.

4680 The `iobl_num_elts` argument identifies the requested number of IOBL elements to be allocated
4681 for this LMR handle. The Implementation may allocate more than the requested amount. The
4682 actual allocated will be at least that requested and may be queried using the [it_lmr_query](#)
4683 routine.

4684 The `flags` argument is a bitwise OR of zero or more of the following options. The value
4685 `IT_LMR_FLAG_NONE` or 0 may be used to specify no options.

4686 `IT_LMR_FLAG_NONCOHERENT` Controls whether an LMR is created in coherent or non-
4687 coherent mode. Coherent mode is the default and is supported
4688 by all Implementations. Non-coherent mode is not supported
4689 by all Implementations, and
4690 `IT_LMR_FLAG_NONCOHERENT` is silently ignored on
4691 such Implementations.

4692 IT_LMR_FLAG_NON_SHAREABLE Controls whether an LMR may be shared. If set, then the
4693 LMR is not eligible to be shared. Otherwise, the LMR may
4694 be shared.

4695 Set IT_LMR_FLAG_NONCOHERENT to create an LMR in non-coherent mode. Non-coherent
4696 mode may yield higher throughput for large DTOs, but may also increase latency for small
4697 DTOs. The downside of requesting non-coherent mode is that the Consumer must synchronize
4698 between local and remote access to the memory region using the *it_lmr_refresh_from_mem* and
4699 *it_lmr_flush_to_mem* calls.

4700 The coherency mode of an LMR is inherited by any RMR that is linked to it.

4701 The LMR is created in the unlinked state. For an unlinked LMR, the IT_LMR_FLAG_SHARED
4702 flag for the LMR will be reported as clear when queried.

4703 The newly created LMR Handle is returned in the *lmr_handle* argument.

4704 **RETURN VALUE**

4705 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

4706 IT_ERR_INVALID_FLAGS The *flags* value was invalid.

4707 IT_ERR_INVALID_PZ The Protection Zone Handle (*pz_handle*) was invalid.

4708 IT_ERR_OP_NOT_SUPPORTED The IA has no LMR Link support (*lmr_link_support* in
4709 *it_ia_info_t* is IT_FALSE).

4710 IT_ERR_NO_PERMISSION The Consumer is not Privileged.

4711 IT_ERR_RESOURCE_IOBL_LENGTH The underlying transport could not allocate an LMR with
4712 the requested number of IOBLs (*iobl_num_elts*) at this
4713 time.

4714 IT_ERR_RESOURCES The requested operation failed due to insufficient
4715 resources.

4716 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the
4717 disabled state. None of the output parameters from this
4718 routine are valid. See *it_ia_info_t* for a description of the
4719 disabled state.

4720 **APPLICATION USAGE**

4721 Memory that is to be the Source or Destination of a DTO must first be registered. Using
4722 *it_lmr_create_unlinked*, the Consumer requests allocation of an unlinked LMR with a capacity
4723 for a number of IOBL entries. The Consumer then calls *it_lmr_link* with an IOBL with less than
4724 or equal to *iobl_num_elts* entries to register the memory with the Interface Adapter. If the LMR
4725 is linked with flags that enable remote access, then the Consumer typically passes the returned
4726 RMR Context to a remote peer using a DTO. The remote peer uses the RMR Context in RDMA
4727 calls such as *it_post_rdma_write* that access memory within the range of the LMR.

4728 Having linked an LMR, the Consumer may unlink it locally using *it_lmr_unlink* or also by
4729 setting the IT_UNLINK_LOCAL_SINK flag on RDMA Read WRs or may unlink it remotely
4730 using *it_post_send_and_unlink*. Having unlinked an LMR, the Consumer may subsequently link
4731 it again.

4732 If the LMR created with *it_lmr_create_unlinked* becomes shared, it will no longer be possible to
4733 unlink the LMR (via a local *it_lmr_link* or via a remote *it_post_send_and_unlink*) – instead the
4734 Consumer must free the LMR (*it_lmr_free*). A Consumer wishing to prevent an LMR becoming
4735 shared as a side-effect of another Consumer creating a shared LMR should set the
4736 IT_LMR_NON_SHAREABLE flag. For more details on sharing LMRs, see Application Usage
4737 of *it_lmr_create*.

4738 **SEE ALSO**

4739 *it_lmr_link()*, *it_lmr_unlink()*, *it_lmr_free()*, *it_lmr_query()*, *it_lmr_modify()*,
4740 *it_lmr_flush_to_mem()*, *it_lmr_refresh_from_mem()*, *it_lmr_create()*,
4741 *it_post_send_and_unlink()*, *it_addr_mode_t*, *it_mem_priv_t*

it_lmr_flush_to_mem()

4742

4743 NAME

4744 it_lmr_flush_to_mem – make memory changes visible to an incoming RDMA Read or Atomic
4745 operation

4746 SYNOPSIS

```
4747 #include <it_api.h>  
4748  
4749 it_status_t it_lmr_flush_to_mem(  
4750     IN const it_lmr_triplet_t *local_segments,  
4751     IN size_t num_segments  
4752 );
```

4753 DESCRIPTION

4754 *local_segments* Array of buffer segments.

4755 *num_segments* Number of segments in the array.

4756 The *it_lmr_flush_to_mem* routine is needed if and only if an LMR was created in non-coherent
4757 mode using IT_LMR_FLAG_NONCOHERENT.

4758 If a Local Memory Region is created in non-coherent mode, then the Consumer must call
4759 *it_lmr_flush_to_mem* after modifying data in a memory range in this region that will be the
4760 target of an *incoming* RDMA Read or Atomic operation. *it_lmr_flush_to_mem* must be called
4761 after the Consumer has modified the memory range but before the RDMA Read or Atomic
4762 operation starts, and the memory range that will be accessed by the RDMA Read or Atomic
4763 operation must be supplied by the caller in the *local_segments* array. After this call returns, the RDMA
4764 Read or Atomic operation may safely see the modified contents of the memory range. It is
4765 permissible to batch synchronizations for multiple RDMA Read operations in a single call, by
4766 passing a *local_segments* array that includes all modified memory ranges. The *local_segments*
4767 entries need not contain the same LMR, and need not be in the same Protection Zone.

4768 If an RDMA Read or Atomic operation on an LMR created in non-coherent mode attempts to
4769 read from a memory range that is not properly synchronized using *it_lmr_flush_to_mem*, the
4770 returned contents are undefined.

4771 If the *local_segments* specified by the Consumer contain both normal LMRs (i.e., those created
4772 with IT_LMR_FLAG_NONCOHERENT cleared) and non-coherent LMRs (i.e., those created
4773 with IT_LMR_FLAG_NONCOHERENT set), only the non-coherent LMRs will be
4774 synchronized by this routine; the normal LMRs will be unaffected.

4775 RETURN VALUE

4776 This call is a no-op and always returns successfully if the Implementation does not support non-
4777 coherent mode, or if none of the LMRs in *local_segments* were created using the
4778 IT_LMR_FLAG_NONCOHERENT flag.

4779 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

4780 IT_ERR_RANGE The address range for a local segment fell outside the boundaries
4781 of the corresponding Local Memory Region and the Local
4782 Memory Region was created in non-coherent mode.

4783 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the disabled
4784 state. None of the output parameters from this routine are valid.
4785 See *it_ia_info_t* for a description of the disabled state.

4786 **APPLICATION USAGE**

4787 Determining when an RDMA Read or Atomic will start and what memory range it will read is
4788 the Consumer's responsibility. One possibility is to have the Consumer that is modifying
4789 memory call *it_lmr_flush_to_mem* and then post a Send DTO message that identifies the range
4790 in the body of the Send. The Consumer wishing to do the RDMA Read or Atomic can receive
4791 this message and thus know when it is safe to initiate the RDMA Read or Atomic operation.

4792 **SEE ALSO**

4793 *it_lmr_create()*, *it_lmr_refresh_from_mem()*, *it_lmr_triplet_t*

it_lmr_free()

4794

4795 NAME

4796 `it_lmr_free` – destroy a Local Memory Region

4797 SYNOPSIS

```
4798 #include <it_api.h>
4799
4800 it_status_t it_lmr_free(
4801     IN it_lmr_handle_t lmr_handle
4802 );
```

4803 DESCRIPTION

4804 *lmr_handle* Handle of Local Memory Region to be destroyed.

4805 The *it_lmr_free* routine destroys the Local Memory Region *lmr_handle*. On return, the Handle
4806 *lmr_handle* and associated RMR context (if any) may no longer be used. A Local Memory
4807 Region may not be destroyed if it has an RMR linked to it; an attempt to do so will fail and
4808 neither the LMR, its RMR Context (if any), nor any linked RMR(s) will be affected. *it_lmr_free*
4809 does not invalidate the memory range represented by *lmr_handle*, and the caller may continue to
4810 reference memory in this range for non-transport operations. If the memory range was locked in
4811 physical memory as a side-effect of the corresponding *it_lmr_create* call, then it will be
4812 unlocked immediately if no portion of the range overlaps with the range of other non-freed
4813 LMRs. Otherwise, the unlock operation may be deferred until the overlapping LMRs are
4814 themselves freed. Note that these may include LMRs created by other Consumers if the range is
4815 in shared memory.

4816 LMRs with memory ranges that overlap the range of *lmr_handle* are not affected by its
4817 destruction. Outstanding DTOs, Link, and Unlink operations that use an LMR that has been
4818 destroyed may or may not complete successfully.

4819 RETURN VALUE

4820 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

4821	IT_ERR_INVALID_LMR	The Local Memory Region Handle (<i>lmr_handle</i>) was
4822		invalid.
4823	IT_ERR_LMR_BUSY	The Local Memory Region was still referenced by a
4824		Remote Memory Region.
4825	IT_ERR_IA_CATASTROPHE	The IA has experienced a catastrophic error and is in the
4826		disabled state. None of the output parameters from this
4827		routine are valid. See <i>it_ia_info_t</i> for a description of the
4828		disabled state.

4829 SEE ALSO

4830 *it_lmr_create()*, *it_lmr_create_unlinked()*, *it_lmr_query()*, *it_lmr_modify()*

it_lmr_link()

4831

4832 NAME

4833 `it_lmr_link` – post operation to Link an unlinked Local Memory Region to an I/O Address range

4834 SYNOPSIS

```
4835 #include <it_api.h>
4836
4837 it_status_t it_lmr_link(
4838     IN    it_lmr_handle_t    lmr_handle,
4839     IN    const void         *addr,
4840     IN    const it_iobl_t    *iobl,
4841     IN    it_length_t        length,
4842     IN    it_addr_mode_t     addr_mode,
4843     IN    it_mem_priv_t      privs,
4844     IN    it_ep_handle_t     ep_handle,
4845     IN    it_dto_cookie_t    cookie,
4846     IN    it_dto_flags_t     dto_flags,
4847     OUT   it_rmr_context_t   *rmr_context
4848 );
```

4849 APPLICABILITY

4850 `it_lmr_link` is applicable only to Privileged Mode Endpoints created for the RC service type and
4851 is supported only if the Interface Adapter attribute `lmr_link_support` (see [it_ia_info_t](#)) is
4852 `IT_TRUE`.

4853 DESCRIPTION

4854	<code>lmr_handle</code>	Unlinked LMR to be linked.
4855	<code>addr</code>	Absolute address when <code>addr_mode</code> is <code>IT_ADDR_MODE_ABSOLUTE</code> .
4856		The <code>addr</code> argument is ignored if <code>addr_mode</code> is
4857		<code>IT_ADDR_MODE_RELATIVE</code> .
4858	<code>iobl</code>	I/O Buffer List specifying the I/O Address range of the LMR to be linked
4859		jointly with <code>length</code> .
4860	<code>length</code>	Length of I/O Address range in bytes. Must not be 0.
4861	<code>addr_mode</code>	Addressing mode of the LMR to be linked.
4862	<code>privs</code>	Bitwise OR of remote access privilege flags for the linked LMR, taken
4863		from it_mem_priv_t .
4864	<code>ep_handle</code>	Endpoint on which to post the Link operation.
4865	<code>cookie</code>	Consumer-provided cookie that is returned to the Consumer in the
4866		Completion Event corresponding to the LMR Link operation.
4867	<code>dto_flags</code>	Bitwise OR of options for operation handling.
4868	<code>rmr_context</code>	Returned Context allowing remote access to the linked LMR.

4869 The *it_lmr_link* routine posts to Endpoint *ep_handle* an operation to Link the currently unlinked
4870 Local Memory Region *lmr_handle* (see *it_lmr_create_unlinked* and *it_lmr_unlink*) to the
4871 memory specified by the *iobl* and *length* arguments (and possibly the *addr* argument).

4872 The Base Address of the LMR is given by *addr* for Absolute Addressing and by the I/O Address
4873 of the first byte of the region, identified by the *iobl* argument, for Relative Addressing. See also
4874 *addr_mode* below for addressing modes. The region extends for *length* bytes.

4875 The *iobl* argument (*it_iobl_t*) provides an I/O Buffer List and, jointly with the *length* argument,
4876 specifies the I/O Address range of the region to be linked. The first byte of this region is
4877 identified by the First-Byte Offset (*fbo*) member of the IOBL. The length of the IOBL is the
4878 number of bytes from the first byte of the region to the last byte of the IOBL. The *length*
4879 argument specifies the total size of the LMR in bytes, which is upper-bounded by the length of
4880 the IOBL.

4881 If the *addr_mode* argument (*it_addr_mode_t*) selects Absolute Addressing, then DTOs will
4882 access the LMR through absolute addresses in the given linear address space. If the *addr_mode*
4883 argument selects Relative Addressing, then DTOs will access the LMR through offsets relative
4884 to the Base Address of the LMR. Consumers can check the IA attribute
4885 *addr_mode_relative_support* to determine whether Relative Addressing is supported.

4886 If Absolute Addressing is used and the IOBL is a Page List (IOBL element length is guaranteed
4887 to be a power of two), the absolute address (*addr*) modulo the length (*elt_len*) of the first IOBL
4888 element must equal the First-Byte Offset (*fbo*) in the IOBL. If Relative Addressing is used, the
4889 LMR is fully specified by the IOBL and the *addr* parameter is ignored.

4890 An LMR can be linked only if the Endpoint *ep_handle* is in IT_EP_STATE_CONNECTED
4891 state and the Endpoint is Privileged.

4892 An LMR can be linked only if it is not currently in linked state.

4893 The LMR handle must be valid. The Protection Zones of the LMR and Endpoint must match.

4894 The type of access granted is specified by the *privs* argument as the bitwise-inclusive OR of one
4895 or more of the bit values IT_PRIV_LOCAL_READ, IT_PRIV_LOCAL_WRITE,
4896 IT_PRIV_REMOTE_READ, and IT_PRIV_REMOTE_WRITE. Unless otherwise noted below,
4897 all bit combinations are allowed. See *it_mem_priv_t* for bit definitions and predefined bit
4898 combinations. It is invalid to set *privs* to 0 (no access privileges), or to include
4899 IT_PRIV_REMOTE_WRITE without also including IT_PRIV_LOCAL_WRITE. Consumers
4900 are advised to use IT_PRIV_LOCAL_READ or IT_PRIV_LOCAL (which equals
4901 IT_PRIV_LOCAL_READ | IT_PRIV_LOCAL_WRITE) as local access privileges for
4902 maximum portability between transports (see also Extended Description). The access privileges
4903 required for a DTO's Source or Destination buffer are specified on the corresponding DTO
4904 reference page.

4905 The *cookie* (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work
4906 Request. This identifier is completely under Consumer control and opaque to the
4907 Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted
4908 Work Request.

4909 Request handling is specified by the *dto_flags* argument and is the bitwise OR of zero or more of
4910 the following flags:

4911 IT_COMPLETION_FLAG
4912 IT_NOTIFY_FLAG
4913 IT_BARRIER_FENCE_FLAG

4914 For the definition of these flags, see *it_dto_flags_t*. In addition, *it_lmr_link* automatically fences
4915 all DTO, Link, and Unlink operations subsequently submitted on the Endpoint *ep_handle* such
4916 that none of these operations starts until the currently posted Link operation has completed.

4917 An RMR Context allowing remote access to the memory region will be created if the *privs*
4918 argument includes either IT_PRIV_REMOTE_READ or IT_PRIV_REMOTE_WRITE. The
4919 Context will be returned in the location pointed to by *rmr_context* if the input value of
4920 *rmr_context* is not IT_NO_ADDR. The returned *rmr_context* is only valid if *privs* includes
4921 remote privileges and the call returns successfully; otherwise, it is undefined. The RMR Context
4922 may also be retrieved using *it_lmr_query*. The *rmr_context* is returned in network byte order,
4923 and may be passed by value to any remote process that wishes to use the Context in DTOs that
4924 target the corresponding LMR.

4925 The value of *rmr_context* is immediately available when *it_lmr_link* returns, but it may not be
4926 used by a remote host for an RDMA operation until the Link Completion Event occurs.
4927 Violation of this rule may result in an error and a broken Connection for the reliable Connection
4928 Endpoint on which the RDMA operation is posted. See Application Usage for more details.

4929 The completion of the posted Link operation is reported asynchronously to the Consumer
4930 according to the rules defined in *it_dto_flags_t*. An LMR Link Completion Event is of type
4931 *it_dto_cmpl_event_t*. Any LMR Link Completion Event generated manifests on the EVD
4932 associated with the Endpoint Send Queue. The Event type is IT_LMR_LINK_CMPL_EVENT.

4933 The linking remains valid until the Unlink operation completes successfully, or until the RMR is
4934 destroyed. A Link operation will never be partially successful over a subset of the requested
4935 memory range.

4936 If *it_lmr_link* returns successfully, but the Link Completion Event status indicates failure, then
4937 RMR Context is invalid.

4938 Any Work Request posted to an Endpoint's Send Queue after a call to *it_lmr_link* will not begin
4939 execution until the Link operation has completed.

4940 **EXTENDED DESCRIPTION**

4941 For certain Implementations, not all LMRs may be unlinked. This may be the case for [IB-R1.2],
4942 if an LMR is created via a Verb that does not pass an IOBL (PBL); i.e., the IB Register Memory
4943 Region verb. Consumers may get a completion error when trying to unlink LMRs created with
4944 *it_lmr_create*. An LMR created with *it_lmr_create_unlinked* and subsequently linked may
4945 always be unlinked. Any successfully unlinked LMR may be linked again.

4946 **RETURN VALUE**

4947 A successful call returns IT_SUCCESS. Otherwise, an immediate error is returned and, if the
4948 RMR was previously linked, the previous linking and RMR Context remains valid. It is possible
4949 for *it_lmr_link* to return success but for the Completion Event to indicate failure.

4950	Posting to an Endpoint that is not in the <code>IT_EP_STATE_CONNECTED</code> or	
4951	<code>IT_EP_STATE_NONOPERATIONAL</code> state will return the <code>IT_ERR_INVALID_EP_STATE</code>	
4952	immediate error.	
4953	The possible immediate errors for <code>it_lmr_link</code> are listed below:	
4954	<code>IT_ERR_INVALID_ADDR_MODE</code>	The addressing mode (<code>addr_mode</code>) was invalid or
4955		unsupported.
4956	<code>IT_ERR_INVALID_EP</code>	The Endpoint Handle (<code>ep_handle</code>) was invalid.
4957	<code>IT_ERR_INVALID_EP_STATE</code>	The Endpoint was not in the proper state for the attempted
4958		operation.
4959	<code>IT_ERR_INVALID_EP_TYPE</code>	The attempted operation was invalid for the Service Type of
4960		the Endpoint.
4961	<code>IT_ERR_INVALID_DTO_FLAGS</code>	The Data Transfer Operation flags (<code>dto_flags</code>) value was
4962		invalid.
4963	<code>IT_ERR_INVALID_IOBL</code>	The IOBL specified with <code>iobl</code> was invalid or of an
4964		unsupported type.
4965	<code>IT_ERR_INVALID_LMR</code>	The Local Memory Region Handle (<code>lmr_handle</code>) was
4966		invalid.
4967	<code>IT_ERR_INVALID_PRIVS</code>	The requested memory privileges (<code>privs</code>) contained an
4968		invalid flag.
4969	<code>IT_ERR_OP_NOT_SUPPORTED</code>	The IA has no LMR Link support (<code>lmr_link_support</code> in
4970		<code>it_ia_info_t</code> is <code>IT_FALSE</code>).
4971	<code>IT_ERR_RESOURCES</code>	The requested operation failed due to insufficient resources.
4972	<code>IT_ERR_TOO_MANY_POSTS</code>	The operation failed due to an overflow of a Work Queue.
4973	<code>IT_ERR_IA_CATASTROPHE</code>	The IA has experienced a catastrophic error and is in the
4974		disabled state. None of the output parameters from this
4975		routine are valid. See <code>it_ia_info_t</code> for a description of the
4976		disabled state.

4977 **ASYNCHRONOUS ERRORS**

4978 For Work Requests posted to an Endpoint in the `IT_EP_STATE_CONNECTED` state, a
4979 completion status (see `it_dto_status_t`) other than `IT_DTO_SUCCESS` will break the Connection
4980 by moving the Endpoint to the `IT_EP_STATE_NONOPERATIONAL` state and deliver an
4981 `IT_CM_MSG_CONN_BROKEN_EVENT` Event to the Connect EVD of `ep_handle`. Once the
4982 Connection is broken, all outstanding and in-progress operations on the Connection will
4983 complete with an error status.

4984 Any posting to an Endpoint that is in the `IT_EP_STATE_NONOPERATIONAL` state will be
4985 flushed with completion status set to `IT_DTO_ERR_FLUSHED`.

4986 An `IT_DTO_ERR_LOCAL_MM_OPERATION` completion status results in case of an invalid
4987 LMR handle, unless the Implementation handles this error as an immediate error.

4988 An IT_DTO_ERR_LOCAL_MM_OPERATION completion status results if the Protection
4989 Zones of the LMR and the Endpoint do not match or if *privs* represents an invalid combination
4990 of access privileges.

4991 If *lmr_handle* represents a linked LMR or if *lmr_handle* is the Direct LMR Handle, a
4992 Completion Error is generated with completion status set to
4993 IT_DTO_ERR_LOCAL_MM_OPERATION.

4994 An IT_DTO_ERR_LOCAL_MM_OPERATION completion status results if the number of
4995 IOBL elements exceeds the *iobl_num_elts* LMR attribute, if the IOBL includes elements of an
4996 unsupported length, or if the IOBL has an invalid First-Byte Offset (*fbo*).

4997 An IT_DTO_ERR_LOCAL_MM_OPERATION completion status also results for IOBLs that
4998 are Page Lists, if an IOBL element has an invalid page size or is not page aligned or, for
4999 Absolute Addressing only, if the absolute address (*addr*) modulo the length (*elt_len*) of the first
5000 IOBL element does not equal the First-Byte Offset (*fbo*) in the IOBL.

5001 If the Endpoint to which the *it_lmr_link* Work Request is posted is not Privileged, then a
5002 Completion Error with the DTO status IT_DTO_ERR_LOCAL_MM_OPERATION shall be
5003 returned.

5004 For the iWARP Transport only, if the *length* of the LMR exceeds the length of the IOBL or if
5005 *length* equals zero, a Completion Error is generated with completion status set to
5006 IT_DTO_ERR_MM_OPERATION_ERROR.

5007 APPLICATION USAGE

5008 The *it_lmr_link* operation is lightweight compared to creating an LMR. An application
5009 concerned with efficiency would typically create one or more unlinked LMRs at initialization
5010 time; later, it would link (unlink) LMRs as needed to enable (disable) local or remote access to
5011 regions of memory, possibly multiple times.

5012 The Consumer should use unique identifiers for *cookie* if they desire to identify each DTO. If the
5013 Consumer does not require a unique DTO identifier, the value of zero or NULL
5014 (IT_NO_ADDR) can be used.

5015 The local Consumer has several options for ensuring that the remote Consumer does not use
5016 *rmr_context* before the Link Completion Event occurs. One is to wait for the Completion Event
5017 on the Send EVD of the specified Endpoint *ep_handle* before sending the *rmr_context* to a peer.
5018 Another option is to send the *rmr_context* to a peer by posting a DTO to the same Endpoint
5019 *ep_handle* that was used to Link the LMR. The barrier-fencing behavior of *it_lmr_link* ensures
5020 that the DTO does not start until the Link Completion Event has occurred. If the Link fails with
5021 a Completion Error, the Connection will be broken and the DTO flushed, so the *rmr_context* will
5022 not be sent.

5023 For reasons already described, the completion of an LMR Link operation represents an important
5024 change that Consumers may need to monitor. One way to do this is to set
5025 IT_COMPLETION_FLAG in *dto_flags*, which will generate a Completion Event to indicate
5026 when the LMR has been linked. Consumers who do not set IT_COMPLETION_FLAG must rely
5027 on ordering semantics to infer when the LMR has been successfully linked. For example, if a
5028 subsequent DTO posted to the Send Queue of the same EP completes successfully, then the Link
5029 operation has completed, because DTOs posted to the Send Queue of an EP must wait for a Link
5030 operation to complete before processing. If the Link operation fails, a Link Completion Event is
5031 generated regardless of the use of IT_COMPLETION_FLAG.

5032 When an EVD holding the DTO Event Stream associated with an Endpoint overflows, on some
5033 transports the behavior of the Endpoint and/or the EVD is undefined, making recovery from
5034 such an overflow either impossible or at the very least IA-dependent. Consumers should
5035 therefore take care to avoid this situation. Each Work Request that a Consumer posts to a Work
5036 Queue can result in a Completion Event being enqueued in an EVD, due to the possibility of an
5037 error occurring during Work Request processing. To avoid any possibility of overflow, the
5038 Consumer must therefore ensure that the total number of Outstanding Operations on all Work
5039 Queues that can enqueue Completion Events into a given EVD never exceeds the minimum
5040 queue size (as reported by *it_evd_query*) of that EVD.

5041 **FUTURE DIRECTIONS**

5042 None.

5043 **SEE ALSO**

5044 *it_lmr_create()*, *it_lmr_create_unlinked()*, *it_lmr_unlink()*, *it_dto_flags_t*, *it_dto_events*,
5045 *it_addr_mode_t*, *it_iobl_t*, *it_mem_priv_t*, *it_ia_info_t*

it_lmr_modify()

5046

5047 NAME

5048 `it_lmr_modify` – modify selected attributes of a Local Memory Region

5049 SYNOPSIS

```
5050 #include <it_api.h>
5051
5052 it_status_t it_lmr_modify(
5053     IN          it_lmr_handle_t      lmr_handle,
5054     IN          it_lmr_param_mask_t mask,
5055     IN  const   it_lmr_param_t      *params
5056 );
```

5057 DESCRIPTION

5058 *lmr_handle* Local Memory Region.

5059 *mask* Bitwise OR of flags for specified parameters.

5060 *params* Structure whose members contain the new parameter values.

5061 The *it_lmr_modify* routine changes selected attributes of the Local Memory Region *lmr_handle*.
5062 Attributes to be modified are specified by flags in *mask*. New values for the attributes are
5063 specified by the corresponding fields in the structure pointed to by *params*. Fields and their
5064 corresponding flag values are shown in *it_lmr_param_t*. Note that attributes represented by
5065 fields of *it_lmr_param_t* that are not shown below cannot be modified. The definition of each
5066 field follows:

5067 *pz* The new Protection Zone Handle for the LMR.

5068 *privs* The new memory access privileges for the LMR. See *it_mem_priv_t*,
5069 *it_lmr_create*, and *it_lmr_link* for flag definitions and restrictions.

5070 If remote access privileges are specified in *privs* and the routine returns successfully, then a new
5071 RMR Context has been created and associated with the LMR. The LMR's new associated RMR
5072 Context may be retrieved using *it_lmr_query*. The Implementation of *it_lmr_modify* strives to
5073 generate a new RMR Context that is different from any RMR Contexts generated previously. As
5074 long as the new RMR Context does not match a previously generated RMR Context, remote
5075 access using a previously generated RMR Context is revoked. If no remote access privileges are
5076 specified in *privs* and the routine returns successfully, then no RMR Context remains associated
5077 with the LMR, and remote access to the LMR is disabled. If the *lmr_handle* is invalidated due to
5078 an error, any associated RMR Context is invalidated, and any IA access to the LMR is disabled.

5079 A Local Memory Region may not be modified if it is still referenced by bound Remote Memory
5080 Regions; an attempt to do so will fail with an error return, and the LMR will not be modified or
5081 affected.

5082 The Consumer should not modify an LMR whose LMR Handle or RMR Context is used in
5083 outstanding DTOs, LMR Link, LMR Unlink, RMR Link, or RMR Unlink operations. The
5084 Consumer must dequeue the Completion Events for all such operations prior to modifying the
5085 LMR. If this rule is not followed, the Outstanding Operations may fail and complete with an
5086 error status.

5087 If the *lmr_handle* or *mask* parameter is invalid, or the LMR has any RMRs linked to it, then the
5088 operation fails and, if *lmr_handle* was valid, the LMR is not modified or affected and any RMRs
5089 linked to the LMR are also unaffected.

5090 For all other errors, the *lmr_handle* and any associated RMR Context are invalidated and may no
5091 longer be used. Any resources that were associated with the LMR are freed, as if the Consumer
5092 was calling *it_lmr_free* on the *lmr_handle*.

5093 **RETURN VALUE**

5094 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

5095	IT_ERR_ACCESS	The Consumer was not allowed to have the requested memory
5096		privileges.
5097	IT_ERR_INVALID_LMR	The Local Memory Region Handle (<i>lmr_handle</i>) was invalid.
5098	IT_ERR_INVALID_MASK	The <i>mask</i> contained invalid flag values.
5099	IT_ERR_INVALID_PRIVS	The requested memory privileges (<i>privs</i>) contained an invalid flag.
5100	IT_ERR_INVALID_PZ	The Protection Zone Handle (<i>pz_handle</i>) was invalid.
5101	IT_ERR_LMR_BUSY	The Local Memory Region was still referenced by a Remote
5102		Memory Region.
5103	IT_ERR_RESOURCES	The requested operation failed due to insufficient resources.
5104	IT_ERR_IA_CATASTROPHE	The IA has experienced a catastrophic error and is in the disabled
5105		state. None of the output parameters from this routine are valid.
5106		See <i>it_ia_info_t</i> for a description of the disabled state.

5107 **APPLICATION USAGE**

5108 Although *it_lmr_modify* can be used to change the remote access privileges for an LMR, this is a
5109 much more expensive operation than linking an RMR to an LMR using *it_rmr_link*.

5110 **SEE ALSO**

5111 *it_lmr_create()*, *it_lmr_free()*, *it_lmr_link()*, *it_lmr_query()*, *it_lmr_unlink()*, *it_rmr_link()*,
5112 *it_rmr_unlink()*, *it_addr_mode_t*, *it_lmr_param_t*, *it_lmr_param_mask_t*, *it_mem_priv_t*

it_lmr_query()

5113

5114 **NAME**

5115 `it_lmr_query` – get attributes of a Local Memory Region

5116 **SYNOPSIS**

```
5117 #include <it_api.h>
5118
5119 it_status_t it_lmr_query(
5120     IN    it_lmr_handle_t    lmr_handle,
5121     IN    it_lmr_param_mask_t mask,
5122     OUT   it_lmr_param_t     *params
5123 );
5124
5125 typedef enum {
5126     IT_LMR_PARAM_ALL           = 0x000001,
5127     IT_LMR_PARAM_IA           = 0x000002,
5128     IT_LMR_PARAM_PZ          = 0x000004,
5129     IT_LMR_PARAM_ADDR         = 0x000008,
5130     IT_LMR_PARAM_LENGTH       = 0x000010,
5131     IT_LMR_PARAM_MEM_PRIV     = 0x000020,
5132     IT_LMR_PARAM_FLAG         = 0x000040,
5133     IT_LMR_PARAM_SHARED_ID    = 0x000080,
5134     IT_LMR_PARAM_RMR_CONTEXT  = 0x000100,
5135     IT_LMR_PARAM_ACTUAL_ADDR  = 0x000200,
5136     IT_LMR_PARAM_ACTUAL_LENGTH = 0x000400,
5137     IT_LMR_PARAM_ADDR_MODE    = 0x000800,
5138     IT_LMR_PARAM_LINKED       = 0x001000,
5139     IT_LMR_PARAM_IOBL_NUM_ELTS = 0x002000,
5140     IT_LMR_PARAM_IOBL_TYPE    = 0x004000
5141 } it_lmr_param_mask_t;
5142
5143 typedef struct {
5144     it_ia_handle_t    ia;           /* IT_LMR_PARAM_IA */
5145     it_pz_handle_t    pz;           /* IT_LMR_PARAM_PZ */
5146     void               *addr;       /* IT_LMR_PARAM_ADDR */
5147     it_length_t        length;      /* IT_LMR_PARAM_LENGTH */
5148     it_mem_priv_t      privs;       /* IT_LMR_PARAM_MEM_PRIV */
5149     it_lmr_flag_t      flags;       /* IT_LMR_PARAM_FLAG */
5150     uint32_t           shared_id;    /* IT_LMR_PARAM_SHARED_ID */
5151     it_rmr_context_t   rmr_context; /* IT_LMR_PARAM_RMR_CONTEXT */
5152     void               *actual_addr; /* IT_LMR_PARAM_ACTUAL_ADDR */
5153     it_length_t        actual_length; /* IT_LMR_PARAM_ACTUAL_LENGTH */
5154     it_addr_mode_t     addr_mode;    /* IT_LMR_PARAM_ADDR_MODE */
5155     it_boolean_t       linked;       /* IT_LMR_PARAM_LINKED */
5156     size_t             iobl_num_elts; /* IT_LMR_PARAM_IOBL_NUM_ELTS */
5157     it_iobl_type_t     iobl_type;    /* IT_LMR_PARAM_IOBL_TYPE */
5158 } it_lmr_param_t;
```

5159 **DESCRIPTION**

5160 `lmr_handle` Local Memory Region.

5161	<i>mask</i>	Bitwise OR of flags for desired parameters.
5162	<i>params</i>	Structure whose members are written with the desired parameters.
5163		The <i>it_lmr_query</i> routine returns the desired attributes of the Local Memory Region <i>lmr_handle</i>
5164		in the structure pointed to by <i>params</i> . On return, each field of <i>params</i> is only valid if the
5165		corresponding flag as shown in the Synopsis is set in the <i>mask</i> argument. The <i>mask</i> value
5166		IT_LMR_PARAM_ALL causes all fields to be returned.
5167		The definition of each field of <i>params</i> follows:
5168	<i>ia</i>	The Interface Adapter Handle specified to create the LMR.
5169	<i>pz</i>	The Protection Zone Handle specified to create the LMR.
5170	<i>addr</i>	The requested starting address or, equivalently, Base Address of the LMR.
5171		To be interpreted as an absolute address, regardless of <i>addr_mode</i> .
5172	<i>length</i>	The requested length in bytes of the LMR.
5173	<i>privs</i>	The memory access privileges of the LMR. For the definition of bits, see
5174		it_mem_priv_t .
5175	<i>flags</i>	Attributes of the LMR according to <i>it_lmr_flags_t</i> . The attribute
5176		corresponding to IT_LMR_FLAG_SHARED will be set if the LMR has
5177		been matched with another LMR (i.e., shared) or clear if the LMR is not.
5178	<i>shared_id</i>	The <i>shared_id</i> specified to create the LMR.
5179	<i>rmr_context</i>	The RMR Context associated with the LMR, or undefined if <i>privs</i> does not
5180		include remote access. Returned in network byte order.
5181	<i>actual_addr</i>	The actual starting address for which bounds checking is done for data
5182		transfers. To be interpreted as an absolute address, regardless of
5183		<i>addr_mode</i> .
5184	<i>actual_length</i>	The actual length for which bounds checking is done for data transfers.
5185	<i>addr_mode</i>	The addressing mode of the LMR.
5186	<i>linked</i>	Flag identifying whether or not the LMR is currently linked.
5187	<i>iobl_num_elts</i>	The number of IOBL entries allocated by this LMR. Valid if LMR was
5188		created using it_lmr_create_unlinked (validity is Implementation-
5189		dependent otherwise).
5190	<i>iobl_type</i>	The type of IOBL linked to this LMR. Valid if LMR was created using
5191		it_lmr_create_unlinked (validity is Implementation-dependent otherwise).

5192 EXTENDED DESCRIPTION

5193 Implementations based on iWARP and Implementations on any transport supporting Relative
5194 Addressing will support LMR creation and bounds checking with byte-level granularity.

5195 Implementations based on InfiniBand 1.2 can be expected to support LMR creation and bounds
5196 checking with byte-level granularity.

5197 For Implementations with byte-level granularity for LMRs, the actual starting address and actual
5198 length will equal the requested starting address and requested length, respectively. For LMRs
5199 with Relative Addressing, LMR byte-level granularity ensures that the starting address of the
5200 LMR is unambiguous.

5201 **RETURN VALUE**

5202 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

5203 IT_ERR_INVALID_LMR The Local Memory Region Handle (*lmr_handle*) was invalid.

5204 IT_ERR_INVALID_MASK The *mask* contained invalid flag values.

5205 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the disabled
5206 state. None of the output parameters from this routine are valid.
5207 See *it_ia_info_t* for a description of the disabled state.

5208 **SEE ALSO**

5209 *it_lmr_create()*, *it_lmr_create_unlinked()*, *it_lmr_free()*, *it_lmr_link()*, *it_lmr_modify()*,
5210 *it_lmr_unlink()*, *it_addr_mode_t*, *it_mem_priv_t*

5211

it_lmr_refresh_from_mem()

5212 NAME

5213 it_lmr_refresh_from_mem – make effects of an incoming RDMA Write or Atomic operation
5214 visible to Consumer

5215 SYNOPSIS

```
5216 #include <it_api.h>
5217
5218 it_status_t it_lmr_refresh_from_mem(
5219     IN const it_lmr_triplet_t *local_segments,
5220     IN size_t num_segments
5221 );
```

5222 DESCRIPTION

5223 *local_segments* Array of buffer segments.

5224 *num_segments* Number of segments in the array.

5225 The *it_lmr_refresh_from_mem* routine is needed if and only if an LMR was created in non-
5226 coherent mode using IT_LMR_FLAG_NONCOHERENT.

5227 If a Local Memory Region is created in non-coherent mode, then the Consumer must call
5228 *it_lmr_refresh_from_mem* before reading data from a memory range in this region that was the
5229 target of an *incoming* RDMA Write operation. *it_lmr_refresh_from_mem* must be called after
5230 the RDMA Write operation completes, and the memory range that was modified by the RDMA
5231 Write must be supplied by the caller in the *local_segments* array. After this call returns, the
5232 Consumer may safely see the modified contents of the memory range. It is permissible to batch
5233 synchronizations of multiple RDMA Write operations in a single call, by passing a
5234 *local_segments* array that includes all modified memory ranges. The *local_segments* entries need
5235 not contain the same LMR, and need not be in the same Protection Zone.

5236 The Consumer must also use *it_lmr_refresh_from_mem* when performing local writes to a
5237 memory range that was or will be the target of incoming RDMA Writes. After performing the
5238 local write, the Consumer must call *it_lmr_refresh_from_mem* before the RDMA Write is
5239 initiated. Conversely, after an RDMA Write completes, the Consumer must call
5240 *it_lmr_refresh_from_mem* before performing a local write to the same range.

5241 If the Consumer attempts to read from a memory range in an LMR that was created in non-
5242 coherent mode, without properly synchronizing using *it_lmr_refresh_from_mem*, the returned
5243 contents are undefined. If the Consumer attempts to write to a memory range without properly
5244 synchronizing, the contents of the memory range become undefined.

5245 If the *local_segments* specified by the Consumer contain both normal LMRs (i.e., those created
5246 with IT_LMR_FLAG_NONCOHERENT cleared) and non-coherent LMRs (i.e., those created
5247 with IT_LMR_FLAG_NONCOHERENT set), only the non-coherent LMRs will be
5248 synchronized by this routine; the normal LMRs will be unaffected.

5249 RETURN VALUE

5250 This call is a no-op and always returns successfully if the Implementation does not support non-
5251 coherent mode, or if none of the LMRs in *local_segments* were created using the
5252 IT_LMR_FLAG_NONCOHERENT flag.

5253 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

5254 IT_ERR_RANGE The address range for a local segment fell outside the boundaries
5255 of the corresponding Local Memory Region and the Local
5256 Memory Region was created in non-coherent mode.

5257 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the disabled
5258 state. None of the output parameters from this routine are valid.
5259 See *it_ia_info_t* for a description of the disabled state.

5260 **APPLICATION USAGE**

5261 Determining when an RDMA Write completes and determining which memory range was
5262 modified is the Consumer's responsibility. One possibility is for the RDMA Write initiator to
5263 post a Send DTO message after each RDMA Write that identifies the range in the body of the
5264 Send. The Consumer at the target of the RDMA Write can receive the message and thus know
5265 when and how to call *it_lmr_refresh_from_mem*.

5266 **SEE ALSO**

5267 *it_lmr_create()*, *it_lmr_flush_to_mem()*, *it_lmr_triplet_t*

it_lmr_unlink()

5268

5269 NAME

5270 it_lmr_unlink – post operation to Unlink a Local Memory Region from its memory range

5271 SYNOPSIS

```
5272 #include <it_api.h>
5273
5274 it_status_t it_lmr_unlink(
5275     IN it_lmr_handle_t lmr_handle,
5276     IN it_ep_handle_t ep_handle,
5277     IN it_dto_cookie_t cookie,
5278     IN it_dto_flags_t dto_flags
5279 );
```

5280 APPLICABILITY

5281 *it_lmr_unlink* is applicable only to Endpoints created for the RC service type and is supported
5282 only if the Interface Adapter attribute *lmr_link_support* (see *it_ia_info_t*) is IT_TRUE.

5283 DESCRIPTION

5284 *lmr_handle* Handle of LMR that will be unlinked.

5285 *ep_handle* Endpoint on which to post the operation.

5286 *cookie* Consumer-provided cookie that is returned to the Consumer in the
5287 Completion Event corresponding to the operation.

5288 *dto_flags* Bitwise OR of options for operation handling.

5289 The *it_lmr_unlink* routine posts to Endpoint *ep_handle* an Unlink operation to unlink the Local
5290 Memory Region specified by *lmr_handle*. The *it_lmr_unlink* routine does not require Privilege
5291 (see Application Usage).

5292 An LMR can be unlinked only if the Endpoint *ep_handle* is in IT_EP_STATE_CONNECTED
5293 state.

5294 The Protection Zones of the LMR and Endpoint must match.

5295 The *cookie* (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work
5296 Request. This identifier is completely under Consumer control and opaque to the
5297 Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted
5298 Work Request.

5299 Request handling is specified by the *dto_flags* argument as the bitwise OR of zero or more of the
5300 following flags:

```
5301 IT_COMPLETION_FLAG
5302 IT_NOTIFY_FLAG
5303 IT_BARRIER_FENCE_FLAG
5304 IT_UNLINK_FENCE_FLAG
```

5305 For the definition of these flags, see *it_dto_flags_t*.

5306 The completion of the posted Unlink operation is reported asynchronously to the Consumer
5307 according to the rules defined in *it_dto_flags_t*. An LMR Unlink Completion Event is of type
5308 *it_dto_cmpl_event_t*. Any generated LMR Unlink Completion Event manifests on the EVD
5309 associated with the Endpoint Send Queue. The Event type is IT_LMR_LINK_CMPL_EVENT.

5310 After a successful Unlink Completion Event, the previous linking for the LMR is invalidated,
5311 and any RMR Context for the LMR is no longer defined. Any RDMA operation that uses a
5312 previous RMR Context will fail with a protection violation; beware that this may include
5313 operations that are outstanding when *it_lmr_unlink* is called. The Consumer must ensure that
5314 such operations have completed prior to calling *it_lmr_unlink* if successful completions are
5315 desired. An Unlink operation will never be partially successful over a subset of the requested
5316 memory range; it either succeeds completely or fails without invalidating any portion of the
5317 previous linking.

5318 If *it_lmr_unlink* returns successfully but the Completion Event status indicates failure, then the
5319 previous linking and RMR Context remains valid.

5320 Any Work Request posted to an Endpoint's Send Queue after a call to *it_lmr_unlink* may begin
5321 execution before the Unlink operation has completed. To ensure deterministic behavior, the
5322 IT_UNLINK_FENCE_FLAG may be specified on subsequent Work Requests.

5323 **EXTENDED DESCRIPTION**

5324 The InfiniBand v1.2 Transport supports LMRs that may be linked and unlinked but also supports
5325 a separate class of LMRs that cannot be linked or unlinked. To guarantee that an LMR may be
5326 linked and unlinked, the Consumer should use the *it_lmr_create_unlinked* routine for LMR
5327 creation.

5328 The InfiniBand v1.1 Transport does not support LMRs that may be linked and unlinked and
5329 reports *lmr_link_support* at IT_FALSE.

5330 **RETURN VALUE**

5331 A successful call returns IT_SUCCESS. Otherwise, an immediate error is returned and the
5332 previous linking of the LMR remains valid. It is possible for *it_lmr_unlink* to return success but
5333 for the Completion Event to indicate failure.

5334 Posting to an Endpoint that is not in the IT_EP_STATE_CONNECTED or
5335 IT_EP_STATE_NONOPERATIONAL state will return the IT_ERR_INVALID_EP_STATE
5336 immediate error.

5337 The possible immediate errors for *it_lmr_unlink* are listed below:

5338 IT_ERR_INVALID_DTO_FLAGS The Data Transfer Operation flags (*dto_flags*) value was
5339 invalid.

5340 IT_ERR_INVALID_EP The Endpoint Handle (*ep_handle*) was invalid.

5341 IT_ERR_INVALID_EP_STATE The Endpoint was not in the proper state for the attempted
5342 operation.

5343 IT_ERR_INVALID_EP_TYPE The attempted operation was invalid for the Service Type of
5344 the Endpoint.

5345 IT_ERR_INVALID_LMR The Region Handle (*lmr_handle*) was invalid.

5346	IT_ERR_OP_NOT_SUPPORTED	The IA has no LMR Link support (<i>lmr_link_support</i> in <i>it_ia_info_t</i> is IT_FALSE).
5347		
5348	IT_ERR_TOO_MANY_POSTS	The operation failed due to an overflow of a Work Queue.
5349	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.
5350		
5351		
5352		

5353 **ASYNCHRONOUS ERRORS**

5354 For Work Requests posted to an Endpoint in the IT_EP_STATE_CONNECTED state, a
5355 completion status (see *it_dto_status_t*) other than IT_DTO_SUCCESS will break the Connection
5356 by moving the Endpoint to the IT_EP_STATE_NONOPERATIONAL state and deliver an
5357 IT_CM_MSG_CONN_BROKEN_EVENT Event to the Connect EVD of *ep_handle*. Once the
5358 Connection is broken, all outstanding and in-progress operations on the Connection will
5359 complete with an error status.

5360 Any posting to an Endpoint that is in the IT_EP_STATE_NONOPERATIONAL state will be
5361 flushed with completion status set to IT_DTO_ERR_FLUSHED.

5362 An IT_DTO_ERR_LOCAL_MM_OPERATION completion status results in case of an invalid
5363 LMR handle, unless the Implementation handles this error as an immediate error.

5364 An IT_DTO_ERR_LOCAL_MM_OPERATION completion status results if the Protection
5365 Zones of the LMR and the Endpoint do not match.

5366 An IT_DTO_ERR_LOCAL_MM_OPERATION completion status results if *lmr_handle* is the
5367 Direct LMR Handle, or if the LMR is in the Shared state, or if the LMR has any RMR bound to
5368 it, or, for InfiniBand only, if the LMR could not be unlinked due to the way it was created.

5369 **APPLICATION USAGE**

5370 *it_lmr_unlink* is used to move an LMR from the linked state to the unlinked state, thereby
5371 revoking local Consumer access and remote Consumer access if that was previously granted.

5372 The *it_lmr_unlink* routine does not require Privilege to invoke but the *it_lmr_link* routine does.
5373 Invoking *it_lmr_unlink* as a non-Privileged Consumer has the consequence that the unlinked
5374 LMR may not be re-linked (as the Consumer has no such Privilege) and may only be freed.

5375 When an EVD holding the DTO Event Stream associated with an Endpoint overflows, on some
5376 transports the behavior of the Endpoint and/or the EVD is undefined, making recovery from
5377 such an overflow either impossible or at the very least IA-dependent. Consumers should
5378 therefore take care to avoid this situation. Each Work Request that a Consumer posts to a Work
5379 Queue can result in a Completion Event being enqueued in an EVD, due to the possibility of an
5380 error occurring during Work Request processing. To avoid any possibility of overflow, the
5381 Consumer must therefore ensure that the total number of Outstanding Operations on all Work
5382 Queues that can enqueue Completion Events into a given EVD never exceeds the minimum
5383 queue size (as reported by *it_evd_query*) of that EVD.

5384 **SEE ALSO**

5385 *it_lmr_create()*, *it_lmr_create_unlinked()*, *it_lmr_link()*, *it_dto_flags_t*

5386

it_make_rdma_addr_absolute()

5387 **NAME**

5388 `it_make_rdma_addr_absolute` – make a platform-independent RDMA address for Absolute
5389 Addressing

5390 **SYNOPSIS**

```
5391 #include <it_api.h>  
5392  
5393 typedef uint64_t it_rdma_addr_t;  
5394  
5395 it_rdma_addr_t it_make_rdma_addr_absolute(  
5396     void *addr  
5397 );
```

5398 **DESCRIPTION**

5399 `addr` Local address.

5400 The `it_make_rdma_addr_absolute` routine takes a local address `addr` that may be the target of a
5401 remote operation, and returns a 64-bit platform-independent representation of that address in
5402 network byte order and suitable for Absolute Addressings, called an RDMA address (see
5403 [it_addr_mode_t](#) for a description of Absolute Addressing). A network peer may use this RDMA
5404 address in RDMA Read and Write operations. `it_make_rdma_addr_absolute` performs no
5405 validity checking on `addr`, so `addr` is not required to lie within a currently registered LMR when
5406 `it_make_rdma_addr_absolute` is called.

5407 **RETURN VALUE**

5408 This function always succeeds and returns a 64-bit RDMA address.

5409 **APPLICATION USAGE**

5410 The returned RDMA address must be communicated to a network peer in order to be used in
5411 RDMA operations. The Consumer is responsible for performing this communication.

5412 Because the RDMA address is in network byte order, a Consumer wishing to perform address
5413 arithmetic must first convert it to host byte order, which may be done using the [it_ntoh64](#)
5414 function. Derived addresses must be converted back to network byte order using [it_hton64](#)
5415 before being used in RDMA operations.

5416 **SEE ALSO**

5417 [it_post_rdma_write\(\)](#), [it_ntoh64](#), [it_hton64](#), [it_make_rdma_addr_relative](#), [it_addr_mode_t](#)

it_mem_map()

5448

5449 NAME

5450 `it_mem_map` – map virtually addressed buffer to I/O address space

5451 SYNOPSIS

```
5452 #include <it_api.h>
5453
5454 it_status_t it_mem_map(
5455     IN  it_ia_handle_t  ia_handle,
5456     IN  void            *vaddr,
5457     IN  uint64_t        aspace_id,
5458     IN  size_t          len,
5459     OUT it_io_addr_t    *io_addr,
5460     OUT size_t          *mapped_len
5461 );
```

5462 APPLICABILITY

5463 `it_mem_map` is usable only by Privileged Mode Consumers.

5464 DESCRIPTION

5465	<i>ia_handle</i>	The Interface Adapter on which <i>io_addr</i> will be used.
5466	<i>vaddr</i>	Local address.
5467	<i>aspace_id</i>	Local address space identifier. An O/S-dependent and architecture-specific value that combines with <i>vaddr</i> to yield a virtual address valid in any O/S context.
5470	<i>len</i>	Length of buffer addressed by <i>vaddr</i> .
5471	<i>io_addr</i>	I/O address of buffer corresponding to the first <i>mapped_len</i> bytes of <i>vaddr</i> . The buffer returned in <i>io_addr</i> is contiguous for <i>mapped_len</i> bytes in the I/O address space of the IA.
5474	<i>mapped_len</i>	Number of bytes of <i>vaddr</i> mapped into <i>io_addr</i> .

5475 The `it_mem_map` routine maps a local virtually addressed buffer *vaddr* to a buffer that is
5476 contiguous in the I/O address space usable by the Interface Adapter identified by *ia_handle*. If
5477 `it_mem_map` is unable to map all of *vaddr* in one contiguous I/O buffer it returns the number of
5478 bytes actually mapped in *mapped_len*.

5479 RETURN VALUE

5480 A successful call returns `IT_SUCCESS`. Otherwise, an immediate error is returned.

5481	<code>IT_ERR_NO_PERMISSION</code>	The Consumer did not have the proper permissions to perform the requested operation.
5482		
5483	<code>IT_ERR_FAULT</code>	Part or all of the supplied address range was invalid.
5484	<code>IT_ERR_INVALID_IA</code>	The Interface Adapter Handle (<i>ia_handle</i>) was invalid.

5485 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the
5486 disabled state. None of the output parameters from this
5487 routine are valid. See *it_ia_info_t* for a description of the
5488 disabled state.

5489 **APPLICATION USAGE**

5490 The Consumer may use *it_mem_map* to construct an IOBL for input to *it_lmr_link* and other
5491 API routines.

5492 When *it_mem_map* returns *mapped_len* as less than *len*, the Consumer should store *io_addr* and
5493 *mapped_len* into an IOBL entry, increment *vaddr* by *mapped_len* bytes, subtract *mapped_len*
5494 bytes from *len*, and call *it_mem_map* again with the revised arguments and store the result into a
5495 subsequent IOBL entry. The Consumer should follow this practice iteratively until the entire
5496 buffer is mapped.

5497 **SEE ALSO**

5498 *it_mem_unmap()*, *it_lmr_link()*, *it_lmr_create()*, *it_io_addr_t*, *it_iobl_t*

it_mem_unmap()

5499

5500 NAME

5501 `it_mem_unmap` – undo a mapping established by `it_mem_map`

5502 SYNOPSIS

```
5503 #include <it_api.h>
5504
5505 it_status_t it_mem_unmap(
5506     IN it_ia_handle_t ia_handle,
5507     IN it_io_addr_t io_addr,
5508     IN size_t mapped_len
5509 );
```

5510 APPLICABILITY

5511 `it_mem_unmap` is usable only by Privileged Mode Consumers.

5512 DESCRIPTION

5513 `ia_handle` The Interface Adapter on which `io_addr` will be used.

5514 `io_addr` I/O address of buffer returned earlier by `it_mem_map`.

5515 `mapped_len` Size of buffer starting at `io_addr` in bytes.

5516 The `it_mem_unmap` routine tears down a mapping established by `it_mem_map` on the Interface
5517 Adapter implicitly identified by `pz_handle`.

5518 Calling `it_mem_unmap` on an address range still in use by the Consumer can result in undefined
5519 behavior up to and including unreported data corruption.

5520 Passing `io_addr` or `mapped_len` values that do not correspond to values returned by `it_mem_map`
5521 will yield an immediate error.

5522 RETURN VALUE

5523 A successful call returns `IT_SUCCESS`. Otherwise, an immediate error is returned.

5524 `IT_ERR_NO_PERMISSION` The Consumer did not have the proper permissions to
5525 perform the requested operation.

5526 `IT_ERR_FAULT` Part or all of the supplied address range was invalid.

5527 `IT_ERR_INVALID_IA` The Interface Adapter Handle (`ia_handle`) was invalid.

5528 `IT_ERR_IA_CATASTROPHE` The IA has experienced a catastrophic error and is in the
5529 disabled state. None of the output parameters from this
5530 routine are valid. See `it_ia_info_t` for a description of the
5531 disabled state.

5532 SEE ALSO

5533 `it_mem_map()`, `it_lmr_link()`, `it_lmr_create()`, `it_io_addr_t`, `it_iobl_t`

it_post_atomic()

5534

5535 NAME

5536 it_post_atomic – post an Atomic DTO to an Endpoint

5537 SYNOPSIS

```
5538 #include <it_api.h>
5539
5540 typedef enum {
5541     IT_COMPARE_AND_SWAP,
5542     IT_FETCH_AND_ADD
5543 } it_atomic_op_t;
5544
5545 it_status_t it_post_atomic (
5546     IN          it_ep_handle_t      ep_handle,
5547     IN          it_atomic_op_t      op,
5548     IN          uint64_t            swap_or_add,
5549     IN          uint64_t            compare,
5550     IN const    it_lmr_triplet_t    *result,
5551     IN          it_dto_cookie_t      cookie,
5552     IN          it_dto_flags_t      dto_flags,
5553     IN          it_rdma_addr_t      rdma_addr,
5554     IN          it_rmr_context_t    rmr_context
5555 );
```

5556 APPLICABILITY

5557 *it_post_atomic* is applicable only to Endpoints created for the RC service type and is supported
5558 only if the Interface Adapter attribute *atomic_support* (see *it_ia_info_t*) is IT_TRUE.

5559 DESCRIPTION

5560	<i>ep_handle</i>	Handle for the Endpoint – the local side of the Connection.
5561	<i>op</i>	Atomic operation to be performed. Must be one of 5562 IT_COMPARE_AND_SWAP or IT_FETCH_AND_ADD.
5563	<i>swap_or_add</i>	Swap operand when <i>op</i> is IT_COMPARE_AND_SWAP. Add operand 5564 when <i>op</i> is IT_FETCH_AND_ADD.
5565	<i>compare</i>	Compare operand when <i>op</i> is IT_COMPARE_AND_SWAP. Not used 5566 when <i>op</i> is IT_FETCH_AND_ADD.
5567	<i>result</i>	One <i>it_lmr_triplet_t</i> data structure that specifies the local buffer where the 5568 result of the operation will be deposited. The local buffer must be at least 5569 eight bytes in size.
5570	<i>cookie</i>	Consumer-provided cookie that is returned to the Consumer in the 5571 Completion Event corresponding to the Atomic.
5572	<i>dto_flags</i>	Flags for posted Atomic.
5573	<i>rdma_addr</i>	The starting address of the remote buffer (target of Atomic operation). 5574 Remote buffer starting address must correspond to an I/O Address that is 5575 aligned to a modulo eight byte boundary at the remote and the remote

5576 buffer must be contiguous in the I/O Address space for eight bytes.
5577 Ensuring that the virtual address of the buffer is aligned to a modulo eight-
5578 byte boundary is sufficient on supported architectures.

5579 *rmr_context* The RMR Context for the remote buffer (target of Atomic operation).

5580 *it_post_atomic* posts a request to the *ep_handle* Endpoint to initiate an Atomic operation on the
5581 remote buffer, *rdma_addr*, via the reliable Connection of the *ep_handle* Endpoint.

5582 Two operations specified in *op* are supported, IT_FETCH_AND_ADD and
5583 IT_COMPARE_AND_SWAP.

5584 When *op* is specified as IT_FETCH_AND_ADD, the contents of *rdma_addr* are fetched from
5585 the remote and stored into *result*. The value in *swap_or_add* is then added to the contents of
5586 *rdma_addr*. The fetch and add operations are performed atomically at the remote. Conversion to
5587 the endianness of the remote system is performed by the Implementation and addition is
5588 performed in the native endianness of the remote system using 2's complement arithmetic
5589 without a carry (i.e., addition may overflow/underflow – wrapping around within the operand).
5590 The value fetched into *result* is converted into the native endianness of the local system. See [IB-
5591 R1.2], Section 9.4.5. It is the Consumer's responsibility to interpret whether or not the operands
5592 and results are signed arithmetically.

5593 When *op* is specified as IT_COMPARE_AND_SWAP, the contents of *rdma_addr* are compared
5594 with that in *compare*. If the values match, the swap is performed as follows: The contents of
5595 *rdma_addr* are fetched from the remote and stored into *result* and the value in *swap_or_add* is
5596 written into the contents of *rdma_addr*. The comparison and swap operations are performed
5597 atomically at the remote. The endiannesses of the local and remote systems are honored: the
5598 comparison is done in the native endianness of the remote. The value fetched into *result* is
5599 converted into the native endianness of the local system. See [IB-R1.2] Section 9.4.5.

5600 All Atomic operation requests made to the same remote IA, referencing the same target buffer
5601 (*rdma_addr*) shall appear serialized with respect to each other.

5602 Atomic operations made to an IA are not guaranteed to be serialized with respect to RDMA
5603 operation requests made to it or other IAs in the system, or with respect to operations performed
5604 by other system components such as processors. Because of this behavior, if Atomic operations
5605 on a particular area of memory are used to implement locks, all accesses to that memory must be
5606 done using Atomic operations. In particular, it is not safe to use an RDMA read or Send/Receive
5607 to see if a lock is held, and it is not safe to use an RDMA write or Send/Receive to clear a lock.

5608 For a remote buffer with Absolute Addressing, the starting address *rdma_addr* must be an
5609 absolute address within the remote linear address space, and for a remote buffer with Relative
5610 Addressing, *rdma_addr* must be a byte offset relative to the first byte of the remote buffer. In
5611 either case, *rdma_addr* must represent a buffer aligned as described above. See *it_addr_mode_t*
5612 for details on addressing modes.

5613 The remote buffer represented by *rmr_context* must have memory access privileges including
5614 both remote read and remote write access.

5615 An LMR used in *result* (data sink) must have local write access.

5616 The cookie (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work
5617 Request. This identifier is completely under Consumer control and opaque to the

5618 Implementation. The cookie is returned to the Consumer in the Completion Event for the posted
5619 Work Request.

5620 The *dto_flags* value is used as specified in *it_dto_flags_t*.

5621 A successful call returns IT_SUCCESS, which means that the Atomic operation was
5622 successfully posted to the transport layer.

5623 The completion of the posted Atomic is reported asynchronously to the Consumer according to
5624 the rules defined in *it_dto_flags_t*. An Atomic Completion Event is of type *it_dto_cmpl_event_t*.
5625 Any generated Atomic Completion Event manifests on the EVD associated with the Endpoint
5626 Send Queue. See *it_ep_rc_create*, *it_dto_status_t*, and *it_dto_events*. Once a successful
5627 Completion Event has been generated for the Atomic, the contents of the local buffer
5628 corresponds to the result of the Atomic operation. Prior to the Completion Event being
5629 generated, the content of the local buffer is Implementation-dependent.

5630 A Consumer shall not modify the local buffer specified by *result* until the DTO is completed.
5631 When a Consumer does not adhere to this rule, the behavior of the Implementation and the
5632 underlying transport is not defined. A Consumer does get back the ownership of the
5633 *it_lmr_triplet_t* represented by *result* (but not the local buffer identified by LMR triplet) when
5634 *it_post_atomic* returns and is free to use LMR triplet for other calls, to modify it, or to destroy it.

5635 If the reported status of the completion DTO Event corresponding to the posted Atomic is not
5636 IT_DTO_SUCCESS, the content of the local buffer is not defined.

5637 The Implementation ensures that the Atomic in no way corresponds to any Send or Recv Data
5638 Transfer Operations over the same Connection.

5639 The Implementation ensures that subsequent Atomic DTOs posted to the same Endpoint start
5640 and complete in post order.

5641 In general, Work Requests following an RDMA Read (or Atomic operation) may start execution
5642 while the RDMA Read (or Atomic operation) is in progress (but may not complete before the
5643 RDMA Read (or Atomic operation) completes). To ensure deterministically that an Atomic
5644 operation following an RDMA Read DTO (or following an Atomic operation) starts after the
5645 RDMA Read (or Atomic operation) completes, specify the IT_BARRIER_FENCE_FLAG on
5646 the following Atomic operation. This technique can be used to ensure that the Atomic DTOs
5647 place their data payloads into their local sink buffer in post order. It can also be used to prevent
5648 exceeding the IRD of the remote Endpoint.

5649 Any Work Request posted to an Endpoint's Send Queue (thus also an Atomic) after a call to
5650 *it_rmr_link* or *it_rmr_unlink* will not begin execution until the Link or Unlink operation has
5651 completed.

5652 The Implementation ensures that all data for a given Atomic operation is transferred from the
5653 remote buffer into the local buffer before an Atomic completion is generated with the status of
5654 IT_DTO_SUCCESS.

5655 **RETURN VALUE**

5656 A successful call returns IT_SUCCESS.

5657 Posting to an Endpoint that is not in the IT_EP_STATE_CONNECTED or IT_EP_STATE_
5658 NONOPERATIONAL state will return the IT_ERR_INVALID_EP_STATE immediate error.

5659	The possible immediate errors for <i>it_post_atomic</i> are listed below:	
5660 5661	IT_ERR_INVALID_DTO_FLAGS	The Data Transfer Operation flags (<i>dto_flags</i>) value was invalid.
5662	IT_ERR_INVALID_ATOMIC_OP	The Atomic operation (<i>op</i>) value was invalid.
5663	IT_ERR_INVALID_EP	The Endpoint Handle (<i>ep_handle</i>) was invalid.
5664 5665	IT_ERR_INVALID_EP_STATE	The Endpoint was not in the proper state for the attempted operation.
5666 5667	IT_ERR_INVALID_EP_TYPE	The attempted operation was invalid for the Service Type of the Endpoint.
5668	IT_ERR_TOO_MANY_POSTS	The operation failed due to an overflow of a Work Queue.
5669 5670 5671 5672	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

5673 **ASYNCHRONOUS ERRORS**

5674 For Work Requests posted to an Endpoint in the IT_EP_STATE_CONNECTED state, a
5675 completion status (see *it_dto_status_t*) other than IT_DTO_SUCCESS will break the Connection
5676 by moving the Endpoint to the IT_EP_STATE_NONOPERATIONAL state and deliver an
5677 IT_CM_MSG_CONN_BROKEN_EVENT Event to the Connect EVD of *ep_handle*. Once the
5678 Connection is broken, all outstanding and in-progress operations on the Connection will
5679 complete with an error status.

5680 Any posting to an Endpoint that is in the IT_EP_STATE_NONOPERATIONAL state will be
5681 flushed with completion status set to IT_DTO_ERR_FLUSHED.

5682 For locally or remotely detected errors that can be reported as Completion Errors, see also
5683 *it_dto_status_t*.

5684 An Atomic operation may result in a remotely detected access violation at the remote buffer.
5685 Such a violation may be due to misalignment or other reasons.

5686 The remote Implementation verifies the accessibility of the remote buffer represented by
5687 *rmr_context* based on the buffer's access rights, the buffer's association with the remote
5688 Endpoint that processes the operation, and the incoming RDMA operations allowed on the
5689 remote Endpoint. It declares an access violation if either the remote buffer has insufficient
5690 access rights, if *rmr_context* represents an LMR or Wide RMR whose Protection Zone does not
5691 match the Protection Zone of the remote Endpoint, if *rmr_context* represents a Narrow RMR that
5692 is associated with a different remote Endpoint, or if the targeted remote Endpoint does not have
5693 incoming Atomic operations enabled. An access violation at the data source is also declared if
5694 the Atomic operation exceeds the bounds of the remote buffer.

5695 An access violation at the data source will surface remotely as an
5696 IT_ASYNC_AFF_EP_L_ACCESS_VIOLATION Affiliated Asynchronous Error on the IB
5697 transport. Atomics are not supported on the iWARP Transport.

5698 An access violation at the remote buffer will surface locally as follows:
5699 For the IB transport, an IT_DTO_ERR_REMOTE_ACCESS Completion Error will occur.
5700 Atomics are not supported on the iWARP Transport.
5701 An Atomic operation may result in a locally detected access violation at the local data sink.
5702 The local Implementation verifies the accessibility of the local buffer represented by result based
5703 on the segment access rights and association with the local Endpoint, and the incoming
5704 operations allowed on the local Endpoint. It declares an access violation if an LMR in a local
5705 segment has insufficient access rights or a Protection Zone that does not match the Protection
5706 Zone of the local Endpoint. An access violation is also declared if the Atomic operation exceeds
5707 the bounds of an LMR in a local segment.
5708 An access violation at the local data sink will surface locally as follows:
5709 For the IB transport, an IT_DTO_ERR_LOCAL_PROTECTION Completion Error will occur.
5710 Atomics are not supported on the iWARP Transport.
5711 Despite using the RC service, an Atomic DTO may fail to successfully deliver the contents of
5712 the section of the remote buffer into the local buffer. This may result in a broken Connection or
5713 lead to data corruption in the local buffer, which is detected locally with high probability. In case
5714 of local data corruption, an IT_DTO_ERR_TRANSPORT Completion Error will occur.

5715 APPLICATION USAGE

5716 This function is used after a Connection has been established to perform atomic Fetch and Add
5717 or Compare and Swap operations on a Consumer-specified remote buffer and yield a result into
5718 a Consumer-specified local buffer.

5719 The Consumer should use unique identifiers for *cookie* if they desire to identify each DTO. If the
5720 Consumer does not require a unique DTO identifier, the value of zero or NULL
5721 (IT_NO_ADDR) can be used.

5722 It is the Consumer's responsibility to ensure *rdma_addr* corresponds to a remote buffer that is
5723 64-bit physically aligned and is physically contiguous for 64 bits. For Absolute Addressing, the
5724 Consumer must ensure that *rdma_addr* is 8-byte aligned. For Relative Addressing, the
5725 Consumer must ensure that "address + *rdma_addr*" is 8-byte aligned where "address" is the
5726 absolute address of the first byte of the remote memory region.

5727 For Relative Addressing, the alignment of the remote buffer cannot be determined at the
5728 Atomic-initiating side by examination of the lower bits of *rdma_addr* alone (since *rdma_addr* is
5729 just a byte offset relative to the starting address of the remote buffer, which may have arbitrary
5730 alignment). It is the Consumer's responsibility to ensure that the Atomic-initiating side has
5731 sufficient information to generate an *rdma_addr* byte offset that results in an 8-byte aligned
5732 absolute address in the remote buffer's address space.

5733 If the remote buffer corresponds to memory registered with the
5734 IT_LMR_FLAG_NONCOHERENT (i.e., the buffer is in non-coherent memory), then it is the
5735 Consumer's responsibility to ensure the synchronization of the view of the memory using the
5736 *it_lmr_flush_to_mem* and *it_lmr_refresh_from_mem* APIs appropriately.

5737 For best Atomic operation performance, the Consumer should align the buffer segment of *result*
5738 to the *dto_alignment_hint* in the IA attributes obtained via *it_ia_query*.

5739 When an EVD holding the DTO Event Stream associated with an Endpoint overflows, on some
5740 transports the behavior of the Endpoint and/or the EVD is undefined, making recovery from
5741 such an overflow either impossible or at the very least IA-dependent. Consumers should
5742 therefore take care to avoid this situation. Each Work Request that a Consumer posts to a Work
5743 Queue can result in a Completion Event being enqueued in an EVD, due to the possibility of an
5744 error occurring during Work Request processing. To avoid any possibility of overflow, the
5745 Consumer must therefore ensure that the total number of Outstanding Operations on all Work
5746 Queues that can enqueue Completion Events into a given EVD never exceeds the minimum
5747 queue size (as reported by *it_evd_query*) of that EVD.

5748 **SEE ALSO**

5749 *it_post_send()*, *it_post_send_and_unlink()*, *it_post_recv()*, *it_post_rdma_read()*,
5750 *it_post_rdma_read_to_rmr()*, *it_post_rdma_write()*, *it_ep_rc_create()*, *it_ia_query()*,
5751 *it_addr_mode_t*, *it_dto_cookie_t*, *it_dto_events*, *it_dto_flags_t*, *it_dto_status_t*, *it_ia_info_t*,
5752 *it_lmr_triplet_t*

it_post_rdma_read()

5753

5754 NAME

5755 `it_post_rdma_read` – post an RDMA Read DTO to an Endpoint

5756 SYNOPSIS

```
5757 #include <it_api.h>
5758
5759 it_status_t it_post_rdma_read (
5760     IN          it_ep_handle_t      ep_handle,
5761     IN  const   it_lmr_triplet_t    *local_segments,
5762     IN          size_t              num_segments,
5763     IN          it_dto_cookie_t     cookie,
5764     IN          it_dto_flags_t      dto_flags,
5765     IN          it_rdma_addr_t      rdma_addr,
5766     IN          it_rmr_context_t     rmr_context
5767 ) ;
```

5768 APPLICABILITY

5769 `it_post_rdma_read` is applicable only to Endpoints created for the RC service type.

5770 DESCRIPTION

5771	<code>ep_handle</code>	Handle for the Endpoint – the local side of the Connection.
5772	<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 (IT_NO_ADDR) for a zero-sized message.
5773		
5774		
5775	<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.
5776		
5777	<code>cookie</code>	Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the RDMA Read.
5778		
5779	<code>dto_flags</code>	Flags for posted RDMA Read.
5780	<code>rdma_addr</code>	The starting address of the section of the remote buffer from which to read.
5781	<code>rmr_context</code>	The RMR Context corresponding to the remote buffer from which reads will occur.
5782		
5783		<code>it_post_rdma_read</code> posts a request to the <code>ep_handle</code> Endpoint to transfer data from the section of the remote buffer (data source) specified by <code>rmr_context</code> and starting address <code>rdma_addr</code> into the local buffer (data sink) specified by <code>local_segments</code> and <code>num_segments</code> , via the reliable Connection of the <code>ep_handle</code> Endpoint. If <code>num_segments</code> is non-zero, then the size of the data transferred is given by the sum of the segment lengths specified by <code>local_segments</code> . A zero-sized message may be transferred. Like all other RDMA Read operations, the maximum number of <code>local_segments</code> is limited by the Endpoint attribute <code>max_rdma_read_segments</code> .
5784		
5785		
5786		
5787		
5788		
5789		
5790		For a remote buffer with Absolute Addressing, the starting address <code>rdma_addr</code> must be an absolute address within the remote linear address space, and for a remote buffer with Relative
5791		

5792 Addressing, *rdma_addr* must be a byte offset relative to the first byte of the remote buffer. See
5793 [it_addr_mode_t](#) for details on addressing modes.

5794 The Implementation ensures that an LMR Triplet supports byte alignment for Data Transfer
5795 Operations. If the buffer segments described by *local_segments* overlap, the resulting content of
5796 the local buffer is undefined.

5797 The remote buffer represented by *rmr_context* (data source) must have memory access privileges
5798 including remote read access.

5799 An LMR used in *local_segments* (data sink) must have memory access privileges including
5800 remote write access if the Interface Adapter attribute *rdma_read_requires_remote_write* equals
5801 IT_TRUE, or including local write access otherwise. See also Extended Description and
5802 Application Usage. The Direct LMR Handle may not be used.

5803 The *cookie* ([it_dto_cookie_t](#)) allows the Consumer to associate an identifier with each Work
5804 Request. This identifier is completely under Consumer control and opaque to the
5805 Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted
5806 Work Request.

5807 The *dto_flags* value is used as specified in [it_dto_flags_t](#).

5808 A successful call returns IT_SUCCESS, which means that the RDMA Read operation was
5809 successfully posted to the transport layer.

5810 The completion of the posted RDMA Read is reported asynchronously to the Consumer
5811 according to the rules defined in [it_dto_flags_t](#). An RDMA Read Completion Event is of type
5812 [it_dto_cmpl_event_t](#). Any generated RDMA Read Completion Event manifests on the EVD
5813 associated with the Endpoint Send Queue. See [it_ep_rc_create](#), [it_dto_status_t](#), and
5814 [it_dto_events](#). Once a successful Completion Event has been generated for the RDMA Read, the
5815 order of the bytes in the local buffer specified by *num_segments* and *local_segments* corresponds
5816 to the order defined by the section of the remote buffer unless there is local overlap. If there is
5817 local overlap, the content of the local buffer is undefined. Prior to the Completion Event being
5818 generated, the content of the local buffer is Implementation-dependent.

5819 A Consumer shall not modify the local buffer specified by *num_segments* and *local_segments*
5820 until the DTO is completed. When a Consumer does not adhere to this rule, the behavior of the
5821 Implementation and the underlying transport is not defined. A Consumer does get back the
5822 ownership of the array specified by the *local_segments* and *num_segments* arguments (but not
5823 the local buffer identified by this array) when *it_post_rdma_read* returns and is free to use this
5824 array for other calls, to modify it, or to destroy it.

5825 If the reported status of the completion DTO Event corresponding to the posted RDMA Read is
5826 not IT_DTO_SUCCESS, the content of the local buffer is not defined.

5827 The Implementation ensures that the RDMA Read in no way corresponds to any Send or Recv
5828 Data Transfer Operations over the same Connection.

5829 The Implementation ensures that subsequent RDMA Read DTOs posted to the same Endpoint
5830 start and complete in post order. However, the Implementation does not ensure that the RDMA
5831 Read DTOs place their data payloads into their local sink buffers in post order; if the local sink
5832 buffers overlap, their contents will be indeterminate.

5833 In general, Work Requests following an RDMA Read may start execution while the RDMA
5834 Read is in progress (but may not complete before the RDMA Read completes). To ensure

5835 deterministically that an RDMA Read DTO following an RDMA Read DTO starts after the first
5836 RDMA Read completes, specify the `IT_BARRIER_FENCE_FLAG` on the RDMA Read
5837 following the first RDMA Read. This technique can be used to ensure that the RDMA Read
5838 DTOs place their data payloads into their local sink buffers in post order. It can also be used to
5839 prevent exceeding the IRD of the remote Endpoint.

5840 Any Work Request posted to an Endpoint's Send Queue (thus also an RDMA Read) after a call
5841 to `it_rmr_link` or `it_rmr_unlink` will not begin execution until the Link or Unlink operation has
5842 completed.

5843 The Implementation ensures that all data for a given RDMA Read operation is transferred from
5844 the section of the remote buffer into the local buffer before an RDMA Read completion is
5845 generated with the status of `IT_DTO_SUCCESS`.

5846 **EXTENDED DESCRIPTION**

5847 For InfiniBand, the IA writing to a local buffer (data sink) in an RDMA Read DTO is considered
5848 a local write access, as indicated by the `it_ia_info_t` attribute `rdma_read_requires_remote_write`
5849 being `IT_FALSE`; consequently, the local buffer need only have the local write access privilege
5850 set and the Endpoint `ep_handle` need not have the `rdma_write_enable` attribute set to `IT_TRUE`.

5851 For iWARP, however, the IA writing to a local buffer in an RDMA Read DTO is considered a
5852 remote write access (by an incoming RDMA Read Response message), as indicated by the
5853 `it_ia_info_t` attribute `rdma_read_requires_remote_write` being `IT_TRUE`; hence, the local buffer
5854 must have the remote write access privilege set and the Endpoint `ep_handle` must have the
5855 `rdma_write_enable` attribute set to `IT_TRUE`.

5856 **RETURN VALUE**

5857 A successful call returns `IT_SUCCESS`.

5858 Posting to an Endpoint that is not in the `IT_EP_STATE_CONNECTED` or `IT_EP_STATE_`
5859 `NONOPERATIONAL` state will return the `IT_ERR_INVALID_EP_STATE` immediate error.

5860 The possible immediate errors for `it_post_rdma_read` are listed below:

5861	<code>IT_ERR_INVALID_DTO_FLAGS</code>	The Data Transfer Operation flags (<code>dto_flags</code>) value was 5862 invalid.
5863	<code>IT_ERR_INVALID_EP</code>	The Endpoint Handle (<code>ep_handle</code>) was invalid.
5864	<code>IT_ERR_INVALID_EP_STATE</code>	The Endpoint was not in the proper state for the 5865 attempted operation.
5866	<code>IT_ERR_INVALID_EP_TYPE</code>	The attempted operation was invalid for the Service 5867 Type of the Endpoint.
5868	<code>IT_ERR_INVALID_NUM_SEGMENTS</code>	The requested number of segments (<code>num_segments</code>) was 5869 larger than the Endpoint supports.
5870	<code>IT_ERR_TOO_MANY_POSTS</code>	The operation failed due to an overflow of a Work 5871 Queue.
5872	<code>IT_ERR_IA_CATASTROPHE</code>	The IA has experienced a catastrophic error and is in the 5873 disabled state. None of the output parameters from this

5874 routine are valid. See *it_ia_info_t* for a description of the
5875 disabled state

5876 **ASYNCHRONOUS ERRORS**

5877 For Work Requests posted to an Endpoint in the IT_EP_STATE_CONNECTED state, a
5878 completion status (see *it_dto_status_t*) other than IT_DTO_SUCCESS will break the Connection
5879 by moving the Endpoint to the IT_EP_STATE_NONOPERATIONAL state and deliver an
5880 IT_CM_MSG_CONN_BROKEN_EVENT Event to the Connect EVD of *ep_handle*.

5881 An Affiliated Asynchronous Error that is associated with an Endpoint will move that Endpoint to
5882 the IT_EP_STATE_NONOPERATIONAL state; if this state transition was from the
5883 IT_EP_STATE_CONNECTED state, an IT_CM_MSG_CONN_BROKEN_EVENT Event will
5884 be delivered to the Connect EVD of *ep_handle*.

5885 Once the Connection is broken, all outstanding and in-progress operations on the Connection
5886 will complete with an error status.

5887 Any posting to an Endpoint that is in the IT_EP_STATE_NONOPERATIONAL state will be
5888 flushed with completion status set to IT_DTO_ERR_FLUSHED.

5889 For locally or remotely detected errors that can be reported as Completion Errors, see also
5890 *it_dto_status_t*.

5891 The handling of remotely detected errors and some locally detected errors is transport and
5892 Implementation-dependent. For the iWARP Transport, an IA may provide advanced RDMA
5893 Read error handling support; i.e., RDMAP/DDP support for mapping an error due to an inbound
5894 RDMA Read Response to a Completion Error, support for on-the-fly conversion of an RDMAP
5895 Terminate message into a Completion Error and, optionally, early detection of a local access
5896 violation due to an RDMA Read Request.

5897 An RDMA Read operation may result in a remotely detected access violation (IRRQ access
5898 violation for iWARP) at the data source.

5899 The remote Implementation verifies the accessibility of the remote buffer represented by
5900 *rmr_context* based on the buffer's access rights, the buffer's association with the remote
5901 Endpoint that processes the operation, and the incoming RDMA operations allowed on the
5902 remote Endpoint. In case of a non-zero sized data transfer, it declares an access violation if
5903 *rmr_context* does not represent a valid and linked remote buffer, if the remote buffer has
5904 insufficient access rights, if *rmr_context* represents an LMR or Wide RMR whose Protection
5905 Zone does not match the Protection Zone of the remote Endpoint, if *rmr_context* represents a
5906 Narrow RMR that is associated with a different remote Endpoint, if the targeted remote
5907 Endpoint does not have incoming RDMA Read operations enabled, or if the RDMA Read
5908 operation exceeds the bounds of the remote buffer. In case of a zero-sized data transfer, no
5909 access violation will result regardless of the values of *rmr_context* and *rdma_addr*.

5910 An access violation at the data source will surface remotely as an IT_ASYNC_AFF_
5911 EP_L_ACCESS_VIOLATION Affiliated Asynchronous Error on the IB transport, and as an
5912 IT_ASYNC_AFF_EP_L_IRRQ_ACCESS_VIOLATION Affiliated Asynchronous Error on the
5913 iWARP Transport.

5914 An access violation at the data source will surface locally as follows:

5915 For the IB transport, an IT_DTO_ERR_REMOTE_ACCESS Completion Error will occur.

5916 For the iWARP Transport and an IA with advanced RDMA Read error handling support, an
5917 IT_DTO_ERR_REMOTE_ACCESS Completion Error will occur. For the iWARP Transport
5918 and an IA without such support, an IT_ASYNC_AFF_EP_R_ACCESS_VIOLATION Affiliated
5919 Asynchronous Error will surface.

5920 An RDMA Read operation may result in a locally detected access violation at the data sink.

5921 The local Implementation verifies the accessibility of the local buffer represented by
5922 *local_segments* and *num_segments* based on each segment's access rights and association with
5923 the local Endpoint, and the incoming RDMA operations allowed on the local Endpoint. In case
5924 of a non-zero sized data transfer, it declares an access violation if a local segment does not refer
5925 to a valid and linked LMR, if an LMR in a local segment has insufficient access rights or a
5926 Protection Zone that does not match the Protection Zone of the local Endpoint, if the
5927 *rdma_read_requires_remote_write* IA attribute equals IT_TRUE and the local Endpoint has
5928 incoming RDMA Write operations disabled, or if the RDMA Read operation exceeds the bounds
5929 of an LMR in a local segment. In case of a zero-sized data transfer, no access violation will
5930 result regardless of *num_segments* and any (zero-sized) segments in the *local_segments* vector.

5931 An access violation at the data sink will surface locally as follows:

5932 For the IB transport, an IT_DTO_ERR_LOCAL_PROTECTION Completion Error will occur.

5933 For the iWARP Transport and an IA with advanced RDMA Read error handling support, an
5934 IT_DTO_ERR_LOCAL_PROTECTION Completion Error will occur. For the iWARP
5935 Transport and an IA without such support, an IT_ASYNC_AFF_EP_L_ACCESS_VIOLATION
5936 Affiliated Asynchronous Error will surface.

5937 Despite using the RC service, an RDMA Read DTO may fail to successfully deliver the contents
5938 of the section of the remote buffer into the local buffer. A failure such as an unfinished data
5939 delivery or a CRC error that cannot be corrected by the transport will manifest locally as
5940 follows:

5941 For the IB transport, an IT_DTO_ERR_TRANSPORT Completion Error will occur.

5942 For the iWARP Transport and an IA with advanced RDMA Read error handling support, an
5943 IT_DTO_ERR_TRANSPORT Completion Error will occur. For the iWARP Transport and an
5944 IA without such support, an IT_ASYNC_AFF_EP_L_LLQ_ERROR Affiliated Asynchronous
5945 Error will surface.

5946 If a type of remotely detected error can manifest locally in different ways – i.e., as an Affiliated
5947 Asynchronous Error or as a Completion Error – transport-independent programming requires
5948 Consumers to be prepared to deal with either.

5949 If the Direct LMR Handle is used to directly specify a local buffer in an RDMA Read Work
5950 Request, then a Completion Error with DTO status IT_DTO_ERR_LOCAL_PROTECTION
5951 shall be returned.

5952 **APPLICATION USAGE**

5953 This function is used after a Connection has been established to transfer data from a Consumer-
5954 specified section of a remote buffer to a Consumer-specified local buffer.

5955 For writing transport-independent code, the Consumer should specify the remote and local write
5956 access privileges of an RDMA Read DTO's data sink and the *rdma_write_enable* attribute of the
5957 local Endpoint according to the IA attribute *rdma_read_requires_remote_write* obtained via

5958 *it_ia_query*. The Consumer may also enable both remote write and local write access for the data
5959 sink, regardless of the *rdma_read_requires_remote_write* attribute.

5960 The Consumer should use unique identifiers for *cookie* if they desire to identify each DTO. If the
5961 Consumer does not require a unique DTO identifier, the value of zero or NULL
5962 (IT_NO_ADDR) can be used.

5963 For best RDMA Read operation performance, the Consumer should align each buffer segment of
5964 *local_segments* to the *dto_alignment_hint* in the IA attributes obtained via *it_ia_query*.

5965 When an EVD holding the DTO Event Stream associated with an Endpoint overflows, on some
5966 transports the behavior of the Endpoint and/or the EVD is undefined, making recovery from
5967 such an overflow either impossible or at the very least IA-dependent. Consumers should
5968 therefore take care to avoid this situation. Each Work Request that a Consumer posts to a Work
5969 Queue can result in a Completion Event being enqueued in an EVD, due to the possibility of an
5970 error occurring during Work Request processing. To avoid any possibility of overflow, the
5971 Consumer must therefore ensure that the total number of Outstanding Operations on all Work
5972 Queues that can enqueue Completion Events into a given EVD never exceeds the minimum
5973 queue size (as reported by *it_evd_query*) of that EVD.

5974 **SEE ALSO**

5975 *it_post_atomic()*, *it_post_send()*, *it_post_send_and_unlink()*, *it_post_recv()*,
5976 *it_post_rdma_read_to_rmr()*, *it_post_rdma_write()*, *it_ep_rc_create()*, *it_addr_mode_t*,
5977 *it_ia_query()*, *it_dto_cookie_t*, *it_dto_events*, *it_dto_flags_t*, *it_dto_status_t*, *it_ia_info_t*,
5978 *it_lmr_triplet_t*

it_post_rdma_read_to_rmr()

5979

5980 NAME

5981 `it_post_rdma_read_to_rmr` – post an RDMA Read DTO to an Endpoint

5982 SYNOPSIS

```
5983 #include <it_api.h>
5984
5985 it_status_t it_post_rdma_read_to_rmr (
5986     IN          it_ep_handle_t      ep_handle,
5987     IN  const   it_rmr_triplet_t    *local_segments,
5988     IN          size_t              num_segments,
5989     IN          it_dto_cookie_t     cookie,
5990     IN          it_dto_flags_t     dto_flags,
5991     IN          it_rdma_addr_t     rdma_addr,
5992     IN          it_rmr_context_t    rmr_context
5993 );
```

5994 APPLICABILITY

5995 `it_post_rdma_read_to_rmr` is applicable only to Endpoints created for the RC service type and is
5996 supported only if the Interface Adapter attribute `rdma_read_local_extensions` (see [it_ia_info_t](#))
5997 is `IT_TRUE`.

5998 DESCRIPTION

5999	<code>ep_handle</code>	Handle for the Endpoint – the local side of the Connection.
6000	<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 <code>NULL</code> (<code>IT_NO_ADDR</code>) for a zero-sized message.
6001		
6002		
6003	<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.
6004		
6005	<code>cookie</code>	Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the RDMA Read.
6006		
6007	<code>dto_flags</code>	Flags for posted RDMA Read.
6008	<code>rdma_addr</code>	The starting address of the section of the remote buffer from which to read.
6009	<code>rmr_context</code>	The RMR Context corresponding to the remote buffer from which reads will occur.
6010		
6011		<code>it_post_rdma_read_to_rmr</code> posts a request to the <code>ep_handle</code> Endpoint to transfer data from the section of the remote buffer (data source) specified by <code>rmr_context</code> and <code>rdma_addr</code> into the local buffer (data sink) specified by <code>local_segments</code> and <code>num_segments</code> , via the reliable Connection of the <code>ep_handle</code> Endpoint. If <code>num_segments</code> is non-zero, then the size of the data transferred is given by the sum of the segment lengths specified by <code>local_segments</code> . A zero-sized message may be transferred. Like all other RDMA Read operations, the maximum number of <code>local_segments</code> is limited by the Endpoint attribute <code>max_rdma_read_segments</code> .
6012		
6013		
6014		
6015		
6016		
6017		

6018 For a remote buffer with Absolute Addressing, the starting address *rdma_addr* must be an
6019 absolute address within the remote linear address space, and for a remote buffer with Relative
6020 Addressing, *rdma_addr* must be a byte offset relative to the first byte of the remote buffer. See
6021 [it_addr_mode_t](#) for details on addressing modes.

6022 The Implementation ensures that an RMR Triplet supports byte alignment for Data Transfer
6023 Operations. If the buffer segments described by *local_segments* overlap, the resulting content of
6024 the local buffer is undefined.

6025 The remote buffer represented by *rmr_context* (data source) must have memory access privileges
6026 including remote read access.

6027 An RMR used in *local_segments* (data sink) must have memory access privileges including
6028 remote write access.

6029 The *cookie* ([it_dto_cookie_t](#)) allows the Consumer to associate an identifier with each Work
6030 Request. This identifier is completely under Consumer control and opaque to the
6031 Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted
6032 Work Request.

6033 The *dto_flags* value is used as specified in [it_dto_flags_t](#).

6034 A successful call returns IT_SUCCESS, which means that the RDMA Read operation was
6035 successfully posted to the transport layer.

6036 The completion of the posted RDMA Read is reported asynchronously to the Consumer
6037 according to the rules defined in [it_dto_flags_t](#). An RDMA Read Completion Event is of type
6038 [it_dto_cmpl_event_t](#). Any generated RDMA Read Completion Event manifests on the EVD
6039 associated with the Endpoint Send Queue. See [it_ep_rc_create](#), [it_dto_status_t](#), and
6040 [it_dto_events](#). Once a successful Completion Event has been generated for the RDMA Read, the
6041 order of the bytes in the local buffer specified by *num_segments* and *local_segments* corresponds
6042 to the order defined by the section of the remote buffer unless there is local overlap. If there is
6043 local overlap, the content of the local buffer is undefined. Prior to the Completion Event being
6044 generated, the content of the local buffer is Implementation-dependent.

6045 A Consumer shall not modify the local buffer specified by *num_segments* and *local_segments*
6046 until the DTO is completed. When a Consumer does not adhere to this rule, the behavior of the
6047 Implementation and the underlying transport is not defined. A Consumer does get back the
6048 ownership of the array specified by the *local_segments* and *num_segments* arguments (but not
6049 the local buffer identified by this array) when *it_post_rdma_read_to_rmr* returns and is free to
6050 use this array for other calls, to modify it, or to destroy it.

6051 If the reported status of the completion DTO Event corresponding to the posted RDMA Read is
6052 not IT_DTO_SUCCESS, the content of the local buffer is not defined.

6053 The Implementation ensures that the RDMA Read in no way corresponds to any Send or Recv
6054 Data Transfer Operations over the same Connection.

6055 The Implementation ensures that subsequent RDMA Read DTOs posted to the same Endpoint
6056 start and complete in post order. However, the Implementation does not ensure that the RDMA
6057 Read DTOs place their data payloads into their local sink buffers in post order; if the local sink
6058 buffers overlap, their contents will be indeterminate.

6059 In general, Work Requests following an RDMA Read may start execution while the RDMA
6060 Read is in progress (but may not complete before the RDMA Read completes). To ensure

6061 deterministically that an RDMA Read DTO following an RDMA Read DTO starts after the first
6062 RDMA Read completes, specify the `IT_BARRIER_FENCE_FLAG` on the RDMA Read
6063 following the first RDMA Read. This technique can be used to ensure that the RDMA Read
6064 DTOs place their data payloads into their local sink buffers in post order. It can also be used
6065 to prevent exceeding the IRD of the remote Endpoint.

6066 Any Work Request posted to an Endpoint's Send Queue (thus also an RDMA Read) after a call
6067 to `it_rmr_link` or `it_rmr_unlink` will not begin execution until the Link or Unlink operation has
6068 completed.

6069 The Implementation ensures that all data for a given RDMA Read operation is transferred from
6070 the section of the remote buffer into the local buffer before an RDMA Read completion is
6071 generated with the status of `IT_DTO_SUCCESS`.

6072 **EXTENDED DESCRIPTION**

6073 This call is supported only by iWARP. It allows using local RMRs as data sinks.

6074 For the iWARP Transport, the IA writing to a local buffer (data sink) in an RDMA Read DTO is
6075 considered a remote write access (by an incoming RDMA Read Response message), as indicated
6076 by the `it_ia_info_t` attribute `rdma_read_requires_remote_write` being `IT_TRUE`; hence, the local
6077 buffer must have the remote write access privilege set and the Endpoint `ep_handle` must have the
6078 `rdma_write_enable` attribute set to `IT_TRUE`.

6079 **RETURN VALUE**

6080 A successful call returns `IT_SUCCESS`.

6081 Posting to an Endpoint that is not in the `IT_EP_STATE_CONNECTED` or `IT_EP_STATE_`
6082 `NONOPERATIONAL` state will return the `IT_ERR_INVALID_EP_STATE` immediate error.

6083 The possible immediate errors for `it_post_rdma_read_to_rmr` are listed below:

6084 6085	<code>IT_ERR_INVALID_DTO_FLAGS</code>	The Data Transfer Operation flags (<code>dto_flags</code>) value was invalid.
6086	<code>IT_ERR_INVALID_EP</code>	The Endpoint Handle (<code>ep_handle</code>) was invalid.
6087 6088	<code>IT_ERR_INVALID_EP_STATE</code>	The Endpoint was not in the proper state for the attempted operation.
6089 6090	<code>IT_ERR_INVALID_EP_TYPE</code>	The attempted operation was invalid for the Service Type of the Endpoint.
6091 6092	<code>IT_ERR_INVALID_NUM_SEGMENTS</code>	The requested number of segments (<code>num_segments</code>) was larger than the Endpoint supports.
6093	<code>IT_ERR_OP_NOT_SUPPORTED</code>	The operation failed as it is not supported on the IA.
6094	<code>IT_ERR_TOO_MANY_POSTS</code>	The operation failed due to an overflow of a work queue.
6095 6096 6097 6098	<code>IT_ERR_IA_CATASTROPHE</code>	The IA has experienced a catastrophic error and is in the disabled state. None of the output parameters from this routine are valid. See <code>it_ia_info_t</code> for a description of the disabled state

6099 **ASYNCHRONOUS ERRORS**

6100 For Work Requests posted to an Endpoint in the `IT_EP_STATE_CONNECTED` state, a
6101 completion status (see *it_dto_status_t*) other than `IT_DTO_SUCCESS` will break the Connection
6102 by moving the Endpoint to the `IT_EP_STATE_NONOPERATIONAL` state and deliver an
6103 `IT_CM_MSG_CONN_BROKEN_EVENT` Event to the Connect EVD of *ep_handle*.

6104 An Affiliated Asynchronous Error that is associated with an Endpoint will move that Endpoint to
6105 the `IT_EP_STATE_NONOPERATIONAL` state; if this state transition was from the
6106 `IT_EP_STATE_CONNECTED` state, an `IT_CM_MSG_CONN_BROKEN_EVENT` Event will
6107 be delivered to the Connect EVD of *ep_handle*.

6108 Once the Connection is broken, all outstanding and in-progress operations on the Connection
6109 will complete with an error status.

6110 Any posting to an Endpoint that is in the `IT_EP_STATE_NONOPERATIONAL` state will be
6111 flushed with completion status set to `IT_DTO_ERR_FLUSHED`.

6112 For locally or remotely detected errors that can be reported as Completion Errors, see also
6113 *it_dto_status_t*.

6114 The handling of remotely detected errors and some locally detected errors is transport and
6115 Implementation-dependent. For the iWARP Transport, an IA may provide advanced RDMA
6116 Read error handling support; i.e., RDMAP/DDP support for mapping an error due to an inbound
6117 RDMA Read Response to a Completion Error, support for on-the-fly conversion of an RDMAP
6118 Terminate message into a Completion Error and, optionally, early detection of a local access
6119 violation due to an RDMA Read Request.

6120 An RDMA Read operation may result in a remotely detected access violation (`IRRQ` access
6121 violation for iWARP) at the data source.

6122 The remote Implementation verifies the accessibility of the remote buffer represented by
6123 *rmr_context* based on the buffer's access rights, the buffer's association with the remote
6124 Endpoint that processes the operation, and the incoming RDMA operations allowed on the
6125 remote Endpoint. In case of a non-zero sized data transfer, it declares an access violation if
6126 *rmr_context* does not represent a valid and linked remote buffer, if the remote buffer has
6127 insufficient access rights, if *rmr_context* represents an LMR or Wide RMR whose Protection
6128 Zone does not match the Protection Zone of the remote Endpoint, if *rmr_context* represents a
6129 Narrow RMR that is associated with a different remote Endpoint, if the targeted remote
6130 Endpoint does not have incoming RDMA Read operations enabled, or if the RDMA Read
6131 operation exceeds the bounds of the remote buffer. In case of a zero-sized data transfer, no
6132 access violation will result regardless of the values of *rmr_context* and *rdma_addr*.

6133 An access violation at the data source will surface remotely as an `IT_ASYNC_AFF_EP_L_`
6134 `ACCESS_VIOLATION` Affiliated Asynchronous Error on the IB transport, and as an
6135 `IT_ASYNC_AFF_EP_L_IRRQ_ACCESS_VIOLATION` Affiliated Asynchronous Error on the
6136 iWARP Transport.

6137 An access violation at the data source will surface locally as follows:

6138 For the IB transport, an `IT_DTO_ERR_REMOTE_ACCESS` Completion Error will occur.

6139 For the iWARP Transport and an IA with advanced RDMA Read error handling support, an
6140 `IT_DTO_ERR_REMOTE_ACCESS` Completion Error will occur. For the iWARP Transport

6141 and an IA without such support, an IT_ASYNC_AFF_EP_R_ACCESS_VIOLATION Affiliated
6142 Asynchronous Error will surface.

6143 An RDMA Read operation may result in a locally detected access violation at the data sink.

6144 The local Implementation verifies the accessibility of the local buffer represented by
6145 *local_segments* and *num_segments* based on each segment's access rights and association with
6146 the local Endpoint and the incoming RDMA operations allowed on the local Endpoint. In case of
6147 a non-zero sized data transfer, it declares an access violation if a local segment does not refer to
6148 a valid and linked RMR, if an RMR in a local segment has insufficient access rights or a
6149 Protection Zone that does not match the Protection Zone of the local Endpoint, if an RMR in a
6150 local segment is a Narrow RMR that is associated with a different local Endpoint, if the
6151 *rdma_read_requires_remote_write* IA attribute equals IT_TRUE and the local Endpoint has
6152 incoming RDMA Write operations disabled, or if the RDMA Read operation exceeds the bounds
6153 of an RMR in a local segment. In case of a zero-sized data transfer, no access violation will
6154 result regardless of *num_segments* and any (zero-sized) segments in the *local_segments* vector.

6155 An access violation at the data sink will surface locally as follows:

6156 For the IB transport, an IT_DTO_ERR_LOCAL_PROTECTION Completion Error will occur.

6157 For the iWARP Transport and an IA with advanced RDMA Read error handling support, an
6158 IT_DTO_ERR_LOCAL_PROTECTION Completion Error will occur. For the iWARP
6159 Transport and an IA without such support, an IT_ASYNC_AFF_EP_L_ACCESS_VIOLATION
6160 Affiliated Asynchronous Error will surface.

6161 Despite using the RC service, an RDMA Read DTO may fail to successfully deliver the contents
6162 of the section of the remote buffer into the local buffer. A failure such as an unfinished data
6163 delivery or a CRC error that cannot be corrected by the transport will manifest locally as
6164 follows:

6165 For the IB transport, an IT_DTO_ERR_TRANSPORT Completion Error will occur.

6166 For the iWARP Transport and an IA with advanced RDMA Read error handling support, an
6167 IT_DTO_ERR_TRANSPORT Completion Error will occur. For the iWARP Transport and an
6168 IA without such support, an IT_ASYNC_AFF_EP_L_LL_P_ERROR Affiliated Asynchronous
6169 Error will surface.

6170 If a type of remotely detected errors can manifest locally in different ways – i.e., as an Affiliated
6171 Asynchronous Error or as a Completion Error – transport-independent programming requires
6172 Consumers to be prepared to deal with either.

6173 APPLICATION USAGE

6174 This function is used after a Connection has been established to transfer data from a Consumer-
6175 specified section of a remote buffer to a Consumer-specified local buffer.

6176 For writing transport-independent code, the Consumer should specify the remote write access
6177 privilege of an RDMA Read DTO's data sink and the *rdma_write_enable* attribute of the local
6178 Endpoint according to *rdma_read_requires_remote_write* in the IA attributes obtained via
6179 *it_ia_query*. The Consumer may also enable both remote write and local write access for the data
6180 sink, regardless of the *rdma_read_requires_remote_write* attribute.

6181 The Consumer should use unique identifiers for *cookie* if they desire to identify each DTO. If the
6182 Consumer does not require a unique DTO identifier, the value of zero or NULL
6183 (IT_NO_ADDR) can be used.

6184 For best RDMA Read operation performance, the Consumer should align each buffer segment of
6185 *local_segments* to the *dto_alignment_hint* in the IA attributes obtained via *it_ia_query*.

6186 When an EVD holding the DTO Event Stream associated with an Endpoint overflows, on some
6187 transports the behavior of the Endpoint and/or the EVD is undefined, making recovery from
6188 such an overflow either impossible or at the very least IA-dependent. Consumers should
6189 therefore take care to avoid this situation. Each Work Request that a Consumer posts to a Work
6190 Queue can result in a Completion Event being enqueued in an EVD, due to the possibility of an
6191 error occurring during Work Request processing. To avoid any possibility of overflow, the
6192 Consumer must therefore ensure that the total number of Outstanding Operations on all Work
6193 Queues that can enqueue Completion Events into a given EVD never exceeds the minimum
6194 queue size (as reported by *it_evd_query*) of that EVD.

6195 **SEE ALSO**

6196 *it_post_atomic()*, *it_post_send()*, *it_post_send_and_unlink()*, *it_post_sendto()*, *it_post_recv()*,
6197 *it_post_recvfrom()*, *it_post_rdma_read()*, *it_post_rdma_write()*, *it_rmr_link()*, *it_rmr_unlink()*,
6198 *it_ep_rc_create()*, *it_ia_query()*, *it_addr_mode_t*, *it_dto_cookie_t*, *it_dto_events*, *it_dto_flags_t*,
6199 *it_dto_status_t*, *it_ia_info_t*, *it_rmr_triplet_t*

it_post_rdma_write()

6200

6201 NAME

6202 `it_post_rdma_write` – post an RDMA Write DTO to an Endpoint

6203 SYNOPSIS

```
6204 #include <it_api.h>
6205
6206 it_status_t it_post_rdma_write (
6207     IN          it_ep_handle_t      ep_handle,
6208     IN  const   it_lmr_triplet_t    *local_segments,
6209     IN          size_t              num_segments,
6210     IN          it_dto_cookie_t     cookie,
6211     IN          it_dto_flags_t      dto_flags,
6212     IN          it_rdma_addr_t      rdma_addr,
6213     IN          it_rmr_context_t    rmr_context
6214 ) ;
```

6215 APPLICABILITY

6216 `it_post_rdma_write` is applicable only to Endpoints created for the RC service type.

6217 DESCRIPTION

6218	<code>ep_handle</code>	Handle for the Endpoint – the local side of the Connection.
6219	<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 (IT_NO_ADDR) for a zero-sized message.
6220		
6221		
6222	<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.
6223		
6224	<code>cookie</code>	Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the RDMA Write.
6225		
6226	<code>dto_flags</code>	Flags for posted RDMA Write.
6227	<code>rdma_addr</code>	The starting address of the section of the remote buffer to which to write.
6228	<code>rmr_context</code>	The RMR Context corresponding to the remote buffer to which writes will occur.
6229		
6230		<code>it_post_rdma_write</code> posts a request to the <code>ep_handle</code> Endpoint to transfer all the data from the local buffer (data source) specified by <code>local_segments</code> and <code>num_segments</code> into the section of the remote buffer (data sink) specified by <code>rmr_context</code> and <code>rdma_addr</code> , via the reliable Connection of the <code>ep_handle</code> Endpoint. If <code>num_segments</code> is non-zero, then the size of the data transferred is given by the sum of the segment lengths specified by <code>local_segments</code> . A zero-sized message may be transferred.
6231		
6232		
6233		
6234		
6235		
6236		For a remote buffer with Absolute Addressing, the starting address <code>rdma_addr</code> must be an absolute address within the remote linear address space, and for a remote buffer with Relative Addressing, <code>rdma_addr</code> must be a byte offset relative to the first byte of the remote buffer. See it_addr_mode_t for details on addressing modes.
6237		
6238		
6239		

6240 The Implementation ensures that an LMR Triplet supports byte alignment for Data Transfer
6241 Operations. The buffer segments described by *local_segments* can overlap; it is safe to use
6242 overlapping buffer segments as a data source.

6243 An LMR used in *local_segments* (data source) must have memory access privileges including
6244 local read access. The Direct LMR Handle may be used by Privileged Mode Consumers (see
6245 Asynchronous Errors for more details).

6246 The remote buffer represented by *rmr_context* (data sink) must have memory access privileges
6247 including remote write access.

6248 The *cookie* (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work
6249 Request. This identifier is completely under Consumer control and opaque to the
6250 Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted
6251 Work Request.

6252 The *dto_flags* value is used as specified in *it_dto_flags_t*.

6253 A successful call returns IT_SUCCESS, which means that the RDMA Write operation was
6254 successfully posted to the transport layer.

6255 The completion of the posted RDMA Write is reported asynchronously to the Consumer
6256 according to the rules defined in *it_dto_flags_t*. An RDMA Write Completion Event is of type
6257 *it_dto_cmpl_event_t*. Any generated RDMA Write Completion Event manifests on the EVD
6258 associated with the Endpoint Send Queue. See *it_ep_rc_create*, *it_dto_status_t*, and
6259 *it_dto_events*. The Completion Event for the *it_post_rdma_write* call indicates to the Consumer
6260 that the local buffer is under Consumer control again and that the contents of the local buffer
6261 will be transported reliably, but does not guarantee that these contents have been successfully
6262 delivered into the memory of the remote Consumer; wherever necessary, this restriction is made
6263 explicit by designating a Completion as local or by stating that operations complete locally. See
6264 Section 5.4 of [DDP-IETF] for background information. However, once the contents of the local
6265 buffer reach the remote Consumer memory, the order of the bytes in the remote memory
6266 corresponds to the order defined by the *local_segments*.

6267 A Consumer should not modify the local buffer specified by *num_segments* and *local_segments*
6268 until the DTO is locally completed. When a Consumer does not adhere to this rule, the behavior
6269 of the Implementation and the underlying transport is not defined. A Consumer does get back the
6270 ownership of the array specified by the *local_segments* and *num_segments* arguments (but not
6271 the local buffer identified by this array) when *it_post_rdma_write* returns and is free to use this
6272 array for other calls, to modify it, or to destroy it.

6273 The Implementation ensures that the RDMA Write in no way corresponds to any Receive Data
6274 Transfer Operations over the same Connection.

6275 The Implementation ensures that subsequent RDMA Write DTOs posted to the same Endpoint
6276 start and complete locally in post order. However, the Implementation does not ensure that the
6277 RDMA Write DTOs place their data payloads into their remote buffers in post order; if the
6278 targeted sections of the remote buffers overlap, their contents will be indeterminate.

6279 The Implementation ensures that each RDMA Write DTO posted to an Endpoint prior to a Send
6280 DTO posted to the same Endpoint has its complete data payload delivered to the remote memory
6281 prior to the completion of the Receive DTO at the remote side matching that Send. However, the
6282 Implementation does not ensure that the RDMA Write DTO places its entire data payload into
6283 its remote buffer before the Receive DTO places the data payload of the incoming Send into its

6284 receive buffer; if the targeted sections of the remote buffers overlap, their contents will be
6285 indeterminate.

6286 In general, Work Requests following an RDMA Read may start execution while the RDMA
6287 Read is in progress (but may not complete before the RDMA Read completes). To ensure
6288 deterministically that an RDMA Write DTO following an RDMA Read DTO starts after the
6289 RDMA Read completes, specify the `IT_BARRIER_FENCE_FLAG` on the RDMA Write DTO.

6290 Any Work Request posted to an Endpoint's Send Queue (thus also an RDMA Write) after a call
6291 to *it_rmr_link* or *it_rmr_unlink* will not begin execution until the Link or Unlink operation has
6292 completed.

6293 **EXTENDED DESCRIPTION**

6294 See *it_lmr_refresh_from_mem* for a discussion of ramifications of RDMA Write use on non-
6295 coherent systems.

6296 On the InfiniBand Transport, the Implementation ensures that subsequent RDMA Write DTOs
6297 or an RDMA Write DTO followed by a Send DTO posted to the same Endpoint cause data
6298 payloads to be placed in remote memory in post order; relying on such placement ordering
6299 represents a transport-dependent programming practice.

6300 **RETURN VALUE**

6301 A successful call returns `IT_SUCCESS`.

6302 Posting to an Endpoint that is not in the `IT_EP_STATE_CONNECTED` or `IT_EP_STATE_`
6303 `NONOPERATIONAL` state will return the `IT_ERR_INVALID_EP_STATE` immediate error.

6304 The possible immediate errors for *it_post_rdma_write* are listed below:

6305	<code>IT_ERR_INVALID.DTO_FLAGS</code>	The Data Transfer Operation flags (<i>dto_flags</i>) value was 6306 invalid.
6307	<code>IT_ERR_INVALID_EP</code>	The Endpoint Handle (<i>ep_handle</i>) was invalid.
6308	<code>IT_ERR_INVALID_EP_STATE</code>	The Endpoint was not in the proper state for the attempted 6309 operation.
6310	<code>IT_ERR_INVALID_EP_TYPE</code>	The attempted operation was invalid for the Service Type 6311 of the Endpoint.
6312	<code>IT_ERR_INVALID_NUM_SEGMENTS</code>	The requested number of segments (<i>num_segments</i>) was 6313 larger than the Endpoint supports.
6314	<code>IT_ERR_TOO_MANY_POSTS</code>	The operation failed due to an overflow of a Work Queue.
6315	<code>IT_ERR_IA_CATASTROPHE</code>	The IA has experienced a catastrophic error and is in the 6316 disabled state. None of the output parameters from this 6317 routine are valid. See <i>it_ia_info_t</i> for a description of the 6318 disabled state.

6319 **ASYNCHRONOUS ERRORS**

6320 For Work Requests posted to an Endpoint in the `IT_EP_STATE_CONNECTED` state, a
6321 completion status (see *it_dto_status_t*) other than `IT.DTO_SUCCESS` will break the Connection
6322 by moving the Endpoint to the `IT_EP_STATE_NONOPERATIONAL` state and deliver an
6323 `IT_CM_MSG_CONN_BROKEN_EVENT` Event to the Connect EVD of *ep_handle*. Once the

6324 Connection is broken, all outstanding and in-progress operations on the Connection will
6325 complete with an error status.

6326 An Affiliated Asynchronous Error that is associated with an Endpoint will move that Endpoint to
6327 the `IT_EP_STATE_NONOPERATIONAL` state; if this state transition was from the
6328 `IT_EP_STATE_CONNECTED` state, an `IT_CM_MSG_CONN_BROKEN_EVENT` Event will
6329 be delivered to the Connect EVD of *ep_handle*.

6330 Any posting to an Endpoint that is in the `IT_EP_STATE_NONOPERATIONAL` state will be
6331 flushed with completion status set to `IT_DTO_ERR_FLUSHED`.

6332 For locally or remotely detected errors that can be reported as Completion Errors, see also
6333 [*it_dto_status_t*](#).

6334 The handling of remotely detected errors is transport and Implementation-dependent; end-to-end
6335 completions are not supported for certain transports or DTOs.

6336 An RDMA Write operation may result in a remotely detected access violation, also referred to as
6337 a protection error by some transports.

6338 The remote Implementation verifies the accessibility of the remote buffer represented by
6339 *rmr_context* based on the buffer's access rights, the buffer's association with the remote
6340 Endpoint that processes the operation, and the RDMA operations allowed on the remote
6341 Endpoint. In case of a non-zero sized data transfer, it declares an access violation if *rmr_context*
6342 does not represent a valid and linked remote buffer, if the remote buffer has insufficient access
6343 rights, if *rmr_context* represents an LMR or Wide RMR whose Protection Zone does not match
6344 the Protection Zone of the remote Endpoint, if *rmr_context* represents a Narrow RMR that is
6345 associated with a different remote Endpoint, if the targeted remote Endpoint has incoming
6346 RDMA Write operations disabled, or if the RDMA Write operation exceeds the bounds of the
6347 remote buffer. In case of a zero-sized data transfer, no access violation will result regardless of
6348 the values of *rmr_context* and *rdma_addr*.

6349 An access violation due to an RDMA Write operation will surface remotely as an
6350 `IT_ASYNC_AFF_EP_L_ACCESS_VIOLATION` Affiliated Asynchronous Error on both the IB
6351 and iWARP Transports.

6352 An access violation due to an RDMA Write operation will surface locally as follows:

6353 For the IB transport, an `IT_DTO_ERR_REMOTE_ACCESS` Completion Error will occur.

6354 For the iWARP Transport, either an `IT_ASYNC_AFF_EP_R_ACCESS_VIOLATION`
6355 Affiliated Asynchronous Error will surface after the DTO has already completed with
6356 `IT_DTO_SUCCESS` or, if the Implementation is able to convert a Terminate message to a
6357 Completion Error, an `IT_DTO_ERR_REMOTE_ACCESS` Completion Error will occur.

6358 Despite using the RC service, an RDMA Write operation may fail to successfully deliver the
6359 contents of the local buffer into the remote buffer. A failure such as an unfinished data delivery
6360 or a CRC error that cannot be corrected by the transport is detected as a transport error and
6361 manifests as follows:

6362 For the IB transport, a transport error causes the RDMA Write DTO to complete with an
6363 `IT_DTO_ERR_TRANSPORT` Completion Error.

6364 For the iWARP Transport, a transport error will surface remotely as an
6365 `IT_ASYNC_AFF_EP_L_LL_P_ERROR` Affiliated Asynchronous Error, and locally as an

6366 IT_ASYNC_AFF_EP_R_ERROR Affiliated Asynchronous Error after the RDMA Write DTO
6367 has already completed with IT_DTO_SUCCESS.

6368 If a type of remotely detected error can manifest locally in different ways – i.e., as an Affiliated
6369 Asynchronous Error or as a Completion Error – transport-independent programming requires
6370 Consumers to be prepared to deal with either.

6371 If the Direct LMR Handle is used to specify a local buffer in a Work Request and the Endpoint
6372 to which the Work Request is posted is not Privileged, then a Completion Error with the DTO
6373 status IT_DTO_ERR_LOCAL_PROTECTION shall be returned.

6374 APPLICATION USAGE

6375 This function is used after a Connection has been established to transfer data from a Consumer-
6376 specified local buffer to a section of a remote buffer on the other side of the Connection.

6377 The Consumer should use unique identifiers for *cookie* if they desire to identify each DTO. If the
6378 Consumer does not require a unique DTO identifier, the value of zero or NULL
6379 (IT_NO_ADDR) can be used.

6380 For best RDMA Write operation performance, the Consumer should align each buffer segment
6381 of *local_segments* to the *dto_alignment_hint* in the IA attributes obtained via *it_ia_query*.

6382 There are a variety of ways to guarantee the delivery of a local buffer via RDMA Write into the
6383 memory of the remote Consumer. One way would be for the Consumer to send a message over
6384 the Connection after the RDMA Write had completed, and then wait for the remote Consumer to
6385 reply to that message. When the remote Consumer reaps the Receive Completion for the Send
6386 from the local Consumer, the payload of the RDMA Write must have been delivered into the
6387 remote memory.

6388 When an EVD holding the DTO Event Stream associated with an Endpoint overflows, on some
6389 transports the behavior of the Endpoint and/or the EVD is undefined, making recovery from
6390 such an overflow either impossible or at the very least IA-dependent. Consumers should
6391 therefore take care to avoid this situation. Each Work Request that a Consumer posts to a Work
6392 Queue can result in a Completion Event being enqueued in an EVD, due to the possibility of an
6393 error occurring during Work Request processing. To avoid any possibility of overflow, the
6394 Consumer must therefore ensure that the total number of Outstanding Operations on all Work
6395 Queues that can enqueue Completion Events into a given EVD never exceeds the minimum
6396 queue size (as reported by *it_evd_query*) of that EVD.

6397 SEE ALSO

6398 *it_post_atomic()*, *it_post_send()*, *it_post_send_and_unlink()*, *it_post_sendto()*, *it_post_rcv()*,
6399 *it_post_rcvfrom()*, *it_post_rdma_read()*, *it_post_rdma_read_to_rmr()*, *it_rmr_link()*,
6400 *it_rmr_unlink()*, *it_ep_rc_create()*, *it_ia_query()*, *it_addr_mode_t*, *it_dto_cookie_t*,
6401 *it_dto_events*, *it_dto_status_t*, *it_dto_flags_t*, *it_ia_info_t*, *it_lmr_triplet_t*

it_post_recv()

6402

6403 NAME

6404

it_post_recv – post a Receive DTO to an Endpoint or Shared Receive Queue

6405 SYNOPSIS

6406

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

6407

```
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  
);
```

6415 APPLICABILITY

6416

it_post_recv is applicable only to Endpoints created for the RC service type or to Shared Receive Queues.

6417

6418 DESCRIPTION

6419

handle Handle for the Endpoint or S-RQ.

6420

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 (IT_NO_ADDR) for a zero-sized message.

6421

6422

6423

num_segments Number of *it_lmr_triplet_t* data structures in *local_segments*. Can be zero for a zero-sized message.

6424

6425

cookie Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the Receive.

6426

6427

dto_flags Flags for posted Receive.

6428

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.

6429

6430

6431

6432

6433

6434

6435

6436

An LMR used in *local_segments* (data sink) must have memory access privileges including local write access. The Direct LMR Handle may be used by Privileged Mode Consumers (see Asynchronous Errors for more details).

6437

6438

6439

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.

6440

6441

6442 The *cookie* (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work
6443 Request. This identifier is completely under Consumer control and opaque to the
6444 Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted
6445 Work Request.

6446 The *dto_flags* value is used as specified in *it_dto_flags_t*.

6447 A successful call returns IT_SUCCESS, which means that the Receive operation was
6448 successfully posted to the transport layer.

6449 If an Endpoint has an associated S-RQ and an incoming Send arrives for that Endpoint, then the
6450 Implementation behaves as if a Receive Work Request is dequeued from the S-RQ and enqueued
6451 onto the Endpoint's local RQ. Receive Work Requests are not necessarily dequeued from the S-
6452 RQ in the order in which Send messages arrive. Depending on Implementation, an out-of-order
6453 arrival of a Send message may cause more than one Receive WR to be dequeued from the S-RQ.
6454 An Interface Adapter may support the Endpoint Soft High Watermark and/or Endpoint Hard
6455 High Watermark mechanisms (see *it_ep_attributes_t* and *it_ia_info_t*), allowing Consumers to
6456 bound the effects of out-of-order arrivals.

6457 The completion of the posted Receive is reported asynchronously to the Consumer according to
6458 the rules defined in *it_dto_flags_t*. A Receive Completion Event is of type *it_dto_cmpl_event_t*.
6459 Exactly one Receive Completion Event is always generated and manifests on the EVD
6460 associated with the Endpoint to which the corresponding incoming Send operation was directed
6461 (see *it_ep_rc_create*).

6462 If the reported *dto_status* of the Receive Completion Event (see *it_dto_cmpl_event_t*) is
6463 IT_DTO_SUCCESS (see *it_dto_status_t*), then all data from an incoming Send message has
6464 been transferred into the Receive buffer and the size of the received data can be retrieved
6465 through the *transferred_length* member of the Event. Reception of a zero-sized message is
6466 supported and will consume a posted Receive. Once a successful Completion Event has been
6467 generated for the Receive, the order of the bytes in the local buffer specified by *local_segments*
6468 and *num_segments* corresponds to the order defined by the *local_segments* of the corresponding
6469 Send operation unless there is overlap among the segments of the local Receive buffer. If there
6470 is such an overlap, the content of the local Receive buffer after the Completion Event has been
6471 generated is undefined. Prior to the Completion Event being generated, the content of the local
6472 buffer is Implementation-dependent.

6473 A Consumer should not modify the local buffer specified by *num_segments* and *local_segments*
6474 until the DTO is completed. When a Consumer does not adhere to this rule, the behavior of the
6475 Implementation and the underlying transport is not defined. A Consumer does get back the
6476 ownership of the array specified by the *local_segments* and *num_segments* arguments (but not
6477 the local buffer identified by this array) when *it_post_rcv* returns and is free to use this array for
6478 other calls, to modify it, or to destroy it.

6479 The Implementation ensures that each Receive corresponds to one and only one remote Send and
6480 in no way corresponds to any RDMA Read or RDMA Write Data Transfer Operations over the
6481 same Connection.

6482 The Implementation ensures that an RDMA Write DTO from a remote Endpoint preceding a
6483 Send DTO from the same remote Endpoint has fully delivered its payload prior to the
6484 completion of the Receive DTO corresponding to the Send DTO. However, the Implementation
6485 does not ensure that the RDMA Write DTO places its entire data payload into its Destination

6486 buffer before the Receive DTO places the data payload of the incoming Send into its Receive
6487 buffer; if these two buffers overlap, their contents will be indeterminate.

6488 The Implementation ensures that Receive DTOs matching subsequent Send DTOs from the same
6489 remote Endpoint complete in the post order of these Send DTOs. However, the Implementation
6490 does not ensure that the matching Receive DTOs place the data payloads into their Receive
6491 buffers in the post order of the Send DTOs; if the Receive buffers overlap, their contents will be
6492 indeterminate. If a collection of Endpoints is using an S-RQ, the Receive Completion Events
6493 will be generated for a particular Endpoint in the same order that the corresponding Sends were
6494 done on the associated remote Endpoint.

6495 For Receives that are posted to an Endpoint, the Implementation ensures that all Receives start
6496 and complete in the order posted. The same is not true for Receives that are posted to an S-RQ;
6497 such Receive Work Requests do not necessarily complete in the order posted, nor in the order in
6498 which Send messages arrive, nor in the order in which the Receive Work Requests are dequeued
6499 from the S-RQ.

6500 There is no order relationship between completions of Receive DTOs and any other Work
6501 Requests posted to an Endpoint.

6502 RETURN VALUE

6503 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

6504	IT_ERR_INVALID.DTO_FLAGS	The Data Transfer Operation flags (<i>dto_flags</i>) value
6505		was invalid.
6506	IT_ERR_INVALID_EP	The input handle (<i>handle</i>) was invalid.
6507	IT_ERR_INVALID_EP_TYPE	The attempted operation was invalid for the Service
6508		Type of the Endpoint.
6509	IT_ERR_INVALID_NUM_SEGMENTS	The requested number of segments (<i>num_segments</i>)
6510		was larger than the Endpoint or S-RQ supports.
6511	IT_ERR_INVALID_SRQ	The S-RQ handle was invalid.
6512	IT_ERR_TOO_MANY_POSTS	The operation failed due to an overflow of a Work
6513		Queue.
6514	IT_ERR_IA_CATASTROPHE	The IA has experienced a catastrophic error and is in
6515		the disabled state. None of the output parameters from
6516		this routine are valid. See <i>it_ia_info_t</i> for a description
6517		of the disabled state.

6518 ASYNCHRONOUS ERRORS

6519 A completion status (see *it_dto_status_t*) other than IT.DTO_SUCCESS will break the
6520 Connection associated with the Endpoint that received the corresponding Send message by
6521 moving that Endpoint to the IT_EP_STATE_NONOPERATIONAL state and delivering an
6522 IT_CM_MSG_CONN_BROKEN_EVENT Event to the Connect EVD of that Endpoint. Once
6523 that Connection is broken, all outstanding and in-progress operations on that Connection will
6524 complete with an error status. If the reported completion status is not IT.DTO_SUCCESS, the
6525 content of the local buffer is not defined.

6526 An Affiliated Asynchronous Error that is associated with an Endpoint will move that Endpoint to
6527 the `IT_EP_STATE_NONOPERATIONAL` state; if this state transition was from the
6528 `IT_EP_STATE_CONNECTED` state, an `IT_CM_MSG_CONN_BROKEN_EVENT` Event will
6529 be delivered to the Connect EVD of *ep_handle*.

6530 Any posting to an Endpoint that is in the `IT_EP_STATE_NONOPERATIONAL` state will be
6531 flushed with completion status set to `IT_DTO_ERR_FLUSHED`.

6532 For errors that can be reported as Completion Errors, see also *it_dto_status_t*.

6533 If no Receive buffers are posted in time, then an incoming Send will fail to deliver its data. In
6534 order to avoid this error, the Consumer should post Receive resources prior to the remote
6535 Consumer posting a Send. The lack of a Receive buffer is handled as follows:

6536 For IB and iWARP, an `IT_ASYNC_AFF_EP_L_RECV_ACCESS_VIOLATION` Affiliated
6537 Asynchronous Error is generated.

6538 For the IB transport, the Send DTO that was posted remotely completes with an
6539 `IT_DTO_ERR_REMOTE_RESPONDER` error status.

6540 For the iWARP Transport, an `IT_ASYNC_AFF_EP_R_RECV_ACCESS_VIOLATION`
6541 Affiliated Asynchronous Error will surface remotely after the Send DTO posted remotely has
6542 already completed with `IT_DTO_SUCCESS`.

6543 If the Receive buffer is not sufficient in size for the data payload of the incoming Send message,
6544 a length error will occur. In order to avoid length errors, the Consumer should post a buffer large
6545 enough for the incoming Send data. Length errors are handled as follows:

6546 For both the IB and iWARP Transports, the Receive DTO completes with an
6547 `IT_DTO_ERR_LOCAL_LENGTH` error status.

6548 For the IB transport, the Send DTO that was posted remotely completes with an
6549 `IT_DTO_ERR_REMOTE_RESPONDER` error status.

6550 For the iWARP Transport, an `IT_ASYNC_AFF_EP_R_RECV_LENGTH_ERROR` Affiliated
6551 Asynchronous Error will surface remotely after the Send DTO posted remotely has already
6552 completed with `IT_DTO_SUCCESS`.

6553 Despite using the RC service, an incoming Send message may fail to successfully deliver its
6554 data. A failure such as an unfinished data delivery or a CRC error that cannot be corrected by the
6555 transport is detected as a transport error and manifests as follows:

6556 For the IB transport, a transport error causes the Send DTO posted remotely and the Receive
6557 DTO matching the Send DTO to complete with an `IT_DTO_ERR_TRANSPORT` Completion
6558 Error.

6559 For the iWARP Transport, a transport error causes the Receive DTO to complete with an
6560 `IT_DTO_ERR_TRANSPORT` Completion Error and an `IT_ASYNC_AFF_EP_R_ERROR`
6561 Affiliated Asynchronous Error to surface remotely after the Send DTO posted remotely has
6562 already completed with `IT_DTO_SUCCESS`.

6563 If the Direct LMR Handle is used to specify a local buffer in a Work Request and the Endpoint
6564 to which the Work Request is posted is not Privileged, then a Completion Error with the DTO
6565 status `IT_DTO_ERR_LOCAL_PROTECTION` shall be returned.

6566 If the Direct LMR Handle is used to specify a local buffer in a Receive Work Request posted to
6567 an S-RQ and the Endpoint that finally processes the Receive Work Request is not Privileged,
6568 then a Completion Error with the DTO status IT_DTO_ERR_LOCAL_PROTECTION shall be
6569 returned.

6570 APPLICATION USAGE

6571 This function is used to post a Receive DTO to an Endpoint or Shared Receive Queue,
6572 requesting the transfer of data into a Consumer-specified local buffer from a buffer specified by
6573 the corresponding Send operation on the other side of the Connection.

6574 A Receive DTO can be posted to an Endpoint in any RC Endpoint state (see *it_ep_state_t*) other
6575 than IT_EP_STATE_NONOPERATIONAL – i.e., even before Connection establishment – in
6576 order to prepare the Endpoint for reception.

6577 The Consumer should use unique identifiers for *cookie* if they desire to identify each DTO. If the
6578 Consumer does not require a unique DTO identifier, the value of zero or NULL
6579 (IT_NO_ADDR) can be used.

6580 For best Receive operation performance, the Consumer should align each buffer segment of
6581 *local_segments* to the *dto_alignment_hint* in the IA attributes obtained via *it_ia_query*.

6582 When an EVD holding the DTO Event Stream associated with an Endpoint overflows, on some
6583 transports the behavior of the Endpoint and/or the EVD is undefined, making recovery from
6584 such an overflow either impossible or at the very least IA-dependent. Consumers should
6585 therefore take care to avoid this situation. Each Work Request that a Consumer posts to a Work
6586 Queue can result in a Completion Event being enqueued in an EVD, due to the possibility of an
6587 error occurring during Work Request processing. To avoid any possibility of overflow, the
6588 Consumer must therefore ensure that the total number of Outstanding Operations on all Work
6589 Queues that can enqueue Completion Events into a given EVD never exceeds the minimum
6590 queue size (as reported by *it_evd_query*) of that EVD.

6591 SEE ALSO

6592 *it_post_atomic()*, *it_post_send()*, *it_post_send_and_unlink()*, *it_post_sendto()*,
6593 *it_post_recvfrom()*, *it_post_rdma_read()*, *it_post_rdma_read_to_rmr()*, *it_post_rdma_write()*,
6594 *it_dto_status_t*, *it_dto_events*, *it_dto_flags_t*, *it_ep_rc_create()*, *it_lmr_triplet_t*, *it_ia_query()*,
6595 *it_dto_cookie_t*, *it_ia_info_t*, *it_ep_attributes_t*, *it_ep_state_t*

it_post_recvfrom()

6596

6597 NAME

6598 `it_post_recvfrom` – post a Receive DTO to a datagram Endpoint

6599 SYNOPSIS

```
6600 #include <it_api.h>
6601
6602 it_status_t it_post_recvfrom(
6603     IN          it_ep_handle_t      ep_handle,
6604     IN  const  it_lmr_triplet_t    *local_segments,
6605     IN          size_t              num_segments,
6606     IN          it_dto_cookie_t     cookie,
6607     IN          it_dto_flags_t      dto_flags
6608 );
```

6609 APPLICABILITY

6610 `it_post_recvfrom` is applicable only to Endpoints created for the UD service type.

6611 DESCRIPTION

6612	<code>ep_handle</code>	Handle for the local datagram Endpoint.
6613	<code>local_segments</code>	Vector of <code>it_lmr_triplet_t</code> data structures that specifies the local buffer that will contain the received data. Local buffer must be at least 40 bytes.
6615	<code>num_segments</code>	Number of <code>it_lmr_triplet_t</code> data structures in <code>local_segments</code> . Must be at least one.
6617	<code>cookie</code>	Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the Receive.
6619	<code>dto_flags</code>	Flags for posted Receive.

6620 `it_post_recvfrom` posts a request to the `ep_handle` datagram Endpoint to deposit all incoming data corresponding to a single Send from a remote datagram Endpoint into the local Receive buffer (data sink) specified by `local_segments` and `num_segments`.

6623 The size of the Receive buffer is given by the sum of the segment lengths specified by `local_segments`. The first 40 bytes of the Consumer's local buffer are reserved for Implementation use. For a zero-sized message, the minimum size for the local buffer in `local_segments` is 40 bytes. To accommodate a larger message, the Consumer should provide a local buffer in `local_segments` at least 40 bytes bigger than their expected incoming message size. See `it_dto_events` for more details.

6629 An LMR used in `local_segments` (data sink) must have memory access privileges including local write access. The Direct LMR Handle may be used by Privileged Mode Consumers (see Asynchronous Errors for more details).

6632 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.

6635 The *cookie* (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work
6636 Request. This identifier is completely under Consumer control and opaque to the
6637 Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted
6638 Work Request.

6639 The *dto_flags* value is used as specified in *it_dto_flags_t*.

6640 A successful call returns IT_SUCCESS, which means that the Receive operation was
6641 successfully posted to the transport layer.

6642 The completion of the posted Receive is reported asynchronously to the Consumer according to
6643 the rules defined in *it_dto_flags_t*. Exactly one Receive Completion Event of type
6644 *it_all_dto_cmpl_event_t* is always generated and manifests on the EVD associated with the
6645 Endpoint Receive Queue (see *it_ep_ud_create*).

6646 If the reported *dto_status* of the Receive Completion Event (see *it_all_dto_cmpl_event_t*) is
6647 IT_DTO_SUCCESS (see *it_dto_status_t*), then the size of the received data can be retrieved
6648 through the *transferred_length* member of the Event. Once a successful Completion Event has
6649 been generated for the Receive, the order of the bytes in the local buffer specified by
6650 *local_segments* and *num_segments* corresponds to the order defined by the *local_segments* of the
6651 corresponding Send operation unless there is overlap among the segments of the local Receive
6652 buffer. If there is such an overlap, the content of the local buffer after the Completion Event has
6653 been generated is undefined. Prior to the Completion Event being generated, the content of the
6654 local buffer is Implementation-dependent. A successful Completion Event indicates that the data
6655 has been delivered uncorrupted into the local buffer.

6656 If the reported completion status is not IT_DTO_SUCCESS, the content of the local buffer is not
6657 defined.

6658 A Consumer should not modify the local buffer specified by *num_segments* and *local_segments*
6659 until the DTO is completed. When a Consumer does not adhere to this rule, the behavior of the
6660 Implementation and the underlying transport is not defined. A Consumer does get back the
6661 ownership of the array specified by the *local_segments* and *num_segments* arguments (but not
6662 the local buffer identified by this array) when *it_post_recvfrom* returns and is free to use this
6663 array for other calls, to modify it, or to destroy it.

6664 The Implementation ensures that each Receive corresponds to one and only one remote Send.

6665 There is no relationship guaranteed on the order of Receive completions and the order of the
6666 posting of the corresponding Sends at remote datagram Endpoints.

6667 The Implementation ensures that all Receives start and complete in the order posted.

6668 The Implementation ensures that all data from a given Send operation is transferred from the
6669 remote buffer and into the local buffer before a Receive completion is generated with
6670 IT_DTO_SUCCESS.

6671 **RETURN VALUE**

6672 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

6673	IT_ERR_INVALID_DTO_FLAGS	The Data Transfer Operation flags (<i>dto_flags</i>)
6674		value was invalid.
6675	IT_ERR_INVALID_EP	The Endpoint Handle (<i>ep_handle</i>) was invalid.

6676 6677	IT_ERR_INVALID_EP_TYPE	The attempted operation was invalid for the Service Type of the Endpoint.
6678 6679 6680	IT_ERR_INVALID_NUM_SEGMENTS	The requested number of segments (<i>num_segments</i>) was larger than the Endpoint supports.
6681 6682	IT_ERR_TOO_MANY_POSTS	The operation failed due to an overflow of a Work Queue.
6683 6684 6685 6686	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.

6687 **ASYNCHRONOUS ERRORS**

6688 Any posting to an Endpoint that is in the IT_EP_STATE_UD_NONOPERATIONAL state will
6689 be flushed with completion status (*it_dto_status_t*) set to IT_DTO_ERR_FLUSHED.

6690 An Affiliated Asynchronous Error that is associated with an Endpoint will move that Endpoint to
6691 the IT_EP_STATE_UD_NONOPERATIONAL state.

6692 For errors that can be reported as Completion Errors, see also [it_dto_status_t](#).

6693 If no Receive buffers are posted in time, then an incoming Send will fail to deliver its data. In
6694 order to avoid this error, the Consumer should post Receive resources prior to the remote
6695 Consumer posting a Send. The lack of a Receive buffer is handled as follows:

6696 For the IB transport, an IT_ASYNC_AFF_EP_L_RECV_ACCESS_VIOLATION Affiliated
6697 Asynchronous Error is generated.

6698 If the size of an incoming message is larger than the size of the local buffer or larger than the
6699 MTU of the local Spigot, a length error will occur. In order to avoid length errors, the Consumer
6700 should post a buffer large enough for the incoming Send data. Length errors are handled as
6701 follows:

6702 For the IB transport, the Receive DTO completes with an IT_DTO_ERR_LOCAL_LENGTH
6703 error status.

6704 If the Direct LMR Handle is used to specify a local buffer in a Work Request and the Endpoint
6705 to which the Work Request is posted is not Privileged, then a Completion Error with the DTO
6706 status IT_DTO_ERR_LOCAL_PROTECTION shall be returned.

6707 **APPLICATION USAGE**

6708 This function is used to transfer data into a Consumer-specified local buffer from a buffer
6709 specified by the corresponding Send operation at the remote Endpoint.

6710 The Consumer should use unique identifiers for *cookie* if they desire to identify each DTO. If the
6711 Consumer does not require a unique DTO identifier, the value of zero or NULL
6712 (IT_NO_ADDR) can be used.

6713 For best Receive operation performance, the Consumer should align each buffer segment of
6714 *local_segments* to the *dto_alignment_hint* in the IA attributes obtained via [it_ia_query](#).

6715 When an EVD holding the DTO Event Stream associated with an Endpoint overflows, on some
6716 transports the behavior of the Endpoint and/or the EVD is undefined, making recovery from

6717 such an overflow either impossible or at the very least IA-dependent. Consumers should
6718 therefore take care to avoid this situation. Each Work Request that a Consumer posts to a Work
6719 Queue can result in a Completion Event being enqueued in an EVD, due to the possibility of an
6720 error occurring during Work Request processing. To avoid any possibility of overflow, the
6721 Consumer must therefore ensure that the total number of Outstanding Operations on all Work
6722 Queues that can enqueue Completion Events into a given EVD never exceeds the minimum
6723 queue size (as reported by *it_evd_query*) of that EVD.

6724 **SEE ALSO**

6725 *it_post_atomic()*, *it_post_send()*, *it_post_send_and_unlink()*, *it_post_sendto()*, *it_post_recv()*,
6726 *it_post_rdma_read()*, *it_post_rdma_read_to_rmr()*, *it_post_rdma_write()*, *it_dto_status_t*,
6727 *it_dto_events*, *it_dto_flags_t*, *it_ep_ud_create()*, *it_lmr_triplet_t*, *it_ia_query()*, *it_dto_cookie_t*,
6728 *it_ia_info_t*

it_post_send()

6729

6730 NAME

6731 it_post_send – post a Send DTO to a connected Endpoint

6732 SYNOPSIS

```
6733 #include <it_api.h>
6734
6735 it_status_t it_post_send(
6736     IN          it_ep_handle_t      ep_handle,
6737     IN const    it_lmr_triplet_t    *local_segments,
6738     IN          size_t              num_segments,
6739     IN          it_dto_cookie_t     cookie,
6740     IN          it_dto_flags_t      dto_flags
6741 );
```

6742 APPLICABILITY

6743 *it_post_send* is applicable only to Endpoints created for the RC service type.

6744 DESCRIPTION

6745 *ep_handle* Handle for the Endpoint – the local side of the Connection.

6746 *local_segments* Vector of *it_lmr_triplet_t* data structures that specifies the local buffer that
6747 contains data to be transferred. Can be NULL (IT_NO_ADDR) for a zero-
6748 sized message.

6749 *num_segments* Number of *it_lmr_triplet_t* data structures in *local_segments*. Can be zero
6750 for a zero-sized message.

6751 *cookie* Consumer-provided cookie that is returned to the Consumer in the
6752 Completion Event corresponding to the Send.

6753 *dto_flags* Flags for posted Send.

6754 *it_post_send* posts a request to the *ep_handle* Endpoint to transfer over the reliable Connection
6755 of that Endpoint all the data from the local buffer (data source) specified by *local_segments* and
6756 *num_segments* into a remote buffer specified by a single corresponding Receive on the other side
6757 of the Connection. If *num_segments* is non-zero, then the size of the data transferred is given by
6758 the sum of the segment lengths specified by *local_segments*. A zero-sized message may be
6759 transferred over the Connection and will consume a buffer specified by the corresponding
6760 Receive.

6761 The Implementation ensures that an LMR Triplet supports byte alignment for Data Transfer
6762 Operations. The buffer segments described by *local_segments* can overlap; it is safe to use
6763 overlapping buffer segments as a data source.

6764 An LMR used in *local_segments* (data source) must have memory access privileges including
6765 local read access. The Direct LMR Handle may be used by Privileged Mode Consumers (see
6766 Asynchronous Errors for more details).

6767 The *cookie* (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work
6768 Request. This identifier is completely under Consumer control and opaque to the

6769 Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted
6770 Work Request.

6771 The *dto_flags* value is used as specified in *it_dto_flags_t*.

6772 A successful call returns IT_SUCCESS, which means that the Send operation was successfully
6773 posted to the transport layer.

6774 The completion of the posted Send is reported asynchronously to the Consumer according to the
6775 rules defined in *it_dto_flags_t*. A Send Completion Event is of type *it_dto_cmpl_event_t*. Any
6776 generated Send Completion Event manifests on the EVD associated with the Endpoint Send
6777 Queue. See *it_ep_rc_create*, *it_dto_status_t*, and *it_dto_events*. The Completion Event for the
6778 *it_post_send* call indicates to the Consumer that the local buffer is under Consumer control again
6779 and that the contents of the local buffer will be transported reliably, but does not guarantee that
6780 these contents have been successfully received by the remote Consumer; wherever necessary,
6781 this restriction is made explicit by designating a Completion as local or by stating that operations
6782 complete locally. See Section 5.4 of [DDP-IETF] for background information. The contents of
6783 the local buffer are only guaranteed to have reached the remote Consumer's memory when the
6784 remote Consumer reaps a successful completion for the Receive operation that matches the Send
6785 initiated by the *it_post_send* call. However, once the contents of the local buffer reach the
6786 remote Consumer memory, the order of the bytes in the remote buffer specified by the Receive
6787 operation at the remote Endpoint corresponds to the order defined by the Send side
6788 *local_segments*, subject to overlap constraints. See *it_post_recv* for details on overlap
6789 constraints.

6790 A Consumer should not modify the local buffer specified by *num_segments* and *local_segments*
6791 until the DTO is locally completed. When a Consumer does not adhere to this rule, the content
6792 of the local buffer at the receiving side is undefined after the matching Receive operation
6793 completes. A Consumer does get back the ownership of the *num_segments* and *local_segments*
6794 arguments (but not the local buffer identified by them) when *it_post_send* returns and is free to
6795 use the *num_segments* and *local_segments* arguments for other calls, to modify them, or to
6796 destroy them. A Consumer does get back the ownership of the array specified by the
6797 *local_segments* and *num_segments* arguments (but not the local buffer identified by this array)
6798 when *it_post_send* returns and is free to use this array for other calls, to modify it, or to destroy
6799 it.

6800 The Implementation ensures that each Send corresponds to one and only one remote Receive and
6801 in no way corresponds to any locally or remotely posted RDMA Read or RDMA Write Data
6802 Transfer Operations over the same Connection.

6803 The Implementation ensures that each RDMA Write DTO posted to an Endpoint prior to a Send
6804 DTO posted to the same Endpoint has its complete data payload delivered to the remote memory
6805 prior to the completion of the Receive DTO at the remote side matching the Send DTO.
6806 However, the Implementation does not ensure that the RDMA Write DTO places its entire data
6807 payload into its remote buffer before the Receive DTO places the data payload of the incoming
6808 Send into its Receive buffer; if the remote buffers overlap, their contents will be indeterminate.

6809 The Implementation ensures that subsequent Send DTOs posted to the same Endpoint start and
6810 complete locally in post order and that the Receive DTOs at the remote side matching the Send
6811 DTOs complete in the same order. However, the Implementation does not ensure that the
6812 matching Receive DTOs place the data payloads into their Receive buffers in the post order of
6813 the Send DTOs; if the Receive buffers overlap, their contents will be indeterminate.

6814 In general, Work Requests following an RDMA Read may start execution while the RDMA
6815 Read is in progress (but may not complete before the RDMA Read completes). To ensure
6816 deterministically that a Send DTO following an RDMA Read DTO starts after the RDMA Read
6817 completes, specify the `IT_BARRIER_FENCE_FLAG` on the Send DTO.

6818 Any Work Request posted to the Send Queue of an Endpoint (thus also a Send) after a call to
6819 `it_rmr_link` or `it_rmr_unlink` will not begin execution until the Link or Unlink operation has
6820 completed.

6821 EXTENDED DESCRIPTION

6822 On the InfiniBand Transport, the Implementation ensures that an RDMA Write DTO followed
6823 by a Send DTO posted to the same Endpoint cause data payloads to be placed in remote memory
6824 in post order; relying on such placement ordering represents a transport-dependent programming
6825 practice.

6826 RETURN VALUE

6827 A successful call returns `IT_SUCCESS`.

6828 Posting to an Endpoint that is not in the `IT_EP_STATE_CONNECTED` or `IT_EP_STATE_`
6829 `NONOPERATIONAL` state will return the `IT_ERR_INVALID_EP_STATE` immediate error.

6830 The possible immediate errors for `it_post_send` are listed below:

6831 `IT_ERR_INVALID.DTO_FLAGS` The Data Transfer Operation flags (`dto_flags`) value was
6832 invalid.

6833 `IT_ERR_INVALID_EP` The Endpoint Handle (`ep_handle`) was invalid.

6834 `IT_ERR_INVALID_EP_STATE` The Endpoint was not in the proper state for the attempted
6835 operation.

6836 `IT_ERR_INVALID_EP_TYPE` The attempted operation was invalid for the Service Type
6837 of the Endpoint.

6838 `IT_ERR_INVALID_NUM_SEGMENTS` The requested number of segments (`num_segments`) was
6839 larger than the Endpoint supports.

6840 `IT_ERR_TOO_MANY_POSTS` The operation failed due to an overflow of a Work Queue.

6841 `IT_ERR_INVALID_LMR` The `local_segments` contains the Direct LMR Handle on an
6842 IA that does not support Direct LMRs.

6843 `IT_ERR_IA_CATASTROPHE` The IA has experienced a catastrophic error and is in the
6844 disabled state. None of the output parameters from this
6845 routine are valid. See `it_ia_info_t` for a description of the
6846 disabled state.

6847 ASYNCHRONOUS ERRORS

6848 For Work Requests posted to an Endpoint in the `IT_EP_STATE_CONNECTED` state, a
6849 completion status (see `it_dto_status_t`) other than `IT.DTO_SUCCESS` will break the Connection
6850 by moving the Endpoint to the `IT_EP_STATE_NONOPERATIONAL` state and deliver an
6851 `IT_CM_MSG_CONN_BROKEN_EVENT` Event to the Connect EVD of `ep_handle`. Once the
6852 Connection is broken, all outstanding and in-progress operations on the Connection will
6853 complete with an error status.

6854 An Affiliated Asynchronous Error that is associated with an Endpoint will move that Endpoint to
6855 the `IT_EP_STATE_NONOPERATIONAL` state; if this state transition was from the
6856 `IT_EP_STATE_CONNECTED` state, an `IT_CM_MSG_CONN_BROKEN_EVENT` Event will
6857 be delivered to the Connect EVD of *ep_handle*.

6858 Any posting to an Endpoint that is in the `IT_EP_STATE_NONOPERATIONAL` state will be
6859 flushed with completion status set to `IT_DTO_ERR_FLUSHED`.

6860 For locally or remotely detected errors that can be reported as Completion Errors, see also
6861 *it_dto_status_t*.

6862 The handling of remotely detected errors is transport and Implementation-dependent; end-to-end
6863 completions are not supported for certain transports or DTOs.

6864 If no Receive buffers are posted in time at the remote end, then a Send will fail to successfully
6865 deliver the contents of the local buffer. In order to avoid this error, the remote Consumer should
6866 post Receive resources prior to the local Consumer posting the Send. The lack of a remote
6867 Receive buffer is handled as follows:

6868 For IB and iWARP, an `IT_ASYNC_AFF_EP_L_RECV_ACCESS_VIOLATION` Affiliated
6869 Asynchronous Error will surface remotely.

6870 For the IB transport, the Send DTO completes with an `IT_DTO_ERR_REMOTE_RESPONDER`
6871 error status.

6872 For the iWARP Transport, an `IT_ASYNC_AFF_EP_R_RECV_ACCESS_VIOLATION`
6873 Affiliated Asynchronous Error will surface after the Send DTO has already completed with
6874 `IT_DTO_SUCCESS`.

6875 If the Receive buffer of the Receive DTO at the remote side matching the Send DTO is not
6876 sufficient in size for the data payload of the Send DTO, a length error will occur. In order to
6877 avoid length errors, the remote Consumer should post a buffer large enough for the incoming
6878 Send data. Remotely detected length errors are handled as follows:

6879 For both the IB and iWARP Transports, the Receive DTO at the remote side completes with an
6880 `IT_DTO_ERR_LOCAL_LENGTH` error status.

6881 For the IB transport, the Send DTO completes with an `IT_DTO_ERR_REMOTE_RESPONDER`
6882 error status.

6883 For the iWARP Transport, an `IT_ASYNC_AFF_EP_R_RECV_LENGTH_ERROR` Affiliated
6884 Asynchronous Error will surface after the Send DTO has already completed with
6885 `IT_DTO_SUCCESS`.

6886 Despite using the RC service, a Send operation may fail to successfully deliver the contents of
6887 the local buffer. A failure such as an unfinished data delivery or a CRC error that cannot be
6888 corrected by the transport is detected as a transport error and manifests as follows:

6889 For the IB transport, a transport error causes the Send DTO and possibly the Receive DTO
6890 matching the Send DTO to complete with an `IT_DTO_ERR_TRANSPORT` Completion Error.

6891 For the iWARP Transport, a transport error causes the Receive DTO matching the Send DTO to
6892 complete with an `IT_DTO_ERR_TRANSPORT` Completion Error and an
6893 `IT_ASYNC_AFF_EP_R_ERROR` Affiliated Asynchronous Error to surface locally after the
6894 Send DTO has already completed with `IT_DTO_SUCCESS`.

6895 If a type of remotely detected error can manifest locally in different ways – i.e., as an Affiliated
6896 Asynchronous Error or as a Completion Error – transport-independent programming requires
6897 Consumers to be prepared to deal with either.

6898 If the Direct LMR Handle is used to specify a local buffer in a Work Request and the Endpoint
6899 to which the Work Request is posted is not Privileged, then a Completion Error with the DTO
6900 status `IT_DTO_ERR_LOCAL_PROTECTION` shall be returned.

6901 **APPLICATION USAGE**

6902 This function is used after a Connection has been established to transfer data from a Consumer-
6903 specified local buffer to a buffer specified by the corresponding Receive operation on the other
6904 side of the Connection.

6905 The Consumer should use unique identifiers for *cookie* if they desire to identify each DTO. If the
6906 Consumer does not require a unique DTO identifier, the value of zero or NULL
6907 (`IT_NO_ADDR`) can be used.

6908 For best Send operation performance, the Consumer should align each buffer segment of
6909 *local_segments* to the *dto_alignment_hint* in the IA attributes obtained via *it_ia_query*.

6910 When an EVD holding the DTO Event Stream associated with an Endpoint overflows, on some
6911 transports the behavior of the Endpoint and/or the EVD is undefined, making recovery from
6912 such an overflow either impossible or at the very least IA-dependent. Consumers should
6913 therefore take care to avoid this situation. Each Work Request that a Consumer posts to a Work
6914 Queue can result in a Completion Event being enqueued in an EVD, due to the possibility of an
6915 error occurring during Work Request processing. To avoid any possibility of overflow, the
6916 Consumer must therefore ensure that the total number of Outstanding Operations on all Work
6917 Queues that can enqueue Completion Events into a given EVD never exceeds the minimum
6918 queue size (as reported by *it_evd_query*) of that EVD.

6919 **SEE ALSO**

6920 *it_post_atomic()*, *it_post_send_and_unlink()*, *it_post_sendto()*, *it_post_recv()*,
6921 *it_post_recvfrom()*, *it_post_rdma_read()*, *it_post_rdma_read_to_rmr()*, *it_post_rdma_write()*,
6922 *it_dto_status_t*, *it_dto_events*, *it_dto_flags_t*, *it_ep_rc_create()*, *it_lmr_triplet_t*, *it_ia_query()*,
6923 *it_dto_cookie_t*, *it_ia_info_t*

it_post_send_and_unlink()

6924
6925 **NAME**
6926 `it_post_send_and_unlink` – post a Send and Unlink DTO to a connected Endpoint

6927 **SYNOPSIS**
6928

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


6929
6930

```
it_status_t it_post_send_and_unlink(  
6931     IN          it_ep_handle_t      ep_handle,  
6932     IN  const   it_lmr_triplet_t    *local_segments,  
6933     IN          size_t              num_segments,  
6934     IN          it_dto_cookie_t     cookie,  
6935     IN          it_dto_flags_t     dto_flags,  
6936     IN          it_rmr_context_t    rmr_context  
6937 ) ;
```

6938 **APPLICABILITY**
6939 `it_post_send_and_unlink` is applicable only to Endpoints created for the RC service type and is
6940 supported only if the Interface Adapter attribute `post_send_unlink_remote_support` (see
6941 [it_ia_info_t](#)) is IT_TRUE.

6942 **DESCRIPTION**

6943	<code>ep_handle</code>	Handle for the Endpoint – the local side of the Connection.
6944	<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 (IT_NO_ADDR) for a zero-sized message.
6945		
6946		
6947	<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.
6948		
6949	<code>cookie</code>	Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the Send and Unlink operation.
6950		
6951	<code>dto_flags</code>	Flags for posted Send and Unlink operation.
6952	<code>rmr_context</code>	RMR Context to be unlinked at Remote.

6953 `it_post_send_and_unlink` posts a request to the `ep_handle` Endpoint to transfer over the reliable
6954 Connection of that Endpoint all the data from the local buffer (data source) specified by
6955 `local_segments` and `num_segments` into a remote buffer specified by a single corresponding
6956 Receive on the other side of the Connection. In addition, on successful transmission to the
6957 remote, the remote buffer registered remotely using the RMR Context `rmr_context` is unlinked.

6958 If `num_segments` is non-zero, then the size of the data transferred is given by the sum of the
6959 segment lengths specified by `local_segments`. A zero-sized message may be transferred over the
6960 Connection and will consume a buffer specified by the corresponding Receive.

6961 The Implementation ensures that an LMR Triplet supports byte alignment for Data Transfer
6962 Operations. The buffer segments described by `local_segments` can overlap; it is safe to use
6963 overlapping buffer segments as a data source.

6964 An LMR used in *local_segments* (data source) must have memory access privileges including
6965 local read access. The Direct LMR Handle may be used by Privileged Mode Consumers (see
6966 Asynchronous Errors for more details).

6967 The *cookie* (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work
6968 Request. This identifier is completely under Consumer control and opaque to the
6969 Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted
6970 Work Request.

6971 The *dto_flags* value is used as specified in *it_dto_flags_t*.

6972 The *rmr_context* is used to specify an LMR or RMR at the remote that, if the Send portion of
6973 this operation completes successfully, will be unlinked after the Send data has been placed at the
6974 remote.

6975 A successful call returns IT_SUCCESS, which means that the Send operation was successfully
6976 posted to the transport layer.

6977 The completion of the posted Send is reported asynchronously to the Consumer according to the
6978 rules defined in *it_dto_flags_t*. A Send Completion Event is of type *it_dto_cmpl_event_t*. Any
6979 generated Send Completion Event manifests on the EVD associated with the Endpoint Send
6980 Queue. See *it_ep_rc_create*, *it_dto_status_t*, and *it_dto_events*. The Completion Event for the
6981 *it_post_send_and_unlink* call indicates to the Consumer that the local buffer is under Consumer
6982 control again and that the contents of the local buffer will be transported reliably, but does not
6983 guarantee that these contents have been successfully received by the remote Consumer;
6984 wherever necessary, this restriction is made explicit by designating a Completion as local or by
6985 stating that operations complete locally. See Section 5.4 of [DDP-IETF] for background
6986 information. The contents of the local buffer are only guaranteed to have reached the remote
6987 Consumer's memory when the remote Consumer reaps a successful completion for the Receive
6988 operation that matches the Send initiated by the *it_post_send_and_unlink* call. However, once
6989 the contents of the local buffer reach the remote Consumer memory, the order of the bytes in the
6990 remote buffer specified by the Receive operation at the remote Endpoint corresponds to the order
6991 defined by the Send side *local_segments*, subject to overlap constraints. See *it_post_recv* for
6992 details on overlap constraints.

6993 A Consumer should not modify the local buffer specified by *num_segments* and *local_segments*
6994 until the DTO is locally completed. When a Consumer does not adhere to this rule, the content
6995 of the local buffer at the receiving side is undefined after the matching Receive operation
6996 completes. A Consumer does get back the ownership of the *num_segments* and *local_segments*
6997 arguments (but not the local buffer identified by them) when *it_post_send_and_unlink* returns
6998 and is free to use the *num_segments* and *local_segments* arguments for other calls, to modify
6999 them, or to destroy them. A Consumer does get back the ownership of the array specified by the
7000 *local_segments* and *num_segments* arguments (but not the local buffer identified by this array)
7001 when *it_post_send_and_unlink* returns and is free to use this array for other calls, to modify it, or
7002 to destroy it.

7003 The Implementation ensures that each Send corresponds to one and only one remote Receive and
7004 in no way corresponds to any locally or remotely posted RDMA Read or RDMA Write Data
7005 Transfer Operations over the same Connection.

7006 The Implementation ensures that each RDMA Write DTO posted to an Endpoint prior to a Send
7007 DTO posted to the same Endpoint has its complete data payload delivered to the remote memory
7008 prior to the completion of the Receive DTO at the remote side matching the Send DTO.

7009 However, the Implementation does not ensure that the RDMA Write DTO places its entire data
7010 payload into its remote buffer before the Receive DTO places the data payload of the incoming
7011 Send into its Receive buffer; if the remote buffers overlap, their contents will be indeterminate.

7012 The Implementation ensures that subsequent Send DTOs posted to the same Endpoint start and
7013 complete locally in post order and that the Receive DTOs at the remote side matching the Send
7014 DTOs complete in the same order. However, the Implementation does not ensure that the
7015 matching Receive DTOs place the data payloads into their Receive buffers in the post order of
7016 the Send DTOs; if the Receive buffers overlap, their contents will be indeterminate.

7017 In general, Work Requests following an RDMA Read may start execution while the RDMA
7018 Read is in progress (but may not complete before the RDMA Read completes). To ensure
7019 deterministically that a Send DTO following an RDMA Read DTO starts after the RDMA Read
7020 completes, specify the `IT_BARRIER_FENCE_FLAG` on the Send DTO.

7021 Any Work Request posted to the Send Queue of an Endpoint (thus also a Send) after a call to
7022 *it_rmr_link* or *it_rmr_unlink* will not begin execution until the Link or Unlink operation has
7023 completed.

7024 EXTENDED DESCRIPTION

7025 On the InfiniBand Transport, the Implementation ensures that an RDMA Write DTO followed
7026 by a Send DTO posted to the same Endpoint cause data payloads to be placed in remote memory
7027 in post order; relying on such placement ordering represents a transport-dependent programming
7028 practice.

7029 RETURN VALUE

7030 A successful call returns `IT_SUCCESS`.

7031 Posting to an Endpoint that is not in the `IT_EP_STATE_CONNECTED` or `IT_EP_STATE_`
7032 `NONOPERATIONAL` state will return the `IT_ERR_INVALID_EP_STATE` immediate error.

7033 The possible immediate errors for *it_post_send_and_unlink* are listed below:

7034 `IT_ERR_INVALID_DTO_FLAGS` The Data Transfer Operation flags (*dto_flags*) value was
7035 invalid.

7036 `IT_ERR_INVALID_EP` The Endpoint Handle (*ep_handle*) was invalid.

7037 `IT_ERR_INVALID_EP_STATE` The Endpoint was not in the proper state for the attempted
7038 operation.

7039 `IT_ERR_INVALID_EP_TYPE` The attempted operation was invalid for the Service Type
7040 of the Endpoint.

7041 `IT_ERR_INVALID_NUM_SEGMENTS` The requested number of segments (*num_segments*) was
7042 larger than the Endpoint supports.

7043 `IT_ERR_TOO_MANY_POSTS` The operation failed due to an overflow of a Work Queue.

7044 `IT_ERR_INVALID_LMR` The *local_segments* contains the Direct LMR Handle on an
7045 IA that does not support Direct LMRs.

7046 `IT_ERR_IA_CATASTROPHE` The IA has experienced a catastrophic error and is in the
7047 disabled state. None of the output parameters from this

7048 routine are valid. See [it_ia_info_t](#) for a description of the
7049 disabled state.

7050 **ASYNCHRONOUS ERRORS**

7051 For Work Requests posted to an Endpoint in the IT_EP_STATE_CONNECTED state, a
7052 completion status (see [it_dto_status_t](#)) other than IT_DTO_SUCCESS will break the Connection
7053 by moving the Endpoint to the IT_EP_STATE_NONOPERATIONAL state and deliver an
7054 IT_CM_MSG_CONN_BROKEN_EVENT Event to the Connect EVD of *ep_handle*. Once the
7055 Connection is broken, all outstanding and in-progress operations on the Connection will
7056 complete with an error status.

7057 An Affiliated Asynchronous Error that is associated with an Endpoint will move that Endpoint to
7058 the IT_EP_STATE_NONOPERATIONAL state; if this state transition was from the
7059 IT_EP_STATE_CONNECTED state, an IT_CM_MSG_CONN_BROKEN_EVENT Event will
7060 be delivered to the Connect EVD of *ep_handle*.

7061 Any posting to an Endpoint that is in the IT_EP_STATE_NONOPERATIONAL state will be
7062 flushed with completion status set to IT_DTO_ERR_FLUSHED.

7063 For locally or remotely detected errors that can be reported as Completion Errors, see also
7064 [it_dto_status_t](#).

7065 The handling of remotely detected errors is transport and Implementation-dependent; end-to-end
7066 completions are not supported for certain transports or DTOs.

7067 If no Receive buffers are posted in time at the remote end, then a Send will fail to successfully
7068 deliver the contents of the local buffer. In order to avoid this error, the remote Consumer should
7069 post Receive resources prior to the local Consumer posting the Send. The lack of a remote
7070 Receive buffer is handled as follows:

7071 For IB and iWARP, an IT_ASYNC_AFF_EP_L_RECV_ACCESS_VIOLATION Affiliated
7072 Asynchronous Error will surface remotely.

7073 For the IB transport, the Send DTO completes with an IT_DTO_ERR_REMOTE_RESPONDER
7074 error status.

7075 For the iWARP Transport, an IT_ASYNC_AFF_EP_R_RECV_ACCESS_VIOLATION
7076 Affiliated Asynchronous Error will surface after the Send DTO has already completed with
7077 IT_DTO_SUCCESS.

7078 If the Receive buffer of the Receive DTO at the remote side matching the Send DTO is not
7079 sufficient in size for the data payload of the Send DTO, a length error will occur. In order to
7080 avoid length errors, the remote Consumer should post a buffer large enough for the incoming
7081 Send data. Remotely detected length errors are handled as follows:

7082 For both the IB and iWARP Transports, the Receive DTO at the remote side completes with an
7083 IT_DTO_ERR_LOCAL_LENGTH error status.

7084 For the IB transport, the Send DTO completes with an IT_DTO_ERR_REMOTE_RESPONDER
7085 error status.

7086 For the iWARP Transport, an IT_ASYNC_AFF_EP_R_RECV_LENGTH_ERROR Affiliated
7087 Asynchronous Error will surface after the Send DTO has already completed with
7088 IT_DTO_SUCCESS.

7089 Despite using the RC service, a Send operation may fail to successfully deliver the contents of
7090 the local buffer. A failure such as an unfinished data delivery or a CRC error that cannot be
7091 corrected by the transport is detected as a transport error and manifests as follows:

7092 For the IB transport, a transport error causes the Send DTO and possibly the Receive DTO
7093 matching the Send DTO to complete with an IT_DTO_ERR_TRANSPORT Completion Error.

7094 For the iWARP Transport, a transport error causes the Receive DTO matching the Send DTO to
7095 complete with an IT_DTO_ERR_TRANSPORT Completion Error and an
7096 IT_ASYNC_AFF_EP_R_ERROR Affiliated Asynchronous Error to surface locally after the
7097 Send DTO has already completed with IT_DTO_SUCCESS.

7098 If a type of remotely detected error can manifest locally in different ways – i.e., as an Affiliated
7099 Asynchronous Error or as a Completion Error – transport-independent programming requires
7100 Consumers to be prepared to deal with either.

7101 If the Direct LMR Handle is used to specify a local buffer in a Work Request and the Endpoint
7102 to which the Work Request is posted is not Privileged, then a Completion Error with the DTO
7103 status IT_DTO_ERR_LOCAL_PROTECTION shall be returned.

7104 Unlinking a remote RMR Context (RMR/LMR) through *it_post_send_and_unlink* may result in
7105 a remotely detected error if the remote RMR Context is invalid or corresponds to the remote
7106 Direct LMR Handle, or if the remote PZ of the remote RMR Context does not match the remote
7107 PZ of the remote EP, or if the remote RMR is a Wide RMR, or if the remote RMR is a linked
7108 Narrow RMR associated with an Endpoint other than the remote Endpoint of the connection, or
7109 if the remote RMR is a narrow RMR in another PZ than that of the remote Endpoint of the
7110 connection, or if the RMR Context corresponds to a remote LMR that has bound RMRs or the
7111 LMR is shared or the LMR does not have remote access or the LMR cannot be unlinked due to
7112 the way it was linked (IB-only) or if the LMR was already in the unlinked state. Such a
7113 remotely detected error manifests remotely by returning a Completion Error for the matching
7114 Receive and by terminating the connection. The remote RMR/LMR remains linked. Such
7115 remotely detected errors manifest locally in a transport-dependent manner – for InfiniBand,
7116 subsequent Work Requests to the same Endpoint will fail with an
7117 IT_DTO_ERR_REMOTE_RESPONDER Completion Error; for iWARP, the connection will be
7118 torn down abortively and no Completion Error will be generated.

7119 APPLICATION USAGE

7120 This function is used after a Connection has been established to transfer data from a Consumer-
7121 specified local buffer to a buffer specified by the corresponding Receive operation on the other
7122 side of the Connection. In addition, this function supplies an RMR Context that is to be unlinked
7123 at the remote – the remote Consumer should examine the RMR Context found in the
7124 corresponding Receive Completion Event to determine if one of their local Memory Regions
7125 (LMR or RMR) has been unlinked.

7126 The Consumer should use unique identifiers for *cookie* if they desire to identify each DTO. If the
7127 Consumer does not require a unique DTO identifier, the value of zero or NULL
7128 (IT_NO_ADDR) can be used.

7129 For best Send operation performance, the Consumer should align each buffer segment of
7130 *local_segments* to the *dto_alignment_hint* in the IA attributes obtained via *it_ia_query*.

7131 When an EVD holding the DTO Event Stream associated with an Endpoint overflows, on some
7132 transports the behavior of the Endpoint and/or the EVD is undefined, making recovery from

7133 such an overflow either impossible or at the very least IA-dependent. Consumers should
7134 therefore take care to avoid this situation. Each Work Request that a Consumer posts to a Work
7135 Queue can result in a Completion Event being enqueued in an EVD, due to the possibility of an
7136 error occurring during Work Request processing. To avoid any possibility of overflow, the
7137 Consumer must therefore ensure that the total number of Outstanding Operations on all Work
7138 Queues that can enqueue Completion Events into a given EVD never exceeds the minimum
7139 queue size (as reported by *it_evd_query*) of that EVD.

7140 **SEE ALSO**

7141 *it_post_atomic()*, *it_post_send()*, *it_post_sendto()*, *it_post_recv()*, *it_post_recvfrom()*,
7142 *it_post_rdma_read()*, *it_post_rdma_read_to_rmr()*, *it_post_rdma_write()*, *it_dto_status_t*,
7143 *it_dto_events*, *it_dto_flags_t*, *it_ep_rc_create()*, *it_lmr_triplet_t*, *it_ia_query()*, *it_dto_cookie_t*,
7144 *it_ia_info_t*

it_post_sendto()

7145

7146 NAME

7147

it_post_sendto – post a Send DTO to a datagram Endpoint

7148 SYNOPSIS

7149

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

7150

```
it_status_t it_post_sendto(  
    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,  
    IN const    it_dg_remote_ep_addr_t *remote_ep_addr  
);
```

7158

7159 APPLICABILITY

7160

it_post_sendto is applicable only to Endpoints created for the UD service type.

7161 DESCRIPTION

7162

ep_handle Handle for the local datagram Endpoint.

7163

local_segments Vector of *it_lmr_triplet_t* that specifies the local buffer that contains data to be transferred. Can be NULL (IT_NO_ADDR) for a zero-sized message.

7164

7165

num_segments Number of *it_lmr_triplet_t* data structures in *local_segments*. Can be zero for a zero-sized message.

7166

7167

cookie Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the Send.

7168

7169

dto_flags Flags for posted Send.

7170

remote_ep_addr Remote datagram Endpoint address.

7171

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.

7172

7173

7174

7175

7176

7177

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.

7178

7179

7180

An LMR used in *local_segments* (data source) must have memory access privileges including local read access. The Direct LMR Handle may be used by Privileged Mode Consumers (see Asynchronous Errors for more details).

7181

7182

7183

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

7184

7185 Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted
7186 Work Request.

7187 The *dto_flags* value is used as specified in *it_dto_flags_t*.

7188 *remote_ep_addr* specifies the Destination for the *it_post_sendto* operation. See
7189 *it_dg_remote_ep_addr_t* for details on the format of this data structure.

7190 A successful call returns IT_SUCCESS, which means that the Send operation was successfully
7191 posted to the transport layer.

7192 The completion of the posted Send is reported asynchronously to the Consumer according to the
7193 rules defined in *it_dto_flags_t*. A Send Completion Event is of type *it_all_dto_cmpl_event_t*.
7194 Any generated Send Completion Event manifests on the EVD associated with the Endpoint Send
7195 Queue. See *it_ep_ud_create*, *it_dto_status_t* and *it_dto_events*.

7196 Once a successful Completion Event has been generated at the receiver, the order of the bytes in
7197 the remote buffer specified by the Receive operation at the remote Endpoint corresponds to the
7198 order defined by the Send side *local_segments* subject to overlap constraints. See
7199 *it_post_recvfrom* for details on overlap constraints.

7200 A Consumer should not modify the local buffer specified by *num_segments* and *local_segments*
7201 until the DTO is completed. When a Consumer does not adhere to this rule, the behavior of the
7202 Implementation and the underlying transport is not defined. A Consumer does get back the
7203 ownership of the array specified by the *local_segments* and *num_segments* arguments (but not
7204 the local buffer identified by this array) when *it_post_sendto* returns and is free to use this array
7205 for other calls, to modify it, or to destroy it.

7206 The Implementation ensures that all Sends start and complete in the order posted.

7207 The Implementation makes no delivery order guarantees for Unreliable Datagrams.

7208 There is no delivery order or completion order between Receive Data Transfer Operations on
7209 different Destinations that correspond to the Sends posted in order to the same Unreliable
7210 Datagram Endpoint.

7211 When *it_post_sendto* completes with IT_DTO_SUCCESS or IT_DTO_ERR_LOCAL_EP there
7212 is no guarantee that the DTO has reached the remote Endpoint.

7213 **RETURN VALUE**

7214 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

7215	IT_ERR_INVALID_AH	The Address Handle within <i>remote_ep_addr</i> was
7216		invalid or the does not match the <i>spigot_id</i> of the
7217		Endpoint.
7218	IT_ERR_INVALID_DTO_FLAGS	The Data Transfer Operation flags (<i>dto_flags</i>) value
7219		was invalid.
7220	IT_ERR_INVALID_EP	The Endpoint Handle (<i>ep_handle</i>) was invalid.
7221	IT_ERR_INVALID_EP_TYPE	The attempted operation was invalid for the Service
7222		Type of the Endpoint.
7223	IT_ERR_INVALID_NUM_SEGMENTS	The requested number of segments (<i>num_segments</i>)
7224		was larger than the Endpoint supports.

7225	IT_ERR_TOO_MANY_POSTS	The operation failed due to an overflow of a Work Queue.
7226		
7227	IT_ERR_INVALID_LMR	The <i>local_segments</i> contains the Direct LMR Handle on an IA that does not support Direct LMRs.
7228		
7229	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.
7230		
7231		
7232		

7233 **ASYNCHRONOUS ERRORS**

7234 Any posting to an Endpoint that is in the IT_EP_STATE_UD_NONOPERATIONAL state will
7235 be flushed with completion status (*it_dto_status_t*) set to IT_DTO_ERR_FLUSHED.

7236 An Affiliated Asynchronous Error that is associated with an Endpoint will move that Endpoint to
7237 the IT_EP_STATE_UD_NONOPERATIONAL state.

7238 For errors that can be reported as Completion Errors, see also *it_dto_status_t*.

7239 If no Receive buffers are posted in time at the remote end, then a Send will fail to successfully
7240 deliver the contents of the local buffer. In order to avoid this error, the remote Consumer should
7241 post Receive resources prior to the local Consumer posting the Send. The lack of a remote
7242 Receive buffer is handled as follows:

7243 For the IB transport, an IT_ASYNC_AFF_EP_L_RECV_ACCESS_VIOLATION Affiliated
7244 Asynchronous Error will surface remotely.

7245 If the Receive buffer of the Receive DTO at the remote side matching the Send DTO is not
7246 sufficient in size for the data payload of the Send DTO, a length error will occur. In order to
7247 avoid length errors, the remote Consumer should post a buffer large enough for the incoming
7248 Send data. Remotely detected length errors are handled as follows:

7249 For the IB transport, the Receive DTO at the remote side completes with an
7250 IT_DTO_ERR_LOCAL_LENGTH error status.

7251 A completion status IT_DTO_ERR_LOCAL_EP may be returned in case of an Address Handle
7252 inconsistency (see Application Usage).

7253 If the Direct LMR Handle is used to specify a local buffer in a Work Request and the Endpoint
7254 to which the Work Request is posted is not Privileged, then a Completion Error with the DTO
7255 status IT_DTO_ERR_LOCAL_PROTECTION shall be returned.

7256 **APPLICATION USAGE**

7257 This function is used to transfer data from a Consumer-specified local buffer to a buffer
7258 specified by the corresponding Receive operation at a remote datagram Endpoint.

7259 The Consumer should use unique identifiers for *cookie* if they desire to identify each DTO. If the
7260 Consumer does not require a unique DTO identifier, the value of zero or NULL
7261 (IT_NO_ADDR) can be used.

7262 For best Send operation performance, the Consumer should align each buffer segment of
7263 *local_segments* to the *dto_alignment_hint* in the IA attributes obtained via *it_ia_query*.

7264 An Address Handle corresponds to a specific Spigot on an IA. Attempting to *it_post_sendto* on
7265 an Endpoint using an Address Handle that does not correspond to the Spigot associated with the

7266 Endpoint must be avoided by the Consumer. If the Consumer persists in this practice, they must
7267 write error handling code to deal with three possible error cases: One – the *it_post_sendto* call
7268 will return the *IT_ERR_INVALID_AH* error immediately. Or two – the DTO will complete in
7269 error with the *it_dto_status* set to *IT_DTO_ERR_LOCAL_EP*. Or three – there will be no
7270 indication of error. The three possible cases represent the allowable implementations of the
7271 underlying technology.

7272 When an EVD holding the DTO Event Stream associated with an Endpoint overflows, on some
7273 transports the behavior of the Endpoint and/or the EVD is undefined, making recovery from
7274 such an overflow either impossible or at the very least IA-dependent. Consumers should
7275 therefore take care to avoid this situation. Each Work Request that a Consumer posts to a Work
7276 Queue can result in a Completion Event being enqueued in an EVD, due to the possibility of an
7277 error occurring during Work Request processing. To avoid any possibility of overflow, the
7278 Consumer must therefore ensure that the total number of Outstanding Operations on all Work
7279 Queues that can enqueue Completion Events into a given EVD never exceeds the minimum
7280 queue size (as reported by *it_evd_query*) of that EVD.

7281 **SEE ALSO**

7282 *it_post_atomic()*, *it_post_send()*, *it_post_send_and_unlink()*, *it_post_rcv()*, *it_post_rcvfrom()*,
7283 *it_post_rdma_read()*, *it_post_rdma_read_to_rmr()*, *it_post_rdma_write()*, *it_dto_status_t*,
7284 *it_dto_events*, *it_dg_remote_ep_addr_t*, *it_dto_flags_t*, *it_ep_ud_create()*, *it_lmr_triplet_t*,
7285 *it_ia_query()*, *it_dto_cookie_t*, *it_ia_info_t*

it_pz_create()

7286

7287 NAME

7288 `it_pz_create` – create a new Protection Zone

7289 SYNOPSIS

```
7290 #include <it_api.h>
7291
7292 it_status_t it_pz_create(
7293     IN  it_ia_handle_t  ia_handle,
7294     OUT it_pz_handle_t  *pz_handle
7295 );
```

7296 DESCRIPTION

7297 *ia_handle* Interface Adapter on which the Protection Zone will be created.

7298 *pz_handle* Handle of new Protection Zone

7299 The *it_pz_create* routine creates a new Protection Zone that may be used to create Local
7300 Memory Regions, Remote Memory Regions, transport Endpoints, or Address Handles on the
7301 Interface Adapter identified by *ia_handle*. The Protection Zone is returned in *pz_handle*.

7302 RETURN VALUE

7303 A successful call returns `IT_SUCCESS`. Otherwise, returns an error code as described below:

7304 `IT_ERR_RESOURCES` The requested operation failed due to insufficient resources.

7305 `IT_ERR_INVALID_IA` The Interface Adapter Handle (*ia_handle*) was invalid.

7306 `IT_ERR_IA_CATASTROPHE` The IA has experienced a catastrophic error and is in the disabled
7307 state. None of the output parameters from this routine are valid.
7308 See *it_ia_info_t* for a description of the disabled state.

7309 APPLICATION USAGE

7310 An LMR, RMR, Endpoint, or Address Handle cannot be created without supplying a Protection
7311 Zone. An LMR, RMR, or Address Handle may only be used in concert with an Endpoint having
7312 the same Protection Zone. In DTO, Link, and Unlink operations, the Protection Zone of the local
7313 Endpoint and the LMR must match, or the operation will fail. In RDMA operations, the
7314 Protection Zone of the RMR associated with the RMR Context must match that of the remote
7315 Endpoint. In datagram DTO operations, the Protection Zone of the local Address Handle
7316 identifying the Destination must match that of the local Endpoint. In Link and Unlink operations,
7317 the Protection Zone of the LMR and RMR must match.

7318 SEE ALSO

7319 *it_pz_free()*, *it_pz_query()*

it_pz_free()

7320

7321 NAME

7322 `it_pz_free` – destroy a Protection Zone

7323 SYNOPSIS

```
7324 #include <it_api.h>
7325
7326 it_status_t it_pz_free(
7327     IN it_pz_handle_t pz_handle
7328 );
```

7329 DESCRIPTION

7330 *pz_handle* Handle of Protection Zone to be destroyed.

7331 The *it_pz_free* routine destroys the Protection Zone *pz_handle*. On successful return, the
7332 *pz_handle* may no longer be used. An attempt to free a Protection Zone that is still referenced by
7333 undestroyed Endpoints, Local Memory Regions, Remote Memory Regions, or Address Handles
7334 will fail with IT_ERR_PZ_BUSY, and the Protection Zone will be unaffected.

7335 RETURN VALUE

7336 A successful call returns IT_SUCCESS. Otherwise, returns an error code as described below:

7337 IT_ERR_INVALID_PZ The Protection Zone Handle (*pz_handle*) was invalid.

7338 IT_ERR_PZ_BUSY The Protection Zone was still in use.

7339 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the disabled
7340 state. None of the output parameters from this routine are valid.
7341 See *it_ia_info_t* for a description of the disabled state.

7342 SEE ALSO

7343 [*it_pz_create\(\)*](#), [*it_pz_query\(\)*](#)

it_pz_query()

7344

7345 NAME

7346 `it_pz_query` – get attributes of a Protection Zone

7347 SYNOPSIS

```
7348 #include <it_api.h>
7349
7350 it_status_t it_pz_query(
7351     IN    it_pz_handle_t    pz_handle,
7352     IN    it_pz_param_mask_t mask,
7353     OUT   it_pz_param_t     *params
7354 );
7355
7356 typedef enum {
7357     IT_PZ_PARAM_ALL = 0x01,
7358     IT_PZ_PARAM_IA  = 0x02
7359 } it_pz_param_mask_t;
7360
7361 typedef struct {
7362     it_ia_handle_t ia; /* IT_PZ_PARAM_IA */
7363 } it_pz_param_t;
```

7364 DESCRIPTION

7365 *pz_handle* Protection Zone.

7366 *mask* Bitwise OR of flags for desired parameters.

7367 *params* Structure whose members are written with the desired parameters.

7368 The *it_pz_query* routine returns the desired parameters of the Protection Zone *pz_handle* in the
7369 structure pointed to by *params*. On return, each field of *params* is only valid if the corresponding
7370 flag as shown in the Synopsis is set in the mask argument. The mask value
7371 IT_PZ_PARAM_ALL causes all fields to be returned.

7372 The definition of each field of *params* follows:

7373 *ia* The Interface Adapter Handle specified to create the Protection Zone.

7374 RETURN VALUE

7375 A successful call returns IT_SUCCESS. Otherwise, returns an error code as described below:

7376 IT_ERR_INVALID_PZ The Protection Zone Handle (*pz_handle*) was invalid.

7377 IT_ERR_INVALID_MASK The *mask* contained invalid flag values.

7378 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the disabled
7379 state. None of the output parameters from this routine are valid.
7380 See *it_ia_info_t* for a description of the disabled state.

7381 SEE ALSO

7382 *it_pz_create()*, *it_pz_free()*

it_reject()

7383

7384 NAME

7385 `it_reject` – reject an incoming Connection Request or Connection Reply

7386 SYNOPSIS

```
7387 #include <it_api.h>
7388
7389 it_status_t it_reject(
7390     IN          it_cn_est_identifier_t  cn_est_id,
7391     IN const unsigned char             *private_data,
7392     IN          size_t                 private_data_length
7393 );
7394
7395 typedef uint64_t it_cn_est_identifier_t;
```

7396 APPLICABILITY

7397 `it_reject` is applicable only to the RC service type.

7398 DESCRIPTION

7399 `cn_est_id` Connection Establishment Identifier associated with the Connection
7400 Request to be rejected. Calling `it_reject` destroys the identifier. See
7401 [it_ep_accept](#) for a definition of this data type.

7402 `private_data` Opaque Private Data to be sent in the IT_CM_MSG_CONN_PEER_
7403 REJECT_EVENT Event delivered to the Remote Consumer. If the IA does
7404 not support Private Data, `private_data_length` must be zero.

7405 `private_data_length` Length of `private_data`. This field must be 0 if the IA does not support
7406 Private Data.

7407 `it_reject` rejects an incoming Connection Request or Connection Reply. The Remote Endpoint
7408 will receive an IT_CM_MSG_CONN_PEER_REJECT_EVENT Event on its IT_CM_MSG_
7409 EVENT_STREAM Simple Event Dispatcher, and that Endpoint will transition into the
7410 IT_EP_STATE_NONOPERATIONAL state.

7411 For two-way Connection establishment, `it_reject` can only be called on the Passive side in
7412 response to the IT_CM_REQ_CONN_REQUEST_EVENT Event.

7413 For three-way Connection establishment, `it_reject` can be called on the Passive side in response
7414 to the IT_CM_REQ_CONN_REQUEST_EVENT, or on the Active side in response to the
7415 IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT. If `it_reject` is called on the active side,
7416 the local Endpoint associated with the Connection establishment transitions to the IT_EP_
7417 STATE_NONOPERATIONAL state. See the [it_ep_state_t](#) reference page for a description of
7418 this Endpoint state.

7419 Once the Endpoint is in the IT_EP_STATE_NONOPERATIONAL state, any pending Data
7420 Transfer Operations or Link or Unlink operations on the Endpoint will be flushed and will
7421 generate Completion Events with a Status of IT_DTO_ERR_FLUSHED.

7422 The Connection Establishment Identifier, `cn_est_id`, is freed by `it_reject`.

7423 **RETURN VALUE**

7424 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

7425 IT_ERR_INVALID_CN_EST_ID The Connection Establishment Identifier (*cn_est_id*)
7426 was invalid.

7427 IT_ERR_PDATA_NOT_SUPPORTED Private Data was supplied by the Consumer but this
7428 Interface Adapter does not support Private Data. See
7429 [it_ia_query](#) for the IAs capabilities to support Private
7430 Data.

7431 IT_ERR_INVALID_PDATA_LENGTH The Interface Adapter supports Private Data, but the
7432 length specified exceeded the Interface Adapter's
7433 capabilities.

7434 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in
7435 the disabled state. None of the output parameters from
7436 this routine are valid. See [it_ia_info_t](#) for a description
7437 of the disabled state.

7438 **APPLICATION USAGE**

7439 1. The Consumer is responsible for coordinating the use of functions that free a Connection
7440 Establishment Identifier (*cn_est_id*) such as [it_ep_accept](#), [it_reject](#), [it_ep_disconnect](#), and
7441 [it_handoff](#). The behavior of functions that are passed in an invalid Connection
7442 Establishment Identifier is indeterminate.

7443 2. Calling [it_reject](#) does not necessarily mean that the remote end will receive a peer reject
7444 Event; they might receive a non-peer reject Event (e.g., because the REJ message got lost
7445 on IB and the connection establishment attempt timed out). If the remote does get a peer
7446 reject Event, the Private Data contained within that Event will be exactly what was
7447 furnished to [it_reject](#).

7448 **SEE ALSO**

7449 [it_ep_accept\(\)](#), [it_ep_connect\(\)](#), [it_cm_req_events](#), [it_cm_msg_events](#), [it_ep_state_t](#),
7450 [it_handoff\(\)](#), [it_ia_query\(\)](#)

it_rmr_create()

7451

7452 NAME

7453 `it_rmr_create` – create a Remote Memory Region (RMR)

7454 SYNOPSIS

```
7455 #include <it_api.h>
7456
7457 it_status_t it_rmr_create(
7458     IN  it_pz_handle_t    pz_handle,
7459     IN  it_rmr_type_t    rmr_type,
7460     OUT it_rmr_handle_t  *rmr_handle
7461 );
```

7462 APPLICABILITY

7463 Remote Memory Regions can be used only for the RC service type.

7464 DESCRIPTION

7465 *pz_handle* Protection Zone in which the Remote Memory Region will be created.

7466 *rmr_type* Desired type for the Remote Memory Region. RMR types are defined in
7467 [it_rmr_type_t](#).

7468 *rmr_handle* Handle of new Remote Memory Region.

7469 The `it_rmr_create` routine creates a Remote Memory Region within a Protection Zone identified
7470 by the *pz_handle* argument and associated with the Interface Adapter implicitly identified by
7471 *pz_handle*. RMRs can be accessed only remotely (for setting access privileges, see [it_rmr_link](#)).
7472 The restriction to remote access does not preclude a special case where a “local RMR” is used
7473 locally in a DTO – see the Extended Description for details.

7474 The desired RMR type is specified through the *rmr_type* argument. The allowable RMR types
7475 are Implementation-dependent (see [it_rmr_type_t](#) for details).

7476 If the Consumer specifies an *rmr_type* of `IT_RMR_TYPE_DEFAULT`, the Implementation will
7477 choose a type of RMR that is supported by the underlying IA, which may be either a Narrow or a
7478 Wide RMR. Requesting an RMR type that is not supported by the Implementation will cause
7479 `it_rmr_create` to fail.

7480 An RMR cannot be used as a target for DTOs until it has been linked with [it_rmr_link](#).

7481 An unlinked Wide or Narrow RMR may be used as a target for Link operations via all Endpoints
7482 in the PZ of the RMR.

7483 A linked Wide RMR may be used as a target for DTOs and Link/Unlink operations via all
7484 Endpoints in the PZ of the RMR. A linked Narrow RMR may be used as a target for DTOs and
7485 Unlink operations only via the Endpoint through which it was linked.

7486 EXTENDED DESCRIPTION

7487 InfiniBand Implementations at least support Wide RMRs and, if Verb Extensions (BMM) are
7488 provided, also Narrow RMRs; iWARP Implementations at least support Narrow RMRs.

7489 For iWARP only, RMRs can also be used locally, namely as data sinks in an
7490 *it_post_rdma_read_to_rmr* call; see the Extended Description of *it_post_rdma_read_to_rmr* for
7491 why iWARP nevertheless considers the writing to a local data sink of an RDMA Read DTO a
7492 remote write access.

7493 **BACKWARDS COMPATIBILITY**

7494 *it_rmr_create* (v2.0) called with *rmr_type* set to IT_RMR_TYPE_WIDE behaves identically as
7495 *it_rmr_create* (v1.0), which creates Wide RMRs only.

7496 *it_rmr_create* (v2.0) called with *rmr_type* set to IT_RMR_TYPE_DEFAULT selects the
7497 appropriate RMR type for the given Implementation.

7498 See also Appendix C.

7499 **RETURN VALUE**

7500 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

7501 IT_ERR_INVALID_PZ The Protection Zone Handle (*pz_handle*) was invalid.

7502 IT_ERR_INVALID_RMR_TYPE The RMR type (*rmr_type*) was invalid or not supported
7503 by the Implementation.

7504 IT_ERR_RESOURCES The requested operation failed due to insufficient
7505 resources.

7506 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the
7507 disabled state. None of the output parameters from this
7508 routine are valid. See *it_ia_info_t* for a description of the
7509 disabled state.

7510 **APPLICATION USAGE**

7511 The returned RMR must be linked to a Local Memory Region by means of *it_rmr_link* before it
7512 can be used in Data Transfer Operations. Remote use of the linked RMR also requires exposing
7513 the RMR to a remote Consumer through an associated RMR Context.

7514 Creating an RMR is a relatively expensive operation. Once created, however, an RMR may be
7515 linked repeatedly to different LMR address ranges using the more efficient *it_rmr_link* call, as
7516 long as a linked RMR is first unlinked through *it_rmr_unlink*. Linking an RMR is much more
7517 efficient than changing remote access privileges using *it_lmr_modify*.

7518 **SEE ALSO**

7519 *it_rmr_link()*, *it_rmr_free()*, *it_rmr_query()*

it_rmr_free()

7520

7521 NAME

7522 `it_rmr_free` – destroy a Remote Memory Region

7523 SYNOPSIS

```
7524 #include <it_api.h>
7525
7526 it_status_t it_rmr_free(
7527     IN it_rmr_handle_t rmr_handle
7528 );
```

7529 APPLICABILITY

7530 Remote Memory Regions can be used only for the RC service type. See [it_rmr_create](#).

7531 DESCRIPTION

7532 `rmr_handle` Handle of Remote Memory Region to be destroyed.

7533 The `it_rmr_free` routine destroys the Remote Memory Region `rmr_handle`. If the RMR is
7534 currently linked to an LMR, then the RMR linking is also destroyed. On return, the `rmr_handle`
7535 may no longer be used, and the associated RMR Context may no longer be used. RMRs with
7536 memory ranges that overlap the range of `rmr_handle` are not affected by its destruction.

7537 Outstanding remote DTOs that use the RMR Context of this RMR may either complete
7538 successfully or fail with an access violation error.

7539 RETURN VALUE

7540 A successful call returns `IT_SUCCESS`. Otherwise, an error code is returned as described below:

7541 `IT_ERR_INVALID_RMR` The Remote Memory Region Handle (`rmr_handle`) was invalid.

7542 `IT_ERR_IA_CATASTROPHE` The IA has experienced a catastrophic error and is in the disabled
7543 state. None of the output parameters from this routine are valid.
7544 See [it_ia_info_t](#) for a description of the disabled state.

7545 APPLICATION USAGE

7546 Since the number of possible RMR Context values is finite, the Implementation will eventually
7547 re-use previously freed values in a new linking. If a DTO using an RMR Context is posted after
7548 that Context is freed, it is theoretically possible for the Context to be re-used before the DTO
7549 completes, and for the DTO to complete under the new linking for the Context, resulting in data
7550 corruption. To avoid this, the Consumer should not free an RMR which may be the target of
7551 outstanding DTOs. This may require coordination between local and remote Consumers, and
7552 such coordination is the Consumer's responsibility.

7553 SEE ALSO

7554 [it_rmr_create\(\)](#), [it_rmr_query\(\)](#)

it_rmr_link()

7555
7556 **NAME**
7557 *it_rmr_link* – post operation to Link a Remote Memory Region to a memory range

7558 SYNOPSIS

```
7559 #include <it_api.h>  
7560  
7561 it_status_t it_rmr_link(  
7562     IN    it_rmr_handle_t    rmr_handle,  
7563     IN    it_lmr_handle_t    lmr_handle,  
7564     IN    void                *addr,  
7565     IN    it_length_t        length,  
7566     IN    it_addr_mode_t     addr_mode,  
7567     IN    it_mem_priv_t      privs,  
7568     IN    it_ep_handle_t     ep_handle,  
7569     IN    it_dto_cookie_t    cookie,  
7570     IN    it_dto_flags_t     dto_flags,  
7571     OUT   it_rmr_context_t   *rmr_context  
7572 );
```

7573 APPLICABILITY

7574 *it_rmr_link* is applicable only to Endpoints created for the RC service type.

7575 DESCRIPTION

7576	<i>rmr_handle</i>	Handle of RMR that will be linked.
7577	<i>lmr_handle</i>	LMR to which RMR will be linked.
7578	<i>addr</i>	Starting address of RMR to be linked.
7579	<i>length</i>	Length in bytes of RMR to be linked. Must not be 0.
7580	<i>addr_mode</i>	Addressing mode of RMR to be linked.
7581	<i>privs</i>	Bitwise OR of remote access privilege flags for linked RMR, taken from it_mem_priv_t .
7582		
7583	<i>ep_handle</i>	Endpoint on which to post the Link operation.
7584	<i>cookie</i>	Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the RMR Link operation.
7585		
7586	<i>dto_flags</i>	Bitwise OR of options for operation handling.
7587	<i>rmr_context</i>	Returned Context allowing remote access to the linked RMR.
7588		The <i>it_rmr_link</i> routine posts to Endpoint <i>ep_handle</i> an operation to Link the Remote Memory Region <i>rmr_handle</i> to the segment of an LMR specified by the <i>lmr_handle</i> , <i>addr</i> , and <i>length</i> arguments. It returns a new <i>rmr_context</i> value in network byte order that can be transferred by the local Consumer to a remote Consumer to be used for an RDMA operation.
7589		
7590		
7591		

7592 The arguments *addr* and *length* provide the starting address, also known as the Base Address,
7593 and the length in bytes, respectively, of the region to be linked. The starting address *addr* must
7594 be in the same linear address space as used by the underlying LMR given by *lmr_handle*. The
7595 specified address range must fall within the address range of the underlying LMR.

7596 If the *addr_mode* argument (*it_addr_mode_t*) selects Absolute Addressing, then DTOs will
7597 access the RMR through absolute addresses in the given linear address space. If the *addr_mode*
7598 argument selects Relative Addressing, then DTOs will access the RMR through offsets relative
7599 to the Base Address of the RMR, which is within the address range of the underlying LMR.
7600 Consumers can check the IA attribute *addr_mode_relative_support* to determine whether
7601 Relative Addressing is supported.

7602 An RMR can be linked only if the Endpoint *ep_handle* is in IT_EP_STATE_CONNECTED
7603 state.

7604 While any RMR is associated with a Protection Zone, successfully linking a Narrow RMR
7605 causes the RMR to be also associated with the Endpoint *ep_handle* to which the RMR Link
7606 operation is posted (see *it_rmr_type_t*).

7607 A Narrow RMR can be linked if it is not currently in linked state, while a Wide RMR can be
7608 (re)linked regardless of whether it is currently linked.

7609 The RMR handle must be valid. The LMR handle must be valid. For any RMR type, the
7610 Protection Zones of the RMR, underlying LMR, and Endpoint must match. For any RMR type
7611 and regardless of the *addr_mode* argument, the underlying LMR must use Absolute Addressing.
7612 An RMR of type IT_RMR_TYPE_NARROW must be in unlinked state to be linkable.

7613 Remote access to an RMR is enforced with byte-level granularity.

7614 RMRs can be accessed only remotely, and any local access privileges specified in the *privs*
7615 argument are ignored. The type of remote access to be allowed is specified by the *privs*
7616 argument as a bitwise-inclusive OR of one or more of the bit values IT_PRIV_
7617 REMOTE_READ and IT_PRIV_REMOTE_WRITE. Either or both of IT_PRIV_
7618 REMOTE_READ and IT_PRIV_REMOTE_WRITE must be included; see *it_mem_priv_t* for
7619 bit definitions and predefined bit combinations.

7620 It is legal to request remote access rights that exceed the remote access rights of the underlying
7621 LMR.

7622 The *cookie* (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work
7623 Request. This identifier is completely under Consumer control and opaque to the
7624 Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted
7625 Work Request.

7626 Request handling is specified by the *dto_flags* argument and is the bitwise OR of zero or more of
7627 the following flags:

7628 IT_COMPLETION_FLAG
7629 IT_NOTIFY_FLAG
7630 IT_BARRIER_FENCE_FLAG

7631 For the definition of these flags, see *it_dto_flags_t*. In addition, *it_rmr_link* automatically fences
7632 all DTO, Link, and Unlink operations subsequently submitted on the Endpoint *ep_handle* such
7633 that none of these operations starts until the currently posted Link operation has completed.

7634 The completion of the posted Link operation is reported asynchronously to the Consumer
7635 according to the rules defined in *it_dto_flags_t*. An RMR Link Completion Event is of type
7636 *it_dto_cmpl_event_t*. Any RMR Link Completion Event generated manifests on the EVD
7637 associated with the Endpoint Send Queue. The Event type is IT_RMR_LINK_CMPL_EVENT.

7638 The value of *rmr_context* is immediately available when *it_rmr_link* returns, but it may not be
7639 used by a remote host for an RDMA operation until the Link Completion Event occurs.
7640 Violation of this rule may result in an error and a broken Connection for the reliable Connection
7641 Endpoint on which the RDMA operation is posted. See Application Usage for more details.

7642 After a successful Link Completion Event, any previous linking for the RMR is invalidated.
7643 Successfully linking an RMR results in a new RMR Context being associated with the RMR and
7644 revokes remote access using any previous RMR Contexts, as long as the new RMR Context is
7645 not a re-use of a previously generated RMR Context. The revocation of remote access may affect
7646 operations that are outstanding when *it_rmr_link* is called. Consumers should make sure that
7647 such operations complete before calling *it_rmr_link*.

7648 The new linking remains valid until the next Link or Unlink operation completes successfully, or
7649 until the RMR is destroyed. A Link operation will never be partially successful over a subset of
7650 the requested memory range; it either succeeds completely or fails without invalidating any
7651 portion of the previous linking.

7652 If *it_rmr_link* returns successfully, but the Link Completion Event status indicates failure, then
7653 any previous linking and RMR Context remains valid.

7654 Any Work Request posted to an Endpoint's Send Queue after a call to *it_rmr_link* will not begin
7655 execution until the Link operation has completed.

7656 **EXTENDED DESCRIPTION**

7657 For the InfiniBand Transport, specifying the IT_PRIV_REMOTE_WRITE access privilege for
7658 the RMR requires that the underlying LMR be enabled for local write access.

7659 For the InfiniBand Transport, linking a Wide RMR with Relative Addressing is typically not
7660 supported (even if the IA attribute *addr_mode_relative_support* indicates support for Relative
7661 Addressing); in this case, an IT_ERR_INVALID_ADDR_MODE immediate error is returned.

7662 **BACKWARDS COMPATIBILITY**

7663 *it_rmr_link* called for a Wide RMR and with *addr_mode* set to IT_ADDR_MODE_ABSOLUTE
7664 behaves identically as *it_rmr_bind* (v1.0).

7665 See also Appendix C.

7666 **RETURN VALUE**

7667 A successful call returns IT_SUCCESS. Otherwise, an immediate error is returned and, if the
7668 RMR was previously linked, the previous linking and RMR Context remains valid. It is possible
7669 for *it_rmr_link* to return success but for the Completion Event to indicate failure.

7670 Posting to an Endpoint that is not in the IT_EP_STATE_CONNECTED or
7671 IT_EP_STATE_NONOPERATIONAL state will return the IT_ERR_INVALID_EP_STATE
7672 immediate error.

7673 The possible immediate errors for *it_rmr_link* are listed below.

7674 7675	IT_ERR_ADDRESS	The address (<i>addr</i>) fell outside the boundaries specified by the Local Memory Region.
7676 7677 7678	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.
7679 7680	IT_ERR_INVALID_DTO_FLAGS	The Data Transfer Operation flags (<i>dto_flags</i>) value was invalid.
7681	IT_ERR_INVALID_EP	The Endpoint Handle (<i>ep_handle</i>) was invalid.
7682 7683	IT_ERR_INVALID_EP_STATE	The Endpoint was not in the proper state for the attempted operation.
7684 7685	IT_ERR_INVALID_EP_TYPE	The attempted operation was invalid for the Service Type of the Endpoint.
7686 7687	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.
7688 7689	IT_ERR_INVALID_LMR	The Local Memory Region Handle (<i>lmr_handle</i>) was invalid.
7690 7691	IT_ERR_INVALID_PRIVS	The requested memory privileges (<i>privs</i>) contained an invalid flag.
7692 7693	IT_ERR_INVALID_RMR	The Remote Memory Region Handle (<i>rmr_handle</i>) was invalid.
7694	IT_ERR_TOO_MANY_POSTS	The operation failed due to an overflow of a Work Queue.
7695 7696 7697 7698	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.

7699 **ASYNCHRONOUS ERRORS**

7700 For Work Requests posted to an Endpoint in the IT_EP_STATE_CONNECTED state, a
7701 completion status (see *it_dto_status_t*) other than IT_DTO_SUCCESS will break the Connection
7702 by moving the Endpoint to the IT_EP_STATE_NONOPERATIONAL state and deliver an
7703 IT_CM_MSG_CONN_BROKEN_EVENT Event to the Connect EVD of *ep_handle*. Once the
7704 Connection is broken, all outstanding and in-progress operations on the Connection will
7705 complete with an error status.

7706 Any posting to an Endpoint that is in the IT_EP_STATE_NONOPERATIONAL state will be
7707 flushed with completion status set to IT_DTO_ERR_FLUSHED.

7708 If the RMR address range specified with *addr* and *length* exceeds the address range of the
7709 underlying LMR, a Completion Error is generated with completion status set to
7710 IT_RMR_OPERATION_FAILED.

7711 An IT_RMR_OPERATION_FAILED completion status also results in case of an invalid RMR
7712 handle or invalid LMR handle, unless the Implementation handles these errors as immediate
7713 errors.

7714 An `IT_RMR_OPERATION_FAILED` completion status also results if the Protection Zones of
7715 the RMR, underlying LMR, and Endpoint do not match, if the underlying LMR does not use
7716 Absolute Addressing, or if the RMR is a Narrow RMR and not in unlinked state.

7717 For InfiniBand, it is invalid to request remote write access if the memory access privileges of the
7718 underlying LMR do not include local write access. Such an attempt also causes an
7719 `IT_RMR_OPERATION_FAILED` completion status.

7720 Binding a Narrow or Wide RMR with Remote Read permission when the LMR permissions do
7721 not include Local Read results in a Completion Error with the `IT_RMR_OPERATION_FAILED`
7722 completion status.

7723 Linking an RMR to an LMR that uses Relative Addressing is invalid and results in a Completion
7724 Error with the `IT_DTO_ERR_LOCAL_MM_OPERATION` completion status.

7725 For the InfiniBand Transport, linking a Narrow RMR with a length of zero is invalid and results
7726 in a Completion Error with the `IT_DTO_ERR_LOCAL_MM_OPERATION` completion status.

7727 APPLICATION USAGE

7728 The `it_rmr_link` operation is lightweight compared to creating an RMR or an LMR. An
7729 application concerned with efficiency would typically create one or more RMRs at initialization
7730 time that could be linked multiple times to enable remote access to different peers as needed.

7731 The Consumer should use unique identifiers for `cookie` if they desire to identify each DTO. If the
7732 Consumer does not require a unique DTO identifier, the value of zero or NULL
7733 (`IT_NO_ADDR`) can be used.

7734 The local Consumer has several options for ensuring that the remote Consumer does not use
7735 `rmr_context` before the Link Completion Event occurs. One is to wait for the Completion Event
7736 on the Send EVD of the specified Endpoint `ep_handle` before sending the `rmr_context` to a peer.
7737 Another option is to send the `rmr_context` to a peer by posting a DTO to the same Endpoint
7738 `ep_handle` that was used to Link the RMR. The barrier-fencing behavior of `it_rmr_link` ensures
7739 that the DTO does not start until the Link Completion Event has occurred. If the Link fails with
7740 a Completion Error, the Connection will be broken and the DTO flushed, so the `rmr_context` will
7741 not be sent.

7742 For reasons already described, the completion of an RMR Link operation represents an
7743 important change that Consumers may need to monitor. One way to do this is to set
7744 `IT_COMPLETION_FLAG` in `dto_flags`, which will generate a Completion Event to indicate
7745 when the RMR has been linked. Consumers who do not set `IT_COMPLETION_FLAG` must
7746 rely on ordering semantics to infer when the RMR has been successfully linked. For example, if
7747 a subsequent DTO posted to the Send Queue of the same EP completes successfully, then the
7748 Link operation has completed, because DTOs posted to the Send Queue of an EP must wait for a
7749 Link operation to complete before processing. If the Link operation fails, a Link Completion
7750 Event is generated regardless of the use of `IT_COMPLETION_FLAG`.

7751 On the InfiniBand Transport, it is possible to prevent further accesses to a (linked) RMR by
7752 linking it with `privs` set to 0, but it is recommended and preferable to use `it_rmr_unlink` instead.

7753 When an EVD holding the DTO Event Stream associated with an Endpoint overflows, on some
7754 transports the behavior of the Endpoint and/or the EVD is undefined, making recovery from
7755 such an overflow either impossible or at the very least IA-dependent. Consumers should
7756 therefore take care to avoid this situation. Each Work Request that a Consumer posts to a Work
7757 Queue can result in a Completion Event being enqueued in an EVD, due to the possibility of an

7758 error occurring during Work Request processing. To avoid any possibility of overflow, the
7759 Consumer must therefore ensure that the total number of Outstanding Operations on all Work
7760 Queues that can enqueue Completion Events into a given EVD never exceeds the minimum
7761 queue size (as reported by *it_evd_query*) of that EVD.

7762 **FUTURE DIRECTIONS**

7763 Calling *it_rmr_link* on a Wide RMR that is already in the linked state is currently allowed, while
7764 the same operation is not allowed for a linked Narrow RMR. A future version of the IT-API may
7765 require the Consumer to unlink any linked RMR using *it_rmr_unlink* before linking it, regardless
7766 of the transport.

7767 **SEE ALSO**

7768 *it_lmr_create()*, *it_rmr_create()*, *it_rmr_unlink()*, *it_dto_flags_t*, *it_dto_events*, *it_addr_mode_t*,
7769 *it_mem_priv_t*

it_rmr_query()

7770

7771 NAME

7772

it_rmr_query – get attributes of a Remote Memory Region

7773 SYNOPSIS

7774

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

7775

7776

```
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  
);
```

7781

7782

```
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;
```

7795

7796

```
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;
```

7808 APPLICABILITY

7809

Remote Memory Regions can be used only for the RC service type. See [it_rmr_create](#).

7810 DESCRIPTION

7811

rmr_handle Remote Memory Region.

7812

mask Bitwise OR of flags for desired parameters.

7813

params Structure whose members are written with the desired parameters.

7814

The *it_rmr_query* routine returns the desired attributes of the Remote Memory Region

7815

rmr_handle in the structure pointed to by *params*. The *mask* argument specifies which fields of

7816 *params* are returned, and the values returned in other fields are undefined. See the Synopsis for
7817 the correspondence between *mask* values and fields. The *mask* value IT_RMR_PARAM_ALL
7818 causes all fields to be returned.

7819 The definition of each field of *params* follows, some of which depend on whether the RMR is
7820 currently linked to an LMR as a result of using *it_rmr_link*:

7821	<i>ia</i>	The Interface Adapter Handle specified to create the RMR.
7822	<i>pz</i>	The Protection Zone Handle specified to create the RMR.
7823	<i>linked</i>	IT_TRUE if the RMR is currently linked; otherwise, IT_FALSE.
7824	<i>lmr</i>	The Local Memory Region to which the RMR is currently linked. Undefined if 7825 RMR is not linked.
7826	<i>addr</i>	The currently linked starting address or, equivalently, Base Address of the 7827 RMR. To be interpreted as an absolute address, regardless of <i>addr_mode</i> . 7828 Undefined if RMR is not linked.
7829	<i>length</i>	The currently linked length in bytes of the RMR. Undefined if RMR is not 7830 linked.
7831	<i>privs</i>	The currently linked memory access privileges of the RMR. Undefined if RMR 7832 is not linked. For the definition of bits, see <i>it_mem_priv_t</i> .
7833	<i>rmr_context</i>	The currently linked RMR Context associated with the RMR. Undefined if 7834 RMR is not linked. Returned in network byte order.
7835	<i>type</i>	The type of the RMR, either IT_RMR_TYPE_NARROW or 7836 IT_RMR_TYPE_WIDE.
7837	<i>addr_mode</i>	The addressing mode of the RMR. Undefined if RMR is not linked.

7838 If the Consumer calls *it_rmr_query* after posting an RMR Link or RMR Unlink operation, and
7839 before dequeuing the Completion Event of such an operation, then the returned *linked*, *lmr*,
7840 *addr*, *length*, *privs*, *rmr_context*, and *addr_mode* fields may represent the RMR state as it was
7841 prior to posting, or a new RMR state. The Consumer should not rely on the value of these fields
7842 during this time.

7843 **RETURN VALUE**

7844 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

7845	IT_ERR_INVALID_MASK	The <i>mask</i> contained invalid flag values.
7846	IT_ERR_INVALID_RMR	The Remote Memory Region Handle (<i>rmr_handle</i>) was invalid.
7847	IT_ERR_IA_CATASTROPHE	The IA has experienced a catastrophic error and is in the disabled 7848 state. None of the output parameters from this routine are valid. 7849 See <i>it_ia_info_t</i> for a description of the disabled state.

7850 **SEE ALSO**

7851 *it_rmr_create()*, *it_rmr_free()*, *it_rmr_link()*, *it_rmr_context_t*, *it_addr_mode_t*, *it_mem_priv_t*,
7852 *it_rmr_type_t*

it_rmr_unlink()

7853

7854 NAME

7855 `it_rmr_unlink` – post operation to Unlink a Remote Memory Region from its memory range

7856 SYNOPSIS

```
7857 #include <it_api.h>
7858
7859 it_status_t it_rmr_unlink(
7860     IN it_rmr_handle_t rmr_handle,
7861     IN it_ep_handle_t ep_handle,
7862     IN it_dto_cookie_t cookie,
7863     IN it_dto_flags_t dto_flags
7864 );
```

7865 APPLICABILITY

7866 `it_rmr_unlink` is applicable only to Endpoints created for the RC service type.

7867 DESCRIPTION

7868 *rmr_handle* Handle of RMR that will be unlinked.

7869 *ep_handle* Endpoint on which to post the operation.

7870 *cookie* Consumer-provided cookie that is returned to the Consumer in the
7871 Completion Event corresponding to the operation.

7872 *dto_flags* Bitwise OR of options for operation handling.

7873 The `it_rmr_unlink` routine posts to Endpoint *ep_handle* an Unlink operation to Unlink the
7874 Remote Memory Region specified by *rmr_handle*.

7875 An RMR can be unlinked only if the Endpoint *ep_handle* is in `IT_EP_STATE_CONNECTED`
7876 state.

7877 For any RMR type, the Protection Zones of the RMR and Endpoint must match. For a linked
7878 Narrow RMR, the Endpoint *ep_handle* to which the Unlink operation is posted must match the
7879 Endpoint through which the RMR was linked (see *it_rmr_type_t*).

7880 An RMR can be unlinked regardless of whether it is currently linked or unlinked.

7881 The *cookie* (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work
7882 Request. This identifier is completely under Consumer control and opaque to the
7883 Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted
7884 Work Request.

7885 Request handling is specified by the *dto_flags* argument as the bitwise OR of zero or more of the
7886 following flags:

```
7887 IT_COMPLETION_FLAG
7888 IT_NOTIFY_FLAG
7889 IT_BARRIER_FENCE_FLAG
7890 IT_UNLINK_FENCE_FLAG
```

7891 For the definition of these flags, see *it_dto_flags_t*.

7892 The completion of the posted Unlink operation is reported asynchronously to the Consumer
7893 according to the rules defined in *it_dto_flags_t*. An RMR Unlink Completion Event is of type
7894 *it_dto_cmpl_event_t*. Any generated RMR Unlink Completion Event manifests on the EVD
7895 associated with the Endpoint Send Queue. The Event type is IT_RMR_LINK_CMPL_EVENT.

7896 After a successful Unlink Completion Event, any previous linking for the RMR is invalidated,
7897 and the RMR Context for the RMR is no longer defined. Any RDMA operation that uses the
7898 previous RMR Context will fail with a protection violation; beware that this may include
7899 operations that are outstanding when *it_rmr_unlink* is called. The Consumer must ensure that
7900 such operations have completed prior to calling *it_rmr_unlink* if successful completions are
7901 desired. An Unlink operation will never be partially successful over a subset of the requested
7902 memory range; it either succeeds completely or fails without invalidating any portion of the
7903 previous linking.

7904 If *it_rmr_unlink* returns successfully but the Completion Event status indicates failure, then any
7905 previous linking and RMR Context remains valid.

7906 Any Work Request posted to an Endpoint's Send Queue after a call to *it_rmr_unlink* will not
7907 begin execution until the Unlink operation has completed.

7908 RETURN VALUE

7909 A successful call returns IT_SUCCESS. Otherwise, an immediate error is returned and, if the
7910 RMR was previously linked the previous linking of the RMR remains valid. It is possible for
7911 *it_rmr_unlink* to return success but for the Completion Event to indicate failure.

7912 Posting to an Endpoint that is not in the IT_EP_STATE_CONNECTED or
7913 IT_EP_STATE_NONOPERATIONAL state will return the IT_ERR_INVALID_EP_STATE
7914 immediate error.

7915 The possible immediate errors for *it_rmr_unlink* are listed below:

7916	IT_ERR_INVALID_DTO_FLAGS	The Data Transfer Operation flags (<i>dto_flags</i>) value was
7917		invalid.
7918	IT_ERR_INVALID_EP	The Endpoint Handle (<i>ep_handle</i>) was invalid.
7919	IT_ERR_INVALID_EP_STATE	The Endpoint was not in the proper state for the attempted
7920		operation.
7921	IT_ERR_INVALID_EP_TYPE	The attempted operation was invalid for the Service Type of
7922		the Endpoint.
7923	IT_ERR_INVALID_RMR	The Remote Memory Region Handle (<i>rmr_handle</i>) was
7924		invalid.
7925	IT_ERR_TOO_MANY_POSTS	The operation failed due to an overflow of a Work Queue.
7926	IT_ERR_IA_CATASTROPHE	The IA has experienced a catastrophic error and is in the
7927		disabled state. None of the output parameters from this routine
7928		are valid. See <i>it_ia_info_t</i> for a description of the disabled
7929		state.

7930 **ASYNCHRONOUS ERRORS**

7931 For Work Requests posted to an Endpoint in the `IT_EP_STATE_CONNECTED` state, a
7932 completion status (see *it_dto_status_t*) other than `IT_DTO_SUCCESS` will break the Connection
7933 by moving the Endpoint to the `IT_EP_STATE_NONOPERATIONAL` state and deliver an
7934 `IT_CM_MSG_CONN_BROKEN_EVENT` Event to the Connect EVD of *ep_handle*. Once the
7935 Connection is broken, all outstanding and in-progress operations on the Connection will
7936 complete with an error status.

7937 Any posting to an Endpoint that is in the `IT_EP_STATE_NONOPERATIONAL` state will be
7938 flushed with completion status set to `IT_DTO_ERR_FLUSHED`.

7939 An `IT_RMR_OPERATION_FAILED` completion status results in case of an invalid RMR
7940 handle, unless the Implementation handles this error as an immediate error.

7941 An `IT_RMR_OPERATION_FAILED` completion status also results if the Protection Zones of
7942 the RMR and the Endpoint do not match or, for a linked Narrow RMR, if the Endpoint to which
7943 the Unlink operation is posted does not match the Endpoint through which the RMR was linked.

7944 **APPLICATION USAGE**

7945 *it_rmr_unlink* may be used to revoke remote Consumer access to an RMR that was previously
7946 granted. In addition, the Consumer must Unlink all RMRs that refer to an LMR in order to
7947 destroy or modify the LMR. Note that the RMR is not considered unlinked until a successful
7948 Completion Event is generated; thus, the Consumer should dequeue the Completion Event
7949 before calling *it_lmr_free*.

7950 A difficulty can arise if the Endpoint that the Consumer was using to Link the RMR has become
7951 disconnected, because an Unlink operation can only be posted to a connected Endpoint. The
7952 preferred solution is to destroy the RMR by calling *it_rmr_free*. In case of a Wide RMR and for
7953 the InfiniBand Transport only, an alternative is for the Consumer to create a special pair of
7954 Endpoints within the same Protection Zone as the Wide RMR and to connect these Endpoints in
7955 loopback mode to each other; the Wide RMR can then be unlinked through one of the special
7956 Endpoints.

7957 For reasons already described, the completion of an RMR Unlink operation represents an
7958 important change that Consumers may need to monitor. One way to do this is to set
7959 `IT_COMPLETION_FLAG` in *dto_flags*, which will generate a Completion Event to indicate
7960 when the RMR has been unlinked. Consumers who do not set `IT_COMPLETION_FLAG` must
7961 rely on ordering semantics to infer when the RMR has been successfully unlinked. For example,
7962 if a subsequent DTO posted to the Send Queue of the same EP completes successfully, then the
7963 Link operation has completed, because DTOs posted to the Send Queue of an EP must wait for a
7964 Link operation to complete before processing. If the Unlink operation fails, a Completion Event
7965 is generated regardless of the use of `IT_COMPLETION_FLAG`.

7966 When an EVD holding the DTO Event Stream associated with an Endpoint overflows, on some
7967 transports the behavior of the Endpoint and/or the EVD is undefined, making recovery from
7968 such an overflow either impossible or at the very least IA-dependent. Consumers should
7969 therefore take care to avoid this situation. Each Work Request that a Consumer posts to a Work
7970 Queue can result in a Completion Event being enqueued in an EVD, due to the possibility of an
7971 error occurring during Work Request processing. To avoid any possibility of overflow, the
7972 Consumer must therefore ensure that the total number of Outstanding Operations on all Work
7973 Queues that can enqueue Completion Events into a given EVD never exceeds the minimum
7974 queue size (as reported by *it_evd_query*) of that EVD.

7975 **SEE ALSO**
7976 *it_rmr_create(), it_rmr_link(), it_dto_flags_t*

it_set_consumer_context()

7977

7978 NAME

7979 `it_set_consumer_context` – associate a Consumer Context with an IT Object Handle

7980 SYNOPSIS

```
7981 #include <it_api.h>
7982
7983 it_status_t it_set_consumer_context(
7984     IN it_handle_t handle,
7985     IN it_context_t context
7986 );
```

7987 DESCRIPTION

7988 *handle* Handle for the IT-API object to be associated with the Consumer Context.

7989 *context* The Consumer Context to be associated with the object Handle.

7990 *it_set_consumer_context* associates a Consumer Context with the specified *handle*. See
7991 [it_handle_t](#) for a description of the valid Handle types.

7992 Only a single Consumer Context is provided for any IT Object Handle. If there is a previous
7993 Consumer Context associated with the specified Handle, the new Context replaces the old one.
7994 The value of Context is opaque to the Implementation. The Consumer can disassociate the
7995 existing Context by providing a NULL (IT_NO_ADDR) value for the Context. The
7996 Implementation makes no attempt to synchronize access to the Context.

7997 RETURN VALUE

7998 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

7999 IT_ERR_INVALID_HANDLE The *handle* was invalid.

8000 IT_ERR_IA_CATASTROPHE The IA has experienced a catastrophic error and is in the disabled
8001 state. None of the output parameters from this routine are valid.
8002 See [it_ia_info_t](#) for a description of the disabled state.

8003 EXAMPLES

8004 The following code example demonstrates the use of a cast in the call to
8005 *it_set_consumer_context*. The *lmr* object is cast to the generic *it_handle_t* type for the call.

```
8006 it_lmr_handle_t lmr;
8007 it_context_t cxt = 1234;
8008 it_set_consumer_context( (it_handle_t) lmr, cxt);
```

8009 SEE ALSO

8010 [it_get_consumer_context\(\)](#), [it_context_t](#), [it_handle_t](#)

it_socket_convert()

8011
8012 **NAME**
8013 `it_socket_convert` – convert a connected socket into a connected IT-API Endpoint

8014 SYNOPSIS

```
8015 #include <it_api.h>
8016
8017 it_status_t it_socket_convert(
8018     IN      int      sd,
8019     IN      it_ep_handle_t ep_handle,
8020     IN      it_sc_flags_t flags,
8021     IN const unsigned char* msg,
8022     IN      size_t   len
8023 );
8024
8025 typedef enum {
8026     IT_SC_DEFAULT      = 0x0000,
8027     IT_SC_NO_REQ_REP  = 0x0001,
8028 } it_sc_flags_t;
```

8029 APPLICABILITY

8030 `it_socket_convert` is supported only if the Interface Adapter attribute `socket_conversion_support`
8031 (see `it_ia_info_t`) is `IT_TRUE`.

8032 DESCRIPTION

8033 `sd` Socket descriptor.

8034 `ep_handle` Handle for an instance of the local Endpoint.

8035 `flags` Flags for the Conversion attempt.

8036

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

8037

8038 `msg` Last ULP streaming mode message for delivery to the remote responding
8039 Consumer. Must be NULL (`IT_NO_ADDR`) when the Consumer is in the
8040 Conversion Responder role.

8041 `len` Length of `msg`. This field must be zero when the Consumer is in the
8042 Conversion Responder role.

8043 The `it_socket_convert` routine converts a connected TCP/IP socket (`sd`) into a connected IT-API
8044 Endpoint (`ep_handle`).

8045 Calling *it_socket_convert* with an *sd* parameter not corresponding to a TCP connection or with
8046 an *ep_handle* corresponding to an Endpoint created on an IA where *socket_conversion_support*
8047 is *IT_FALSE* shall yield the return code *IT_ERR_OP_NOT_SUPPORTED*.

8048 This routine is for use by the Consumer both when initiating and when responding to a
8049 Conversion attempt.

8050 The Consumer must provide a connected socket, *sd*, to the *it_socket_convert* routine. Prior to
8051 calling the *it_socket_convert* routine, the Consumer must quiesce all traffic in both directions on
8052 the connection. How the Consumer quiesces the connection is outside the scope of this
8053 specification. Failure to quiesce the connection prior to calling this routine may result in
8054 breaking the TCP/IP connection.

8055 The connection parameter, *sd*, is a socket descriptor as returned by the *socket()* API routine.

8056 After this call returns with *IT_SUCCESS* and before the Conversion process completes, the *sd*
8057 must not be used by the Consumer. Use of the *sd* prior to the Conversion process completing
8058 may abort the Conversion process and cause the generation of a Completion Error. After the
8059 Conversion process has completed, the Consumer can only legally use the *sd* in two other system
8060 calls: they can pass it to the *setsockopt(2)* system call to modify only the *SO_KEEPALIVE*
8061 option, or they pass it to *close(2)* to close it. Support for doing anything else with the *sd* is
8062 Implementation-specific. Support for modifying the *SO_KEEPALIVE* option on a converted
8063 socket is optional; if the Implementation doesn't support this, *setsockopt(2)* will return an error
8064 when the Consumer attempts to do so, but no undefined behavior will result. If the Consumer
8065 attempts to pass *sd* to any other system call besides the two previous specified, or attempts to use
8066 *setsockopt(2)* to modify any option other than *SO_KEEPALIVE*, undefined behavior up to and
8067 including unreported data corruption may result. Calling *close(2)* on the *sd* has no effect upon
8068 the Endpoint.

8069 The Consumer must provide an Endpoint, *ep_handle*, of the Reliable Connected service type to
8070 the *it_socket_convert* routine. The Endpoint must be in the *IT_EP_STATE_UNCONNECTED*
8071 state when input to the routine.

8072 The *flags* parameter for this routine allows the Consumer to control use of underlying features of
8073 the iWARP Transport. Unless stated otherwise, the bitwise OR of any combination of *flags*
8074 values is allowed.

8075 The *flags* value *IT_SC_DEFAULT* causes the underlying implementation to enable MPA
8076 startup.

8077 Setting the bit *IT_SC_NO_REQ_REP* in *flags* allows the Consumer to specify that the
8078 underlying Implementation not use the IETF MPA startup. See [MPA-RDMAC] for more
8079 details.

8080 To initiate the Conversion process, the Consumer must provide a Last ULP Streaming-Mode
8081 Message buffer *msg* of length *len* greater than zero to the *it_socket_convert* routine. The Last
8082 ULP Streaming-Mode Message will be conveyed to the remote Consumer via the underlying
8083 connection. The Consumer is required to supply a Last ULP Streaming-Mode Message to initiate
8084 the Conversion process regardless of whether or not the IETF MPA Startup is used.

8085 Prior to initiating the Conversion process by calling *it_socket_convert* with a non-NULL (i.e.,
8086 not equal to *IT_NO_ADDR*) *msg* buffer, the Consumer should post at least one Receive DTO to
8087 the Endpoint *ep_handle* passed to the *it_socket_convert* routine. Failure to do so may result in
8088 the Endpoint failing to reach the *IT_EP_STATE_CONNECTED* state.

8089 To respond to an incoming Conversion attempt by a remote Consumer, the Consumer must
8090 specify the *msg* parameter as NULL (IT_NO_ADDR) and specify the length, *len*, parameter as
8091 zero. The Consumer should not call the *it_socket_convert* routine in responding mode prior to
8092 consuming the Last ULP Streaming-Mode Message sent by the remote initiating Consumer (e.g.,
8093 the local Consumer should *read()* from the *sd* to consume the remote Consumer's Last ULP
8094 Streaming-Mode Message). The Consumer should receive the peer's Last ULP Streaming-Mode
8095 Message completely so that the *sd* is quiesced prior to calling *it_socket_convert*. How the
8096 Consumer determines the size of the peer's Last ULP Streaming-Mode Message on the
8097 responding side is outside of the scope of this specification. Failure to consume the peer's Last
8098 ULP Streaming-Mode Message completely and thereby to quiesce *sd* before calling
8099 *it_socket_convert* may result in the TCP/IP connection breaking. The Consumer also must not
8100 attempt to send any further streaming mode messages on the *sd* prior to calling *it_socket_convert*
8101 as the Conversion Responder.

8102 How a ULP chooses which side of the connection initiates Conversion and which side responds
8103 to Conversion is outside of the scope of this specification.

8104 In order to safely post RDMA Read operations to an Endpoint, the Consumer must ensure that
8105 the Endpoint at the other end of the converted Connection has compatible *rdma_read_ord*
8106 (ORD) and *rdma_read_ird* (IRD) attributes. See *it_ep_attributes_t* for a description of these
8107 attributes. See Chapter 5 for more details on use of IRD and ORD.

8108 The *it_socket_convert* routine returns control to the Consumer prior to completion of the
8109 Conversion operation. The completion of the Conversion operation is reported asynchronously
8110 via the *connect_sevd_handle* parameter furnished to the *it_ep_rc_create* routine on Endpoint
8111 creation. Successful completion of the Conversion process is indicated by an
8112 IT_CM_MSG_CONN_ESTABLISHED_EVENT queued on the IT_CM_MSG_EVENT_
8113 STREAM SEVD represented by *connect_sevd_handle*. The Consumer may distinguish the
8114 Event corresponding to their Conversion attempt by comparing the *it_ep_handle_t* returned in
8115 the event with that of the Endpoint, *ep_handle*, passed into the *it_socket_convert* routine.

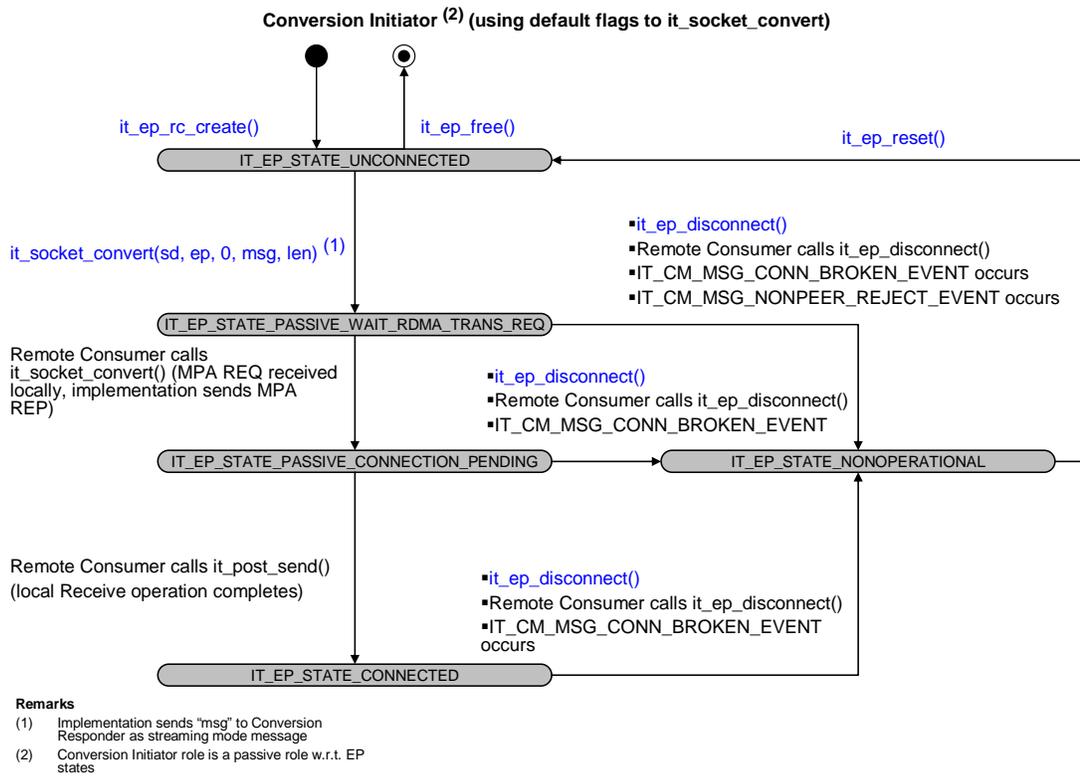
8116 For a complete definition of Endpoint state and a more complete description of the state
8117 transitions, see the Extended Description below. If for any reason an Endpoint Connection fails
8118 to be established the Endpoint will transition into the IT_EP_STATE_NONOPERATIONAL
8119 state and any Receive DTO operations that were successfully posted to the Endpoint will be
8120 completed with an IT_DTO_ERR_FLUSHED status.

8121 The Conversion process may not be completed for the Conversion Initiating Consumer unless
8122 the Conversion Responding Consumer posts a Send message via *it_post_send* to the remote
8123 Endpoint that is connected to, *ep_handle*. The Consumer may abort a Conversion attempt in
8124 progress by making use of the *it_ep_disconnect* routine or the *it_ep_free* routine.

8125 **EXTENDED DESCRIPTION**

8126 The Endpoint state transitions due to the Conversion process depend on the use of the *flags*
8127 parameter.

8128 Setting *flags* to the value IT_SC_DEFAULT yields the following state diagrams for the
8129 initiating and responding sides:

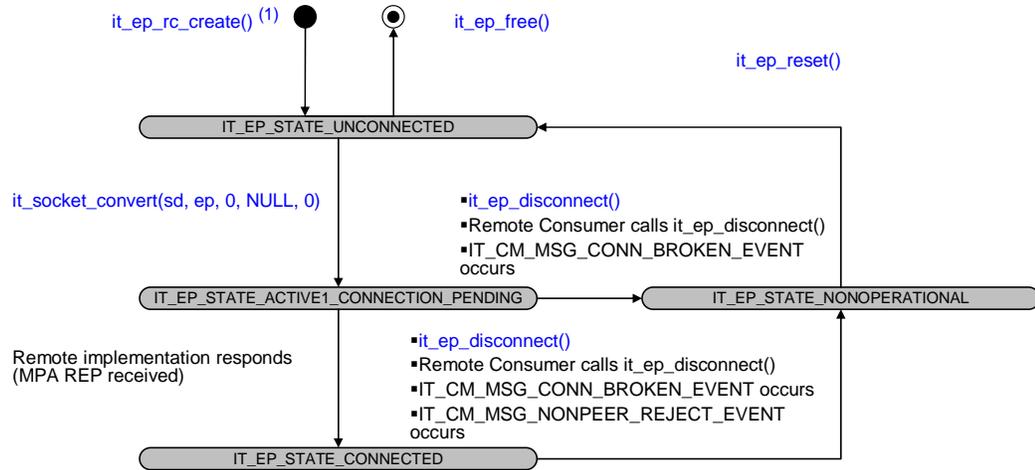


8130

8131

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

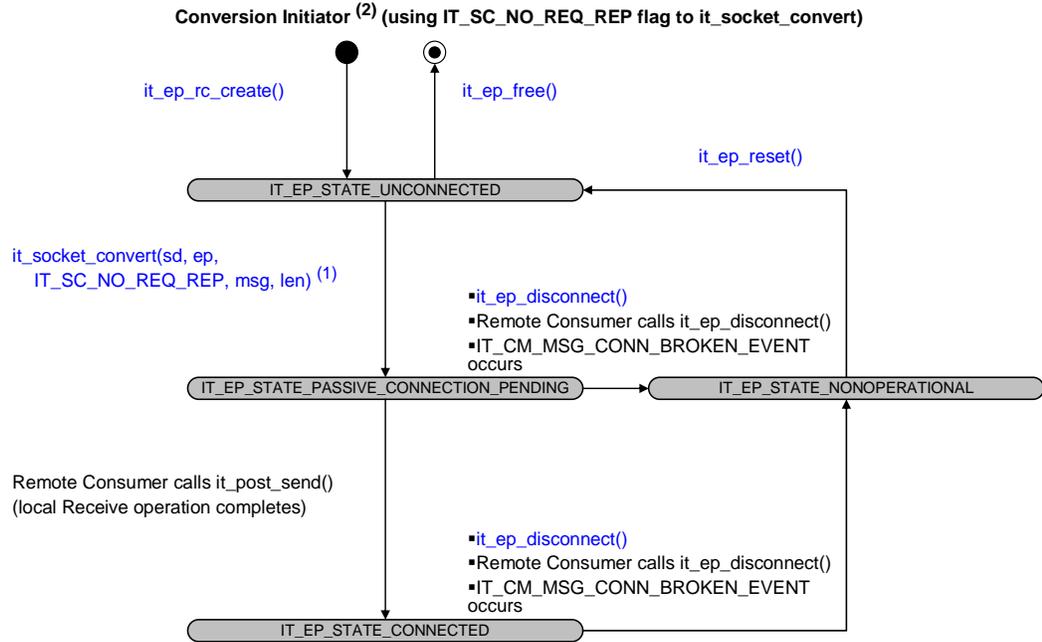
8132

8133

Figure 5: Conversion Responder with *flags* set to `IT_SC_DEFAULT`

8134
8135

Setting the `IT_SC_NO_REQ_REP` bit in *flags* yields the following state diagrams for the initiating and responding sides:



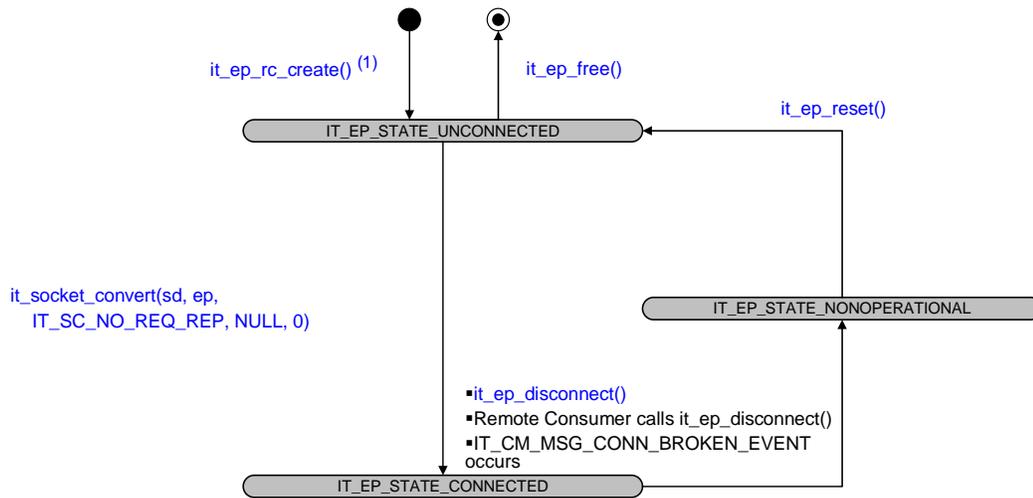
Remarks

- (1) Implementation sends "msg" to Conversion Responder as streaming mode message
- (2) Conversion Initiator role is a passive role w.r.t. EP states

8136
8137

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

8138

8139

Figure 7: Conversion Responder with the IT_SC_NO_REQ_REP bit set in flags

8140

8141

8142

8143

8144

8145

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.

8146

8147

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.

8148

8149

8150

8151

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*).

8152

RETURN VALUE

8153

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

8154

IT_ERR_OP_NOT_SUPPORTED

The operation failed as it is not supported on the IA.

8155

IT_ERR_RESOURCES

The operation failed due to resource limitations.

8156

IT_ERR_INVALID_SD

The socket descriptor *sd* was invalid.

8157	IT_ERR_INVALID_SD_STATE	The socket descriptor <i>sd</i> is not in the proper state to be converted.
8158		
8159	IT_ERR_INVALID_EP	The <i>ep_handle</i> was invalid.
8160	IT_ERR_INVALID_EP_STATE	The Endpoint is not in the proper state to be connected.
8161	IT_ERR_INVALID_EP_TYPE	The Endpoint Service Type does not support this operation.
8162		
8163	IT_ERR_INVALID_FLAGS	The <i>flags</i> values was invalid.
8164	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.
8165		
8166		
8167		

8168 ASYNCHRONOUS ERRORS

8169 If the Conversion Initiator uses a Non-permissive AV-RNIC/IETF and the Conversion
8170 Responder uses an RDMAC AV-RNIC, then an IT_CM_MSG_NONPEER_REJECT_EVENT
8171 with the IT_CN_REJ_BAD_CONN_PARMS reject reason code indicating incompatible
8172 protocol versions will be generated at both Conversion Initiator and Conversion Responder. See
8173 [\[INTEROP-IETF\]](#).

8174 Conversely, if the Conversion Initiator uses an RDMAC AV-RNIC and the Conversion
8175 Responder uses a Non-permissive AV-RNIC/IETF, then an IT_CM_MSG_CONN_BROKEN_
8176 EVENT will be generated at the Conversion Initiator and an IT_CM_MSG_NONPEER_
8177 REJECT_EVENT with an IT_CN_REJ_BAD_CONN_PARMS reject reason code indicating
8178 incompatible protocol versions will be generated at the Conversion Responder. See [\[INTEROP-IETF\]](#).
8179

8180 APPLICATION USAGE

8181 The *it_socket_convert* call may be used for implementing iSER and SDP ULPs.

8182 It is the Consumer's responsibility to monitor forward progress of the Conversion process. The
8183 *it_socket_convert* call will never time out waiting for a reply. The *it_socket_convert* call may
8184 abort if the LLP closes due to an error. Consumers are advised not to perform any operations on
8185 *sd* while the Conversion of *sd* is in progress. Consumers may *close(2)* the *sd* after Conversion,
8186 which indicates only that resources associated with *sd* are no longer needed and has no effect on
8187 the Endpoint provided to *it_socket_convert*.

8188 The iWARP transport currently does not provide a standardized, Endpoint-based interface for
8189 manipulating LLP options after Socket Conversion, such as an iWARP equivalent to the
8190 *setsockopt(2)* system call. Some iWARP Interface Adapters may allow setting a limited set of
8191 LLP options (such as the TCP SO_KEEPALIVE socket option) after a completed Conversion by
8192 *accepting setsockopt(2)* on the *sd* that was provided to *it_socket_convert*. Consumers having a
8193 need to set LLP options are therefore advised to either use *setsockopt(2)* prior to Conversion or
8194 to be prepared for handling failures of *setsockopt(2)* calls if invoked after a completed
8195 Conversion.

8196 SEE ALSO

8197 *it_ep_rc_create()*, *it_ep_modify()*, *it_ep_free()*, *it_ep_disconnect()*, *it_ia_info_t*, *it_post_send()*,
8198 *it_post_recv()*, *it_post_rdma_write()*, *it_post_rdma_read()*

it_srq_create()

8199
8200 **NAME**
8201 `it_srq_create` – create a Shared Receive Queue (S-RQ) on an Interface Adapter

8202 SYNOPSIS

```
8203 #include <it_api.h>  
8204  
8205 it_status_t it_srq_create(  
8206     IN  it_pz_handle_t    pz_handle,  
8207     IN  size_t            max_recv_segments,  
8208     IN  size_t            max_recv_dtos,  
8209     OUT it_srq_handle_t  *srq_handle  
8210 );
```

8211 APPLICABILITY

8212 `it_srq_create` is applicable only to Endpoints created for the RC service type and is supported
8213 only if the Interface Adapter attribute `srq_support` (see `it_ia_info_t`) is `IT_TRUE`. Endpoints of
8214 the UD service type cannot use an S-RQ.

8215 DESCRIPTION

8216 `pz_handle` Protection Zone in which to create the Shared Receive Queue.

8217 `max_recv_segments` Maximum number of data segments for a local buffer that the Consumer
8218 may specify for a posted Receive DTO.

8219 `max_recv_dtos` Maximum number of outstanding Receive DTOs that the Consumer can
8220 have outstanding on the S-RQ.

8221 `srq_handle` Returned Handle for the Shared Receive Queue.

8222 `it_srq_create` creates a Shared Receive Queue on the Interface Adapter implicitly identified by
8223 the input `pz_handle`. The `srq_handle` output is only valid if the call to `it_srq_create` returns
8224 `IT_SUCCESS`. An S-RQ is an IT Object to which Receive DTOs can be posted using the
8225 `it_post_recv` routine. By default when a Reliable Connection Endpoint is created (via the
8226 `it_ep_rc_create` routine) it has its own private Receive Queue, but the Consumer can override
8227 the default and instead specify that the Endpoint should use a Shared Receive Queue that the
8228 Consumer created using the `it_srq_create` routine.

8229 The Protection Zone `pz_handle` allows the Consumer to control what local memory an incoming
8230 Send operation can access. An incoming Send DTO that is paired with a Receive DTO posted to
8231 an S-RQ can only access memory that is in the same Protection Zone as the S-RQ. In contrast,
8232 an incoming RDMA Read or RDMA Write DTO targeted at an Endpoint that is using an S-RQ
8233 can only access memory that is in the same Protection Zone as the Endpoint. The Protection
8234 Zone of the Endpoint and the Protection Zone of the S-RQ that the Endpoint is using are allowed
8235 to be different. See the Application Usage section below for an example of how this could be
8236 used.

8237 When an Endpoint is using an S-RQ and an incoming Send operation targets that Endpoint, the
8238 Implementation shall attempt to dequeue a Receive DTO from the S-RQ and provide it to the
8239 targeted Endpoint for further processing. The semantics associated with the processing of the

8240 Receive operation are described in the *it_post_recv* routine. If an incoming Send must be
8241 processed and no Receive DTO is available to be dequeued in the S-RQ, or if an error occurs
8242 while processing the Receive DTO (e.g., the Send DTO it gets paired with contains more data
8243 than the Receive DTO can handle) the Connection of the targeted Endpoint shall be broken and
8244 the Endpoint moved to the IT_EP_STATE_NONOPERATIONAL state, but other Endpoints
8245 that are using the same S-RQ shall not be affected. Endpoints that use a common S-RQ maintain
8246 the same isolation from errors on other Endpoints that they would have if they weren't using an
8247 S-RQ.

8248 When an S-RQ is first created, the S-RQ Low Watermark mechanism is disabled. It can only be
8249 enabled by calling *it_srq_modify* to explicitly enable it.

8250 If the call to *it_srq_create* returns IT_SUCCESS the total number of segments allowed in a
8251 Receive operation posted to the S-RQ will be greater than or equal to the *max_recv_segments*
8252 that were requested. To find out exactly how many segments will be allowed, use *it_srq_query*.
8253 Similarly, if the call to *it_srq_create* returns IT_SUCCESS the total number of Receive DTOs
8254 that can be outstanding for the S-RQ will be greater than or equal to the *max_recv_dtos* that
8255 were requested. To find out exactly how many Receive operations can safely be outstanding, use
8256 *it_srq_query*. (If the Consumer attempts to have more Receive DTOs outstanding on an S-RQ
8257 than is given by *it_srq_query*, the Implementation may refuse to allow more Receive DTOs to be
8258 posted; see *it_post_recv* for details.)

8259 RETURN VALUE

8260 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

8261	IT_ERR_INVALID_PZ	The Protection Zone Handle (<i>pz_handle</i>) was invalid.
8262	IT_ERR_SRQ_NOT_SUPPORTED	The IA associated with <i>pz_handle</i> does not support Shared Receive Queues.
8263		
8264	IT_ERR_RESOURCES	The requested operation failed due to insufficient resources.
8265		
8266	IT_ERR_RESOURCE_RECV.DTO	The underlying transport could not allocated the requested <i>max_recv_dtos</i> resources at this time.
8267		
8268	IT_ERR_RESOURCE_RSEG	The underlying transport could not allocated the requested <i>max_recv_segments</i> resources at this time.
8269		
8270	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.
8271		
8272		
8273		

8274 APPLICATION USAGE

8275 If a Consumer wishes to use an S-RQ they must first create the S-RQ by calling *it_srq_create*.
8276 They can then furnish the handle for the S-RQ as one of the attributes of the Endpoint when
8277 creating the Endpoint using the *it_ep_rc_create* routine.

8278 Before creating a connection with an Endpoint that is using an S-RQ the Consumer will typically
8279 ensure that the Endpoint that it is connecting will be able to process at least one incoming Send
8280 operation by using the *it_post_recv* routine to post a Receive DTO to the S-RQ.

8281 One example of a situation where it could be useful to allow the Protection Zone associated with
8282 an S-RQ and the Protection Zone associated with an Endpoint that is using the S-RQ to be
8283 different is when multiple clients are performing RDMA Write operations to the same server and
8284 the server wants to prevent an RDMA Write operation from one client from accessing the
8285 RDMA target buffer associated with a different client. Assume that we have two clients C1 and
8286 C2, and that the server processes incoming DTOs from C1 through Endpoint E1, and similarly
8287 processes incoming DTOs from C2 through Endpoint E2. The server can then prevent C1 from
8288 accessing C2's RDMA target buffer (and *vice versa*) by placing the RDMA target buffer for C1
8289 in Protection Zone P1 and assigning P1 to E1, and by placing the RDMA target buffer for C2 in
8290 Protection Zone P2 and assigning P2 to E2. The server also needs to receive incoming Send
8291 operations from both C1 and C2, and may therefore decide to have both E1 and E2 share a single
8292 S-RQ. Since an S-RQ and all Receive buffers used on the S-RQ must have the same Protection
8293 Zone, the PZ associated with the S-RQ will be one of P1 or P2 and thus be different from the PZ
8294 of either E2 or E1, respectively. The PZ associated with the S-RQ could also be a third PZ that is
8295 different from P1 and P2.

8296 **SEE ALSO**

8297 *it_srq_free(), it_srq_query(), it_srq_modify(), it_post_rcv(), it_ep_rc_create()*

it_srq_free()

8298

8299 NAME

8300 `it_srq_free` – destroy a Shared Receive Queue

8301 SYNOPSIS

```
8302 #include <it_api.h>
8303
8304 it_status_t it_srq_free(
8305     IN it_srq_handle_t srq_handle
8306 );
```

8307 APPLICABILITY

8308 `it_srq_free` is applicable only to Endpoints created for the RC service type and is supported only
8309 if the Interface Adapter attribute `srq_support` (see [it_ia_info_t](#)) is IT_TRUE. Endpoints of the
8310 UD service type cannot use an S-RQ.

8311 DESCRIPTION

8312 `srq_handle` Handle of the Shared Receive Queue to be destroyed.

8313 The `it_srq_free` routine destroys the Shared Receive Queue `srq_handle`. On return, the Handle
8314 `srq_handle` may no longer be used. A Shared Receive Queue may not be destroyed if it is still
8315 being used by an Endpoint; an attempt to do so will fail and the S-RQ will not be affected.

8316 If this routine returns success, for each Receive DTO that was outstanding on the S-RQ the
8317 Consumer is guaranteed that either a Completion Event has been enqueued or that no
8318 Completion Event will be enqueued. In addition, the Implementation shall release ownership of
8319 the local buffers associated with all Receive DTOs that were outstanding on the S-RQ back to
8320 the Consumer.

8321 RETURN VALUE

8322 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

8323	IT_ERR_INVALID_SRQ	The Shared Receive Queue Handle <code>srq_handle</code> was
8324		invalid.
8325	IT_ERR_SRQ_BUSY	The Shared Receive Queue was still referenced by an
8326		Endpoint.
8327	IT_ERR_IA_CATASTROPHE	The IA has experienced a catastrophic error and is in the
8328		disabled state. None of the output parameters from this
8329		routine are valid. See it_ia_info_t for a description of the
8330		disabled state.

8331 SEE ALSO

8332 [it_srq_create](#), [it_srq_query\(\)](#), [it_srq_modify\(\)](#)

it_srq_modify()

8333

8334 NAME

8335 `it_srq_modify` – modify selected attributes of a Shared Receive Queue

8336 SYNOPSIS

```
8337 #include <it_api.h>
8338
8339 it_status_t it_srq_modify(
8340     IN          it_srq_handle_t      srq_handle,
8341     IN          it_srq_param_mask_t mask,
8342     IN  const   it_srq_param_t      *params
8343 );
```

8344 APPLICABILITY

8345 `it_srq_modify` is applicable only to Endpoints created for the RC service type and is supported
8346 only if the Interface Adapter attribute `srq_support` (see `it_ia_info_t`) is `IT_TRUE`. Endpoints of
8347 the UD service type cannot use an S-RQ.

8348 DESCRIPTION

8349 `srq_handle` The Shared Receive Queue whose attributes are to be modified.

8350 `mask` Bitwise OR of flags for specified parameters.

8351 `params` Structure whose members contain the new parameter values.

8352 The `it_srq_modify` routine changes selected attributes of the Shared Receive Queue `srq_handle`.
8353 Attributes to be modified are specified by flags in `mask`. New values for the attributes are
8354 specified by the corresponding fields in the structure pointed to by `params`. Fields and their
8355 corresponding flag values are shown in `it_srq_param_t`. Note that attributes represented by
8356 fields of `it_srq_param_t` that are not shown below cannot be modified. The definition of each
8357 field follows:

8358 `max_recv_dtos` The maximum number of Receive DTOs that the Consumer can safely have
8359 outstanding on the S-RQ. This can only be modified if the IA associated
8360 with the S-RQ supports dynamically resizing the S-RQ. See `it_ia_info_t`
8361 for details of how to determine this. If the Consumer attempts to modify this to
8362 a value that is less than the total number of Receive DTOs that are currently
8363 enqueued within the S-RQ, an error shall be returned.

8364 `low_watermark` The S-RQ Low Watermark threshold. If the Consumer attempts to modify
8365 this to a value of zero, an error shall be returned. If the Consumer attempts
8366 to modify this to a value that is greater than `max_recv_dtos`, an error shall
8367 be returned. If the Consumer modifies this to any non-zero value (including
8368 the existing non-zero value if the S-RQ Low Watermark mechanism is
8369 already armed) and `it_srq_modify` returns success, the S-RQ Low
8370 Watermark mechanism shall be armed/rearmed. Once the mechanism is
8371 armed, it shall remain armed until either the S-RQ is destroyed by a call to
8372 `it_srq_free`, or the total number of Receive Work Requests outstanding on
8373 the S-RQ falls below the threshold. If the mechanism is disarmed because
8374 the total number of Receive Work Requests outstanding on the S-RQ falls
8375 below this threshold, an Affiliated Asynchronous Event shall be enqueued

8376 on the Affiliated Asynchronous Event Stream for the IA that the S-RQ is
8377 associated with; see *it_affiliated_event_t* for details. Once the S-RQ Low
8378 Watermark mechanism is disarmed, it can only be rearmed by calling
8379 *it_srq_modify* to establish a new *low_watermark* value.

8380 Modifying the maximum number of Receive DTOs that the S-RQ can safely have outstanding
8381 while there are Endpoints actively using the S-RQ can have an adverse impact upon
8382 performance, and can potentially cause the Connections on Endpoints that are using the S-RQ to
8383 break. The Consumer is therefore advised to only modify this attribute of the S-RQ when data
8384 transfer activity on the Endpoints associated with the S-RQ has been quiesced to minimize the
8385 risk of Connections breaking.

8386 The Consumer is allowed to call *it_srq_modify* to modify the S-RQ Low Watermark for an S-
8387 RQ whose S-RQ Low Watermark mechanism is already armed. The result of doing so is that the
8388 mechanism shall remain armed, but the new S-RQ Low Watermark value shall replace the old
8389 one, subject to the normal bounds checking of the supplied S-RQ Low Watermark value that is
8390 done by the *it_srq_modify* routine.

8391 *it_srq_modify* shall succeed in modifying all the requested attributes atomically; if the attempt to
8392 modify any of the requested attributes generates an error, none of the other attributes supplied to
8393 the call will be applied.

8394 **RETURN VALUE**

8395 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

8396	IT_ERR_INVALID_SRQ	The Shared Receive Queue Handle <i>srq_handle</i> was
8397		invalid.
8398	IT_ERR_INVALID_MASK	The <i>mask</i> contained invalid flag values.
8399	IT_ERR_RESOURCES	The requested operation failed due to insufficient
8400		resources.
8401	IT_ERR_INVALID_SRQ_SIZE	An attempt was made to set the total number of entries in
8402		the S-RQ (<i>max_rcv_dtos</i>) to a value that is less than the
8403		total number of Receive DTOs currently enqueued in the
8404		S-RQ.
8405	IT_ERR_SRQ_LOW_WATERMARK	An attempt was made to either set the S-RQ Low
8406		Watermark (<i>low_watermark</i>) to a value greater than
8407		<i>max_rcv_dtos</i> , or to set the S-RQ Low Watermark to a
8408		value of zero.
8409	IT_ERR_IA_CATASTROPHE	The IA has experienced a catastrophic error and is in the
8410		disabled state. None of the output parameters from this
8411		routine are valid. See <i>it_ia_info_t</i> for a description of the
8412		disabled state.

8413 **SEE ALSO**

8414 *it_srq_create()*, *it_srq_free()*, *it_srq_query()*

it_srq_query()

8415

8416 NAME

8417 `it_srq_query` – get attributes of a Shared Receive Queue

8418 SYNOPSIS

```
8419 #include <it_api.h>
8420
8421 it_status_t it_srq_query(
8422     IN    it_srq_handle_t    srq_handle,
8423     IN    it_srq_param_mask_t mask,
8424     OUT   it_srq_param_t     *params
8425 );
8426
8427 typedef enum {
8428     IT_SRQ_PARAM_ALL           = 0x000001,
8429     IT_SRQ_PARAM_IA           = 0x000002,
8430     IT_SRQ_PARAM_PZ           = 0x000004,
8431     IT_SRQ_PARAM_MAX_RECV_DTO = 0x000008,
8432     IT_SRQ_PARAM_MAX_RECV_SEG = 0x000010,
8433     IT_SRQ_PARAM_LOW_WATERMARK = 0x000020
8434 } it_srq_param_mask_t;
8435
8436 typedef struct {
8437     it_ia_handle_t ia;           /* IT_SRQ_PARAM_IA */
8438     it_pz_handle_t pz;           /* IT_SRQ_PARAM_PZ */
8439     size_t max_recv_dtos;       /* IT_SRQ_PARAM_MAX_RECV_DTO */
8440     size_t max_recv_segs;       /* IT_SRQ_PARAM_MAX_RECV_SEG */
8441     size_t low_watermark;       /* IT_SRQ_PARAM_LOW_WATERMARK */
8442 } it_srq_param_t;
```

8443 APPLICABILITY

8444 `it_srq_query` is applicable only to Endpoints created for the RC service type and is supported
8445 only if the Interface Adapter attribute `srq_support` (see `it_ia_info_t`) is `IT_TRUE`. Endpoints of
8446 the UD service type cannot use an S-RQ.

8447 DESCRIPTION

8448 `srq_handle` The Shared Receive Queue whose attributes are being queried.

8449 `mask` Bitwise OR of flags for desired parameters.

8450 `params` Structure whose members are written with the desired parameters.

8451 The `it_srq_query` routine returns the desired attributes of the Shared Receive Queue `srq_handle`
8452 in the structure pointed to by `params`. On return, each field of `params` is only valid if the
8453 corresponding flag as shown in the Synopsis is set in the `mask` argument. The `mask` value
8454 `IT_SRQ_PARAM_ALL` causes all fields to be returned.

8455 The definition of each field of `params` follows:

8456 `ia` The Interface Adapter Handle associated with the S-RQ.

8457 `pz` The Protection Zone Handle specified to create the S-RQ.

8458	<i>max_rcv_dtos</i>	The maximum number of Receive DTOs that the Consumer can safely have outstanding on the S-RQ.
8459		
8460	<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.
8461		
8462	<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.
8463		
8464		

8465 **RETURN VALUE**

8466 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

8467	IT_ERR_INVALID_SRQ	The Shared Receive Queue Handle (<i>srq_handle</i>) was invalid.
8468	IT_ERR_INVALID_MASK	The <i>mask</i> contained invalid flag values.
8469	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.
8470		
8471		

8472 **SEE ALSO**

8473 *it_srq_create()*, *it_srq_free()*, *it_srq_modify()*

8474

it_ud_service_reply()

8475 NAME

8476 it_ud_service_reply – return the information necessary to communicate via Unreliable Datagram
8477 (UD) messages with the entity specified by the Connection Qualifier in the UD Service Request
8478 Event

8479 SYNOPSIS

```

8480 #include <it_api.h>
8481
8482 it_status_t it_ud_service_reply (
8483     IN          it_ud_svc_req_identifier_t  ud_svc_req_id,
8484     IN          it_ud_svc_req_status_t     status,
8485     IN          it_remote_ep_info_t       ep_info,
8486     IN const    unsigned char             *private_data,
8487     IN          size_t                     private_data_length
8488 );
8489
8490 typedef uint64_t it_ud_svc_req_identifier_t;

```

8491 APPLICABILITY

8492 *it_ud_service_reply* is applicable only to the UD service type.

8493 DESCRIPTION

8494 8495 8496	<i>ud_svc_req_id</i>	Unique identifier from the IT_CM_REQ_UD_SERVICE_REQUEST_EVENT Event generated from the UD Service Request that is being responded to with this invocation of <i>it_ud_service_reply</i> .
8497 8498	<i>status</i>	Status to return in the IT_CM_MSG_UD_SERVICE_REPLY_EVENT data indicating the outcome of the UD Service Request.
8499 8500	<i>ep_info</i>	End-point information to be used by the UD Service Requester to communicate with this UD Service.
8501 8502 8503	<i>private_data</i>	Opaque Private Data provided by the Consumer which will be sent as part of the <i>it_ud_service_reply</i> . If the IA does not support Private Data, <i>private_data_length</i> must be 0.
8504 8505	<i>private_data_length</i>	Length of the <i>private_data</i> provided by the Consumer. If the IA does not support Private Data, this field must be 0.

8506 The *it_ud_service_reply* routine will be called by the Consumer to respond to an
8507 IT_CM_REQ_UD_SERVICE_REQUEST_EVENT Event. The IT_CM_REQ_UD_SERVICE_
8508 REQUEST_EVENT Event data (*it_ud_svc_request_event_t*) contains a unique Service Request
8509 Handle, the Connection Qualifier of interest, Source address information, and optional Private
8510 Data. The recipient of the IT_CM_REQ_UD_SERVICE_REQUEST_EVENT Event needs to
8511 respond to the request by calling *it_ud_service_reply*.

8512 The *ud_svc_req_id* is a unique identifier allowing this response to be correlated to the request
8513 being responded to. The *ud_svc_req_id* should be copied from the IT_CM_REQ_UD_
8514 SERVICE_REQUEST_EVENT Event data, *ud_svc_req_id* field. Once *it_ud_service_reply* has

8515 been successfully invoked, the supplied *ud_svc_req_id* is no longer valid. The resources
8516 associated with the *ud_svc_req_id* are released and the *ud_svc_req_id* cannot be re-used.

8517 Valid status codes for the *status* field are defined in *it_ud_svc_req_status_t*. A valid status code
8518 must be provided. IT_UD_REQ_REDIRECTED cannot be supplied as input for this parameter,
8519 even though it may appear in the Event given to the requester. The Implementation is
8520 responsible for redirection, not the Consumer.

8521 The *ep_info* is only used by this routine if the *status* field is set to IT_UD_SVC_EP_
8522 INFO_VALID. See *it_ud_svc_req_status_t* for details.

8523 The IA can be queried via *it_ia_query* to determine whether it supports the transfer of Private
8524 Data. This is indicated by the *private_data_support* field of the *it_ia_info_t* structure. If Private
8525 Data is not supported, *private_data_length* must be 0. The maximum length of *private_data* can
8526 be determined by examining the *ud_rep_private_data_len* member of the *it_ia_info_t* structure.

8527 EXTENDED DESCRIPTION

8528 *it_ud_service_reply* is called by the Consumer in response to receiving an IT_CM_REQ_UD_
8529 SERVICE_REQUEST_EVENT Event. The Consumer chooses how to respond to the Service
8530 Request and makes that choice known via the value of *status* passed into the *it_ud_service_reply*
8531 call. The value of *status* determines whether the Implementation uses the *ep_info* input
8532 parameter. The table below describes the meaning of each *status* value, and whether the
8533 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.

8534 In order for the Implementation to be able to correctly correlate this *it_ud_service_reply* call
8535 with the request Event being responded to, the Consumer must supply the *ud_svc_req_id* from
8536 the *it_ud_svc_request_event_t* as the *ud_svc_req_id* passed into the *it_ud_service_reply* call.
8537

8538 It is possible to receive duplicate UD Service Requests as a result of the active side retrying an
8539 *it_ud_service_request* operation. It is the Consumer's responsibility to detect and handle
8540 duplicate requests. Requests are uniquely identified by a combination of the *ud_svc_req_id* and

8541 the *source_addr* from the *it_ud_svc_request_event_t* data. This combination can be used to
8542 detect duplicate UD Service Requests.

8543 **RETURN VALUE**

8544 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

8545 IT_ERR_PDATA_NOT_SUPPORTED Private Data was supplied by the Consumer, but this
8546 Interface Adapter does not support Private Data.

8547 IT_ERR_INVALID_PDATA_LENGTH The Interface Adapter supports Private Data, but the
8548 length specified exceeded the Interface Adapter's
8549 capabilities.

8550 IT_ERR_INVALID_UD_SVC_REQ_ID The Unreliable Datagram Service Request ID
8551 (*ud_svc_req_id*) was invalid.

8552 IT_ERR_INVALID_UD_STATUS The Unreliable Datagram Service Request *status* was
8553 invalid.

8554 IT_ERR_IA_CATASTROPHE The Interface Adapter has experienced a catastrophic
8555 error and is in the disabled state. None of the output
8556 parameters from this routine are valid. See *it_ia_info_t*
8557 for a description of the disabled state.

8558 **SEE ALSO**

8559 *it_ia_query()*, *it_ud_service_request()*, *it_ep_attributes_t*, *it_cm_msg_events*, *it_cm_req_events*

it_ud_service_request()

8560

8561 NAME

8562 it_ud_service_request – request that the recipient of this message return the information
8563 necessary to communicate via Unreliable Datagram (UD) messages to the entity specified by the
8564 UD Service Handle

8565 SYNOPSIS

```
8566 #include <it_api.h>
8567
8568 it_status_t it_ud_service_request (
8569     IN it_ud_svc_req_handle_t ud_svc_handle
8570 );
```

8571 APPLICABILITY

8572 *it_ud_service_request* is applicable only to the UD service type.

8573 DESCRIPTION

8574 *ud_svc_handle* UD Service Request Handle created by a call to *it_ud_service_request_*
8575 *handle_create*. This Handle uniquely identifies this UD Service Request
8576 operation. The UD Service Request Handle is associated with a specific UD
8577 Service described during the creation of the UD Service Request Handle.

8578 The *it_ud_service_request* routine is called by a Consumer to request a remote entity specified
8579 by the UD Service Handle to return information necessary to communicate via Unreliable
8580 Datagram messages.

8581 The *ud_svc_handle* provides the Consumer with a means of correlating this
8582 *it_ud_service_request* with the IT_CM_MSG_UD_SERVICE_REPLY_EVENT Event that the
8583 Consumer will receive when the remote Endpoint responds to this UD Service Request. See
8584 *it_cm_msg_events*.

8585 Due to the nature of Unreliable Datagrams, even though an invocation of *it_ud_service_request*
8586 returns success, the target of the UD Service Request may not receive it. Therefore, the
8587 Consumer may have to call *it_ud_service_request* multiple times with the same *ud_svc_handle*
8588 before the recipient actually receives the request and is able to reply to it. In addition, if the
8589 Consumer issues multiple requests with the same *ud_svc_handle*, the Consumer may receive
8590 multiple replies. It is up to the Consumer to detect and handle duplicate replies.

8591 The *ud_svc_req_id* (*it_ud_svc_req_identifier_t*) associated with a given *ud_svc_handle* does not
8592 change. Therefore, all retries using a given *ud_svc_handle* will result in the same *ud_svc_req_id*
8593 being presented to the recipient of the IT_CM_REQ_UD_SERVICE_REQUEST_EVENT Event
8594 in the Event data (*it_ud_svc_request_event_t*).

8595 Upon a successful invocation and transmission of the *it_ud_service_request*, once the recipient
8596 of the request replies via *it_ud_service_reply*, the Consumer will receive an
8597 IT_CM_MSG_UD_SERVICE_REPLY_EVENT Event. The IT_CM_MSG_UD_SERVICE_
8598 REPLY_EVENT Event data (*it_ud_svc_reply_event_t*) contains the results of the Service
8599 Request query. The *status* field of the *it_ud_svc_reply_event_t* structure in the
8600 IT_CM_MSG_UD_SERVICE_REPLY_EVENT indicates the state of the information in the
8601 *it_ud_svc_reply_event_t* structure. See *it_cm_msg_events*.

8602 **EXTENDED DESCRIPTION**

8603 The *ud_service_request* call requests information from the remote UD Service. Once that remote
 8604 UD Service responds, an IT_CM_MSG_UD_SERVICE_REPLY Event will be generated. The
 8605 data associated with the Event, *it_ud_svc_reply_event_t*, contains information the Consumer
 8606 needs in order to perform Data Transfer Operations with the remote UD Service. The *status*
 8607 (*it_ud_svc_req_status_t*) field of the *it_ud_svc_reply_event_t* indicates the validity of other
 8608 fields in the structure. The *status* field should be checked by the Consumer prior to making any
 8609 assumptions about the data in the rest of the structure. The table below summarizes the *status*
 8610 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.

8611 **RETURN VALUE**

8612 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

8613 IT_ERR_INVALID_UD_SVC The Unreliable Datagram Service Handle (*ud_svc_handle*) was
8614 invalid.

8615 IT_ERR_IA_CATASTROPHE The Interface Adapter has experienced a catastrophic error and is
8616 in the disabled state. None of the output parameters from this
8617 routine are valid. See *it_ia_info_t* for a description of the disabled
8618 state.

8619 **SEE ALSO**

8620 *it_ud_service_request_handle_create()*, *it_ud_service_reply()*, *it_cm_msg_events*, *it_path_t*,
8621 *it_ep_attributes_t*

8622 **it_ud_service_request_handle_create()**

8623 **NAME**

8624 `it_ud_service_request_handle_create` – create an Unreliable Datagram (UD) Service Request
8625 Handle

8626 **SYNOPSIS**

```
8627 #include <it_api.h>
8628
8629 it_status_t it_ud_service_request_handle_create (
8630     IN  const  it_conn_qual_t      *conn_qual,
8631     IN  it_evd_handle_t           reply_evd,
8632     IN  const  it_path_t          *destination_path,
8633     IN  const  unsigned char      *private_data,
8634     IN  size_t   private_data_length,
8635     OUT          it_ud_svc_req_handle_t *ud_svc_handle
8636 );
```

8637 **APPLICABILITY**

8638 `it_ud_service_request_handle_create` is applicable only to the UD service type.

8639 **DESCRIPTION**

8640 `conn_qual` The Connection Qualifier describing the UD Service for which the
8641 Consumer is requesting information.

8642 `reply_evd` The Simple EVD on which the IT_CM_MSG_UD_SERVICE_
8643 REPLY_EVENT Event will be received. `reply_evd` must be of the
8644 IT_CM_MSG_EVENT_STREAM Event Stream Type. See
8645 [it_cm_msg_events](#).

8646 `destination_path` `destination_path` specifies a Path to the Destination of the
8647 [it_ud_service_request](#) operation.

8648 `private_data` Opaque Private Data provided by the Consumer which will be sent as part
8649 of the [it_ud_service_request](#). If the IA does not support Private Data,
8650 `private_data_length` must be 0.

8651 `private_data_length`: Length of the `private_data` provided by the Consumer. If the IA does not
8652 support Private Data, this field must be 0.

8653 `ud_svc_handle` UD Service Request Handle created by this call. This Handle will be used
8654 in a call to [it_ud_service_request](#).

8655 The `it_ud_service_request_handle_create` routine is called by the Consumer to create an
8656 Unreliable Datagram Service Request Handle to be used in a call to [it_ud_service_request](#).

8657 The `destination_path` can be obtained by calling [it_get_pathinfo](#). The `spigot_id` in the `it_path_t`
8658 will be the Spigot Identifier used for this UD Service Request.

8659 The IA can be queried via [it_ia_query](#) to determine whether it supports the transfer of Private
8660 Data. This is indicated by the `private_data_support` field of the `it_ia_info_t` structure. If Private
8661 Data is not supported, `private_data_length` must be 0. The maximum length of `private_data` can
8662 be determined by examining the `ud_req_private_data_len` member of the `it_ia_info_t` structure.

8663 The returned *ud_svc_handle* is used to identify the UD Service Request. It provides the
 8664 Consumer with a means of correlating this *it_ud_service_request* with the
 8665 IT_CM_MSG_UD_SERVICE_REPLY_EVENT Event that the Consumer will receive when the
 8666 remote Endpoint responds to this UD Service Request.

8667 The *ud_svc_req_id* (*it_ud_svc_req_identifier_t*) associated with a given *ud_svc_handle* does not
 8668 change. Therefore, all retries using a given *ud_svc_handle* will result in the same *ud_svc_req_id*
 8669 being presented to the recipient of the IT_CM_REQ_UD_SERVICE_REQUEST_EVENT Event
 8670 in the Event data (*it_ud_svc_request_event_t*).

8671 **EXTENDED DESCRIPTION**

8672 The members of the *it_path_t* structure that are pertinent for creating a UD Service Request
 8673 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

8674 **RETURN VALUE**

8675 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

8676 IT_ERR_INVALID_CONN_EVD	The Connection Simple Event Dispatcher Handle was 8677 invalid.
8678 IT_ERR_INVALID_EVD_TYPE	The Event Stream Type for the Event Dispatcher was 8679 invalid.
8680 IT_ERR_PDATA_NOT_SUPPORTED	Private Data was supplied by the Consumer, but this 8681 Interface Adapter does not support Private Data.
8682 IT_ERR_INVALID_PDATA_LENGTH	The Interface Adapter supports Private Data, but the 8683 length specified exceeded the Interface Adapter's 8684 capabilities.
8685 IT_ERR_INVALID_CONN_QUAL	The Connection Qualifier (<i>conn_qual</i>) was invalid.
8686 IT_ERR_INVALID_SOURCE_PATH	One of the components of the Source portion of the 8687 supplied Path was invalid.
8688 IT_ERR_INVALID_SPIGOT	An invalid Spigot ID was specified (<i>spigot_id</i> member of 8689 the <i>destination_path</i>).
8690 IT_ERR_RESOURCES	The requested operation failed due to insufficient 8691 resources.
8692 IT_ERR_IA_CATASTROPHE	The Interface Adapter has experienced a catastrophic 8693 error and is in the disabled state. None of the output

8694 parameters from this routine are valid. See *it_ia_info_t*
8695 for a description of the disabled state.

8696 **APPLICATION USAGE**

8697 The resulting *ud_svc_handle* (*it_ud_svc_req_handle_t*) produced by this call will be used in
8698 calls to *it_ud_service_request* to obtain information describing how to communicate with the
8699 remote UD Service described by *conn_qual* (*it_conn_qual_t*).

8700 The *it_ud_service_request* call requests information from the remote UD Service. Once that
8701 remote UD Service responds, an IT_CM_MSG_UD_SERVICE_REPLY Event will be
8702 generated. The data associated with the Event, *it_ud_svc_reply_event_t*, contains an
8703 *it_remote_ep_info_t* structure and other information. The *ud_ep_id* and *ud_ep_key* from the
8704 *it_remote_ep_info_t*, combined with the information from the *destination_path* (*it_path_t*),
8705 provides the Consumer the necessary information to perform Data Transfer Operations with the
8706 remote UD Service.

8707 Note that the *spigot_id* of the Endpoint that will be used for Data Transfer Operations with the
8708 UD Service being requested must match the *spigot_id* in the *destination_path*.

8709 See *it_ud_service_request* and *it_ud_service_reply* for more information.

8710 **SEE ALSO**

8711 *it_ud_service_request_handle_free()*, *it_ud_request_handle_query()*, *it_ia_query()*,
8712 *it_ud_service_request()*, *it_get_pathinfo()*, *it_path_t*, *it_cm_msg_events*, *it_ep_attributes_t*

8713 **it_ud_service_request_handle_free()**

8714 **NAME**

8715 `it_ud_service_request_handle_free` – free a previously created `it_ud_svc_req_handle_t`

8716 **SYNOPSIS**

```
8717 #include <it_api.h>
8718
8719 it_status_t it_ud_service_request_handle_free (
8720     IN it_ud_svc_req_handle_t ud_svc_handle
8721 );
```

8722 **APPLICABILITY**

8723 `it_ud_service_request_handle_free` is applicable only to the UD service type.

8724 **DESCRIPTION**

8725 `ud_svc_handle` Unreliable Datagram (UD) Service Request Handle previously created by a
8726 call to `it_ud_service_request_handle_create`.

8727 `it_ud_service_request_handle_free` removes an existing UD Service Request Handle and frees
8728 all associated underlying resources. Once `it_ud_service_request_handle_free` returns,
8729 `ud_svc_handle` can no longer be used in UD Service Request operations. In addition, once
8730 `it_ud_service_request_handle_free` returns, any replies to outstanding UD Service Request
8731 operations associated with this `ud_svc_handle` will be silently dropped.

8732 Any IT_CM_MSG_UD_SERVICE_REPLY_EVENT Events associated with this request that
8733 have been enqueued on the Event Dispatcher (EVD) will not be removed. It is the Consumer's
8734 responsibility to dequeue and dispose of them.

8735 **RETURN VALUE**

8736 A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:

8737 IT_ERR_INVALID_UD_SVC The Unreliable Datagram Service Handle (`ud_svc_handle`) was
8738 invalid.

8739 IT_ERR_IA_CATASTROPHE The Interface Adapter has experienced a catastrophic error and
8740 is in the disabled state. None of the output parameters from this
8741 routine are valid. See `it_ia_info_t` for a description of the
8742 disabled state.

8743 **SEE ALSO**

8744 `it_ud_service_request_handle_create()`, `it_ud_service_request_handle_query()`,
8745 `it_cm_msg_events`

8746

it_ud_service_request_handle_query()

8747 NAME

8748 `it_ud_service_request_handle_query` – return information about a specified
8749 `it_ud_svc_req_handle_t`

8750 SYNOPSIS

```
8751 #include <it_api.h>
8752
8753 it_status_t it_ud_service_request_handle_query (
8754     IN    it_ud_svc_req_handle_t    ud_svc_handle,
8755     IN    it_ud_svc_req_param_mask_t mask,
8756     OUT   it_ud_svc_req_param_t    *ud_svc_handle_info
8757 );
8758
8759 typedef enum {
8760     IT_UD_PARAM_ALL                = 0x00000001,
8761     IT_UD_PARAM_IA_HANDLE          = 0x00000002,
8762     IT_UD_PARAM_REQ_ID             = 0x00000004,
8763     IT_UD_PARAM_REPLY_EVD         = 0x00000008,
8764     IT_UD_PARAM_CONN_QUAL         = 0x00000010,
8765     IT_UD_PARAM_DEST_PATH         = 0x00000020,
8766     IT_UD_PARAM_PRIV_DATA         = 0x00000040,
8767     IT_UD_PARAM_PRIV_DATA_LENGTH  = 0x00000080
8768 } it_ud_svc_req_param_mask_t;
8769
8770 /*
8771  * The it_ud_svc_req_param_mask_t value in the comment above
8772  * each attribute in the it_ud_svc_req_param_t structure below
8773  * is the mask value used to select that attribute in a call
8774  * to it_ud_service_request_handle_query.
8775  */
8776 typedef struct {
8777     it_ia_handle_t    ia;                /* IT_UD_PARAM_IA_HANDLE */
8778     uint32_t          request_id;        /* IT_UD_PARAM_REQ_ID */
8779     it_evd_handle_t   reply_evd;        /* IT_UD_PARAM_REPLY_EVD */
8780     it_conn_qual_t    conn_qual;        /* IT_UD_PARAM_CONN_QUAL */
8781     it_path_t         destination_path; /* IT_UD_PARAM_DEST_PATH */
8782     unsigned char     private_data[IT_MAX_PRIV_DATA];
8783                                     /* IT_UD_PARAM_PRIV_DATA */
8784     size_t            private_data_length;
8785                                     /* IT_UD_PARAM_PRIV_DATA_LENGTH */
8786 } it_ud_svc_req_param_t;
```

8787 APPLICABILITY

8788 `it_ud_service_request_handle_query` is applicable only to the UD service type.

8789 DESCRIPTION

8790 `ud_svc_handle` Unreliable Datagram (UD) Service Request Handle previously created by a
8791 call to `it_ud_service_request_handle_create`.

8792	mask	Bitwise OR of flags for the requested UD Service Request Handle parameters.
8793		
8794	ud_svc_handle_info	Data structure containing information about the UD Service Request Handle, <i>ud_svc_handle</i> .
8795		
8796	<i>it_ud_service_request_handle_query</i> collects the desired information about the <i>ud_svc_handle</i> passed in and returns that information in the <i>it_ud_svc_req_param_t</i> structure provided in <i>ud_svc_handle_info</i> . On return, each field of <i>ud_svc_handle_info</i> is only valid if the corresponding flag is set in the <i>mask</i> argument. The flag values for the <i>mask</i> appear in the comments above each of the fields in the <i>it_ud_svc_req_param_t</i> structure. The <i>mask</i> value IT_UD_PARAM_ALL causes all fields to be returned.	
8797		
8798		
8799		
8800		
8801		
8802	The definition of each field in the <i>it_ud_svc_req_param_t</i> structure is as follows:	
8803	<i>ia</i>	Handle for the Interface Adapter associated with this UD Service Request.
8804		
8805	<i>request_id</i>	Unique identifier associated with the <i>it_ud_svc_req_handle_t</i> .
8806	<i>reply_evd</i>	The Simple EVD for reply Events associated with the <i>it_ud_svc_req_handle_t</i> .
8807		
8808	<i>conn_qual</i>	Connection Qualifier describing the UD Service associated with the <i>it_ud_svc_req_handle_t</i> .
8809		
8810	<i>destination_path</i>	Path to the Destination of the <i>it_ud_service_request</i> operation associated with the <i>it_ud_svc_req_handle_t</i> .
8811		
8812	<i>private_data</i>	Opaque Private Data provided by the Consumer if the IA supports Private Data.
8813		
8814	<i>private_data_length</i>	Length of the Private Data supplied by the Consumer.
8815	RETURN VALUE	
8816	A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below:	
8817	IT_ERR_INVALID_UD_SVC	The Unreliable Datagram Service Handle (<i>ud_svc_handle</i>) was invalid.
8818		
8819	IT_ERR_INVALID_MASK	The <i>mask</i> contained invalid flag values.
8820	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.
8821		
8822		
8823		
8824	SEE ALSO	
8825	<i>it_ud_service_request_handle_create()</i> , <i>it_ud_service_request_handle_free()</i> ,	
8826	<i>it_ud_service_request()</i>	

<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_io_addr_t</i>	O/S-dependent structure describing a bus or physical address.
<i>it_ioobl_t</i>	I/O buffer list.
<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

8828

it_addr_mode_t

8829 **NAME**

8830 it_addr_mode_t – definition of addressing mode of a Local Memory Region or Remote Memory
8831 Region

8832 **SYNOPSIS**

```
8833 #include <it_api.h>  
8834  
8835 typedef enum {  
8836     IT_ADDR_MODE_ABSOLUTE = 0,  
8837     IT_ADDR_MODE_RELATIVE = 1  
8838 } it_addr_mode_t;
```

8839 **DESCRIPTION**

8840 Addressing mode of an LMR or RMR, with two possible values as follows:

8841	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.
8857	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

8875 relative to the first byte of the RMR. The total offset relative to the
8876 first byte of the underlying LMR is obtained by adding to this byte
8877 offset the difference between the *addr* attributes of the RMR and the
8878 LMR established at RMR Link time.

8879 **EXTENDED DESCRIPTION**

8880 On InfiniBand, Relative Addressing is supported only if the Verb Extensions are supported
8881 including the “ZBVA” option. Moreover, Relative Addressing may be available only for Narrow
8882 RMRs (Type 2 Memory Windows).

8883 **APPLICATION USAGE**

8884 The desired addressing mode of an LMR must be specified when the LMR is created or linked.
8885 The desired addressing mode of an RMR must be specified when the RMR is linked.

8886 The *addr_mode_relative_support* field of the *it_ia_info_t* structure, which can be queried
8887 through *it_ia_query*, indicates whether the IA supports Relative Addressing.

8888 **SEE ALSO**

8889 *it_lmr_create()*, *it_rmr_link()*, *it_post_atomic()*, *it_post_send()*, *it_post_sendto()*, *it_post_rcv()*,
8890 *it_post_rcvfrom()*, *it_post_rdma_write()*, *it_post_rdma_read()*, *it_lmr_triplet_t*, *it_rmr_triplet_t*

it_aevd_notification_event_t

8891

8892 NAME

8893 it_aevd_notification_event_t – Aggregate Event Dispatcher Notification Event type

8894 SYNOPSIS

```
8895 #include <it_api.h>
8896
8897 typedef struct {
8898     it_event_type_t event_number;
8899     it_evd_handle_t aevd;
8900     it_evd_handle_t sevd;
8901 } it_aevd_notification_event_t;
```

8902 DESCRIPTION

8903 *event_number* Identifier of the Event type.
8904 Valid values: IT_AEVD_NOTIFICATION_EVENT

8905 *aevd* Handle for the Aggregate Event Dispatcher (AEVD) where the Event was
8906 queued.

8907 *sevd* Handle to the Simple Event Dispatcher (SEVD) that experienced a
8908 Notification Event.

8909 An IT_AEVD_NOTIFICATION_EVENT_STREAM Event is generated when a Notification
8910 has occurred on an SEVD associated with an AEVD with the IT_EVD_DEQUEUE_
8911 NOTIFICATIONS *evd_flag* set (see *it_evd_create*).

8912 The AEVD Notification Event passes the Handle for the associated SEVD on which a
8913 Notification Event has occurred.

8914 The AEVD Notification Event only applies to AEVDs. AEVDs do not overflow.

8915 SEE ALSO

8916 *it_event_t*, *it_evd_create()*, *it_evd_wait()*

it_affiliated_event_t

8917

8918 NAME

8919 it_affiliated_event_t – Affiliated Asynchronous Event type

8920 SYNOPSIS

```
8921 #include <it_api.h>
8922
8923 typedef struct {
8924     it_event_type_t event_number;
8925     it_evd_handle_t evd;
8926
8927     union {
8928         it_evd_handle_t sevd;
8929         it_ep_handle_t ep;
8930         it_srq_handle_t srq;
8931     } cause;
8932 } it_affiliated_event_t;
```

8933 DESCRIPTION

8934	<i>event_number</i>	Identifier of the Event type. Valid values: IT_ASYNC_AFF_SEVD_ENQUEUE_FAILURE, 8935 IT_ASYNC_AFF_EP_FAILURE, 8936 IT_ASYNC_AFF_EP_BAD_TRANSPORT_OPCODE, 8937 IT_ASYNC_AFF_EP_REQ_DROPPED, 8938 IT_ASYNC_AFF_EP_RDMAW_ACCESS_VIOLATION, 8939 IT_ASYNC_AFF_EP_RDMAW_CORRUPT_DATA, 8940 IT_ASYNC_AFF_EP_RDMAR_ACCESS_VIOLATION, 8941 IT_ASYNC_AFF_EP_LOCAL_ACCESS_VIOLATION, 8942 IT_ASYNC_AFF_EP_L_ACCESS_VIOLATION, 8943 IT_ASYNC_AFF_EP_L_RECV_ACCESS_VIOLATION, 8944 IT_ASYNC_AFF_EP_L_IRRQ_ACCESS_VIOLATION, 8945 IT_ASYNC_AFF_EP_L_TRANSPORT_ERROR, 8946 IT_ASYNC_AFF_EP_L_LLQ_ERROR, IT_ASYNC_AFF_EP_R_ERROR, 8947 IT_ASYNC_AFF_EP_R_ACCESS_VIOLATION, 8948 IT_ASYNC_AFF_EP_R_RECV_ACCESS_VIOLATION, 8949 IT_ASYNC_AFF_EP_R_RECV_LENGTH_ERROR, 8950 IT_ASYNC_AFF_EP_SOFT_HI_WATERMARK 8951 IT_ASYNC_AFF_EP_SOFT_HI_WATERMARK 8952 IT_ASYNC_AFF_SRQ_LOW_WATERMARK
8953	<i>evd</i>	Handle for the Event Dispatcher where the Event was queued.
8954	<i>sevd</i>	The Handle for the SEVD for which the Implementation failed to enqueue an 8955 Event. Valid only for the IT_ASYNC_AFF_SEVD_ENQUEUE_FAILURE Event 8956 type.
8957	<i>ep</i>	The Handle for the Endpoint that experienced the Event. Valid for all 8958 asynchronous errors affiliated with Endpoint other than 8959 IT_ASYNC_AFF_SEVD_ENQUEUE_FAILURE and 8960 IT_ASYNC_AFF_SRQ_LOW_WATERMARK.
8961	<i>srq</i>	The Handle for the S-RQ whose Low Watermark threshold has just been crossed.

8962
8963
8964

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*.

8965
8966

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.

8967
8968

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_LLQ_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> .

8969
8970
8971

All Events on an IT_ASYNC_AFF_EVENT_STREAM SEVD cause Notification. See *it_evd_create* for details of Notification.

8972
8973
8974
8975
8976
8977
8978

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.

8979
8980
8981
8982

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.)

8983
8984
8985
8986

<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

8987
8988
8989
8990

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_event_type_t Value	VIA “VipErrorCallback” Name
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.)

8991 **SEE ALSO**

8992 *it_event_t, it_evd_wait(), it_evd_create(), it_srq_modify(), it_ep_attributes*

it_boolean_t

8993

8994 **NAME**

8995 `it_boolean_t` – the Boolean type used by the API

8996 **SYNOPSIS**

```
8997 #include <it_api.h>
8998
8999 typedef enum {
9000     IT_FALSE = 0,
9001     IT_TRUE  = 1
9002 } it_boolean_t;
```

9003 **DESCRIPTION**

9004 The *it_boolean_t* type is used in several data structures in the API to describe a value that can
9005 exist in one of two different states: true (IT_TRUE), or false (IT_FALSE).

9006 **SEE ALSO**

9007 *it_cm_msg_events*, *it_cm_req_events*, *it_ep_attributes_t*, *it_evd_create()*, *it_evd_modify()*,
9008 *it_evd_query()*, *it_ia_info_t*, *it_rmr_query()*

it_cm_msg_events

9009

9010 NAME

9011 Communication Management Message Events – definitions for communication management
9012 Events other than Connection Requests and definition of Unreliable Datagram service resolution
9013 reply Event

9014 SYNOPSIS

```
9015 #include <it_api.h>
9016
9017 #define IT_MAX_PRIV_DATA 256
9018
9019 typedef enum {
9020     IT_CN_REJ_OTHER           = 0,
9021     IT_CN_REJ_TIMEOUT        = 1,
9022     IT_CN_REJ_BAD_PATH       = 2,
9023     IT_CN_REJ_STALE_CONN     = 3,
9024     IT_CN_REJ_BAD_ORD        = 4,
9025     IT_CN_REJ_RESOURCES      = 5,
9026     IT_CN_REJ_BAD_CONN_PARMS = 6
9027 } it_conn_reject_code_t;
9028
9029 typedef struct {
9030     it_event_type_t          event_number;
9031     it_evd_handle_t          evd;
9032     it_cn_est_identifier_t   cn_est_id;
9033     it_ep_handle_t           ep;
9034     uint32_t                 rdma_read_ird;
9035     uint32_t                 rdma_read_ord;
9036     it_path_t                dst_path;
9037     it_conn_reject_code_t    reject_reason_code;
9038     unsigned char            private_data[IT_MAX_PRIV_DATA];
9039     it_boolean_t             private_data_present;
9040 } it_connection_event_t;
9041
9042 typedef enum {
9043     IT_UD_SVC_EP_INFO_VALID   = 0,
9044     IT_UD_SVC_ID_NOT_SUPPORTED = 1,
9045     IT_UD_SVC_REQ_REJECTED    = 2,
9046     IT_UD_NO_EP_AVAILABLE    = 3,
9047     IT_UD_REQ_REDIRECTED      = 4
9048 } it_ud_svc_req_status_t;
9049
9050 typedef struct {
9051     it_event_type_t          event_number;
9052     it_evd_handle_t          evd;
9053     it_ud_svc_req_handle_t   ud_svc;
9054     it_ud_svc_req_status_t   status;
9055     it_remote_ep_info_t      ep_info;
9056     it_path_t                dst_path;
9057     unsigned char            private_data[IT_MAX_PRIV_DATA];
9058     it_boolean_t             private_data_present;
9059 } it_ud_svc_reply_event_t;
```

9060 **DESCRIPTION**

9061 The Communication Management Message Event Stream, `IT_CM_MSG_EVENT_STREAM`,
 9062 generates Events for all of the possible state transitions following a Connection Request as well
 9063 as for Unreliable Datagram Service Resolution replies. These Events are all the Communication
 9064 Management Events except those invoked by incoming requests (see *it_cm_req_events* for
 9065 those).

9066 Only one Event will be generated when a Connection is destroyed for any reason, either the
 9067 `IT_CM_MSG_CONN_DISCONNECT_EVENT` or the `IT_CM_MSG_CONN_BROKEN_`
 9068 `EVENT`, but not both. The Consumer should be ready to handle either of these Events being
 9069 generated even when the local or remote Consumer called *it_ep_disconnect*.

9070 The Connection Events are represented by the *it_connection_event_t* structure and the
 9071 Unreliable Datagram Service Resolution replies are represented by the *it_ud_svc_reply_event_t*
 9072 structure.

9073 The *it_connection_event_t* structure has the following members:

9074	<i>event_number</i>	Identifier of the Event type. Valid values: 9075 <code>IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT</code> , 9076 <code>IT_CM_MSG_CONN_ESTABLISHED_EVENT</code> , 9077 <code>IT_CM_MSG_CONN_PEER_REJECT_EVENT</code> , 9078 <code>IT_CM_MSG_CONN_NONPEER_REJECT_EVENT</code> , 9079 <code>IT_CM_MSG_CONN_DISCONNECT_EVENT</code> , 9080 <code>IT_CM_MSG_CONN_BROKEN_EVENT</code>
9081	<i>evd</i>	Handle for the Event Dispatcher where the Event was queued.
9082	<i>cn_est_id</i>	Identifier for the Connection establishment Event.
9083	<i>ep</i>	Endpoint Handle associated with Connection in progress.
9084	<i>rdma_read_ird</i>	Maximum number of incoming simultaneous RDMA Read 9085 operations supported by the remote EP. Only valid if the 9086 <i>it_ia_info.ird_ord_ia_support</i> value is <code>IT_TRUE</code> and as described 9087 under Application Usage below. Not valid if 9088 <code>IT_LISTEN_SUPPRESS_IRD_ORD</code> is set for <i>it_listen_create</i> or 9089 if <code>IT_CONNECT_SUPPRESS_IRD_ORD</code> is set for 9090 <i>it_ep_connect</i> .
9091	<i>rdma_read_ord</i>	Maximum number of outgoing simultaneous RDMA Read 9092 operations supported by the remote EP. Only valid if the 9093 <i>it_ia_info.ird_ord_ia_support</i> value is <code>IT_TRUE</code> and as described 9094 under Application Usage below. Not valid if 9095 <code>IT_LISTEN_SUPPRESS_IRD_ORD</code> is set for <i>it_listen_create</i> or 9096 if <code>IT_CONNECT_SUPPRESS_IRD_ORD</code> is set for 9097 <i>it_ep_connect</i> .
9098	<i>dst_path</i>	Path to Destination node supporting Service. Valid only if remote 9099 has rejected the proposed Path (<i>reject_reason_code</i> is 9100 <code>IT_CN_REJ_BAD_PATH</code>). Consumer should use <i>dst_path</i> if they 9101 wish to retry Connection attempt.
9102	<i>reject_reason_code</i>	Reason for rejection of Connection attempt.

9103	<i>private_data</i>	Private Data buffer.
9104	<i>private_data_present</i>	When it has the value IT_TRUE then Private Data is present in the
9105		<i>private_data</i> buffer above.
9106	The <i>it_ud_svc_reply_event_t</i> structure has the following members:	
9107	<i>event_number</i>	Identifier of the Event type. Valid values:
9108		IT_CM_MSG_UD_SERVICE_REPLY_EVENT
9109	<i>evd</i>	Handle for the Event Dispatcher where the Event was queued.
9110	<i>ud_svc</i>	Handle for the corresponding Service Request.
9111	<i>status</i>	Completion status for Service Request.
9112	<i>ep_info</i>	Resolution of Connection Qualifier for the UD service to a
9113		specific remote Endpoint. Only valid if <i>status</i> is
9114		IT_UD_SVC_EP_INFO_VALID. See <i>it_ep_attributes_t</i> for the
9115		definition of the <i>it_remote_ep_info_t</i> structure.
9116	<i>dst_path</i>	Path to Destination node supporting Service. Valid only if remote
9117		has redirected (<i>status</i> is IT_UD_REQ_REDIRECTED). Path
9118		returned is complete.
9119	<i>private_data</i>	Private Data buffer.
9120	<i>private_data_present</i>	When it has the value IT_TRUE then Private Data is present in the
9121		<i>private_data</i> buffer above.

9122 **EXTENDED DESCRIPTION**

9123 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.

9124

Table 3: Connection Management Event Definitions

9125

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.

9126

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>

9127

Table 4: Event Management Event Fields

9128

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.

9129

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 <code>timeout</code> member of the <code>it_path_t</code> that was input to <code>it_ep_connect</code> . This <code>reject_reason_code</code> 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 <code>dst_path</code> 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 <code>rdma_read_ird</code> 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

9131 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.

9132 **Table 6: UD Service Resolution Reply Event Definitions**

9133 UD service resolution replies return *status* in the Event data structure as described in the
9134 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.

9135 **Table 7: Service Resolution Reply Status**

9136 All Events on an IT_CM_MSG_EVENT_STREAM SEVD cause Notification. See
9137 [it_evd_create](#) for details of Notification.

9138 Default overflow behavior of an IT_CM_MSG_EVENT_STREAM SEVD is automatic
9139 rearming. This default behavior of the SEVD is equivalent to IT_EVD_OVERFLOW_
9140 DEFAULT cleared and IT_EVD_OVERFLOW_NOTIFY set and IT_EVD_OVERFLOW_
9141 AUTO_RESET set. See [it_evd_create](#) for details of overflow detection.

9142 **EXTENDED DESCRIPTION**

9143 For the Infiniband transport, the following table maps the values in the *reject_reason_code* field
9144 to their corresponding “Rejection Reason Code” for the REJ message as specified in Volume 1,
9145 Chapter 12 of the Infiniband specification. Rejection Reason Codes that are not listed in this
9146 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 an 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

9156

9157

9158

9159

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

9160

9161

APPLICATION USAGE

9162

9163

9164

9165

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.

9166

9167

9168

9169

9170

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.

9171

9172

9173

9174

9175

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.

9176 When using two-way Connection establishment, the passive side Consumer may receive an
9177 IT_CM_MSG_CONN_ESTABLISHED_EVENT Event. The *rdma_read_ird* and
9178 *rdma_read_ord* values found in the Event should be ignored by the Consumer (rationale – the
9179 Implementation could, at best, expose the original IRD/ORD values sent by the active side, but
9180 the active side may choose to change ORD on receipt of the passive side thus invalidating the
9181 original values).

9182 With the three-way handshake Connection establishment method, there is also a potential race
9183 condition between the Implementation generating the IT_CM_MSG_CONN_ACCEPT_
9184 ARRIVAL_EVENT Event and the Consumer calling *it_ep_disconnect* or *it_ep_free*. The
9185 Consumer should not use the *cn_est_id* if the IT_CM_MSG_CONN_ACCEPT_
9186 ARRIVAL_EVENT Event arrives after *it_ep_disconnect* or *it_ep_free* was called, regardless of
9187 whether the call returned yet, and regardless of whether the Event was dequeued before or after
9188 the call was made. If the Consumer does use the *cn_est_id* then the Implementation generates an
9189 IT_ERR_INVALID_CN_EST_ID error, or it may generate a segmentation fault, or other error.

9190 Neither the Active nor the Passive side Consumer should rely upon the
9191 IT_CM_MSG_CONN_ESTABLISHED_EVENT Event containing any Private Data, even if
9192 Private Data is input to the final *it_ep_accept* call that causes the Connection to be established.
9193 The side that makes the final call to *it_ep_accept* will never see any Private Data in the
9194 IT_CM_MSG_CONN_ESTABLISHED_EVENT Event, and because of races and unreliability
9195 inherent to the Connection establishment process of many of the transports that the IT-API
9196 supports, Private Data can sometimes be dropped on the other side as well.

9197 See the *it_ep_disconnect* reference page for details on the guarantee of delivery of Private Data
9198 when disconnecting Connections.

9199 The Consumer is advised to structure their ULP so that the Active side sends the first message
9200 after a Connection has been established. This is good practice because under some circumstances
9201 the completion of the first Receive operation is what causes the IT_CM_MSG_CONN_
9202 ESTABLISHED_EVENT Event to be generated on the Passive side. Depending upon the
9203 Passive side to send the first message after a Connection has been established can potentially
9204 result in the Connection establishment process timing out rather than completing successfully.

9205 Consult the Application Usage section of *it_cm_req_events* for the discussion of use of Private
9206 Data.

9207 When the Consumer receives an IT_CM_MSG_UD_SERVICE_REPLY_EVENT where the
9208 *status* is IT_UD_REQ_REDIRECTED, the Consumer can retry the attempt to retrieve UD
9209 service information. In order to do so the Consumer should free up its old UD Service Request
9210 Handle (by calling *it_ud_service_request_handle_free*), and create a new UD Service Request
9211 Handle by passing the *dst_path* returned in the IT_CM_MSG_UD_SERVICE_REPLY_EVENT
9212 to *it_ud_service_request_handle_create* to create a new Handle.

9213 No ordering guarantee exists between generation of IT_CM_MSG_EVENT_STREAM Events
9214 and generation of IT_DTO_EVENT_STREAM Events. For example, when an Endpoint is
9215 disconnected, the IT_CM_MSG_CONN_DISCONNECT_EVENT can be generated on the
9216 IT_CM_MSG_EVENT_STREAM Event Stream either before or after the flushed Events (i.e.,
9217 Events with an IT_DTO_ERR_FLUSHED status) for pending DTOs are generated on the
9218 IT_DTO_EVENT_STREAM Event Stream.

9219 **SEE ALSO**
9220 *it_event_t, it_evd_create(), it_evd_wait(), it_ep_modify(), it_listen_create(), it_ep_accept(),*
9221 *it_ep_disconnect(), it_reject(), it_address_handle_create(),*
9222 *it_ud_service_request_handle_free(), it_path_t, it_ia_info_t, it_ep_attributes_t*

it_cm_req_events

9223

9224 NAME

9225 Communication Management Request Events – definitions for Connection Request and
9226 Unreliable Datagram service resolution request communication management Events

9227 SYNOPSIS

```
9228 #include <it_api.h>
9229
9230 typedef struct {
9231     it_event_type_t          event_number;
9232     it_evd_handle_t         evd;
9233     it_cn_est_identifier_t   cn_est_id;
9234     it_conn_qual_t          conn_qual;
9235     it_net_addr_t           source_addr;
9236     size_t                   spigot_id;
9237     uint32_t                 max_message_size;
9238     uint32_t                 rdma_read_ird;
9239     uint32_t                 rdma_read_ord;
9240     unsigned char            private_data[IT_MAX_PRIV_DATA];
9241     it_boolean_t             private_data_present;
9242 } it_conn_request_event_t;
9243
9244 typedef struct {
9245     it_event_type_t          event_number;
9246     it_evd_handle_t         evd;
9247     it_ud_svc_req_identifier_t ud_svc_req_id;
9248     it_conn_qual_t          conn_qual;
9249     it_net_addr_t           source_addr;
9250     size_t                   spigot_id;
9251     unsigned char            private_data[IT_MAX_PRIV_DATA];
9252     it_boolean_t             private_data_present;
9253 } it_ud_svc_request_event_t;
```

9254 DESCRIPTION

9255 The Communication Management Request Event Stream, *IT_CM_REQ_EVENT_STREAM*,
9256 generates Events when an incoming Connection Request Event or incoming Unreliable
9257 Datagram service resolution request occurs.

9258 Incoming Connection Request Events are represented by the *it_conn_request_event_t* structure
9259 and incoming Unreliable Datagram Service Resolution requests are represented by the
9260 *it_ud_svc_request_event_t* structure.

9261 The *it_conn_request_event_t* structure has the following members:

9262	<i>event_number</i>	Identifier of the Event type. Valid values: 9263 <i>IT_CM_REQ_CONN_REQUEST_EVENT</i>
9264	<i>evd</i>	Handle for the Event Dispatcher where the Event was queued.
9265	<i>cn_est_id</i>	Identifier for the Connection establishment Event.
9266	<i>conn_qual</i>	Connection Qualifier on which request was received.

9267	<i>source_addr</i>	Source address of requestor.
9268	<i>spigot_id</i>	Local Spigot on which the request was received.
9269	<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.
9270		
9271		
9272	<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> .
9273		
9274		
9275		
9276	<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> .
9277		
9278		
9279		
9280	<i>private_data</i>	Private Data buffer.
9281	<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.
9282		
9283	The <i>it_ud_svc_request_event_t</i> structure has the following members:	
9284	<i>event_number</i>	Identifier of the Event type. Valid values: IT_CM_REQ_UD_SERVICE_REQUEST_EVENT
9285		
9286	<i>evd</i>	Handle for the Event Dispatcher where the Event was queued.
9287	<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.
9288		
9289	<i>conn_qual:</i>	Connection Qualifier on which request was received.
9290	<i>source_addr</i>	Source address of requestor.
9291	<i>spigot_id</i>	Local Spigot on which request was received.
9292	<i>private_data</i>	Private Data buffer.
9293	<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.
9294		

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.

9295 **Table 10: Communication Management Request Event Definitions**

9296 All Events on an IT_CM_REQ_EVENT_STREAM SEVD cause Notification. See *it_evd_create*
9297 for details of Notification.

9298 Default overflow behavior of an IT_CM_REQ_EVENT_STREAM SEVD is overflow
9299 Notification disabled. This default behavior of the SEVD is equivalent to IT_EVD_
9300 OVERFLOW_DEFAULT cleared and IT_EVD_OVERFLOW_NOTIFY cleared. See
9301 *it_evd_create* for details of overflow detection.

9302 **APPLICATION USAGE**

9303 The Consumer should use the *it_event_t* structure if it is desired to wait for both Connection
9304 Request Events (*it_conn_request_event_t*) and Unreliable Datagram service resolution request
9305 Events (*it_ud_svc_request_event_t*) via the same EVD. The *it_event_t* structure is of sufficient
9306 size to hold either Event type.

9307 The Consumer must *it_evd_create* an IT_CM_REQ_EVENT_STREAM Simple EVD and pass
9308 the new EVD and a Connection Qualifier to *it_listen_create* in order to receive
9309 IT_CM_REQ_EVENT_STREAM Events via the *it_evd_wait* or *it_evd_dequeue* calls.

9310 The *private_data_present* field indicates whether Private Data is present in the *private_data*
9311 buffer. It is the Consumer's responsibility to convey the size of the data contained in the Private
9312 Data buffer using their own ULP. Each communication management Event type may have a
9313 different maximum Private Data buffer size. The Consumer can determine the maximum
9314 possible sizes for the Private Data buffers corresponding to each of the Event types from the
9315 *it_ia_info_t* structure (for instance, *connect_private_data_len*).

9316 **SEE ALSO**

9317 *it_event_t*, *it_evd_create()*, *it_evd_wait()*, *it_evd_dequeue()*, *it_ep_modify()*, *it_listen_create()*,
9318 *it_ep_accept()*, *it_ep_disconnect()*, *it_reject()*, *it_ud_service_reply()*, *it_ia_info_t*

it_conn_qual_t

9319

9320 **NAME**

9321 it_conn_qual_t – encapsulates all supported Connection Qualifier types

9322 **SYNOPSIS**

```
9323 #include <it_api.h>
9324
9325 /* Enumerates all the possible Connection Qualifier types supported by
9326 the API. */
9327 typedef enum {
9328
9329     /* IANA (TCP/UDP) Port Number */
9330     IT_IANA_PORT = 0x01,
9331
9332     /* InfiniBand Service ID, as described in Section 12.7.3 of
9333 Volume 1 of the InfiniBand specification. */
9334     IT_IB_SERVICEID = 0x02,
9335
9336     /* VIA Connection Discriminator */
9337     IT_VIA_DISCRIMINATOR = 0x04,
9338
9339     /* iWARP local and remote IP (IANA) port object */
9340     IT_IANA_LR_PORT = 0x08
9341
9342 } it_conn_qual_type_t;
9343
9344 /* Defines the Connection Qualifier format for a VIA "connection
9345 discriminator". The API imposes a fixed upper bound on the
9346 discriminator size. */
9347
9348 #define IT_MAX_VIA_DISC_LEN 64
9349
9350 typedef struct {
9351
9352     /* The total number of bytes in the array below */
9353     /* that are significant. */
9354     uint16_t len;
9355
9356     /* VIA connection discriminator, which is an array of bytes. */
9357     unsigned char discriminator[IT_MAX_VIA_DISC_LEN];
9358
9359 } it_via_discriminator_t;
9360
9361 /* This defines the Connection Qualifier for InfiniBand, which is the
9362 64-bit Service ID. */
9363 typedef uint64_t it_ib_serviceid_t;
9364
9365 /* Defines an additional Connection Qualifier for iWARP that
9366 allows specification of both local and remote IANA ports when
9367 passing this structure to it_ep_connect. */
9368 typedef struct {
9369     uint16_t local;
```

```

9370         uint16_t remote;
9371     } it_iana_lr_port_t;
9372
9373     /* This describes a Connection Qualifier suitable for input to
9374        several routines in the API. */
9375     typedef struct {
9376
9377         /* The discriminator for the union below. */
9378         it_conn_qual_type_t type;
9379
9380         union {
9381
9382             /* IANA Port Number, in network byte order. */
9383             uint16_t port;
9384
9385             /* InfiniBand Service ID, in network byte order. */
9386             it_ib_serviceid_t serviceid;
9387
9388             /* VIA connection discriminator. */
9389             it_via_discriminator_t discriminator;
9390
9391             /* IANA local/remote Port numbers, in network byte order. */
9392             it_iana_lr_port_t lr_port;
9393
9394         } conn_qual;
9395     } it_conn_qual_t;
9396

```

9397 DESCRIPTION

9398 The *it_conn_qual_t* type is used by several routines in the API to encapsulate a Connection
9399 Qualifier. A Connection Qualifier is used by a Consumer on the Active side of the Connection
9400 establishment process in the *it_ep_connect* routine to target the remote Consumer that should be
9401 responding to the Connection establishment attempt. It is used on the Passive side of the
9402 Connection establishment process in the *it_listen_create* routine to steer incoming Connection
9403 Requests to an appropriate EVD for further processing.

9404 Each Spigot on an IA can support one or more types of Connection Qualifier. All Spigots will
9405 support the IANA Port Number type of Connection Qualifier, regardless of which transport the
9406 IA that houses the Spigot is using. Which types of Connection Qualifier a Spigot supports can be
9407 determined using the *it_ia_query* routine.

9408 In order to aid Consumers in writing portable applications that span platforms with different
9409 native byte orders, all Connection Qualifiers that are supported by the API with the exception of
9410 the VIA “connection discriminator” are required to be input to the API in network byte order,
9411 and will be output from the API in network byte order. (The VIA “connection discriminator” is
9412 defined to be an array of bytes, and hence is not affected by which native byte order a platform
9413 uses.)

9414 SEE ALSO

9415 *it_ep_connect()*, *it_listen_create()*, *it_ia_query()*

it_context_t

9416

9417 NAME

9418 `it_context_t` – structure describing a Consumer Context

9419 SYNOPSIS

```
9420 #include <it_api.h>
9421
9422 typedef union {
9423     void * ptr;
9424     uint64_t index;
9425 } it_context_t;
```

9426 DESCRIPTION

9427 The `it_context_t` union describes storage definitions for the Consumer Context associated with
9428 an IT Object Handle.

9429 *ptr* Storage space for an address pointer.

9430 *index* Storage space for an unsigned 64-bit integer.

9431 SEE ALSO

9432 [*it_get_consumer_context\(\)*](#), [*it_set_consumer_context\(\)*](#)

it_dg_remote_ep_addr_t

9433

9434 NAME

9435 `it_dg_remote_ep_addr_t` – datagram transport Endpoint address

9436 SYNOPSIS

```
9437 #include <it_api.h>
9438
9439 typedef struct
9440 {
9441     it_addr_handle_t    addr;
9442     it_remote_ep_info_t ep_info;
9443 } it_ib_ud_addr_t;
9444
9445 typedef enum
9446 {
9447     IT_DG_TYPE_IB_UD
9448 } it_dg_type_t;
9449
9450 typedef struct
9451 {
9452     it_dg_type_t type; /* IT_DG_TYPE_IB_UD */
9453     union {
9454         it_ib_ud_addr_t ud;
9455     } addr;
9456 } it_dg_remote_ep_addr_t;
```

9457 APPLICABILITY

9458 The `it_dg_remote_ep_addr_t` data type is applicable only to Endpoints created for the UD
9459 Service Type.

9460 DESCRIPTION

9461 For datagram transports, the Endpoint address is specified in DTO operations using the
9462 `it_dg_remote_ep_addr_t` data structure. The structure is intended to allow support of more than
9463 one datagram transport type.

9464 The datagram transport type is specified as `type` in the `it_dg_remote_ep_addr_t` structure. In this
9465 revision of the API, only the InfiniBand Unreliable Datagram transport, `IT_DG_TYPE_IB_UD`,
9466 is supported.

9467 For InfiniBand Unreliable Datagram, the transport-specific Endpoint address is contained in the
9468 `it_ib_ud_addr_t` sub-structure of the `it_dg_remote_ep_addr_t`. The components of the
9469 InfiniBand Unreliable Datagram Endpoint address are:

9470 `addr` An Address Handle created by the Consumer using `it_address_handle_create`.

9471 `ep_info` An Endpoint Info structure (see `it_ep_attributes_t`) containing the Endpoint ID,
9472 `ud_ep_id`, and the Endpoint Key, `ud_ep_key`. The Consumer may make use of
9473 `it_ud_service_request` to obtain `ud_ep_id` and `ud_ep_key` or may obtain them by
9474 their own means.

9475 **EXTENDED DESCRIPTION**

9476 For InfiniBand Unreliable Datagram, the Endpoint ID is equivalent to an InfiniBand QP number
9477 and the Endpoint Key is equivalent to an InfiniBand *Q_key*.

9478 **FUTURE DIRECTIONS**

9479 Support for Reliable Datagram Service Type may be provided in a future revision of this API.

9480 **SEE ALSO**

9481 *it_post_sendto()*, *it_post_recvfrom()*, *it_address_handle_create()*, *it_ud_service_request()*,
9482 *it_ep_attributes_t*

it_dto_cookie_t

9483

9484 NAME

9485 `it_dto_cookie_t` – definition of implementation-opaque Consumer cookie

9486 SYNOPSIS

```
9487 #include <it_api.h>  
9488  
9489 typedef uint64_t it_dto_cookie_t;
```

9490 DESCRIPTION

9491 `it_dto_cookie_t` is an object that can be provided by the Consumer on every DTO or RMR
9492 operation and is returned to the Consumer in the corresponding DTO Completion Event (see
9493 [it_dto_events](#)) if a DTO Completion Event is generated (see [it_dto_flags_t](#)). The `it_dto_cookie_t`
9494 object is opaque to the Implementation and is returned unchanged to the Consumer in the DTO
9495 Completion Event corresponding to the posted DTO or RMR.

9496 SEE ALSO

9497 [it_dto_events](#), [it_dto_flags_t](#), [it_post_atomic\(\)](#), [it_post_send\(\)](#), [it_post_sendto\(\)](#), [it_post_rcv\(\)](#),
9498 [it_post_rcvfrom\(\)](#), [it_post_rdma_read\(\)](#), [it_post_rdma_write\(\)](#), [it_rmr_link\(\)](#), [it_rmr_unlink\(\)](#)

9499

9500 **NAME**

9501 DTO and RMR Link/Unlink Completion Event types

9502 **SYNOPSIS**

```

9503 #include <it_api.h>
9504
9505 typedef enum {
9506     IT_UD_IB_GRH_PRESENT = 0x01
9507 } it_dto_ud_flags_t;
9508
9509 typedef struct {
9510     it_event_type_t event_number;
9511     it_evd_handle_t evd;
9512     it_ep_handle_t ep;
9513     it_dto_cookie_t cookie;
9514     it_dto_status_t dto_status;
9515     uint32_t transferred_length;
9516     it_handle_t unlinked_mr_handle;
9517 } it_dto_cmpl_event_t;
9518
9519 typedef struct {
9520     it_event_type_t event_number;
9521     it_evd_handle_t evd;
9522     it_ep_handle_t ep;
9523     it_dto_cookie_t cookie;
9524     it_dto_status_t dto_status;
9525     uint32_t transferred_length;
9526     it_handle_t unlinked_mr_handle;
9527     it_dto_ud_flags_t flags;
9528     it_ud_ep_id_t ud_ep_id;
9529     it_path_t src_path;
9530 } it_all_dto_cmpl_event_t;

```

9531 **APPLICABILITY**

9532 The *it_dto_cmpl_event_t* data type is applicable only to Endpoints created for the RC Service
 9533 Type. The *it_all_dto_cmpl_event_t* data type is applicable to Endpoints of any Service Type.

9534 **DESCRIPTION**

9535 The DTO Completion Event Stream, *IT_DTO_EVENT_STREAM*, generates Events for all Data
 9536 Transfer Operation completions as well as RMR Link and Unlink completions.

9537 Unreliable Datagram Receive Completion Events provide additional data beyond that of the
 9538 other DTO Completion Events. The additional data is large enough to warrant defining a much
 9539 smaller Event structure for all other DTO operations usable by Consumers interested in
 9540 conserving the memory footprint of their application.

9541 With the exception of UD Receive completions, all DTO completions, including RMR Links
 9542 and Unlinks, can be represented by the *it_dto_cmpl_event_t* structure.

9543 UD Receive completions require use of the *it_all_dto_cmpl_event_t* structure. Consumers
 9544 wishing to receive UD Receive and Send Completion Events on one Simple EVD or wishing to

9545 handle all possible DTO completions with one Simple EVD must use the
 9546 *it_all_dto_cmpl_event_t* structure or the encompassing *it_event_t* structure (see *it_event_t*).
 9547 Failure to use the *it_all_dto_cmpl_event_t* structure or *it_event_t* structure for UD Receive
 9548 Completion Events can result in program termination.

9549 The *it_dto_cmpl_event_t* structure has the following members:

9550 *event_number* Identifier of the Event type. Valid values:
 9551 IT_DTO_SEND_CMPL_EVENT,
 9552 IT_DTO_RC_RECV_CMPL_EVENT,
 9553 IT_DTO_RDMA_WRITE_CMPL_EVENT,
 9554 IT_DTO_RDMA_READ_CMPL_EVENT,
 9555 IT_RMR_BIND_CMPL_EVENT,
 9556 IT_RMR_LINK_CMPL_EVENT,
 9557 IT_DTO_FETCH_ADD_CMPL_EVENT,
 9558 IT_DTO_CMP_SWAP_CMPL_EVENT,
 9559 IT_LMR_LINK_CMPL_EVENT

9560 *evd* Handle for the Event Dispatcher where the Event was queued.

9561 *ep* For all Event types except IT_DTO_RC_RECV_CMPL_EVENT, this is
 9562 the Handle for the Endpoint on which the DTO was posted. For the
 9563 IT_DTO_RC_RECV_CMPL_EVENT, this is the Handle for the Endpoint
 9564 targeted by an incoming Send operation that resulted in the completion of a
 9565 Receive DTO.

9566 *cookie* Cookie that the Consumer associated with the DTO when it was posted.
 9567 See *it_dto_cookie_t* for details.

9568 *dto_status* Status of completed DTO.

9569 *transferred_length* Length of transferred message.

9570 *unlinked_mr_handle* Handle of the LMR or RMR that was unlinked if not equal
 9571 IT_NULL_HANDLE. Always IT_NULL_HANDLE for UD receives.

9572 See *it_dto_status_t* for values and definition of *dto_status*.

9573 The *transferred_length* field indicates the amount of data transferred in a Receive operation. The
 9574 content of this field is undefined for Send, RDMA Read, RDMA Write, RMR Link, and RMR
 9575 Unlink operations. This field is also only valid if *dto_status* is IT_DTO_SUCCESS; otherwise,
 9576 the contents are undefined.

9577 The *it_all_dto_cmpl_event_t* structure has the following additional members:

9578 *flags:* Flags indicating additional service specific information.

9579 *ud_ep_id* Remote Endpoint ID from incoming datagram.

9580 *src_path* Partial Source Path information from incoming datagram.

9581 The IT_DTO_UD_RECV_CMPL_EVENT *event_number* is an additional valid value only for
 9582 the *it_all_dto_cmpl_event_t* structure or *it_event_t* structure.

9583 The *flags* parameter indicates whether or not the InfiniBand Global Routing Header (GRH) is
 9584 present in the first 40 bytes of the message payload. If the GRH is present, the IT_UD_IB_

9585 GRH_PRESENT bit will be set in *flags*. If the GRH is not present (IT_UD_IB_GRH_PRESENT
9586 bit cleared in *flags*), the first 40 bytes of the payload are undefined.

9587 For an IT_DTO_UD_RECV_CMPL_EVENT, the *transferred_length* field includes the length of
9588 the transferred message plus 40 bytes regardless of the IT_UD_IB_GRH_PRESENT bit value.

9589 The remote Endpoint ID, *ud_ep_id*, is derived from the incoming datagram. See
9590 [it_ep_attributes_t](#) for more details.

9591 Partial Source address information is returned in a datagram Completion Event in the *src_path*
9592 structure element. See Application Usage below.

9593 The *src_path* can hold more information than is returned in the Completion Event. The members
9594 of the [it_path_t](#) structure that are pertinent to a datagram Completion Event are listed in the table
9595 below. For each member, the corresponding Infiniband datagram addressing information that the
9596 member corresponds to is also identified. For a detailed explanation of the semantics associated
9597 with the Datagram addressing information, see Chapter 11.4.2.1 Poll For Completion in [IB-
9598 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

9599 IT_DTO_EVENT_STREAM Events may or may not cause Notification depending on the use of
9600 DTO flags (see [it_dto_flags_t](#)). See [it_evd_create](#) for details of Notification.
9601

9602 Default overflow behavior of an IT_DTO_EVENT_STREAM SEVD is overflow Notification
9603 enabled (for an IA that supports DTO EVD overflow detection, see below). The behavior of the
9604 SEVD is equivalent to IT_EVD_OVERFLOW_DEFAULT cleared and
9605 IT_EVD_OVERFLOW_NOTIFY set. Once an IT_DTO_EVENT_STREAM SEVD overflows,
9606 it cannot be rearmed. See [it_evd_create](#) for details of overflow detection.

9607 Overflow of an IT_DTO_EVENT_STREAM SEVD is catastrophic for the associated Endpoint
9608 or Endpoints. Each Endpoint is transitioned to the IT_EP_STATE_NONOPERATIONAL state
9609 as defined in [it_ep_state_t](#). The Endpoints are unrecoverable; [it_ep_free](#) must be called for all
9610 Endpoints sharing the same SEVD on which the overflow occurred.

9611 Overflow detection of DTO EVDs is only supported when the *dto_evd_overflow_detection* flag
9612 found in [it_ia_info_t](#) is IT_TRUE.

9613 APPLICATION USAGE

9614 Within an IT_DTO_UD_RECV_CMPL_EVENT Event, the *src_path* member returned contains
9615 insufficient information to identify the remote Endpoint. To resolve the remote Endpoint Path,
9616 the user should pass *src_path* returned in this Event to [it_address_handle_create](#) with the
9617 IT_AH_PATH_COMPLETE bit cleared. [it_address_handle_create](#) will complete the resolution
9618 of the Path.

9619 Overflow of an EVD containing the DTO Event Stream can result in indeterminate behavior
9620 when the IA does not support overflow detection (*dto_evd_overflow_detection* is IT_FALSE).
9621 On such an IA, the Consumer should structure their ULP to avoid any possibility of this
9622 occurring. Indeterminate behavior is constrained to the EVD concerned; overflow on a DTO

9623 Event Stream associated with a particular EVD must not affect any other EVD or Endpoint
9624 associated with other EVDs.

9625 No ordering guarantee exists between generation of IT_CM_MSG_EVENT_STREAM Events
9626 and generation of IT_DTO_EVENT_STREAM Events. For example, when an Endpoint is
9627 disconnected, the IT_CM_MSG_CONN_DISCONNECT_EVENT can be generated on the
9628 IT_CM_MSG_EVENT_STREAM Event Stream either before or after the flushed Events (i.e.,
9629 Events with an IT_DTO_ERR_FLUSHED status) for pending DTOs are generated on the
9630 IT_DTO_EVENT_STREAM Event Stream.

9631 Receipt of a successful incoming *it_post_send_and_unlink* DTO will unlink a local LMR or
9632 RMR and generate an *unlinked_mr_handle* not equal to IT_NULL_HANDLE in the Event. It is
9633 the Consumer's responsibility to correlate the *unlinked_mr_handle* with a local LMR or RMR.

9634 **FUTURE DIRECTIONS**

9635 An additional detailed status code may be added to the DTO Completion Event data structures.

9636 **SEE ALSO**

9637 *it_post_atomic()*, *it_post_send()*, *it_post_sendto()*, *it_post_recv()*, *it_post_recvfrom()*,
9638 *it_post_rdma_read()*, *it_post_rdma_write()*, *it_dto_status_t*, *it_dto_flags_t*, *it_event_t*,
9639 *it_evd_create()*, *it_evd_wait()*, *it_address_handle_create()*, *it_ep_state_t*, *it_ep_free()*,
9640 *it_ep_reset()*, *it_path_t*, *it_dto_cookie_t*, *it_ia_info_t*

it_dto_flags_t

9641

9642 NAME

9643 `it_dto_flags_t` – flags for Send, Receive, RDMA Read, RDMA Write, RMR Link, RMR Unlink,
9644 LMR Link, LMR Unlink, and Atomic Work Requests

9645 SYNOPSIS

```
9646 #include <it_api.h>
9647
9648 typedef enum
9649 {
9650     /* If flag set, completion generates a local event. */
9651     IT_COMPLETION_FLAG = 0x01,
9652
9653     /* If flag set, completion causes local Notification. */
9654     IT_NOTIFY_FLAG = 0x02,
9655
9656     /* If flag set, receipt of DTO at remote will cause Notification
9657        at remote. */
9658     IT_SOLICITED_WAIT_FLAG = 0x04,
9659
9660     /* If flag set, a WR posted to an EP's SQ will not start if
9661        RDMA Reads posted previously to the EP are not complete. */
9662     IT_BARRIER_FENCE_FLAG = 0x08,
9663
9664     /* If flag set, an LMR Unlink or RMR Unlink WR posted to an EP
9665        will not start if WRs posted previously to the EP's SQ are
9666        not complete. */
9667     IT_UNLINK_FENCE_FLAG = 0x10,
9668
9669     /* If flag set, the local data sink is unlinked upon WR
9670        completion */
9671     IT_UNLINK_LOCAL_SINK = 0x20,
9672
9673     /* If flag set, attempt to enqueue this WR for later processing */
9674     IT_COALESCE_WR_FLAG = 0x40
9675 } it_dto_flags_t;
```

9676 DESCRIPTION

9677 `it_dto_flags` Flags for posted WRs: Send, Receive, RDMA Read, RDMA Write, RMR Link,
9678 RMR Unlink, LMR Link, LMR Unlink, Atomic.

9679 Values for `it_dto_flags` are constructed by a bitwise-inclusive OR of flags from the following
9680 discussion.

9681 Any combination of the following may be used subject to restrictions as noted:

9682 IT_COMPLETION_FLAG

9683 If set, generate a Completion Event for this WR, else do not generate a Completion Event unless
9684 there is an error.

9685 If not set, then the completion of a subsequent WR on the same Work Queue of the same
9686 Endpoint with this flag set or with error completion will indicate the successful completion of
9687 prior WR(s) with this flag cleared.

9688 **Restrictions**

9689 IT_COMPLETION_FLAG may be set or cleared only for Send, RDMA Write, RDMA Read,
9690 RMR Link, and RMR Unlink operations on a Reliable Connection Service Type, and may be set
9691 or cleared only for Send operation on an Unreliable Datagram Service Type.

9692 IT_COMPLETION_FLAG must be set for all Receive WRs on all Service Types (Reliable
9693 Connection and Unreliable Datagram). Posting a Receive WR with IT_COMPLETION_FLAG
9694 cleared is an error.

9695 **IT_NOTIFY_FLAG**

9696 If set, generate Notification of completion of the WR. If there is an error, Notification of
9697 completion will be generated regardless of the IT_NOTIFY_FLAG value.

9698 **Restrictions**

9699 IT_NOTIFY_FLAG may be set or cleared on all WRs on a Reliable Connected Service Type,
9700 and may be set or cleared on Send and Receive operations on an Unreliable Datagram Service
9701 Type. It is an error to set IT_NOTIFY_FLAG if IT_COMPLETION_FLAG is clear.

9702 A completion will be generated with the Notification for a Receive WR if the matching received
9703 Send message had been posted at the remote with the IT_SOLICITED_WAIT_FLAG set
9704 regardless of the IT_NOTIFY_FLAG of the posted Receive WR.

9705 **IT_SOLICITED_WAIT_FLAG**

9706 If set, the Send WR will request completion Notification for the matching Receive on the other
9707 side of the Connection or, for Unreliable Datagram, for the matching Receive at the remote
9708 datagram Endpoint.

9709 **Restrictions**

9710 IT_SOLICITED_WAIT_FLAG is supported only for Send operations for all Service Types. It is
9711 an error to specify IT_SOLICITED_WAIT_FLAG on other operations.

9712 If set, requests Notification of completion of the matching remote Receive WR regardless of the
9713 value of the IT_NOTIFY_FLAG on the Receive WR.

9714 **IT_BARRIER_FENCE_FLAG**

9715 If set, a Work Request posted to an Endpoint's SQ will not execute until all RDMA Read
9716 requests previously posted to the Endpoint have completed.

9717	Restrictions
9718	If the service does not support RDMA Read, it is an error to set this flag. Specifically, it is an error to set IT_BARRIER_FENCE_FLAG on a WR using the UD service.
9719	
9720	IT_BARRIER_FENCE_FLAG must be cleared for all Receive WRs on any Service Types. Posting a Receive WR with IT_BARRIER_FENCE_FLAG set is an error.
9721	
9722	IT_UNLINK_FENCE_FLAG
9723	If set, an LMR Unlink or RMR Unlink operation posted to an Endpoint will not execute until all Work Requests previously posted to the Endpoint's SQ have completed.
9724	
9725	Restrictions
9726	If the <i>it_ia_info_t</i> attribute <i>unlink_fence_support</i> is IT_FALSE, then it is an error to set this flag.
9727	
9728	The IT_UNLINK_FENCE_FLAG may be set only for LMR Unlink (<i>it_lmr_unlink</i>) or RMR Unlink (<i>it_rmr_unlink</i>) operations. For InfiniBand only, IT_UNLINK_FENCE_FLAG may not be used with Wide RMRs.
9729	
9730	IT_UNLINK_LOCAL_SINK
9731	If set, the local data sink is unlinked after the WR has completed.
9732	Restrictions
9733	If the <i>it_ia_info_t</i> attribute <i>rdma_read_local_extensions</i> is IT_FALSE, then it is an error to set this flag.
9734	
9735	The IT_UNLINK_LOCAL_SINK may be set only for RDMA Read (<i>it_post_rdma_read</i>) or RDMA Read to RMR (<i>it_post_rdma_read_to_rmr</i>) operations and may only be used if <i>num_segments</i> is equal to one.
9736	
9737	
9738	IT_COALESCE_WR_FLAG
9739	If this bit is set in the <i>dto_flags</i> passed to a routine that posts an operation to an Endpoint Work Queue, that routine shall perform all immediate error checks normally associated with the operation but will attempt to enqueue the operation for later processing rather than processing the operation immediately.
9740	
9741	
9742	
9743	If this bit is clear in the <i>dto_flags</i> passed to a routine that posts an operation (of any type) to an Endpoint Work Queue, that routine shall perform all immediate error checks normally associated with the operation, and will then process all enqueued operations for the Endpoint Work Queue in FIFO order up to and including the operation that was posted with the IT_COALESCE_WR_FLAG bit clear.
9744	
9745	
9746	
9747	
9748	Restrictions
9749	While an operation is enqueued for later processing it consumes space in the Endpoint Work Queue.
9750	

9751
9752
9753
9754

If a Consumer attempts to enqueue more operations for later processing than the Endpoint Work Queue depth as returned by *it_ep_query* in the *max_request_dtos* or *max_rcv_dtos* members, the routine which posts the operation to the Work Queue may return an IT_ERR_TOO_MANY_POSTS error.

9755
9756
9757

The Implementation may opt to process the contents of an Endpoint Work Queue prior to the Consumer turning off the IT_COALESCE_WR_FLAG in the *dto_flags* that are passed to a routine that posts an operation to that Endpoint Work Queue.

9758
9759

EXTENDED DESCRIPTION

The following table lists all Work Requests and details the legal *it_dto_flags* values for each.

Work Request	Legal <i>it_dto_flags</i> Combinations
<i>it_post_send</i>	IT_UNLINK_FENCE_FLAG may not be used. IT_UNLINK_LOCAL_SINK may not be used. All other 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_send_and_unlink</i>	IT_UNLINK_FENCE_FLAG may not be used. IT_UNLINK_LOCAL_SINK may not be used. All other 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. IT_UNLINK_FENCE_FLAG may not be used. IT_UNLINK_LOCAL_SINK 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_rcv</i>	IT_COMPLETION_FLAG must be specified. IT_BARRIER_FENCE_FLAG may not be used. IT_UNLINK_FENCE_FLAG may not be used. IT_SOLICITED_WAIT_FLAG may not be used. IT_UNLINK_LOCAL_SINK may not be used. All other possible combinations of the remaining flags are legal.
<i>it_post_rcvfrom</i>	IT_COMPLETION_FLAG must be specified. IT_BARRIER_FENCE_FLAG may not be used. IT_UNLINK_FENCE_FLAG may not be used. IT_SOLICITED_WAIT_FLAG may not be used. IT_UNLINK_LOCAL_SINK 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. IT_UNLINK_FENCE_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. IT_UNLINK_FENCE_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. IT_UNLINK_FENCE_FLAG may not be used. IT_UNLINK_LOCAL_SINK 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. IT_UNLINK_FENCE_FLAG may not be used. IT_UNLINK_LOCAL_SINK 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. IT_UNLINK_LOCAL_SINK 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_atomic</i>	IT_SOLICITED_WAIT_FLAG may not be used. IT_UNLINK_LOCAL_SINK 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_lmr_link</i>	IT_SOLICITED_WAIT_FLAG may not be used. IT_UNLINK_FENCE_FLAG may not be used. IT_UNLINK_LOCAL_SINK 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_lmr_unlink</i>	IT_SOLICITED_WAIT_FLAG may not be used. IT_UNLINK_LOCAL_SINK 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.

9760
9761

The following table lists the WRs on each Service Type on which each flag value is supported.

it_dto_flags_t Value	Supported WR on RC	Supported WR on UD
IT_COMPLETION_FLAG	All	Sendto, Recvfrom

it_dto_flags_t Value	Supported WR on RC	Supported WR on UD
IT_NOTIFY_FLAG	All	Sendto, Recvfrom
IT_SOLICITED_WAIT_FLAG	Send ⁹	Sendto
IT_BARRIER_FENCE_FLAG	Send, RDMA Read, ¹⁰ RDMA Write, RMR Link, RMR Unlink, Atomic, LMR Link, LMR Unlink	N/A
IT_UNLINK_FENCE_FLAG	RMR Unlink, LMR Unlink	N/A
IT_UNLINK_LOCAL_SINK	RDMA Read	N/A
IT_COALESCE_WR_FLAG	All	All

9762
9763
9764

As stated in the Description section, IT_COMPLETION_FLAG must be set on *Recv* and *Recvfrom* WRs.

9765
9766

As stated in the Description section, IT_BARRIER_FENCE_FLAG must be cleared on *Recv* WRs for RC and must be cleared on *Sendto* WRs as well as cleared on *Recvfrom* WRs for UD.

9767
9768

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
IT_UNLINK_FENCE_FLAG	Local Invalidate Fence
IT_UNLINK_LOCAL_SINK	Local Invalidate
IT_COALESCE_WR_FLAG	Submitting a List of Work Requests

9769
9770
9771

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

⁹ In this table, “Send” corresponds to both *it_post_send* and *it_post_send_and_unlink*.

¹⁰ In this table, “RDMA Read” corresponds to both *it_post_rdma_read* and *it_post_rdma_read_to_rmr*.

it_dto_flags_t Value	VIA Concept
IT_SOLICITED_WAIT_FLAG	Not applicable
IT_BARRIER_FENCE_FLAG	Queue Fence Bit
IT_UNLINK_FENCE_FLAG	Not applicable
IT_UNLINK_LOCAL_SINK	Not applicable
IT_COALESCE_WR_FLAG	Not applicable

9772
9773
9774

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_NOTIFY_FLAG	None but can be supported by Implementation.
IT_SOLICITED_WAIT_FLAG	Solicited Event
IT_BARRIER_FENCE_FLAG	Read Fence Indicator
IT_UNLINK_FENCE_FLAG	Local Fence Indicator
IT_UNLINK_LOCAL_SINK	Invalidate Local STag
IT_COALESCE_WR_FLAG	WR List

9775
9776
9777
9778

APPLICATION USAGE

The IT_COMPLETION_FLAG allows “per-Work-Request Completion Suppression”. That is, it controls whether or not a Completion Event is generated and posted to the Event Dispatcher for a successful Work Request.

9779
9780
9781
9782

The IT_NOTIFY_FLAG allows “per-Work-Request Notification Suppression”. That is, it controls whether or not Notification occurs once a Completion Event has been generated for a successful Work Request. It is possible to generate a Completion Event and not generate a Notification.

9783
9784

See Application Usage in *it_ep_state_t* for discussion of flushing WR completions when the WRs have IT_COMPLETION_FLAG cleared.

9785
9786
9787
9788

When posting Send WRs with Completion Suppression (IT_COMPLETION_FLAG cleared) to an Endpoint, the Consumer is advised to enqueue at least one WR with IT_COMPLETION_FLAG set in every *max_request_dtos* number of postings to the Endpoint, in order to preserve the capability to recover from failures.

9789
9790
9791
9792

SEE ALSO

it_post_atomic(), *it_post_send()*, *it_post_sendto()*, *it_post_recv()*, *it_post_recvfrom()*, *it_post_rdma_read()*, *it_post_rdma_write()*, *it_rmr_link()*, *it_rmr_unlink()*, *it_dto_status_t*, *it_dto_events*, *it_ep_state_t*

it_dto_status_t

9793

9794 **NAME**

9795 `it_dto_status_t` – definition of Work Request (DTO, LMR, or RMR operation) completion status

9796 **SYNOPSIS**

```

9797 #include <it_api.h>
9798
9799 typedef enum {
9800     IT_DTO_SUCCESS                = 0,
9801     IT_DTO_ERR_LOCAL_LENGTH      = 1,
9802     IT_DTO_ERR_LOCAL_EP         = 2,
9803     IT_DTO_ERR_LOCAL_PROTECTION = 3,
9804     IT_DTO_ERR_FLUSHED         = 4,
9805     IT_RMR_OPERATION_FAILED     = 5,
9806     IT_DTO_ERR_BAD_RESPONSE     = 6,
9807     IT_DTO_ERR_REMOTE_ACCESS    = 7,
9808     IT_DTO_ERR_REMOTE_RESPONDER = 8,
9809     IT_DTO_ERR_TRANSPORT        = 9,
9810     IT_DTO_ERR_RECEIVER_NOT_READY = 10,
9811     IT_DTO_ERR_PARTIAL_PACKET   = 11,
9812     IT_DTO_ERR_LOCAL_MM_OPERATION = 12
9813 } it_dto_status_t;

```

9814 **DESCRIPTION**

9815 Any successfully initiated Work Request (i.e., Send, Receive, RDMA Read, RDMA Write,
 9816 Atomic, LMR Link, LMR Unlink, RMR Link, or RMR Unlink) can return its completion status
 9817 asynchronously via an Event enqueued on an SEVD. For some WRs, the Consumer can control
 9818 whether an Event is generated via the `IT_COMPLETION_FLAG` (see [it_dto_flags_t](#)). If an
 9819 Event is generated, the completion status is contained in the `it_dto_status_t`.

9820 If the completion status is anything other than `IT_DTO_SUCCESS` for a Reliable Connected
 9821 Endpoint, the Connection will be broken.

9822 The table below enumerates all of the allowed values for `it_dto_status_t`. For each value, a
 9823 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 Work Requests	
		RC	UD
IT_DTO_SUCCESS	The WR completed successfully.	Atomic Send Recv RDMA Read RDMA Write LMR Link LMR Unlink RMR Link RMR Unlink	Sendto Recvfrom

it_dto_status_t Value	Description	Applicable to Work Requests	
		RC	UD
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_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 WR.	Send Recv RDMA Read RDMA Write RMR Link RMR Unlink Atomic LMR Link LMR Unlink	Sendto Recvfrom

it_dto_status_t Value	Description	Applicable to Work Requests	
		RC	UD
IT_DTO_ERR_LOCAL_PROTECTION	<p>One of the local segments in the DTO or the local Endpoint caused a protection violation when the WR was processed, and the possible causes are as follows:</p> <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. • The LMR handle was the Direct LMR Handle and the EP is not a Privileged Mode Endpoint (<i>priv_ops_enable</i> was not set on EP creation). • For <i>it_post_rdma_read</i> or <i>it_post_rdma_read_to_rmr</i> with the IT_UNLINK_LOCAL_SINK DTO flag set, the local sink LMR had an RMR still bound to it. • An incoming <i>it_post_send_and_unlink</i> caused a completion error due to one of the following reasons: The RMR Context in the DTO was invalid, or the RMR Context PZ did not match the EP PZ, or the RMR was a Wide RMR, or the RMR was a Narrow RMR bound to another EP. Also, if the RMR Context represented a local LMR, the LMR had bound RMRs, or the LMR was in the shared state, or the LMR had no remote access enabled, or the LMR was already unlinked or unlinkable (IB-only). 	Send Recv RDMA Read RDMA Write Atomic	Sendto Recvfrom

it_dto_status_t Value	Description	Applicable to Work Requests	
		RC	UD
IT_DTO_ERR_FLUSHED	The Endpoint entered the IT_EP_STATE_NONOPERATIONAL state before processing of the WR could begin.	Send Recv RDMA Read RDMA Write RMR Link RMR Unlink LMR Link LMR Unlink Atomic	Sendto Recvfrom
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 Atomic	N/A

it_dto_status_t Value	Description	Applicable to Work Requests	
		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"> • <i>rmr_context</i> did not represent a valid and linked remote buffer. • 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 Atomic	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 Atomic	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 Atomic	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 Atomic	N/A

it_dto_status_t Value	Description	Applicable to Work Requests	
		RC	UD
IT_DTO_ERR_PARTIAL_PACKET	The data delivered by the Receive DTO was truncated. The contents of the receiver's buffer are unspecified.	Receive	N/A
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). <p>For <i>it_lmr_link</i>:</p> <ul style="list-style-type: none"> The LMR handle was invalid. The PZ of LMR and EP did not match or <i>privs</i> were invalid. The LMR was already in linked state. The LMR Handle was the Direct LMR Handle. The number of IOBL elements exceeded the <i>iobl_num_elts</i> LMR attribute, the IOBL included elements of an unsupported length, the IOBL had an invalid First-Byte Offset (<i>fbo</i>), or if the <i>length</i> of the LMR exceeded the length of the IOBL or if <i>length</i> equaled zero. For IOBLs that are Page Lists, an IOBL element had an invalid page size or was not page aligned or, for Absolute Addressing only, the absolute address (<i>addr</i>) modulo the length (<i>elt_len</i>) of the first IOBL element did not equal the First-Byte Offset (<i>fbo</i>) in the IOBL. The EP was not in Privileged Mode. <p>For <i>it_lmr_unlink</i>:</p> <ul style="list-style-type: none"> The LMR handle was invalid. The LMR was not in linked state. The LMR Handle was the Direct LMR Handle. The PZ of the LMR did not match the PZ of the EP. The LMR had at least one RMR bound to it (via <i>it_rmr_link</i>). The LMR was in the shared state. The LMR could not be unlinked due to the way it was created (only for InfiniBand). 	RMR Link LMR Link LMR Unlink	N/A

9824 **EXTENDED DESCRIPTION**

9825 For the InfiniBand Transport, the following table maps the values in the *it_dto_status_t*
 9826 enumeration to their corresponding “Completion Return Status” values as specified in Chapter
 9827 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

9828 For the iWARP Transport, the following table maps the values in the *it_dto_status_t*
 9829 enumeration to the Completion Status Codes in Section 9.5.2 of [VERBS-RDMAC].
 9830

it_dto_status_t Value	iWARP “Completion Return Status” Name
IT_DTO_SUCCESS	Success
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_status_t Value	iWARP “Completion Return Status” Name
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	S Tag to Invalidate had Invalid PD or Access Rights, S Tag Not In Invalid State, Invalid Region (Bind), Invalid Window (Bind)

9831
9832
9833
9834

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_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.)

9835 **SEE ALSO**
9836 *it_post_atomic(), it_post_send(), it_post_sendto(), it_post_recv(), it_post_recvfrom(),*
9837 *it_post_rdma_read(), it_post_rdma_read_to_rmr(), it_post_rdma_write(), it_rmr_link(),*
9838 *it_rmr_unlink()*

it_ep_attributes_t

9839

9840 **NAME**

9841 it_ep_attributes – Endpoint attributes

9842 **SYNOPSIS**

```
9843 #include <it_api.h>
9844
9845 typedef uint32_t it_ud_ep_id_t;
9846 typedef uint32_t it_ud_ep_key_t;
9847
9848 typedef enum {
9849     IT_EP_PARAM_ALL                = 0x00000001,
9850     IT_EP_PARAM_IA                 = 0x00000002,
9851     IT_EP_PARAM_SPIGOT            = 0x00000004,
9852     IT_EP_PARAM_STATE             = 0x00000008,
9853     IT_EP_PARAM_SERV_TYPE        = 0x00000010,
9854     IT_EP_PARAM_PATH              = 0x00000020,
9855     IT_EP_PARAM_PZ                = 0x00000040,
9856     IT_EP_PARAM_REQ_SEVD         = 0x00000080,
9857     IT_EP_PARAM_RECV_SEVD        = 0x00000100,
9858     IT_EP_PARAM_CONN_SEVD        = 0x00000200,
9859     IT_EP_PARAM_RDMA_RD_ENABLE   = 0x00000400,
9860     IT_EP_PARAM_RDMA_WR_ENABLE   = 0x00000800,
9861     IT_EP_PARAM_MAX_RDMA_READ_SEG = 0x00001000,
9862     IT_EP_PARAM_MAX_RDMA_WRITE_SEG = 0x00002000,
9863     IT_EP_PARAM_MAX_IRD          = 0x00004000,
9864     IT_EP_PARAM_MAX_ORD          = 0x00008000,
9865     IT_EP_PARAM_EP_ID            = 0x00010000,
9866     IT_EP_PARAM_EP_KEY           = 0x00020000,
9867     IT_EP_PARAM_MAX_PAYLOAD      = 0x00040000,
9868     IT_EP_PARAM_MAX_REQ_DTO      = 0x00080000,
9869     IT_EP_PARAM_MAX_RECV_DTO     = 0x00100000,
9870     IT_EP_PARAM_MAX_SEND_SEG     = 0x00200000,
9871     IT_EP_PARAM_MAX_RECV_SEG     = 0x00400000,
9872     IT_EP_PARAM_SRQ              = 0x00800000,
9873     IT_EP_PARAM_SOFT_HI_WATERMARK = 0x01000000,
9874     IT_EP_PARAM_HARD_HI_WATERMARK = 0x02000000,
9875     IT_EP_PARAM_ATOMICS_ENABLE   = 0x04000000,
9876     IT_EP_PARAM_PRIV_OPS_ENABLE  = 0x08000000
9877 } it_ep_param_mask_t;
9878
9879 /*
9880  * The it_ep_param_mask_t value in the comment beside or
9881  * following each attribute is the mask value used to select
9882  * the attribute in the it_ep_query and it_ep_modify calls
9883  */
9884 typedef struct {
9885     it_boolean_t rdma_read_enable;
9886     /* IT_EP_PARAM_RDMA_RD_ENABLE */
9887     it_boolean_t rdma_write_enable;
9888     /* IT_EP_PARAM_RDMA_WR_ENABLE */
9889     size_t max_rdma_read_segments;
```

```

9890         /* IT_EP_PARAM_MAX_RDMA_READ_SEG */
9891     size_t    max_rdma_write_segments;
9892         /* IT_EP_PARAM_MAX_RDMA_WRITE_SEG */
9893     uint32_t  rdma_read_ird;
9894         /* IT_EP_PARAM_MAX_IRD */
9895     uint32_t  rdma_read_ord;
9896         /* IT_EP_PARAM_MAX_ORD */
9897     it_srq_handle_t  srq;
9898         /* IT_EP_PARAM_SRQ */
9899     size_t    soft_hi_watermark;
9900         /* IT_EP_PARAM_SOFT_HI_WATERMARK */
9901     size_t    hard_hi_watermark;
9902         /* IT_EP_PARAM_HARD_HI_WATERMARK */
9903     it_boolean_t  atomics_enable;
9904         /* IT_EP_PARAM_ATOMICS_ENABLE */
9905
9906 } it_rc_only_attributes_t;
9907
9908 #define IT_HARD_HI_WATERMARK_DISABLE ((size_t) -1)
9909
9910 typedef struct {
9911     it_ud_ep_id_t    ud_ep_id;    /* IT_EP_PARAM_EP_ID */
9912     it_ud_ep_key_t   ud_ep_key;   /* IT_EP_PARAM_EP_KEY */
9913 } it_remote_ep_info_t;
9914
9915 typedef struct {
9916     it_remote_ep_info_t  ep_info;
9917
9918 } it_ud_only_attributes_t;
9919
9920 typedef union {
9921     it_rc_only_attributes_t  rc;
9922     it_ud_only_attributes_t  ud;
9923 } it_service_attributes_t;
9924
9925 typedef struct {
9926     size_t    max_dto_payload_size;    /* IT_EP_PARAM_MAX_PAYLOAD */
9927     size_t    max_request_dtos;        /* IT_EP_PARAM_MAX_REQ_DTO */
9928     size_t    max_recv_dtos;          /* IT_EP_PARAM_MAX_RECV_DTO */
9929     size_t    max_send_segments;      /* IT_EP_PARAM_MAX_SEND_SEG */
9930     size_t    max_recv_segments;      /* IT_EP_PARAM_MAX_RECV_SEG */
9931
9932     it_service_attributes_t  srv;
9933
9934     it_boolean_t  priv_ops_enable;    /* IT_EP_PARAM_PRIV_OPS_ENABLE */
9935 } it_ep_attributes_t;

```

9936 DESCRIPTION

9937 `it_ep_attributes` List of Endpoint attributes. The `it_service_attributes_t` union elements are
9938 discriminated by `service_type` found in the `it_ep_param_t` structure in the
9939 [it_ep_query](#) reference page. Mask values for query and modify of Endpoint
9940 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 <code>IT_EP_STATE_NONOPERATIONAL</code> or in the <code>IT_EP_STATE_UNCONNECTED</code> 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 <code>it_post_send</code>, <code>it_post_rdma_read</code>, etc., routines.</p>	UD and RC	Subject to the setting of the <code>ep_work_queues_resizable</code> field in the <code>it_ia_info_t</code> 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 <code>it_post_rcv</code> or <code>it_post_rcvfrom</code> 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 <code>it_ep_query</code> and any attempt to modify the attribute with <code>it_ep_modify</code> shall return an error.</p> <p>For an Endpoint having an associated S-RQ, a corresponding <code>max_rcv_dtos</code> attribute exists as an S-RQ attribute.</p>	UD and RC	Subject to the setting of the <code>ep_work_queues_resizable</code> field in the <code>it_ia_info_t</code> 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>	<p>Maximum number of incoming RDMA Reads from the remote side of the connected Endpoint that can be outstanding simultaneously.</p> <p>If the IA supports Atomics, then this attribute defines the maximum number of incoming Atomics and RDMA Reads that can be outstanding simultaneously.</p>	RC only	When Endpoint is in the IT_EP_STATE_UNCONNECTED or IT_EP_STATE_ACTIVE2_CONNECTION_PENDING state.
<i>rdma_read_ord</i>	<p>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.</p> <p>If the IA supports Atomics, then this attribute defines the maximum number of outgoing Atomics and RDMA Reads that can be outstanding simultaneously.</p>	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</p>	RC only	In any state.

Attribute	Description	Service Type	Modifiable?
	<p>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 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</p>		

Attribute	Description	Service Type	Modifiable?
	Implementation shall ignore the attempt to change the value, and the mechanism shall stay armed.		
<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.

Attribute	Description	Service Type	Modifiable?
<i>atomics_enable</i>	Flag allowing Consumer to enable or disable incoming Atomics operations on this Endpoint. Only usable if <i>it_ia_info_t</i> boolean <i>atomic_support</i> is IT_TRUE.	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>priv_ops_enable</i>	Flag allowing Consumer to request creation of a Privileged Mode Endpoint.	UD and RC	Only on creation.

9941
9942
9943
9944
9945

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.

9946
9947

Exceeding *max_request_dtos* or *max_rcv_dtos* using the post DTO and post RMR operations will result in the post operation returning an error or completing in error.

9948
9949

Posting more RDMA Read operations than specified in *rdma_read_ord* is not an error and will have no adverse effects.

9950 **EXTENDED DESCRIPTION**

9951
9952
9953
9954
9955

In the discussion above, the term “number of Receive DTOs in progress for an Endpoint” is defined to be the number of Receive DTOs which have been consumed on behalf of the Endpoint and for which data has been placed from incoming Send DTOs, but which have not yet completed due to out-of-order Send DTO arrival. For the InfiniBand Transport, this number can be no greater than one. For the iWARP Transport, this number can be greater than one.

9956 **FUTURE DIRECTIONS**

9957

Some new Service Types may be added in the future.

9958 **SEE ALSO**

9959

it_ep_rc_create(), *it_ep_ud_create()*, *it_ep_query()*, *it_ep_modify()*, *it_ia_info_t*

it_ep_state_t

9960

9961 **NAME**

9962 it_ep_state_t – RC and UD Endpoint state type definition

9963 **SYNOPSIS**

```
9964 #include <it_api.h>
9965
9966 typedef enum
9967 {
9968     IT_EP_STATE_UNCONNECTED = 0,
9969     IT_EP_STATE_ACTIVE1_CONNECTION_PENDING = 1,
9970     IT_EP_STATE_ACTIVE2_CONNECTION_PENDING = 2,
9971     IT_EP_STATE_PASSIVE_CONNECTION_PENDING = 3,
9972     IT_EP_STATE_CONNECTED = 4,
9973     IT_EP_STATE_NONOPERATIONAL = 5,
9974     IT_EP_STATE_PASSIVE_WAIT_RDMA_TRANS_REQ = 6
9975 } it_ep_state_rc_t;
9976
9977 typedef enum
9978 {
9979     IT_EP_STATE_UD_NONOPERATIONAL = 0,
9980     IT_EP_STATE_UD_OPERATIONAL = 1
9981 } it_ep_state_ud_t;
9982
9983 typedef union
9984 {
9985     it_ep_state_rc_t rc;
9986     it_ep_state_ud_t ud;
9987 } it_ep_state_t;
```

9988 **DESCRIPTION**

9989 The following table identifies and describes the RC Endpoint states. The Transport-Independent
9990 Interface (TII) and the Transport-Dependent Interface (TDI) use these states in a consistent way.
9991 For each state, the table lists the API routines that can be legally applied to an RC Endpoint in
9992 that state.

9993 Whenever an Endpoint transitions its state, at most one Event is generated for that transition.
9994 The Endpoints state transitions before the Communication Management Message Event is
9995 enqueued. This guarantees that the Consumer can only dequeue Events after the state transition
9996 has occurred. Subsequent state transitions and their related Events will occur regardless of
9997 whether the Consumer is dequeuing the Events.

9998

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_atomic</i> , <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_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_atomic</i> , <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_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_rcv</i> , <i>it_set_consumer_context</i> , <i>it_ep_disconnect</i>

9999
10000
10001
10002

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

10003
10004
10005

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>

10006
10007
10008
10009
10010
10011
10012
10013

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.

10014

IT_EP_STATE_UNCONNECTED

10015
10016
10017

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.

10018

IT_EP_STATE_ACTIVE1_CONNECTION_PENDING

10019
10020
10021

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.

10022
10023
10024
10025

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_

10026 PEER_REJECT_EVENT Event and the Endpoint will transition into the IT_EP_
10027 STATE_NONOPERATIONAL state.

10028 In the case of three-way Connection establishment, the Active side Endpoint will transition to
10029 IT_EP_STATE_ACTIVE2_CONNECTION_PENDING when an IT_CM_MSG_CONN_
10030 ACCEPT_ARRIVAL_EVENT Event is enqueued for the Active side Consumer. If the Active
10031 Consumer calls *it_reject* after processing the IT_CM_MSG_CONN_ACCEPT_
10032 ARRIVAL_EVENT Event, the Endpoint will transition into the IT_EP_STATE_
10033 NONOPERATIONAL state. If the Passive side rejects the Connection, the Active side will
10034 receive an IT_CM_MSG_CONN_PEER_REJECT_EVENT Event and the Endpoint will
10035 transition into the IT_EP_STATE_NONOPERATIONAL state.

10036 **IT_EP_STATE_ACTIVE2_CONNECTION_PENDING**

10037 In the case of three-way Connection establishment, the Endpoint will transition to the
10038 IT_EP_STATE_CONNECTED state when the Active side Consumer successfully calls
10039 *it_ep_accept*, or the Endpoint will transition to the IT_EP_STATE_NONOPERATIONAL state
10040 if the Active side Consumer successfully calls *it_reject*.

10041 In the case of two-way Connection establishment, this state is transient, and the Endpoint will
10042 transition to the IT_EP_STATE_CONNECTED state when the Connection is successfully
10043 established.

10044 **IT_EP_STATE_PASSIVE_CONNECTION_PENDING**

10045 The Passive side Endpoint transitions into this state when the Consumer calls *it_ep_accept*
10046 during Connection establishment. In this state only Receive Data Transfer Operations can be
10047 posted. For iWARP only, this is also a transient state during Socket Conversion on the
10048 Conversion Initiator side.

10049 From this state the Endpoint will transition into the IT_EP_STATE_CONNECTED state when
10050 Connection establishment completes successfully.

10051 **IT_EP_STATE_CONNECTED**

10052 This state is entered when Connection establishment completes. Upon transition to this state, the
10053 Implementation delivers an IT_CM_MSG_CONN_ESTABLISHED_EVENT Event. All types
10054 of Data Transfer and Link Operations can be posted and will be processed in this state.

10055 If either the Local or Remote Consumer disconnects, the Endpoint will transition into the
10056 IT_EP_STATE_NONOPERATIONAL state, and the Implementation will deliver an
10057 IT_CM_MSG_CONN_DISCONNECT_EVENT Event.

10058 Local or Remote errors (e.g., protection violations) also cause the Endpoint to transition to the
10059 IT_EP_STATE_NONOPERATIONAL state, and the Implementation will deliver an
10060 IT_CM_MSG_CONN_BROKEN_EVENT Event.

10061 The transition out of the IT_EP_STATE_CONNECTED state is surfaced by either the
10062 IT_CM_CONN_DISCONNECT_EVENT Event or the IT_CM_MSG_CONN_BROKEN_
10063 EVENT Event, but never both.

10064 **IT_EP_STATE_NONOPERATIONAL**

10065 In this state no requests will be processed by the Endpoint and any well-formed requests posted
10066 will generate Completions with a failing Completion Status. The Endpoint will remain in this
10067 state until *it_ep_reset* is used to put the Endpoint back into the
10068 IT_EP_STATE_UNCONNECTED state.

10069 **IT_EP_STATE_PASSIVE_WAIT_RDMA_TRANS_REQ**

10070 For iWARP only, the Passive side Endpoint; i.e., the Endpoint of the RDMA Responder or,
10071 equivalently, of the Conversion Initiator, transitions into this state when the Conversion Initiator
10072 calls *it_socket_convert*. In this state only Receive Data Transfer Operations can be posted. From
10073 this state, the Endpoint will transition into the transient IT_EP_STATE_
10074 PASSIVE_CONNECTION_PENDING state when a request for transitioning to RDMA mode is
10075 received from the remote side. When Connection establishment completes successfully, the
10076 Endpoint will finally transition to the IT_EP_STATE_CONNECTED state.

10077 **IT_EP_STATE_UD_OPERATIONAL**

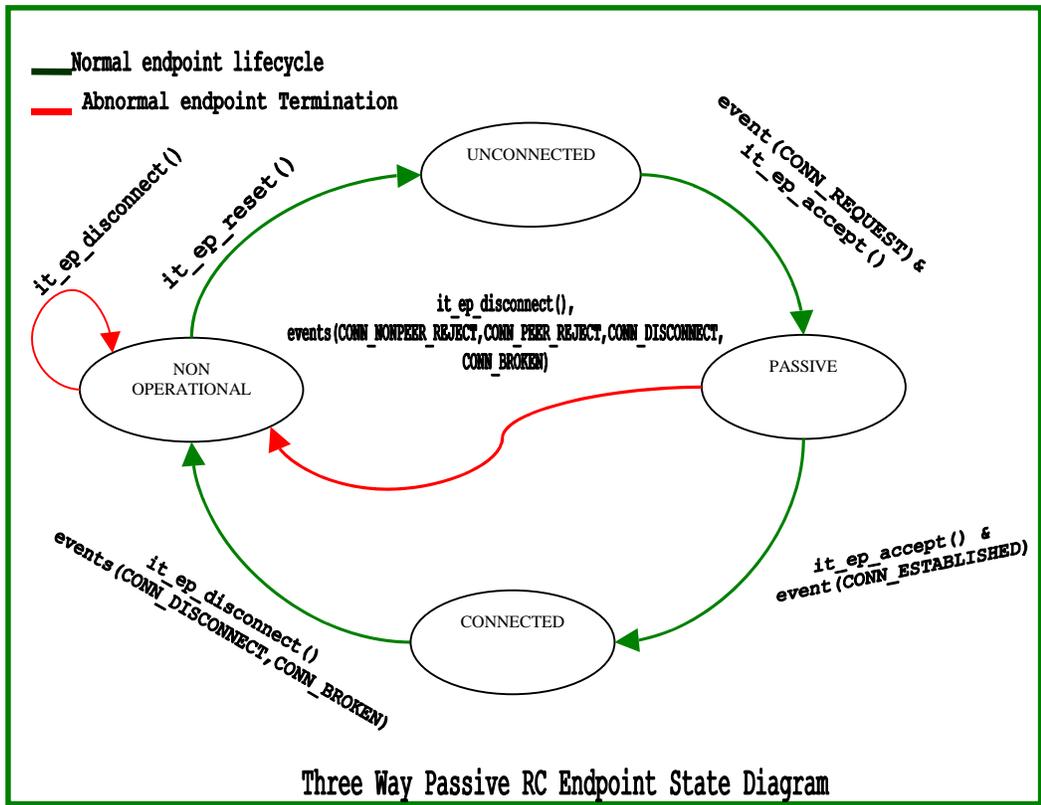
10078 In this state requests will be processed by the Endpoint.

10079 **IT_EP_STATE_UD_NONOPERATIONAL**

10080 In this state no requests will be processed by the Endpoint and any well-formed requests posted
10081 will generate Completions with a failing Completion Status. Once an Unreliable Datagram
10082 Endpoint enters this state it can only be destroyed with *it_ep_free*.

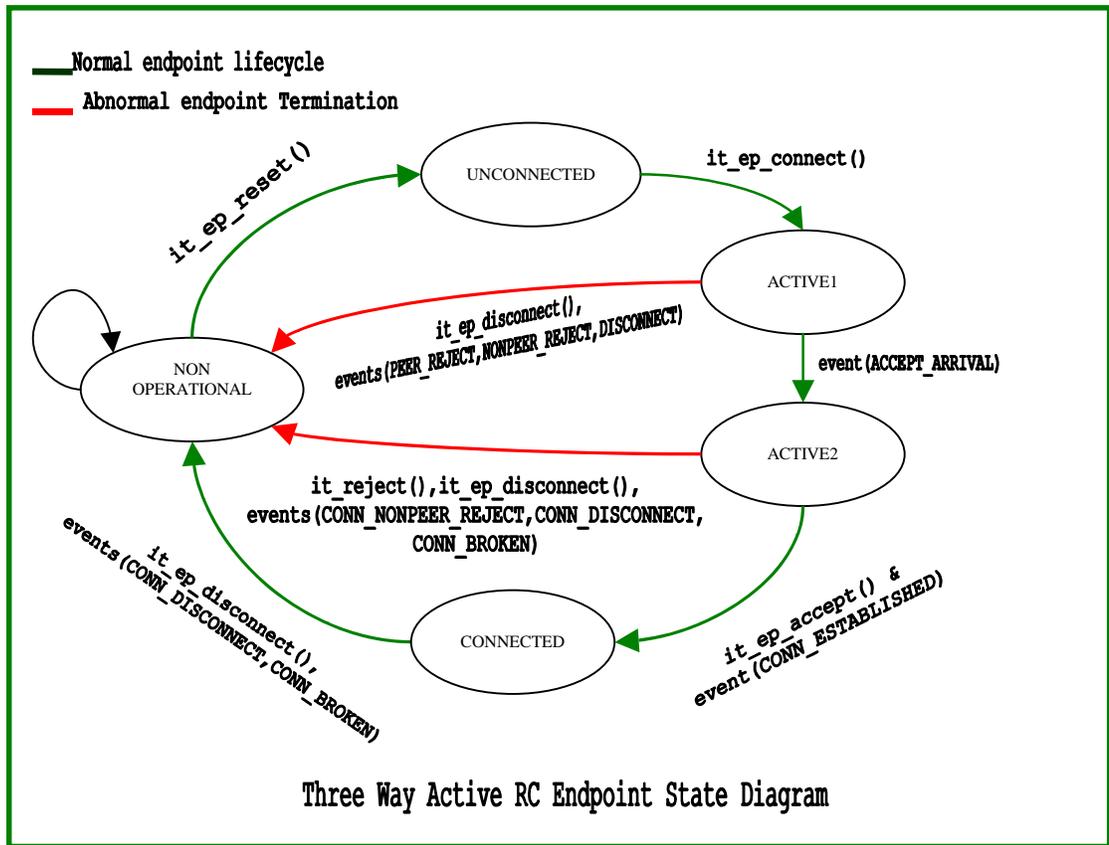
10083 **EXTENDED DESCRIPTION**

10084 For the TII, the state diagrams are shown below, separately for three-way and two-way
10085 Connection establishment, and for the active and passive sides.



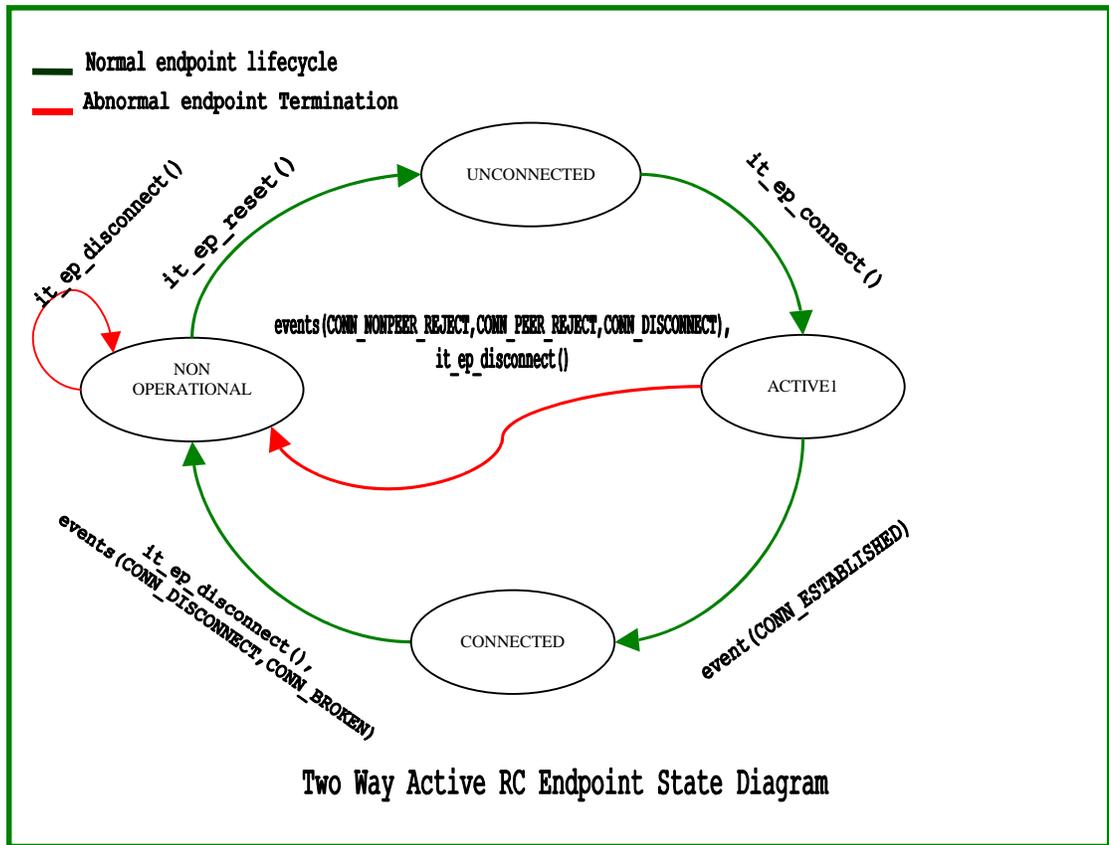
10086
10087

Figure 8: Three Way Passive RC Endpoint State Diagram



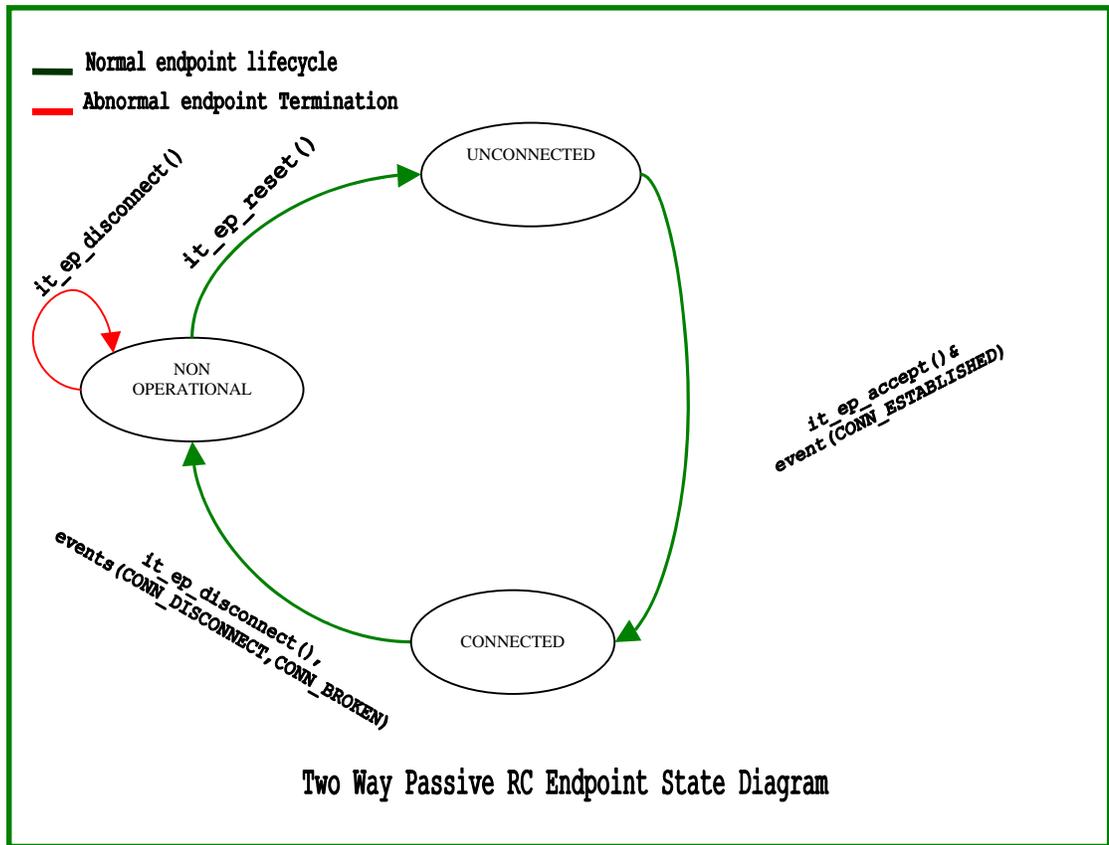
10088
10089

Figure 9: Three Way Active RC Endpoint State Diagram



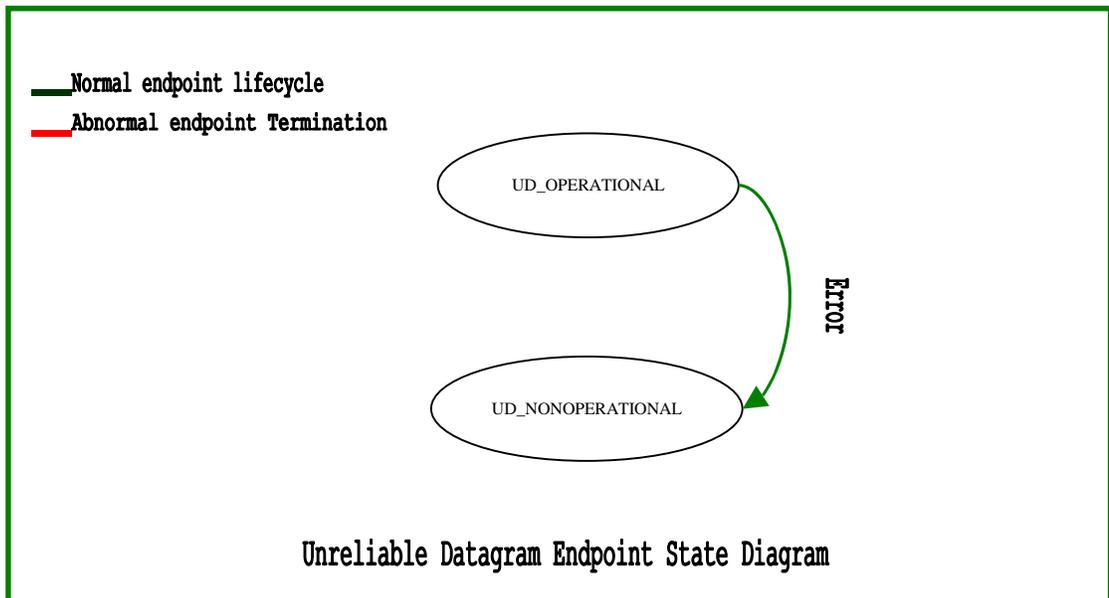
10090
10091

Figure 10: Two Way Active RC Endpoint State Diagram



10092
10093

Figure 11: Two Way Passive RC Endpoint State Diagram



10094
10095
10096

Figure 12: Unreliable Datagram Endpoint State Diagram

For the state diagrams applicable to the TDI, see [it_socket_convert](#).

10097 **APPLICATION USAGE**

- 10098
10099
10100
10101
10102
10103
10104
10105
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.
- 10106
10107
10108
10109
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.

10110 **SEE ALSO**

10111 *it_ep_accept()*, *it_reject()*, *it_ep_disconnect()*, *it_handoff()*, *it_ep_reset()*

10112

10113 **NAME**

10114 it_event – definition of Event data structures

10115 **SYNOPSIS**

```

10116 #include <it_api.h>
10117
10118 #define IT_EVENT_STREAM_MASK    0xff000
10119 #define IT_TIMEOUT_INFINITE    ((uint64_t)(-1))
10120
10121 typedef enum
10122 {
10123     /*
10124      * Event Stream for WR/DTO completions
10125      */
10126     IT_DTO_EVENT_STREAM          = 0x00000,
10127     IT_DTO_SEND_CMPL_EVENT      = 0x00001,
10128     IT_DTO_RC_RECV_CMPL_EVENT  = 0x00002,
10129     IT_DTO_UD_RECV_CMPL_EVENT  = 0x00003,
10130     IT_DTO_RDMA_WRITE_CMPL_EVENT = 0x00004,
10131     IT_DTO_RDMA_READ_CMPL_EVENT = 0x00005,
10132     IT_RMR_BIND_CMPL_EVENT     = 0x00006,
10133     IT_RMR_LINK_CMPL_EVENT     = 0x00006,
10134     IT_DTO_FETCH_ADD_CMPL_EVENT = 0x00007,
10135     IT_DTO_CMP_SWAP_CMPL_EVENT = 0x00008,
10136     IT_LMR_LINK_CMPL_EVENT     = 0x00009,
10137
10138     /*
10139      * Event Stream for Communication Management Request Events
10140      */
10141     IT_CM_REQ_EVENT_STREAM      = 0x01000,
10142     IT_CM_REQ_CONN_REQUEST_EVENT = 0x01001,
10143     IT_CM_REQ_UD_SERVICE_REQUEST_EVENT = 0x01002,
10144
10145     /*
10146      * Event Stream for Communication Management Message Events
10147      */
10148     IT_CM_MSG_EVENT_STREAM      = 0x02000,
10149     IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT = 0x02001,
10150     IT_CM_MSG_CONN_ESTABLISHED_EVENT = 0x02002,
10151     IT_CM_MSG_CONN_DISCONNECT_EVENT = 0x02003,
10152     IT_CM_MSG_CONN_PEER_REJECT_EVENT = 0x02004,
10153     IT_CM_MSG_CONN_NONPEER_REJECT_EVENT = 0x02005,
10154     IT_CM_MSG_CONN_BROKEN_EVENT = 0x02006,
10155     IT_CM_MSG_UD_SERVICE_REPLY_EVENT = 0x02007,
10156
10157     /* Event Stream for Affiliated Asynchronous Events */
10158     IT_ASYNC_AFF_EVENT_STREAM    = 0x04000,
10159     IT_ASYNC_AFF_SEVD_ENQUEUE_FAILURE = 0x04001,
10160     IT_ASYNC_AFF_EP_FAILURE      = 0x04002,
10161     IT_ASYNC_AFF_EP_BAD_TRANSPORT_OPCODE = 0x04003,
10162     IT_ASYNC_AFF_EP_REQ_DROPPED = 0x04005,

```

```

10163     IT_ASYNC_AFF_EP_RDMAW_ACCESS_VIOLATION    = 0x04006,
10164     IT_ASYNC_AFF_EP_RDMAW_CORRUPT_DATA        = 0x04007,
10165     IT_ASYNC_AFF_EP_RDMAR_ACCESS_VIOLATION    = 0x04008,
10166     IT_ASYNC_AFF_EP_LOCAL_ACCESS_VIOLATION    = 0x04020,
10167     IT_ASYNC_AFF_EP_L_ACCESS_VIOLATION        = 0x04020,
10168     IT_ASYNC_AFF_EP_L_RECV_ACCESS_VIOLATION    = 0x04021,
10169     IT_ASYNC_AFF_EP_L_IRRQ_ACCESS_VIOLATION    = 0x04022,
10170     IT_ASYNC_AFF_EP_L_TRANSPORT_ERROR         = 0x04023,
10171     IT_ASYNC_AFF_EP_L_LLP_ERROR               = 0x04024,
10172     IT_ASYNC_AFF_EP_R_ERROR                   = 0x04040,
10173     IT_ASYNC_AFF_EP_R_ACCESS_VIOLATION        = 0x04041,
10174     IT_ASYNC_AFF_EP_R_RECV_ACCESS_VIOLATION    = 0x04042,
10175     IT_ASYNC_AFF_EP_R_RECV_LENGTH_ERROR       = 0x04043,
10176     IT_ASYNC_AFF_EP_SOFT_HI_WATERMARK         = 0x04060,
10177     IT_ASYNC_AFF_SRQ_LOW_WATERMARK            = 0x04100,
10178
10179     /* Event Stream for Unaffiliated Asynchronous Events */
10180     IT_ASYNC_UNAFF_EVENT_STREAM                = 0x08000,
10181     /* 0x08001 is deprecated */
10182     IT_ASYNC_UNAFF_SPIGOT_ONLINE              = 0x08002,
10183     IT_ASYNC_UNAFF_SPIGOT_OFFLINE            = 0x08003,
10184     IT_ASYNC_UNAFF_SEVD_ENQUEUE_FAILURE       = 0x08004,
10185
10186     /* Event Stream for Software Events */
10187     IT_SOFTWARE_EVENT_STREAM                  = 0x10000,
10188     IT_SOFTWARE_EVENT                        = 0x10001,
10189
10190     /* Event Stream for AEVD Notifications */
10191     IT_AEVD_NOTIFICATION_EVENT_STREAM         = 0x20000,
10192     IT_AEVD_NOTIFICATION_EVENT               = 0x20001
10193 } it_event_type_t;
10194
10195 typedef struct {
10196     it_event_type_t  event_number;
10197     it_evd_handle_t  evd;
10198 } it_any_event_t;
10199
10200 typedef union
10201 {
10202     /*
10203     * The following two union elements are
10204     * available for programming convenience.
10205     *
10206     * The event_number may be used to determine the
10207     * it_event_type_t of any Event. it_any_event_t
10208     * allows the EVD to be determined as well.
10209     */
10210     it_event_type_t  event_number;
10211     it_any_event_t   any;
10212
10213     /*
10214     * The remaining union elements correspond to
10215     * the various it_event_type_t types.
10216     */

```

```

10217
10218 /*
10219 * The following two Event structures
10220 * support the IT_DTO_EVENT_STREAM Event Stream.
10221 *
10222 * it_dto_cmpl_event_t supports
10223 * only the following events:
10224 *     IT_DTO_SEND_CMPL_EVENT
10225 *     IT_DTO_RC_RECV_CMPL_EVENT
10226 *     IT_DTO_RDMA_WRITE_CMPL_EVENT
10227 *     IT_DTO_RDMA_READ_CMPL_EVENT
10228 *     IT_RMR_BIND_CMPL_EVENT = IT_RMR_LINK_CMPL_EVENT
10229 *     IT_DTO_FETCH_ADD_CMPL_EVENT
10230 *     IT_DTO_CMP_SWAP_CMPL_EVENT
10231 *     IT_LMR_LINK_CMPL_EVENT
10232 *
10233 * it_all_dto_cmpl_event_t supports all
10234 * possible DTO and RMR events:
10235 *     IT_DTO_SEND_CMPL_EVENT
10236 *     IT_DTO_RC_RECV_CMPL_EVENT
10237 *     IT_DTO_UD_RECV_CMPL_EVENT
10238 *     IT_DTO_RDMA_WRITE_CMPL_EVENT
10239 *     IT_DTO_RDMA_READ_CMPL_EVENT
10240 *     IT_RMR_BIND_CMPL_EVENT = IT_RMR_LINK_CMPL_EVENT
10241 *     IT_DTO_FETCH_ADD_CMPL_EVENT
10242 *     IT_DTO_CMP_SWAP_CMPL_EVENT
10243 *     IT_LMR_LINK_CMPL_EVENT
10244 */
10245 it_dto_cmpl_event_t      dto_cmpl;
10246 it_all_dto_cmpl_event_t all_dto_cmpl;
10247
10248 /*
10249 * The following two Event structures
10250 * support the IT_CM_REQ_EVENT_STREAM Event
10251 * stream:
10252 *
10253 * it_conn_request_event_t supports:
10254 *     IT_CM_REQ_CONN_REQUEST_EVENT
10255 *
10256 * it_ud_svc_request_event_t supports:
10257 *     IT_CM_REQ_UD_SERVICE_REQUEST_EVENT
10258 */
10259 it_conn_request_event_t  conn_req;
10260 it_ud_svc_request_event_t ud_svc_request;
10261
10262 /*
10263 * The following two Event structures
10264 * support the IT_CM_MSG_EVENT_STREAM Event
10265 * stream:
10266 *
10267 * it_connection_event_t supports:
10268 *     IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT
10269 *     IT_CM_MSG_CONN_ESTABLISHED_EVENT
10270 *     IT_CM_MSG_CONN_PEER_REJECT_EVENT

```

```

10271      *      IT_CM_MSG_CONN_NONPEER_REJECT_EVENT
10272      *      IT_CM_MSG_CONN_DISCONNECT_EVENT
10273      *      IT_CM_MSG_CONN_BROKEN_EVENT
10274      *
10275      * it_ud_svc_reply_event_t supports:
10276      *      IT_CM_MSG_UD_SERVICE_REPLY_EVENT
10277      */
10278      it_connection_event_t      conn;
10279      it_ud_svc_reply_event_t    ud_svc_reply;
10280
10281      /*
10282      * it_affiliated_event_t supports
10283      * the following Event Stream:
10284      *      IT_ASYNC_AFF_EVENT_STREAM
10285      */
10286      it_affiliated_event_t      aff_async;
10287
10288      /*
10289      * it_unaffiliated_event_t supports
10290      * the following Event Stream:
10291      *      IT_ASYNC_UNAFF_EVENT_STREAM
10292      */
10293      it_unaffiliated_event_t    unaff_async;
10294
10295      /*
10296      * it_software_event_t supports
10297      * the following Event Stream:
10298      *      IT_SOFTWARE_EVENT_STREAM
10299      */
10300      it_software_event_t        sw;
10301
10302      /*
10303      * it_aevd_notification_event_t supports
10304      * the following Event Stream:
10305      *      IT_AEVD_NOTIFICATION_EVENT_STREAM
10306      */
10307      it_aevd_notification_event_t  aevd_notify;
10308  } it_event_t;

```

10309 DESCRIPTION

10310 The *it_event_t* defines the format for Events for IT-APIs. Each Event consists of a Handle to the
10311 EVD where the Event has been queued, and Event type identifier with the Event type-specific
10312 Event data.

10313 Events for a Simple EVD can be fed from only a single Event Stream type.

10314 Multiple Event numbers that can be on the same Event Stream type form an Event group. The
10315 Event data formats for all Event types of the same Event Stream type are defined in one or more
10316 separate reference page(s) specific to each Event group.

10317 APPLICATION USAGE

10318 The Consumer allocates an *it_event_t* object and passes it into the *it_evd_wait* or *it_evd_dequeue*
10319 calls in order to retrieve Events. The *it_event_t* object is a union of all possible Event Stream
10320 data types, thus it is the size of the largest possible Event type.

10321 If the Consumer wishes to conserve memory and use only the minimally-sized Event data
10322 structures found in the *it_event_t* union, they are free to. Use of *it_event_t* structures that are too
10323 small for an Event Stream may cause program termination.

10324 The Consumer may use the IT_EVENT_STREAM_MASK to convert from an *it_event_type_t*
10325 *event_number* to Event Stream by masking off the lower bits of the Event number.

10326 **SEE ALSO**
10327 *it_aevd_notification_event_t*, *it_affiliated_event_t*, *it_cm_msg_events*, *it_cm_req_events*,
10328 *it_dto_events*, *it_software_event_t*, *it_unaffiliated_event_t*, *it_evd_wait()*, *it_evd_dequeue()*,
10329 *it_evd_create()*

10330

10331 **NAME**

10332 it_handle_t – enumeration and type definitions for IT Handles

10333 **SYNOPSIS**

```

10334 #include <it_api.h>
10335
10336 typedef enum {
10337     IT_HANDLE_TYPE_ADDR,
10338     IT_HANDLE_TYPE_EP,
10339     IT_HANDLE_TYPE_EVD,
10340     IT_HANDLE_TYPE_IA,
10341     IT_HANDLE_TYPE_LISTEN,
10342     IT_HANDLE_TYPE_LMR,
10343     IT_HANDLE_TYPE_PZ,
10344     IT_HANDLE_TYPE_RMR,
10345     IT_HANDLE_TYPE_UD_SVC_REQ,
10346     IT_HANDLE_TYPE_SRQ
10347 } it_handle_type_enum_t;
10348
10349 typedef void *it_handle_t;
10350 #define IT_NULL_HANDLE ((it_handle_t) NULL)
10351
10352 typedef struct it_addr_handle_s      *it_addr_handle_t;
10353 typedef struct it_ep_handle_s        *it_ep_handle_t;
10354 typedef struct it_evd_handle_s       *it_evd_handle_t;
10355 typedef struct it_ia_handle_s        *it_ia_handle_t;
10356 typedef struct it_listen_handle_s    *it_listen_handle_t;
10357 typedef struct it_lmr_handle_s       *it_lmr_handle_t;
10358 typedef struct it_pz_handle_s        *it_pz_handle_t;
10359 typedef struct it_rmr_handle_s       *it_rmr_handle_t;
10360 typedef struct it_ud_svc_req_handle_s *it_ud_svc_req_handle_t;
10361 typedef struct it_srq_handle_s       *it_srq_handle_t;
    
```

10362 **DESCRIPTION**

10363 The *it_handle_type_enum_t* associates an enumerated value with each type of Handle used in the
 10364 API Implementation. The enumeration is used to describe the type of a Handle returned by
 10365 [it_get_handle_type](#).

10366 The table below defines the relationship of IT-API Handle types and the associated
 10367 *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

10368 **SEE ALSO**
10369 [*it_get_handle_type\(\)*](#)

10370

10371 **NAME**

10372 it_ia_info_t – encapsulates all Interface Adapter attributes and Spigot information

10373 **SYNOPSIS**

```

10374 #include <it_api.h>
10375
10376 /* Enumerates all the transport types supported by the API. */
10377 typedef enum {
10378
10379     /* InfiniBand Transport */
10380     IT_IB_TRANSPORT = 1,
10381
10382     /* VIA host Interface using IP transport, supporting
10383      only the Reliable Delivery reliability level */
10384     IT_VIA_IP_TRANSPORT = 2,
10385
10386     /* VIA host Interface, using Fibre Channel transport, supporting
10387      only the Reliable Delivery reliability level */
10388     IT_VIA_FC_TRANSPORT = 3,
10389
10390     /* iWARP over TCP transport */
10391     IT_IWARP_TCP_TRANSPORT = 4,
10392
10393     /* Vendor-proprietary Transport */
10394     IT_VENDOR_TRANSPORT = 1000
10395
10396 } it_transport_type_t;
10397
10398 /* Transport Service Type definitions. */
10399 typedef enum {
10400
10401     /* Reliable Connected Transport Service Type */
10402     IT_RC_SERVICE = 0x1,
10403
10404     /* Unreliable Datagram Transport Service Type */
10405     IT_UD_SERVICE = 0x2,
10406
10407 } it_transport_service_type_t;
10408
10409 /* The following structure describes an Interface Adapter Spigot */
10410 typedef struct {
10411
10412     /* Spigot identifier */
10413     size_t spigot_id;
10414
10415     /* Maximum sized Send operation for the RC service
10416      on this Spigot. */
10417     size_t max_rc_send_len;
10418
10419     /* Maximum sized RDMA Read/Write operation for the RC service on
10420      this Spigot. */

```

```

10421     size_t  max_rc_rdma_len;
10422
10423     /* Maximum sized Send operation for the UD service
10424        on this Spigot. */
10425     size_t  max_ud_send_len;
10426
10427     /* Indicates whether the Spigot is online or offline.
10428        An IT_TRUE value means online. */
10429     it_boolean_t  spigot_online;
10430
10431     /* A mask indicating which Connection Qualifier types this
10432        IA supports for input to it_ep_connect and
10433        it_ud_service_request_handle_create. The bits in the mask are
10434        an inclusive OR of the values for Connection Qualifier types
10435        that this IA supports. */
10436     it_conn_qual_type_t  active_side_conn_qual;
10437
10438     /* A mask indicating which Connection Qualifier types this to
10439        it_listen_create. The bits in the mask are an inclusive OR of
10440        the values for Connection Qualifier types that this IA
10441        supports. */
10442     it_conn_qual_type_t  passive_side_conn_qual;
10443
10444     /* The number of Network Addresses associated with Spigot. */
10445     size_t  num_net_addr;
10446
10447     /* Pointer to array of Network Address addresses. */
10448     it_net_addr_t*  net_addr;
10449
10450 } it_spigot_info_t;
10451
10452 /* The following structure is used to identify the vendor associated
10453    with an IA that uses the IB transport*/
10454 typedef struct {
10455
10456     /* The NodeInfo:VendorID as described in chapter 14 of the
10457        IB spec. */
10458     uint32_t  vendor : 24;
10459
10460     /* The NodeInfo:DeviceID as described in chapter 14 of the
10461        IB spec. */
10462     uint16_t  device;
10463
10464     /* The NodeInfo:Revision as described in chapter 14 of the
10465        IB spec. */
10466     uint32_t  revision;
10467 } it_vendor_ib_t;
10468
10469 /* The following structure is used to identify the vendor associated
10470    with an IA that uses a VIA transport. */
10471 typedef struct {
10472     /* The "Name" member of the VIP_NIC_ATTRIBUTES structure, as
10473        described in the VIA spec. */
10474     char  name[64];

```

```

10475
10476     /* The "HardwareVersion" member of the VIP_NIC_ATTRIBUTES
10477     structure, as described in the VIA spec. */
10478     unsigned long hardware;
10479
10480     /* The "ProviderVersion" member of the VIP_NIC_ATTRIBUTES
10481     structure, as described in the VIA spec. */
10482     unsigned long provider;
10483 } it_vendor_via_t;
10484
10485 /* The following structure is used to identify the vendor associated
10486 with an IA that uses an iWARP TCP transport. */
10487 typedef struct {
10488     /* Indicates whether or not vid field contains valid data. */
10489     it_boolean_t valid_vid;
10490
10491     /* Vendor Identification field - strictly vendor-specific (only
10492     valid if valid_vid field is IT_TRUE). */
10493     unsigned char vid[64];
10494 } it_vendor_iwarp_tcp_t;
10495
10496 /* The following structure is returned by the it_ia_query function. */
10497 typedef struct {
10498
10499     /* Interface Adapter name, as specified in it_ia_create. */
10500     char* ia_name;
10501
10502     /* The major version number of the latest version of the
10503     IT-API that this IA supports. */
10504     uint32_t api_major_version;
10505
10506     /* The minor version number of the latest version of the
10507     IT-API that this IA supports. */
10508     uint32_t api_minor_version;
10509
10510     /* The major version number for the software being used to control
10511     this IA. The IT-API imposes no structure whatsoever on this
10512     number; its meaning is completely IA-dependent. */
10513     uint32_t sw_major_version;
10514
10515     /* The minor version number for the software being used to control
10516     this IA. The IT-API imposes no structure whatsoever on this
10517     number; its meaning is completely IA-dependent. */
10518     uint32_t sw_minor_version;
10519
10520     /* The vendor associated with the IA. This information is useful
10521     if the Consumer wishes to do device-specific programming. This
10522     union is discriminated by transport_type. No vendor
10523     identification is provided for transports not listed below. */
10524     union {
10525
10526         /* Used if transport_type is IT_IB_TRANSPORT. */
10527         it_vendor_ib_t ib;
10528

```

```

10529         /* Used if transport_type is IT_VIA_IP_TRANSPORT or
10530            IT_VIA_FC_TRANSPORT. */
10531         it_vendor_via_t   via;
10532
10533         /* Used if transport_type is IT_IWARP_TCP_TRANSPORT. */
10534         it_vendor_iwarp_tcp_t iwarp;
10535
10536     } vendor;
10537
10538     /* The Interface Adapter and platform provide a data alignment hint
10539        to the Consumer to help the Consumer align their data transfer
10540        buffers in a way that is optimal for the performance of the IA.
10541        For example, if the best throughput is obtained by aligning
10542        buffers to 128-byte boundaries, dto_alignment_hint will have the
10543        value 128. The Consumer may choose to ignore the alignment hint
10544        without any adverse functional impact. (There may be an adverse
10545        performance impact.) */
10546     uint32_t  dto_alignment_hint;
10547
10548     /* The transport type (e.g., InfiniBand) supported by Interface
10549        Adapter. An Interface Adapter supports precisely one transport
10550        type. */
10551     it_transport_type_t  transport_type;
10552
10553     /* The Transport Service Types supported by this IA. This is
10554        constructed by doing an inclusive OR of the Transport Service
10555        Type values.*/
10556     it_transport_service_type_t  supported_service_types;
10557
10558     /* Indicates whether Work Queues are resizable. */
10559     it_boolean_t  ep_work_queues_resizable;
10560
10561     /* Indicates whether the underlying transport used by this IA uses
10562        a three-way handshake for doing Connection establishment. Note
10563        that if the underlying transport supports a three-way handshake
10564        the Consumer can choose whether to use two handshakes or three
10565        when establishing the Connection. If the underlying transport
10566        supports a two-way handshake for establishing a Connection, the
10567        Consumer can only use two handshakes when establishing the
10568        Connection. */
10569     it_boolean_t  three_way_handshake_support;
10570
10571     /* Indicates whether Private Data is supported on Connection
10572        establishment or UD service resolution operations. */
10573     it_boolean_t  private_data_support;
10574
10575     /* Indicates whether the max_message_size field in the
10576        IT_CM_REQ_CONN_REQUEST_EVENT is valid for this IA. */
10577     it_boolean_t  max_message_size_support;
10578
10579     /* Indicates whether or not the IA supports IRD/ORD. Affects
10580        whether the rdma_read_ird or rdma_read_ord fields in the
10581        IT_CM_REQ_CONN_REQUEST_EVENT or the
10582        IT_CM_MSG_CONN_ESTABLISHED_EVENT are valid for this IA.

```

```

10583         Also affects whether IRD/ORD suppression is an option.
10584         Deprecates IT-API 1.0 values "ird_support" and "ord_support". */
10585 it_boolean_t ird_ord_ia_support;
10586
10587 /* Indicates whether IRD/ORD suppression is supported
10588    for this IA. If this member has a value of IT_TRUE, the
10589    Consumer can control IRD/ORD suppression in it_ep_connect
10590    and it_listen_create. Otherwise they cannot. */
10591 it_boolean_t ird_ord_suppressible;
10592
10593 /* Indicates whether the IA generates IT_ASYNC_UNAFF_SPIGOT_ONLINE
10594    Events. See it_unaffiliated_event_t for details. */
10595 it_boolean_t spigot_online_support;
10596
10597 /* Indicates whether the IA generates IT_ASYNC_UNAFF_SPIGOT_OFFLINE
10598    Events. See it_unaffiliated_event_t for details. */
10599 it_boolean_t spigot_offline_support;
10600
10601 /* The maximum number of bytes of Private Data supported for the
10602    it_ep_connect routine. This will be less than or equal to
10603    IT_MAX_PRIV_DATA. */
10604 size_t connect_private_data_len;
10605
10606 /* The maximum number of bytes of Private Data supported for the
10607    it_ep_accept routine. This will be less than or equal to
10608    IT_MAX_PRIV_DATA. */
10609 size_t accept_private_data_len;
10610
10611 /* The maximum number of bytes of Private Data supported for the
10612    it_reject routine. This will be less than or equal to
10613    IT_MAX_PRIV_DATA. */
10614 size_t reject_private_data_len;
10615
10616 /* The maximum number of bytes of Private Data supported for the
10617    it_ep_disconnect routine. This will be less than or equal to
10618    IT_MAX_PRIV_DATA. */
10619 size_t disconnect_private_data_len;
10620
10621 /* The maximum number of bytes of Private Data supported for the
10622    it_ud_service_request_handle_create routine. This will be
10623    less than or equal to IT_MAX_PRIV_DATA. */
10624 size_t ud_req_private_data_len;
10625
10626 /* The maximum number of bytes of Private Data supported for the
10627    it_ud_service_reply routine. This will be less than or equal to
10628    IT_MAX_PRIV_DATA. */
10629 size_t ud_rep_private_data_len;
10630
10631 /* Specifies the number of Spigots associated with this Interface
10632    Adapter. */
10633 size_t num_spigots;
10634
10635 /* An array of Spigot information data structures. The array
10636    contains num_spigots elements. */

```

```

10637     it_spigot_info_t*   spigot_info;
10638
10639     /* The Handle for the EVD that contains the affiliated async Event
10640        Stream. If no EVD contains the Affiliated Async Event Stream,
10641        this member will have the distinguished value IT_NULL_HANDLE. */
10642     it_evd_handle_t   affiliated_err_evd;
10643
10644     /* The Handle for the EVD that contains the Unaffiliated Async
10645        Event Stream. If no EVD contains the Unaffiliated Async Event
10646        Stream, this member will have the distinguished value
10647        IT_NULL_HANDLE. */
10648     it_evd_handle_t   unaffiliated_err_evd;
10649
10650     /* Indicates whether the IA supports the S-RQ feature. */
10651     it_boolean_t      srq_support;
10652
10653     /* Indicates whether the IA supports the Endpoint Hard
10654        High Watermark mechanism for limiting the number of Receive
10655        DTOs that can be in progress on an Endpoint that has an
10656        associated S-RQ. */
10657     it_boolean_t      hard_hi_watermark_support;
10658
10659     /* Indicates whether the IA supports the Endpoint Soft
10660        High Watermark mechanism for generating an Affiliated
10661        Asynchronous Event when the number of Receive DTOs in
10662        progress on an Endpoint that has an associated S-RQ exceeds
10663        the Endpoint Soft High Watermark. */
10664     it_boolean_t      soft_hi_watermark_support;
10665
10666     /* Indicates whether an S-RQ can be resized after it is created. */
10667     it_boolean_t      srq_resizable;
10668
10669     /* Indicates that an iWARP V-RNIC supports a modified qp state
10670        diagram (outside the RDMAC verbs). */
10671     it_boolean_t      extended_iwarp_qp_states;
10672
10673     /* Indicates that Implementation supports socket conversion (TDI).
10674        This attribute is IT_TRUE if and only if transport_type is
10675        IT_IWARP_TCP_TRANSPORT. */
10676     it_boolean_t      socket_conversion_support;
10677
10678     /* Indicates that IA supports increasing ORD
10679        (decreasing ORD is mandatory for all RNICs). */
10680     it_boolean_t      rdma_read_ord_increasable;
10681
10682     /* Indicates that IA supports modifying IRD. */
10683     it_boolean_t      rdma_read_ird_modifiable;
10684
10685     /* Bit set indicating which RMR types are supported.
10686        Possible values are IT_RMR_TYPE_NARROW, IT_RMR_TYPE_WIDE, and
10687        (IT_RMR_TYPE_NARROW|IT_RMR_TYPE_WIDE). See also it_rmr_type_t.*/
10688     it_rmr_type_t     rmr_types_supported;
10689
10690     /* Indicates whether the IA supports Relative Addressing. See also

```

```

10691         it_addr_mode_t. */
10692     it_boolean_t  addr_mode_relative_support;
10693
10694     /* Indicates whether the Destination buffer for an RDMA Read DTO
10695        must have remote or local write permission, and whether or not
10696        the Endpoint to which an RDMA Read DTO is posted must have RDMA
10697        Write access enabled. See also it_post_rdma_read. */
10698     it_boolean_t  rdma_read_requires_remote_write;
10699
10700     /* Indicates whether the IA supports changing the RDMA enables
10701        after EP creation. */
10702     it_boolean_t  ep_rdma_enables_modifiable;
10703
10704     /* Indicates whether the IA supports it_post_rdma_read_to_rmr and
10705        also whether the IT_UNLINK_LOCAL_SINK flag may be used on
10706        it_post_rdma_read or it_post_rdma_read_to_rmr operations. */
10707     it_boolean_t  rdma_read_local_extensions;
10708
10709     /* Indicates whether the IA supports DTO EVD overflow detection. */
10710     it_boolean_t  dto_evd_overflow_detection;
10711
10712     /* Indicates that the transport protocol used by the IA supports
10713        Atomics, and that the IA itself implements that support. If
10714        the underlying transport protocol used by the IA does not
10715        support Atomics, this attribute will have the value IT_FALSE. */
10716     it_boolean_t  atomic_support;
10717
10718     /* Indicates whether the IA supports use of LMR Link capabilities
10719        by Privileged Consumers. */
10720     it_boolean_t  lmr_link_support;
10721
10722     /* Indicates whether the IA supports use of the Direct LMR Handle
10723        by Privileged Consumers. */
10724     it_boolean_t  direct_lmr_handle_support;
10725
10726     /* The Direct LMR Handle. If direct_lmr_handle_support
10727        is IT_FALSE then direct_lmr_handle is reported as
10728        IT_NULL_HANDLE. */
10729     it_lmr_handle_t  direct_lmr_handle;
10730
10731     /* Indicates whether the IA supports use of posting remote unlink
10732        requests in Send calls. */
10733     it_boolean_t  post_send_unlink_remote_support;
10734
10735     /* Indicates whether the IA supports unlinking narrow RMRs. */
10736     it_boolean_t  narrow_rmr_unlink_support;
10737
10738     /* Indicates whether the IA supports unlinking wide RMRs. */
10739     it_boolean_t  wide_rmr_unlink_support;
10740
10741     /* Indicates whether the IA supports local fencing operations. */
10742     it_boolean_t  unlink_fence_support;
10743
10744     /* Defines types of IOBL supported by the IA. */

```

```
10745         it_iobl_type_t   iobl_types_supported;
10746     } it_ia_info_t;
```

10747 **DESCRIPTION**

10748 The *it_ia_info_t* structure is returned by the *it_ia_query* routine.

10749 The *it_ia_info_t* structure specifies the capabilities and attributes of an Interface Adapter. It also
10750 identifies the Interface Adapter's Spigots and their Network Addresses.

10751 Spigot identifiers are required inputs to the *it_get_pathinfo* and *it_listen_create* routines. Spigot
10752 Network Addresses may be used in the advertisement of local Consumer-provided services to
10753 remote Consumers.

10754 The IA can be in one of two states: enabled or disabled. An IA will be in the enabled state when
10755 it is created (via *it_ia_create*). Normally an IA will remain in the enabled state, but if it
10756 encounters a catastrophic error it will move into the disabled state. The Implementation
10757 guarantees that no unreported data corruption has occurred as a result of the IA entering the
10758 disabled state. If the Consumer calls any API routine other than *it_ia_free* while the IA is in the
10759 disabled state, neither the IA nor any of the Implementation data structures associated with it
10760 will be modified in any way. Instead, the routine will return the error code
10761 IT_ERR_IA_CATASTROPHE, and none of the output parameters from the routine will be
10762 valid. Once an IA has entered the disabled state the only recovery action that the Consumer can
10763 perform is to free the IA.

10764 **EXTENDED DESCRIPTION**

10765 The supported I/O Buffer List types are described by *iobl_types_supported*, which is a
10766 combination of bits from *it_iobl_type_t*, as follows:

10767 On the InfiniBand 1.1 Transport, no IOBL types are supported.

10768 On the InfiniBand 1.2 and iWARP Transports, *iobl_types_supported* must include one of the
10769 combinations IT_IOBL_PAGE_LIST (Page List mode), IT_IOBL_BLOCK_LIST (Block List
10770 superset mode), or IT_IOBL_PAGE_LIST | IT_IOBL_BLOCK_LIST (Page List / Block List
10771 discriminating mode), and it may further include IT_IOBL_VE_PAGE_LIST.

10772 **FUTURE DIRECTIONS**

10773 Quality of Service control for VIA and other transports may be added in the future.

10774 **SEE ALSO**

10775 *it_ia_query()*, *it_get_pathinfo()*, *it_listen_create()*, *it_iobl_type_t*

10776

it_io_addr_t

10777 **NAME**

10778 `it_io_addr_t` – O/S-dependent structure describing a bus or physical address

10779 **SYNOPSIS**

10780

```
typedef struct it_io_addr_s      *it_io_addr_t;
```

10781 **DESCRIPTION**

10782 The `it_io_addr_t` structure describes a bus or physical address in an O/S-dependent manner.

10783 **APPLICATION USAGE**

10784 The `it_io_addr_t` is used to describe bus or physical addresses in LMR Triplets as well as in I/O
10785 Buffer Lists used as inputs to the `it_lmr_create` and `it_lmr_link` routines.

10786 **SEE ALSO**

10787 `it_lmr_create()`, `it_post_atomic()`, `it_post_send()`, `it_post_sendto()`, `it_post_recv()`,
10788 `it_post_recvfrom()`, `it_post_rdma_write()`, `it_post_rdma_read()`, `it_addr_mode_t`

10789

10790 **NAME**10791 `it_iobl_t` – I/O Buffer List (IOBL)10792 **SYNOPSIS**

```

10793 #include <it_api.h>
10794
10795 typedef enum {
10796     IT_IOBL_PAGE_LIST      = 0x0001,
10797     IT_IOBL_BLOCK_LIST    = 0x0002,
10798     IT_IOBL_VE_PAGE_LIST  = 0x0004
10799 } it_iobl_type_t;
10800
10801 typedef struct {
10802     /* I/O Address - bus address or physical address.
10803      * it_io_addr_t is an OS-dependent type. */
10804     it_io_addr_t io_addr;
10805
10806     /* IOBL element length for this element */
10807     it_length_t elt_len;
10808 } it_iobl_ve_elt_t; /* Variable-element-length list element */
10809
10810 typedef struct {
10811     it_iobl_type_t iobl_type; /* Union discriminator */
10812     union {
10813         struct {
10814             it_io_addr_t *list_elt; /* Head of list */
10815             it_length_t elt_len; /* Constant IOBL element length */
10816         } ce_list; /* Constant-element-length list */
10817         struct {
10818             it_iobl_ve_elt_t *list_elt; /* Head of list */
10819         } ve_list; /* Variable-element-length list */
10820     } list;
10821     size_t num_elts; /* Number of elements in list */
10822     it_length_t fbo; /* First-Byte Offset */
10823 } it_iobl_t;

```

10824 **DESCRIPTION**

10825 The `it_iobl_t` structure describes a buffer list containing constant-length or variable-length
 10826 elements. Its members are defined as follows:

10827 *iobl_type* IOBL type, discriminating the union *list*. An IOBL can be a Page List (constant or variable
 10828 page size) or a Block List, depending on the supported IOBL types (see the
 10829 *iobl_types_supported* attribute in *it_ia_info_t*). If *iobl_types_supported* includes more than
 10830 one bit, any one of these bits can be selected as *iobl_type*. If *iobl_types_supported* includes
 10831 both `IT_IOBL_PAGE_LIST` and `IT_IOBL_BLOCK_LIST` (Page List / Block List
 10832 discriminating mode), then the Consumer can assist the transport in IOBL
 10833 verification/discrimination by indicating the exact IOBL type rather than declaring each
 10834 IOBL as a Block List.

10835 *list* Either an array of `it_io_addr_t` elements of constant size or an array of elements each
 10836 including an `it_io_addr_t` and a length.

10837 *num_elts* Number of elements in the *list*.

10838 *fbo* First-Byte Offset in first element of list. Must be greater than or equal to 0 and less than

10839 *elt_len* (where *elt_len* is either the constant element length of a *ce_list* or the element length

10840 of the first element of a *ve_list*).

10841

10842 APPLICATION USAGE

10843 The I/O Buffer List is an input parameter describing a local buffer segment for the *it_lmr_link*

10844 and *it_lmr_create* operations.

10845 The following code demonstrates the various uses of the *it_iobl_t* type:

```

10846            #include <it_api.h>
10847
10848            #define PLEN 4096
10849            #define BLEN 512
10850
10851            /*
10852            * for the sake of a simple coding example
10853            * declare a page-aligned buffer
10854            */
10855            #pragma ALIGN 4096
10856            char                    page_buffer[PLEN];
10857
10858            /*
10859            * for the sake of a simple coding example
10860            * declare a block-aligned buffer
10861            */
10862            #pragma ALIGN 512
10863            char                    block_buffer[BLEN];
10864
10865            size_t                   mapped_len;
10866            it_io_addr_t            io_addr[2];
10867            it_iobl_ve_elt_t        ve_elt[2];
10868            it_iobl_t               pl_iobl;
10869            it_iobl_t               bl_iobl;
10870            it_iobl_t               ve_pl_iobl;
10871            it_lmr_handle           lmr_handle;
10872
10873            /* map the page-aligned buffer to an I/O address */
10874            it_mem_map(ia, page_buffer, aspace(page_buffer), PLEN, &io_addr[0],
10875            &mapped_len);
10876
10877            /* create a page list it_iobl_t using the ioaddr from above */
10878            pl_iobl.type = IT_IOBL_PAGE_LIST;
10879            pl_iobl.list.ce_list.list_elt = &io_addr[0];
10880            pl_iobl.list.ce_list.elt_len = PLEN;
10881            pl_iobl.num_elts = 1;
10882            pl_iobl.fbo = 0;
10883
10884            /* create an LMR using the page list and absolute addressing */
10885            it_lmr_create(pz, page_buffer, &pl_iobl, PLEN, IT_ADDR_MODE_ABSOLUTE,
10886            IT_PRIV_LOCAL, IT_LMR_FLAG_NONE, 0, &lmr_handle, IT_NO_ADDR);

```

```

10887
10888      /* map the block-aligned buffer to an I/O address */
10889      it_mem_map(ia, block_buffer, aspace(block_buffer), BLEN, &io_addr[1],
10890      &mapped_len);
10891
10892      /* create a block list it_iobl_t using the ioaddr from above */
10893      bl_iobl.type = IT_IOBL_BLOCK_LIST;
10894      bl_iobl.list.ce_list.list_elt = &io_addr[1];
10895      bl_iobl.list.ce_list.elt_len = BLEN; /* say block size is 512 */
10896      bl_iobl.num_elts = 1;
10897      bl_iobl.fbo = 40; /* as an example of fbo use, set First-Byte Offset of
10898      data to 40 */
10899
10900      /* create an LMR using the page list and relative addressing */
10901      it_lmr_create(pz, 0, &bl_iobl, (BLEN-40), IT_ADDR_MODE_RELATIVE,
10902      IT_PRIV_LOCAL, IT_LMR_FLAG_NONE, 0, &lmr_handle, IT_NO_ADDR);
10903
10904      /* reuse the two mappings to demonstrate VE lists */
10905      ve_elt[0].io_addr = &io_addr[0];
10906      ve_elt[0].elt_len = PLEN;
10907      ve_elt[1].io_addr = &io_addr[1];
10908      ve_elt[2].elt_len = BLEN;
10909      ve_pl_iobl.type = IT_IOBL_VE_PAGE_LIST;
10910      ve_pl_iobl.list.ve_list.list_elt = ve_elt;
10911      ve_pl_iobl.num_elts = 2;
10912      ve_pl_iobl.fbo = 0;
10913
10914      /* create an LMR using the variable element page list */
10915      it_lmr_create(pz, 0, &ve_pl_iobl, (PLEN+BLEN), IT_ADDR_MODE_RELATIVE,
10916      IT_PRIV_LOCAL, IT_LMR_FLAG_NONE, 0, &lmr_handle, IT_NO_ADDR);

```

10917 **SEE ALSO**

10918 *it_lmr_create(), it_lmr_link(), it_post_atomic(), it_post_send(), it_post_sendto(), it_post_recv(),*
10919 *it_post_recvfrom(), it_post_rdma_write(), it_post_rdma_read(), it_addr_mode_t*

it_lmr_triplet_t

10920
10921 **NAME**
10922 `it_lmr_triplet_t` – structure describing a DTO buffer in a Local Memory Region

10923 **SYNOPSIS**
10924

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


10925
10926

```
typedef struct {
```

10927

```
    it_lmr_handle_t lmr;
```

10928

```
    union {
```

10929

```
        void *abs;
```

10930

```
        it_length_t rel;
```

10931

```
        it_io_addr_t io;
```

10932

```
    } addr;
```

10933

```
    it_length_t length;
```

10934

```
} it_lmr_triplet_t;
```

10935 **DESCRIPTION**
10936 The `it_lmr_triplet_t` structure describes a local Source or Destination buffer segment for Data
10937 Transfer Operations. Its members are defined as follows:

10938 *lmr* Handle of LMR in which the local buffer resides.

10939 *addr* Starting address of the local buffer segment, interpreted according to the *addr_mode*
10940 (addressing mode) attribute of the LMR (see [it_addr_mode_t](#)) either as an absolute address
10941 (*abs*) or as an address offset (*rel*) relative to the Base Address of the LMR. If LMR is the
10942 Direct LMR Handle, then *addr* is interpreted as an [it_io_addr_t](#).

10943 *length* Length in bytes of the local buffer segment.

10944 **APPLICATION USAGE**
10945 The LMR Triplet is an input parameter describing a local buffer segment for DTOs such as
10946 [it_post_send](#).

10947 **SEE ALSO**
10948 [it_post_atomic\(\)](#), [it_post_send\(\)](#), [it_post_sendto\(\)](#), [it_post_recv\(\)](#), [it_post_recvfrom\(\)](#),
10949 [it_post_rdma_write\(\)](#), [it_post_rdma_read\(\)](#), [it_addr_mode_t](#)

it_mem_priv_t

10950

10951 NAME

10952 it_mem_priv_t – Memory access privileges for Local Memory Regions and Remote Memory
10953 Regions

10954 SYNOPSIS

```
10955 #include <it_api.h>
10956
10957 typedef enum {
10958     IT_PRIV_LOCAL_READ      = 0x0001,
10959     IT_PRIV_LOCAL_WRITE    = 0x0002,
10960     IT_PRIV_LOCAL          = 0x0003,
10961     IT_PRIV_REMOTE_READ    = 0x0004,
10962     IT_PRIV_REMOTE_WRITE   = 0x0008,
10963     IT_PRIV_REMOTE        = 0x000c,
10964     IT_PRIV_ATOMIC        = 0x0010,
10965     IT_PRIV_ALL           = 0x001f
10966 } it_mem_priv_t;
```

10967 DESCRIPTION

10968 *privs* Memory access privileges for an LMR or RMR, formed by a bitwise-inclusive OR
10969 of values from *it_mem_priv_t*.

10970 Individual bit values defined in *it_mem_priv_t* are as follows:

10971 IT_PRIV_LOCAL_READ	Grants local read access to the IA, enabling the use of an LMR as 10972 a Source buffer for locally posted Send or RDMA Write DTOs.
10973 IT_PRIV_LOCAL_WRITE	Grants local write access to the IA, enabling the use of the LMR as 10974 a Destination buffer for locally posted Receive or RDMA Read 10975 DTOs.
10976 IT_PRIV_REMOTE_READ	Grants remote read access to the IA, enabling the use of an LMR 10977 or RMR as a Source buffer for incoming RDMA Read requests.
10978 IT_PRIV_REMOTE_WRITE	Grants remote write access to the IA, enabling the use of an LMR 10979 or RMR as a Destination buffer for incoming RDMA Write 10980 operations or RDMA Read Responses.
10981 IT_PRIV_ATOMIC	Grants remote Atomic access to the IA, enabling the use of an 10982 LMR or RMR as a Source/Destination buffer for incoming Atomic 10983 operations.

10984 Predefined bit combinations defined in *it_mem_priv_t* are as follows:

10985 IT_PRIV_LOCAL	Local read and local write access, for an LMR.
10986 IT_PRIV_REMOTE	Remote read and remote write access, for an LMR or RMR.
10987 IT_PRIV_ALL	All access privileges, for an LMR.

- 10988 **APPLICATION USAGE**
- 10989 Memory access privileges of a Local Memory Region are specified when the LMR is created.
- 10990 Memory access privileges of a Remote Memory Region are specified when the RMR is linked to
- 10991 an LMR.
- 10992 **SEE ALSO**
- 10993 [*it_lmr_create\(\)*](#), [*it_lmr_query\(\)*](#), [*it_rmr_link\(\)*](#), [*it_rmr_query\(\)*](#)

it_net_addr_t

10994

10995 **NAME**

10996

it_net_addr_t – encapsulates all supported Network Address types

10997 **SYNOPSIS**

10998

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

10999

```
/* Enumerates all the possible Network Address types supported  
by the API. */
```

11000

11001

```
typedef enum {
```

11002

11003

11004

11005

11006

11007

11008

11009

11010

11011

11012

11013

11014

11015

11016

11017

```
    /* IPv4 address */  
    IT_IPV4 = 0x1,
```

11007

11008

11009

11010

11011

11012

11013

11014

11015

11016

11017

```
    /* IPv6 address */  
    IT_IPV6 = 0x2,
```

11010

11011

11012

11013

11014

11015

11016

11017

```
    /* InfiniBand GID */  
    IT_IB_GID = 0x3,
```

11013

11014

11015

11016

11017

```
    /* VIA Network Address */  
    IT_VIA_HOSTADDR = 0x4
```

11016

11017

```
} it_net_addr_type_t;
```

11018

11019

11020

11021

```
/* 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. */
```

11022

11023

11024

11025

11026

11027

11028

11029

```
#define IT_MAX_VIA_ADDR_LEN 64
```

11023

11024

11025

11026

11027

11028

11029

```
typedef struct {
```

11025

11026

11027

11028

11029

11030

11031

11032

11033

11034

11035

11036

11037

11038

11039

11040

11041

11042

11043

11044

```
    /* The number of bytes in the array below that are  
significant. */  
    uint16_t len;
```

11029

11030

11031

11032

11033

11034

```
    /* VIA host address, which is an array of bytes. */  
    unsigned char hostaddr[IT_MAX_VIA_ADDR_LEN];
```

11032

11033

11034

```
} it_via_net_addr_t;
```

11035

11036

11037

11038

```
/* This defines the Network Address format for the InfiniBand  
GID, which is just an IPv6 address. */
```

11036

11037

11038

```
typedef struct in6_addr it_ib_gid_t;
```

11037

11038

11039

11040

11041

11042

11043

11044

```
/* This describes a Network Address suitable for input to several  
routines in the API. */
```

```
typedef struct {
```

11041

11042

11043

11044

```
    /* The discriminator for the union below. */  
    it_net_addr_type_t addr_type;
```

```

11045
11046         union {
11047
11048             /* IPv4 address, in network byte order. */
11049             struct in_addr  ipv4;
11050
11051             /* IPv6 address, in network byte order. */
11052             struct in6_addr  ipv6;
11053
11054             /* InfiniBand GID, in network byte order. */
11055             it_ib_gid_t  gid;
11056
11057             /* VIA Network Address. */
11058             it_via_net_addr_t  via;
11059
11060         } addr;
11061
11062     } it_net_addr_t;

```

11063 **DESCRIPTION**

11064 The *it_net_addr_t* type is used by several routines in the API to encapsulate a Network Address
11065 associated with a Spigot on an IA. The *it_net_addr_t* is the name for a Spigot when it is being
11066 accessed remotely. (When it is being accessed locally, it is named by a Spigot identifier, not by
11067 an *it_net_addr_t*.) Each Spigot on an IA has at least one *it_net_addr_t* associated with it. A
11068 Spigot can have more than one Network Address associated with it, and these Network
11069 Addresses can be of different types. (For example, an InfiniBand HCA might have both an IPv4
11070 address and an InfiniBand GID associated with one of its Spigots.) The set of Network
11071 Addresses that can be used to refer to a Spigot on an IA can be determined using the *it_ia_query*
11072 routine.

11073 In order to aid Consumers in writing portable applications that span platforms with different
11074 native byte orders, all Network Addresses that are supported by the API with the exception of
11075 the VIA “host address” are required to be input to the API in network byte order, and will be
11076 output from the API in network byte order. (The VIA “host address” is defined to be an array of
11077 bytes, and hence is not affected by which native byte order a platform uses.)

11078 **SEE ALSO**

11079 *it_get_pathinfo()*, *it_ia_query()*

it_path_t

11080

11081 **NAME**

11082

it_path_t – describes the Path between a pair of Spigots

11083 **SYNOPSIS**

11084

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

11085

11086

```
/* This is the Path information for the InfiniBand Transport. */
```

11087

```
typedef struct {
```

11088

11089

```
    /* Partition Key, as defined in the REQ message for the IB  
       CM protocol. */
```

11090

```
    uint16_t partition_key;
```

11091

11092

11093

```
    /* Path Packet Payload MTU, as defined in the REQ message  
       for the IB CM protocol. */
```

11094

```
    uint8_t path_mtu : 4;
```

11095

11096

11097

```
    /* PacketLifeTime, as defined in the PathRecord in IB  
       specification. This field is useful for Consumers that  
       wish to use timeout values other than the default ones  
       for doing Connection establishment. */
```

11098

```
    uint8_t packet_lifetime : 6;
```

11099

11100

11101

11102

```
    /* Local Port LID, as defined in the REQ message for the IB  
       CM protocol. The low-order bits of this value also  
       constitute the "Source Path Bits" that are used to  
       create an Address Handle. */
```

11103

```
    uint16_t local_port_lid;
```

11104

11105

11106

11107

11108

```
    /* Remote Port LID, as defined in the REQ message for the  
       IB CM protocol. This is also the "Destination LID" used  
       to create an Address Handle. */
```

11109

```
    uint16_t remote_port_lid;
```

11110

11111

11112

11113

```
    /* Local Port GID in network byte order, as defined in the  
       REQ message for the IB CM protocol. This is also used to  
       determine the appropriate "Source GID Index" to be used  
       when creating an Address Handle. */
```

11114

```
    it_ib_gid_t local_port_gid;
```

11115

11116

11117

11118

11119

```
    /* Remote Port GID in network byte order, as defined in the  
       REQ message for the IB CM protocol. This is also the  
       "Destination GID or MGID" used to create an Address  
       Handle. */
```

11120

```
    it_ib_gid_t remote_port_gid;
```

11121

11122

11123

11124

11125

```
    /* Packet Rate, as defined in the REQ message for the IB CM  
       protocol. This is also the "Maximum Static Rate" to be  
       used when creating an Address Handle. */
```

11126

```
    uint8_t packet_rate : 6;
```

11127

11128

11129

11130

```

11131      /* SL, as defined in the REQ message for the IB CM
11132         protocol. This is also the "Service Level" to be used
11133         when creating an Address Handle. */
11134      uint8_t  sl : 4;
11135
11136      /* Subnet Local, as defined in the REQ message for the IB
11137         CM protocol. When creating an Address Handle, setting
11138         this bit causes a GRH to be included as part of any
11139         Unreliable Datagram sent using the Address Handle. */
11140      uint8_t  subnet_local : 1;
11141
11142      /* Flow Label, as defined in the REQ message for the IB CM
11143         protocol. This is also the "Flow Label" to be used when
11144         creating an Address Handle. This is only valid if
11145         subnet_local is clear. */
11146      uint32_t flow_label : 20;
11147
11148      /* Traffic Class, as defined in the REQ message for the IB
11149         CM protocol. This is also the "Traffic Class" to be
11150         used when creating an Address Handle. This is only
11151         valid if subnet_local is clear. */
11152      uint8_t  traffic_class;
11153
11154      /* Hop Limit, as defined in the REQ message for the IB CM
11155         protocol. This is also the "Hop Limit" to be used when
11156         creating an Address Handle. This is only valid if
11157         subnet_local is clear. */
11158      uint8_t  hop_limit;
11159
11160  } it_ib_net_endpoint_t;
11161
11162      /* This is the Path information for the VIA transport. */
11163      typedef it_via_net_addr_t it_via_net_endpoint_t;
11164
11165      /* This is the Path information for the iWARP Transport. */
11166
11167      /* enum to discriminate path element types in
11168         it_iwarp_net_endpoint_t. */
11169      typedef enum {
11170          IT_IP_VERS_IPV4 = 0x1,
11171          IT_IP_VERS_IPV6 = 0x2
11172      } it_ip_vers_t;
11173
11174      typedef struct {
11175          /* Designates the type of IP address that is found in
11176             both the laddr and raddr unions below. */
11177          it_ip_vers_t  ip_vers;
11178
11179          /* Local path element */
11180          union {
11181              struct in_addr  ipv4;
11182              struct in6_addr ipv6;
11183          } laddr;
11184

```

```

11185         /* Remote path element */
11186         union {
11187             struct in_addr   ipv4;
11188             struct in6_addr  ipv6;
11189         } raddr;
11190     } it_iwarp_net_endpoint_t;
11191
11192     /* This is the Path data structure used by several routines in
11193        the API. */
11194     typedef struct {
11195
11196         /* Identifier for the Spigot to be used on the local IA.
11197            Note that this data structure is always used in a
11198            Context where the IA associated with the Spigot can be
11199            deduced. */
11200         size_t  spigot_id;
11201
11202         /* The transport-independent timeout parameter for how long
11203            to wait, in microseconds, before timing out a Connection
11204            establishment attempt using this Path. The timeout
11205            period for establishing a Connection
11206            can only be specified on the Active side; the timeout
11207            period cannot be changed on the Passive side. */
11208         uint64_t  timeout;
11209
11210         /* The remote component of the Path. */
11211         union {
11212
11213             /* For use with InfiniBand. */
11214             it_ib_net_endpoint_t  ib;
11215
11216             /* For use with VIA. */
11217             it_via_net_endpoint_t  via;
11218
11219             /* For use with iWARP. */
11220             it_iwarp_net_endpoint_t  iwarp;
11221
11222         } remote;
11223     } it_path_t;
11224

```

11225 DESCRIPTION

11226 The *it_path_t* type is used by several routines in the API to encapsulate a Path between two
11227 Spigots. The *it_path_t* contains a local Spigot identifier, a remote Spigot address, and a
11228 specification of all information that determines the properties of the Path that messages will take
11229 between the two Spigots. The local Spigot to be used is identified in a transport-independent
11230 manner, but the remote Spigot and the Path to that Spigot are specified in a transport-dependent
11231 manner.

11232 The *it_path_t* structure also contains a *timeout* member. This *timeout* member defines how long
11233 the local Consumer is willing to wait for a response to its attempt to establish a Connection when
11234 the *it_path_t* is used with the *it_ep_connect* routine. If the Consumer retrieves the Path using the
11235 *it_get_pathinfo* routine, the Implementation will provide a *timeout* that should be sufficiently

11236 long to establish a Connection under most circumstances, and so the Consumer should have no
11237 need to modify this value. If the Consumer chooses to provide their own value for the *timeout*
11238 member, the Consumer should take care to choose a value that is compatible with any
11239 underlying transport-specific timeout values governing Connection establishment that may be
11240 present in the transport-specific portion of the *it_path_t*. Choosing a value of *timeout* that is
11241 incompatible with the transport-specific timeout values governing Connection establishment will
11242 result in an error being returned from the *it_ep_connect* routine.

11243 All data contained within the *it_path_t* data structure appears in host byte order unless otherwise
11244 noted in the comments associated with the members of the data structure. The contents of the
11245 *path_t* returned from the *it_get_pathinfo* routine are only valid on the node on which the routine
11246 was invoked.

11247 **SEE ALSO**

11248 *it_get_pathinfo()*, *it_ep_connect()*, *it_address_handle_create()*,
11249 *it_ud_service_request_handle_create()*

it_rmr_triplet_t

11250
11251 **NAME**
11252 `it_rmr_triplet_t` – structure describing a DTO buffer in a Remote Memory Region

11253 **SYNOPSIS**
11254

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


11255
11256

```
typedef struct {
```


11257

```
    it_rmr_handle_t  rmr;
```


11258

```
    union {
```


11259

```
        void          *abs;
```


11260

```
        it_length_t   rel;
```


11261

```
    } addr;
```


11262

```
    it_length_t       length;
```


11263

```
} it_rmr_triplet_t;
```

11264 **APPLICABILITY**
11265 Remote Memory Regions can be used only for the RC service type. See [it_rmr_create](#).

11266 **DESCRIPTION**
11267 The `it_rmr_triplet_t` structure describes a local Destination buffer segment for an RDMA Read
11268 DTO initiated by the `it_post_rdma_read_to_rmr()` call. Its members are defined as follows:

11269 *rmr* Handle of RMR in which the local buffer resides.

11270 *addr* Starting address of the local buffer segment, interpreted according to the *addr_mode*
11271 (addressing mode) attribute of the RMR (see [it_addr_mode_t](#)) either as an absolute address
11272 (*abs*) or as an address offset (*rel*) relative to the Base Address of the RMR.

11273 *length* Length in bytes of the local buffer segment.

11274 **APPLICATION USAGE**
11275 The RMR Triplet is an input parameter describing a local buffer segment for RDMA Read
11276 DTOs initiated by the `it_post_rdma_read_to_rmr()` call only.

11277 **SEE ALSO**
11278 [it_post_rdma_read_to_rmr\(\)](#), [it_addr_mode_t](#)

it_rmr_type_t

11279

11280 NAME

11281 `it_rmr_type_t` – definition of RMR type

11282 SYNOPSIS

```
11283 #include <it_api.h>
11284
11285 typedef enum {
11286     IT_RMR_TYPE_DEFAULT = 0,
11287     IT_RMR_TYPE_NARROW = 1,
11288     IT_RMR_TYPE_WIDE = 2
11289 } it_rmr_type_t;
```

11290 APPLICABILITY

11291 Remote Memory Regions can be used only for the RC service type. See [it_rmr_create](#).

11292 DESCRIPTION

11293 Type of a Remote Memory Region, with three possible values as follows:

11294 **IT_RMR_TYPE_DEFAULT**

11295 This value can be used only for creating an RMR. Creating an RMR of the default type results in
11296 an RMR of the Narrow or Wide type, depending on the Implementation.

11297 **IT_RMR_TYPE_NARROW**

11298 An unlinked Narrow RMR may be used as a target for Link operations via all Endpoints in the
11299 Protection Zone of the RMR. A linked Narrow RMR may be used as a target for DTOs and
11300 Unlink operations only via the Endpoint through which it was linked.

11301 **IT_RMR_TYPE_WIDE**

11302 An unlinked Wide RMR may be used as a target for Link operations via all Endpoints in the
11303 Protection Zone of the RMR.

11304 A linked Wide RMR may be used as a target for DTOs and Link/Unlink operations via all
11305 Endpoints in the Protection Zone of the RMR.

11306 APPLICATION USAGE

11307 The desired type of a Remote Memory Region must be specified at RMR creation time. The
11308 allowable RMR types are Implementation-dependent. However, InfiniBand Implementations at
11309 least support Wide RMRs and, if Verb Extensions (BMM) are provided, also Narrow RMRs;
11310 iWARP Implementations at least support Narrow RMRs. The `rmr_types_supported` field of the
11311 [it_ia_info_t](#) structure, which can be queried through [it_ia_query](#), indicates which RMR types are
11312 supported by the IA; i.e., Narrow RMRs, Wide RMRs, or both.

11313 SEE ALSO

11314 [it_rmr_create\(\)](#), [it_rmr_link\(\)](#), [it_rmr_query\(\)](#)

it_software_event_t

11315

11316 NAME

11317 it_software_event_t – Software Event type

11318 SYNOPSIS

```
11319 #include <it_api.h>
11320
11321 typedef struct {
11322     it_event_type_t event_number;
11323     it_evd_handle_t evd;
11324     void *data;
11325 } it_software_event_t;
```

11326 DESCRIPTION

11327 *event_number* Identifier of the Event type. Valid values:
11328 IT_SOFTWARE_EVENT

11329 *evd* Handle for the Event Dispatcher where the Event was queued.

11330 *data* The pointer that the Consumer furnished to the *it_evd_post_se* routine.

11331 An IT_SOFTWARE_EVENT_STREAM Event is generated when the Consumer calls the
11332 *it_evd_post_se* routine to post a Software Event.

11333 The IT_SOFTWARE_EVENT_STREAM Event Stream supports Events with the *event_number*
11334 IT_SOFTWARE_EVENT (see *it_event_t*).

11335 The Software Event type passes back the same pointer that the Consumer furnished to the
11336 *it_evd_post_se* routine.

11337 All Events on an IT_SOFTWARE_EVENT_STREAM SEVD cause Notification. See
11338 *it_evd_create* for details of Notification.

11339 The Software Event type EVD does not overflow. Instead, *it_evd_post_se* will generate an
11340 immediate error if the Consumer attempts to queue a software Event on a full Software Event
11341 EVD.

11342 SEE ALSO

11343 *it_evd_post_se()*, *it_evd_create()*, *it_evd_wait()*, *it_event_t*

it_status_t

11344

11345 **NAME**

11346 it_status_t – definition of IT-API call return status

11347 **SYNOPSIS**

```
11348 #include <it_api.h>
11349
11350 typedef enum {
11351     IT_SUCCESS = 0,
11352     IT_ERR_ABORT,
11353     IT_ERR_ACCESS,
11354     IT_ERR_ADDRESS,
11355     IT_ERR_AEVD_NOT_ALLOWED,
11356     IT_ERR_ASYNC_AFF_EVD_EXISTS,
11357     IT_ERR_ASYNC_UNAFF_EVD_EXISTS,
11358     IT_ERR_CALLBACK_EXISTS,
11359     IT_ERR_CANNOT_RESET,
11360     IT_ERR_CONN_QUAL_BUSY,
11361     IT_ERR_EP_BUSY,
11362     IT_ERR_EP_TIMEWAIT,
11363     IT_ERR_EVD_BUSY,
11364     IT_ERR_EVD_WAIT,
11365     IT_ERR_EVD_QUEUE_FULL,
11366     IT_ERR_FAULT,
11367     IT_ERR_HARD_HI_WATERMARK,
11368     IT_ERR_IA_CATASTROPHE,
11369     IT_ERR_INTERRUPT,
11370     IT_ERR_INVALID_ADDR_MODE,
11371     IT_ERR_INVALID_ADDRESS,
11372     IT_ERR_INVALID_AEVD,
11373     IT_ERR_INVALID_AH,
11374     IT_ERR_INVALID_ATIMEOUT,
11375     IT_ERR_INVALID_ATOMIC_OP,
11376     IT_ERR_INVALID_CM_RETRY,
11377     IT_ERR_INVALID_CN_EST_FLAGS,
11378     IT_ERR_INVALID_CN_EST_ID,
11379     IT_ERR_INVALID_CONN_EVD,
11380     IT_ERR_INVALID_CONN_QUAL,
11381     IT_ERR_INVALID_CONVERSION,
11382     IT_ERR_INVALID_DTO_FLAGS,
11383     IT_ERR_INVALID_EP,
11384     IT_ERR_INVALID_EP_ATTR,
11385     IT_ERR_INVALID_EP_KEY,
11386     IT_ERR_INVALID_EP_STATE,
11387     IT_ERR_INVALID_EP_TYPE,
11388     IT_ERR_INVALID_EVD,
11389     IT_ERR_INVALID_EVD_STATE,
11390     IT_ERR_INVALID_EVD_TYPE,
11391     IT_ERR_INVALID_FLAGS,
11392     IT_ERR_INVALID_HANDLE,
11393     IT_ERR_INVALID_IA,
11394     IT_ERR_INVALID_IOBL,
```

11395	IT_ERR_INVALID_LENGTH,
11396	IT_ERR_INVALID_LISTEN,
11397	IT_ERR_INVALID_LMR,
11398	IT_ERR_INVALID_LTIMEOUT,
11399	IT_ERR_INVALID_MAJOR_VERSION,
11400	IT_ERR_INVALID_MASK,
11401	IT_ERR_INVALID_MINOR_VERSION,
11402	IT_ERR_INVALID_NAME,
11403	IT_ERR_INVALID_NETADDR,
11404	IT_ERR_INVALID_NUM_SEGMENTS,
11405	IT_ERR_INVALID_PDATA_LENGTH,
11406	IT_ERR_INVALID_PRIVS,
11407	IT_ERR_INVALID_PZ,
11408	IT_ERR_INVALID_QUEUE_SIZE,
11409	IT_ERR_INVALID_RECV_DTO,
11410	IT_ERR_INVALID_RECV_EVD,
11411	IT_ERR_INVALID_RECV_EVD_STATE,
11412	IT_ERR_INVALID_REQ_EVD,
11413	IT_ERR_INVALID_REQ_EVD_STATE,
11414	IT_ERR_INVALID_RETRY,
11415	IT_ERR_INVALID_RMR,
11416	IT_ERR_INVALID_RNR_RETRY,
11417	IT_ERR_INVALID_RMR_TYPE,
11418	IT_ERR_INVALID_RTIMEOUT,
11419	IT_ERR_INVALID_SOFT_EVD,
11420	IT_ERR_INVALID_SOURCE_PATH,
11421	IT_ERR_INVALID_SPIGOT,
11422	IT_ERR_INVALID_SRQ,
11423	IT_ERR_INVALID_SRQ_SIZE,
11424	IT_ERR_INVALID_THRESHOLD,
11425	IT_ERR_INVALID_UD_STATUS,
11426	IT_ERR_INVALID_UD_SVC,
11427	IT_ERR_INVALID_UD_SVC_REQ_ID,
11428	IT_ERR_INVALID_WATERMARK,
11429	IT_ERR_LMR_BUSY,
11430	IT_ERR_MISMATCH_FD,
11431	IT_ERR_NO_CONTEXT,
11432	IT_ERR_NO_PERMISSION,
11433	IT_ERR_OP_NOT_SUPPORTED,
11434	IT_ERR_PAYLOAD_SIZE,
11435	IT_ERR_PDATA_NOT_SUPPORTED,
11436	IT_ERR_PRIV_OPS_UNAVAILABLE,
11437	IT_ERR_PZ_BUSY,
11438	IT_ERR_QUEUE_EMPTY,
11439	IT_ERR_RANGE,
11440	IT_ERR_RESOURCES,
11441	IT_ERR_RESOURCE_IRD,
11442	IT_ERR_RESOURCE_LMR_LENGTH,
11443	IT_ERR_RESOURCE_ORD,
11444	IT_ERR_RESOURCE_QUEUE_SIZE,
11445	IT_ERR_RESOURCE_RECV_DTO,
11446	IT_ERR_RESOURCE_REQ_DTO,
11447	IT_ERR_RESOURCE_RRSEG,
11448	IT_ERR_RESOURCE_RSEG,

```

11449         IT_ERR_RESOURCE_RWSEG,
11450         IT_ERR_RESOURCE_SSEG,
11451         IT_ERR_SOFT_HI_WATERMARK,
11452         IT_ERR_SRQ_BUSY,
11453         IT_ERR_SRQ_LOW_WATERMARK,
11454         IT_ERR_SRQ_NOT_SUPPORTED,
11455         IT_ERR_TIMEOUT_EXPIRED,
11456         IT_ERR_TOO_MANY_POSTS,
11457         IT_ERR_WAITER_LIMIT
11458     } it_status_t;

```

11459 **DESCRIPTION**

11460 Most IT-API function calls return *it_status_t* on function completion. IT_SUCCESS indicates
11461 that an IT-API operation was invoked successfully; otherwise, the return code indicates the
11462 reason for failure. See each individual reference page for the meaning of a return code in the
11463 Context of the function.

11464 Some API function calls are used to initiate asynchronous operations. For those function calls, a
11465 return value of IT_SUCCESS indicates only that the operation was successfully initiated; it does
11466 not indicate that it was successfully completed. To determine whether an asynchronous
11467 operation was successfully completed, the Completion Event for the asynchronous operation
11468 should be examined.

11469 **SEE ALSO**

11470 *it_address_handle_create()*, *it_address_handle_free()*, *it_address_handle_modify()*,
11471 *it_address_handle_query()*, *it_convert_net_addr()*, *it_ep_accept()*, *it_ep_connect()*,
11472 *it_ep_disconnect()*, *it_ep_free()*, *it_ep_modify()*, *it_ep_query()*, *it_ep_rc_create()*, *it_ep_reset()*,
11473 *it_ep_ud_create()*, *it_evd_create()*, *it_evd_dequeue()*, *it_evd_free()*, *it_evd_modify()*,
11474 *it_evd_post_se()*, *it_evd_query()*, *it_evd_wait()*, *it_get_consumer_context()*,
11475 *it_get_handle_type()*, *it_get_pathinfo()*, *it_handoff()*, *it_ia_create()*, *it_ia_free()*, *it_ia_query()*,
11476 *it_listen_create()*, *it_listen_free()*, *it_listen_query()*, *it_lmr_create()*, *it_lmr_free()*,
11477 *it_lmr_modify()*, *it_lmr_query()*, *it_lmr_flush_to_mem()*, *it_lmr_refresh_from_mem()*,
11478 *it_post_atomic()*, *it_post_rdma_read()*, *it_post_rdma_read_to_rmr()*, *it_post_rdma_write()*,
11479 *it_post_recv()*, *it_post_recvfrom()*, *it_post_send()*, *it_post_sendto()*, *it_pz_create()*, *it_pz_free()*,
11480 *it_pz_query()*, *it_reject()*, *it_rmr_link()*, *it_rmr_create()*, *it_rmr_free()*, *it_rmr_query()*,
11481 *it_rmr_unlink()*, *it_set_consumer_context()*, *it_socket_convert()*, *it_srq_create()*, *it_srq_free()*,
11482 *it_srq_modify()*, *it_srq_query()*, *it_ud_service_reply()*, *it_ud_service_request()*,
11483 *it_ud_service_request_handle_create()*, *it_ud_service_request_handle_free()*,
11484 *it_ud_service_request_handle_query()*, *it_dto_events*

it_unaffiliated_event_t

11485

11486 NAME

11487 it_unaffiliated_event_t – Unaffiliated Asynchronous Event type

11488 SYNOPSIS

```
11489 #include <it_api.h>
11490
11491 typedef struct {
11492     it_event_type_t event_number;
11493     it_evd_handle_t evd;
11494     it_ia_handle_t ia;
11495
11496     size_t spigot_id;
11497 } it_unaffiliated_event_t;
```

11498 DESCRIPTION

11499 *event_number* Identifier of the Event type. Valid values:
11500 IT_ASYNC_UNAFF_SPIGOT_ONLINE,
11501 IT_ASYNC_UNAFF_SPIGOT_OFFLINE,
11502 IT_ASYNC_UNAFF_SEVD_ENQUEUE_FAILURE

11503 *evd* Handle for the Event Dispatcher where the Event was queued.

11504 *ia* The Handle for the IA that experienced the Unaffiliated Event.

11505 *spigot_id* The identifier for the Spigot that changed state on the IA. Valid only for the
11506 IT_ASYNC_UNAFF_SPIGOT_ONLINE and
11507 IT_ASYNC_UNAFF_SPIGOT_OFFLINE Events.

11508 IT_ASYNC_UNAFF_EVENT_STREAM Events are generated when an Unaffiliated
11509 Asynchronous Event occurs. There are several types of Unaffiliated Asynchronous Events, and
11510 each type is identified by *event_number*. The Consumer asks for Unaffiliated Asynchronous
11511 Events to be delivered when it creates an EVD for the Unaffiliated Asynchronous Event Stream
11512 using the *it_evd_create* call.

11513 The following table maps the values in the Unaffiliated Asynchronous Events *it_event_type_t*
11514 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.

11515 **EXTENDED DESCRIPTION**

11516 For the Infiniband transport, the following table maps the values in the Unaffiliated
 11517 Asynchronous Errors *it_event_type_t* enumeration to their corresponding “Unaffiliated
 11518 Asynchronous Events” and “Unaffiliated Asynchronous Errors” as specified in Chapter 11 of
 11519 [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

11520
 11521 For the VIA transport, the following table maps the values in the Unaffiliated Asynchronous
 11522 Errors *it_event_type_t* enumeration to their corresponding descriptions in the
 11523 “VipErrorCallback” reference page in the Appendix of [VIA-V1.0].

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.)

11524
 11525 All Events on an IT_ASYNC_UNAFF_EVENT_STREAM SEVD cause Notification. See
 11526 *it_evd_create* for details of Notification.

11527 Overflow of an IT_ASYNC_UNAFF_EVENT_STREAM SEVD is not visible to the Consumer;
 11528 all subsequent IT_ASYNC_UNAFF_EVENT_STREAM Events are silently dropped until the
 11529 Consumer dequeues at least one Event from the EVD that contains the
 11530 IT_ASYNC_UNAFF_EVENT_STREAM.

11531 When a Consumer has created more than one IA corresponding to an underlying physical
 11532 adapter (say in different processes), then every Unaffiliated Event is replicated to every IA
 11533 instance.

11534 **SEE ALSO**

11535 *it_ia_create()*, *it_event_t*, *it_evd_create()*, *it_evd_wait()*

11536

A Implementer's Guide

11537
11538
11539
11540
11541

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.

11542

it_address_handle_create

11543
11544
11545
11546
11547

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.

11548
11549
11550
11551
11552

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.

11553
11554
11555

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`.

11556

it_affiliated_event_t

11557
11558
11559
11560

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.

11561
11562

`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.

11563

it_cm_msg_events

11564
11565
11566
11567
11568

`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.

11569 In general, all transport-specific Connection Management rejection Events not explicitly defined
11570 in this API should be implemented as IT_CM_MSG_CONN_NONPEER_REJECT_EVENT
11571 with IT_CN_REJ_OTHER reject reason code Events.

11572 When running over the InfiniBand Transport, Implementations have the option to “chew up” a
11573 REJ that is returned with reject reason code 1 (No QP available), 3 (No resources available), or 4
11574 (Timeout) rather than immediately posting an Event with status IT_CN_REJ_RESOURCES (for
11575 REJ codes 1 and 3) or IT_CN_REJ_OTHER (for REJ code 4). The Implementation may wait
11576 until the timeout specified by the Consumer in the *it_path_t* structure input to *it_ep_connect*
11577 expires and then enqueue a non-peer reject Event with an IT_CN_REJ_TIMEOUT status.
11578 Alternatively, within the specified timeout period the Implementation may retry the Connection
11579 establishment attempt on the Consumer's behalf. If a Connection could not be established within
11580 the Consumer-specified timeout period, the Implementation should enqueue a non-peer reject
11581 Event with an IT_CN_REJ_TIMEOUT status after the timeout period has expired.

11582 When running on the IB transport, there are two different things that can signal the
11583 Implementation that it should generate the IT_CM_MSG_CONN_ESTABLISHED_EVENT on
11584 the Passive side: receiving an RTU message, or receiving a “Communication Established”
11585 Affiliated Asynchronous Event from the HCA. (Due to inherent races in the IB Connection
11586 establishment process, it is also possible that both of these conditions could be present.) If the
11587 Implementation receives an RTU message while the Endpoint is in the
11588 IT_EP_STATE_PASSIVE_CONNECTION_PENDING state and within the timeout period
11589 advertised to the Passive side in the REQ message, it should generate an
11590 IT_CM_MSG_CONN_ESTABLISHED_EVENT Event and use the Private Data from the RTU
11591 as part of that Event. If the Implementation receives the “Communication Established” Affiliated
11592 Asynchronous Event without receiving an RTU, the Implementation should generate the
11593 IT_CM_MSG_CONN_ESTABLISHED_EVENT Event with a Private Data size of zero, and
11594 when/if the RTU for the Connection subsequently arrives the Implementation should ignore it.

11595 For the InfiniBand Transport, there is a potential race condition in the three-way handshake
11596 Connection establishment method between the Implementation generating the
11597 IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT Event and the Consumer calling
11598 *it_ep_disconnect* (or *it_ep_free*).

11599 The conditions for the race arise when a Consumer has called *it_ep_connect*, but before the
11600 Connection is successfully established, the Consumer calls *it_ep_disconnect* or *it_ep_free* on the
11601 Endpoint. Within the timeframe of this single Connection attempt, the Implementation must
11602 order Events as follows.

11603 It is acceptable for the Implementation to generate and queue the IT_CM_MSG_CONN_
11604 ACCEPT_ARRIVAL_EVENT Event to the SEVD prior to the IT_CM_MSG_CONN_
11605 DISCONNECT_EVENT Event. But, when the IT_CM_MSG_CONN_DISCONNECT_EVENT
11606 Event is generated, the Implementation must invalidate the *cn_est_id* found in the
11607 IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT Event. Likewise, if the Consumer calls
11608 *it_ep_free*, the Implementation must also invalidate the *cn_est_id*. (Note that it is possible that
11609 the *cn_est_id* may have already been invalidated by a Consumer call to *it_ep_accept*, *it_reject* or
11610 *it_handoff*.)

11611 On the other hand, if the Implementation has generated an IT_CM_MSG_CONN_
11612 DISCONNECT_EVENT Event, and subsequently the Implementation receives indication that

11613 the Connection Request has been accepted by the remote side, the Implementation shall not
 11614 generate an IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT Event. In this situation, the
 11615 Implementation is still responsible for generating any necessary transport-specific response to
 11616 the arrived acceptance message.

11617 **it_conn_qual_t**

11618 When the IANA Port Number Connection Qualifier type is used with the VIA transport, the
 11619 IANA Port Number is mapped into a 2-byte VIA connection discriminator, with byte 0 of the
 11620 connection discriminator containing the upper 8 bits of the 16-bit IANA Port Number, and byte
 11621 1 containing the lower 8 bits of the 16-bit IANA Port Number.

11622 When the IANA Port Number Connection Qualifier type is used with the InfiniBand Transport,
 11623 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	

11624 **it_dto_cmpl_event_t**

11625 The handling of remotely detected errors is Implementation-dependent.

11626 On the iWARP Transport, when a locally posted RDMA DTO results in a remote access
 11627 violation, the remote iWARP Implementation tears down the connection abortively. For
 11628 implementing the IT_DTO_ERR_REMOTE_ACCESS Completion Error on iWARP, the
 11629 RDMAC Verbs provide a “remote termination error” completion status that allows on-the-fly
 11630 conversion of a received Terminate message to a Completion Error. Due to delays in the
 11631 reception of the Terminate message, it may not always be possible to use this mechanism.

11632 **it_dto_flags_t**

11633 The Implementation should attempt to support the use of IT_NOTIFY_FLAG on Receive DTOs.
 11634 Where the underlying transport does not support Receive DTO Notification Suppression it may
 11635 be necessary for the Implementation to generate Receive Notifications regardless of the setting
 11636 of the IT_NOTIFY_FLAG on the Receive DTOs.

11637 **it_ep_accept**

11638 In all cases where a communication manager message changes the state of an Endpoint the
 11639 Implementation must first transition the Endpoint state before generating the Event. This closes a
 11640 race condition where the Consumer may see the Event and call a function expecting the
 11641 Endpoint to be in the state that the Event results in. For example, when the Consumer reaps the
 11642 IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT the Endpoint should be in the
 11643 IT_EP_STATE_ACTIVE2_CONNECTION_PENDING state and ready to be accepted.

11644 **it_ep_attributes_t**

11645 The Implementation must allocate resources needed for the support of the Consumer-requested
11646 attributes. In general, the Implementation can allocate more resources than requested by the
11647 Consumer (a few exceptions are noted in the next paragraph). Some of these resources may
11648 actually be allocated at Connection establishment time for RC, but Connection establishment
11649 cannot fail because resources requested by the Consumer at Endpoint creation time are not
11650 available. Connection establishment may fail when the attributes associated with the local and
11651 remote Endpoints are mismatched. For example, for the InfiniBand and iWARP Transports, if
11652 the local Endpoint's attributes have an *rdma_read_ird* that is less than the remote Endpoint's
11653 *rdma_read_ord*, the Connection establishment attempt will fail.

11654 The attributes that the Implementation must allocate in exactly the quantity that the Consumer
11655 specified are:

11656 `it_rc_only_attributes_t.rdma_read_ord`
11657 `it_ep_attributes_t.max_dto_payload_size`

11658 The reason *rdma_read_ord* is listed above is that for InfiniBand you cannot establish a
11659 Connection unless the ORD value for one side of the Connection is less than or equal to the IRD
11660 value for the other side.

11661 The reason *max_dto_payload_size* is listed above is that for VIA you cannot establish a
11662 Connection unless the passive side and the active side Endpoints have matching values for this
11663 attribute.

11664 Whether the *max_dto_payload_size* limit is actually enforced for data transfers is IA-specific. (It
11665 might not be transport-specific; but it is not clear if all VIA Implementations do this checking.)

11666 An attempt to change *max_request_dtos* or *max_recv_dtos* for an Endpoint of an Interface
11667 Adapter whose *it_ia_info_ep_work_queues_resizable* is clear must not be successful and
11668 IT_ERR_INVALID_EP_STATE is the return value for this case.

11669 When running over the InfiniBand or iWARP Transports, the Implementation must set Signaling
11670 type to Selectable in order to support *it_dto_flags*.

11671 **it_ep_connect**

11672 When running over the InfiniBand Transport, if the Consumer provides a Path to *it_ep_connect*
11673 that contains a *P_Key* that is not in the HCA's *P_Key* table, the Implementation shall return
11674 IT_ERR_INVALID_SOURCE_PATH.

11675 **it_ep_disconnect**

11676 If the EP is already in the IT_EP_STATE_NONOPERATIONAL state, no messages or Events
11677 should be generated. In this case, the transport-level Disconnect Request should have been sent
11678 when the EP transitioned into the non-operational state.

11679 When running over the InfiniBand Transport, if *it_ep_disconnect* is called while the Endpoint is
11680 in the IT_EP_STATE_CONNECTED state, the Implementation should send CM DREQ
11681 (Disconnect Request) message.

11682

it_ep_free

11683
11684
11685
11686

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.

11687
11688
11689
11690
11691

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*.

11692
11693
11694

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.

11695

it_ep_rc_create

11696
11697

The Implementation should ignore the IT_EP_REUSEADDR *it_ep_rc_creation_flag_t* parameter on transports where the timewait state is not applicable.

11698
11699
11700

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.

11701

For the iWARP transport, the QP attribute for MW bind operations shall be set to “enabled”.

11702

it_ep_reset

11703
11704

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.

11705

it_ep_state_t

11706
11707
11708
11709
11710
11711

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.

11712
11713
11714
11715

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*.

11716 **it_evd_create**

11717 An IT-API Implementation should meet the following rules for EVD behavior.

11718 **Definitions**

- 11719 1. An arriving (queued on SEVD) Event is a *notification event* if any one of the following is
11720 true:
- 11721 a. It is an Event for a DTO with IT_NOTIFY_FLAG set when posted.
 - 11722 b. It is an Event for a Recv DTO where matching Send DTO was originally posted
11723 with the IT_SOLICITED_WAIT_FLAG set (regardless of IT_NOTIFY_FLAG on
11724 Recv DTO).
 - 11725 c. It is the *N*th Event to arrive where threshold is set to *N*.
 - 11726 d. It is an Event for a DTO completing in error regardless of IT_NOTIFY_FLAG and
11727 IT_COMPLETION_FLAG.
 - 11728 e. It is a non-DTO Completion Event.

11729 The above are called *arriving notification events*.

11730 The Events of type a, b, d, and e are also called *plain notification events* and retain their
11731 notification status¹¹ on the SEVD queue.

- 11732 2. An arriving Event is a *non-notification event* if none of the above 1a. to 1e. criteria are
11733 met. These are called both *arriving non-notification events* and *plain non-notification*
11734 *events*.
- 11735 3. IT_THRESHOLD_DISABLED stands for no threshold or threshold == infinity.
- 11736 4. The desired semantic for IT-API is: *it_evd_wait* returns only:
- 11737 a. For SEVD – when there is a Notification Event of 1a, 1b, 1d, or 1e type (above) or
11738 when there are a number of Events on the SEVD equal to or more than the
11739 threshold on the SEVD queue.
 - 11740 b. For AEVD – when any one of the associated SEVDs satisfies 4a.

11741 **Rules**

- 11742 1. *it_evd_wait* will block¹² when:
- 11743 a. For SEVD – queue is empty.
 - 11744 b. For AEVD – all associated SEVDs are empty.

¹¹ 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.

¹² This is really an Implementation issue. Semantically, Consumer does not care if it is blocked or not.

- 11745 2. *it_evd_wait* may block if there are no notification events and number of Events is below
11746 the threshold:
- 11747 a. For SEVD – on the SEVD queue.
- 11748 b. For AEVD – on all associated SEVD queues.
- 11749 3. *it_evd_wait* **will** return if there is a *notification event*¹³ or the number of Events is greater
11750 than or equal to the *threshold*:
- 11751 a. For SEVD – on the SEVD queue.
- 11752 b. For AEVD – on any associated SEVDs.
- 11753 4. If threshold > 1, *it_evd_wait* **should** block¹⁴ if less than threshold number of Events and
11754 no *notification events* are¹⁵:
- 11755 a. For SEVD – on the SEVD queue.
- 11756 b. For AEVD – on all SEVD queues of AEVD.
- 11757 5. Each *arriving notification event* **must** unblock at least one waiter¹⁶, but **should** unblock
11758 only one waiter.
- 11759 6. An *arriving notification event* **can** unblock as many waiters as there are Events available.
- 11760 7. *it_evd_dequeue* **will** return an Event if one exists regardless of waiters from:
- 11761 a. For SEVD – the SEVD queue.
- 11762 b. For AEVD with IT_EVD_DEQUEUE_NOTIFICATIONS set – the AEVD queue of
11763 IT_AEVD_NOTIFICATION_EVENTS
- 11764 c. For AEVD with IT_EVD_DEQUEUE_NOTIFICATIONS cleared – any of its
11765 SEVDs.¹⁷

¹³ 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).

¹⁴ 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.

¹⁵ 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.

¹⁶ 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.

¹⁷ 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.

11766 **Heuristic for Wakeup of Multiple Waiters**

11767 Rules 5 and 6 in the EVD Rules (above) require that multiple waiters be awakened on every
11768 Notification Event because of the possibility of Notification coalescing.

11769 The following heuristic is recommended: In the Notification Handler, check the queue length¹⁸,
11770 and wake up that many waiters and no more. This limits the number of threads that wake up, and
11771 reduces the number that wake up and find no Event because some other threads(s) has consumed
11772 them.

11773 The handler algorithm should be something like:

```
11774 evd_handler() {  
11775     for (;;) {  
11776         nToUnblock = min(nmore(),nwaiters());  
11777         for (n = 0; n != nToUnblock; ++n)  
11778             unblockWaiter();  
11779         if (nwaiters() == 0) break;  
11780         rearmHandler();  
11781         if (nmore() == 0) break;  
11782     }  
11783 }
```

11784 **Additional Issues**

11785 An AEVD does not have a queue as such. Potentially, an AEVD can be implemented using a bit
11786 array with an entry for each feeding SEVD. An SEVD bit "set" for an AEVD with
11787 IT_EVD_DEQUEUE_NOTIFICATIONS set means that the SEVD Handle can be returned in an
11788 IT_AEVD_NOTIFICATION_EVENT Event from the AEVD wait or dequeue. The SEVD bit
11789 "set" for an AEVD with IT_EVD_DEQUEUE_NOTIFICATIONS cleared means that the actual
11790 underlying SEVD Event can be returned from the AEVD wait or dequeue.

11791 Ideally, the SEVD bit it cleared as soon as the SEVD is not in the Notification criteria. But it can
11792 be cleared as soon as the SEVD becomes empty or, instead, only when both it becomes empty
11793 and there is direct (waiter on SEVD) or indirect waiter (AEVD waiter or FD) on the SEVD.

11794 If the SEVD is enabled, then an arriving SEVD Event that causes the SEVD to reach
11795 Notification criteria or maintain Notification criteria of the SEVD will set the SEVD bit for the
11796 AEVD.

11797 It is recommended that the Implementation employ a "starvation-free" algorithm in returning
11798 Events via an AEVD from underlying SEVDs. That is, the Implementation should ensure
11799 forward progress on all SEVDs feeding an AEVD.

11800 For IT_EVD_DEQUEUE_NOTIFICATION set, the Implementation should eventually select
11801 every feeding SEVD that has reached Notification status when generating the next AEVD
11802 Notification Event.

11803 For IT_EVD_DEQUEUE_NOTIFICATION cleared, the Implementation of *it_evd_wait* should
11804 eventually select the first Event from every feeding SEVD that has reached Notification status.

¹⁸ The queue length function is not Verbs-compliant, but will be commonly available.

11805 When the requested stream type is `IT_ASYNC_AFF_EVENT_STREAM` or
11806 `IT_ASYNC_UNAFF_EVENT_STREAM`, the Implementation creates the requested EVD, and
11807 fills in the appropriate *evd* field in the associated *it_ia_info_t* structure for that IA.

11808 The Implementation shall not provide to the Consumer Unaffiliated and Affiliated Events that
11809 happened before the Consumer created SEVDs for an Unaffiliated and Affiliated Event Stream.
11810 The Implementation can provide Events that happened during SEVD creation time.

11811 **it_evd_dequeue**

11812 The Implementation should abide by the EVD rules defined in the *it_evd_create* section of this
11813 Implementers Guide.

11814 All synchronization issues between multiple waiters and/or dequeue-ers from the same Event
11815 Dispatcher simultaneously are left to the Implementation.

11816 The Implementation is not required to check that the *Event* structure that the Consumer provides
11817 is sufficient to hold a returned Event.

11818 **it_evd_post_se**

11819 All the synchronization issues between multiple Consumer Contexts trying to post software
11820 Events to an Event Dispatcher instance simultaneously are left to the Implementation.

11821 **it_evd_wait**

11822 The Implementation should abide by the EVD rules defined in the *it_evd_create* section of this
11823 Implementers Guide.

11824 All synchronization issues between multiple waiters and/or dequeue-ers from the same Event
11825 Dispatcher simultaneously are left to the Implementation.

11826 The Implementation is not required to check that the *event* structure that the Consumer provides
11827 is sufficient to hold a returned Event.

11828 **it_event_t**

11829 Each Event Stream has a designated contiguous range of Event numbers with common most-
11830 significant bits (as masked by `IT_EVENT_STREAM_MASK`) representing the Event Stream.

11831 **it_handle_t**

11832 The definition of all object Handles is Implementation-specific, but all object Handles can be
11833 typecast to *it_handle_t* and *vice versa*.

11834 **it_handoff**

11835 Once the IA is open, and until all processes close it, the *ia_name* and Spigot identifier need to
11836 reference the same objects for all processes that are referencing the IA.

11837 **it_ia_create**

11838 The Implementation shall not provide to the Consumer Unaffiliated and Affiliated Events that
11839 happened before the Consumer created SEVDs for the Unaffiliated and Affiliated Event Stream.
11840 The Implementation can provide Events that happened during SEVD creation time.

11841 **it_ia_free**

11842 The Implementation should free the IT Objects in the reverse order of their construction in order
11843 to guarantee that all underlying transport resources will be successfully freed.

11844 **it_ia_info_t**

11845 The 1000+ number range in the *it_transport_type_t* is intended to facilitate the short-term
11846 prototyping efforts of vendors who are developing new transports that work with the IT-API. A
11847 vendor that wishes to productize their prototyping effort should contact the ICSC in order to be
11848 assigned a permanent transport number in the <1000 range. The Implementation is not
11849 responsible for ensuring that two different vendors utilizing the 1000+ range of transport
11850 numbers do not collide.

11851 For the IB transport, when *it_ep_disconnect* is called there are two possible underlying CM
11852 messages that the Private Data could travel with: DREQ or REJ. The value of
11853 *disconnect_private_data_len* for IB should be the lesser of the maximum Private Data supported
11854 in a DREQ message and the maximum Private Data supported in a REJ message.

11855 **it_ia_query**

11856 The Implementation must track any allocation(s) of *it_ia_info_t* structures due to *it_ia_query*
11857 calls so that any allocated *it_ia_info_t* structure(s) can later be freed if necessary when
11858 *it_ia_info_free* or *it_ia_free* is called.

11859 **it_listen_create**

11860 If the EVD where the CM Request Events stream (IT_CM_REQ_EVENT_STREAM) is routed
11861 on the passive side is full, then the Active side shall receive an
11862 IT_CM_MSG_CONN_NONPEER_REJECT_EVENT Event with a reason code of
11863 IT_CN_REJ_TIMEOUT.

11864 **it_lmr_create**

11865 On the InfiniBand Transport, *it_lmr_create* can be implemented by using the Register Physical
11866 Memory Region or Register Memory Region verb. Use of the Register Physical Memory Region
11867 verb on InfiniBand incurs a risk that the InfiniBand Transport may return a differing address
11868 from what was requested by the Consumer (for instance, possibly due to misaligned addresses).
11869 The Register Physical Memory Region verb specifies an I/O Virtual Address (IOVA) modifier
11870 as both input and output. The IOVA output value denotes the “actually assigned IOVA”. In case
11871 BMM is supported, the IOVA output value must equal the IOVA input value for Absolute
11872 Addressing and zero for Relative Addressing (ZBVA). In case BMM is not supported (which

11873 rules out Relative Addressing), the IOVA output value may differ from the IOVA input value if
11874 byte granularity for Memory Regions is not supported; if there is a difference, the
11875 Implementation of *it_lmr_create* shall verify the assertion that the IOVA output value equals the
11876 lower bound of the “Actual Local Protection Bounds enforced by the Channel Interface”
11877 returned by the Query Memory Region verb and, if this assertion is false, deregister the Memory
11878 Region and return the IT_ERR_ABORT immediate error.

11879 Using the Register Physical Memory Region verb has the advantage that control over Memory
11880 Region pinning/unpinning remains in generic, OS-provided code, while using the Register
11881 Memory Region verb would pass such control to IHV-private code. Using the Register Physical
11882 Memory Region verb is thus preferred, since it (i) retains sufficient OS control over pinned
11883 memory, (ii) supports multiple registrations of Memory Regions in a multi-IHV environment,
11884 and (iii) reduces duplication of pinning code in IHV verbs provider drivers.

11885 On the iWARP Transport, *it_lmr_create* can be implemented with the Register Non-Shared
11886 Memory Region verb.

11887 For both InfiniBand and iWARP, the Implementation of *it_lmr_create* must set the Enable
11888 Memory Window Binding input modifier to “true” in order to allow subsequent calls to
11889 *it_rmr_link* using this LMR. The IT_LMR_FLAG_SHARED option can be implemented by
11890 registering either a new or a shared Memory Region. The Implementation should search for a
11891 matching LMR. If a match is found, call the Register Shared Memory Region Verb; if not, call
11892 the Register (Physical) Memory Region verb for InfiniBand and the Register Non-Shared
11893 Memory Region verb for iWARP.

11894 The remote access flag for memory region allocation, registration, or re-registration (iWARP
11895 and IB-1.2:BMM) is not exposed by IT-API and should be set to “true” by the Implementation
11896 when creating an LMR.

11897 The Implementation should protect against race conditions between multiple Consumers and
11898 avoid calling memory registration verbs multiple times for the same memory region.

11899 See the advice under *it_rmr_link* for comments concerning *rmr_context* and byte order.

11900 ***it_lmr_create_unlinked***

11901 For InfiniBand and iWARP, *it_lmr_create_unlinked* corresponds to the Allocate L_Key verb
11902 and the Allocate Non-Shared Memory Region STag verb, respectively.

11903 The remote access flag for memory region allocation, registration, or re-registration (iWARP
11904 and IB-1.2:BMM) is not exposed by IT-API and should be set to “true” by the Implementation
11905 when creating an unlinked LMR.

11906 ***it_lmr_free***

11907 For InfiniBand and iWARP, *it_lmr_free* corresponds to the Deregister Memory Region verb and
11908 the Deallocate STag verb, respectively.

11909 Beware that determining when it is permissible to unlock or unpin physical memory is tricky.
11910 The Implementation must handle multiple LMRs with overlapping ranges, LMRs on different

11911 IAs with overlapping ranges, and LMRs created in overlapping regions of shared memory by
11912 different Consumers. In addition, any of these LMRs may have been created with the
11913 IT_LMR_FLAG_SHARED flag set.

11914 **it_lmr_link**

11915 For InfiniBand and iWARP, *it_lmr_link* corresponds to the Fast Register Physical Memory
11916 Region verb and the Fast-Register Non-Shared Memory Region verb, respectively.

11917 For both InfiniBand and iWARP, the Implementation of *it_lmr_link* must include the Enable
11918 Memory Window Binding input modifier to allow subsequent calls to *it_rmr_link* using this
11919 LMR.

11920 **it_lmr_modify**

11921 On the InfiniBand Transport, *it_lmr_modify* corresponds to the Reregister Physical Memory
11922 Region or Reregister Memory Region verb. When an LMR enabled for remote access is
11923 modified, the Implementation should modify the 8-bit Key Portion of the R_Key (which is
11924 shared with the Key Portion of the L_Key) to reduce the potential for problems with stale
11925 R_Keys.

11926 On the iWARP Transport, *it_lmr_modify* can be implemented with the Reregister Non-Shared
11927 Memory Region verb for a non-shared LMR, and with the Deallocate STag verb followed by the
11928 Register Shared Memory Region verb for a shared LMR. When an LMR enabled for remote
11929 access is modified, the Implementation should modify the 8-bit STag Key to reduce the potential
11930 for problems with stale STags.

11931 **it_lmr_query**

11932 For both InfiniBand and iWARP, *it_lmr_query* can be implemented with the Query Memory
11933 Region verb.

11934 For InfiniBand, the *actual_addr* and *actual_length* fields in *params* should be derived from the
11935 Actual Remote Protection Bounds returned by the verb, because the Consumer will be most
11936 concerned with the degree of exposure of the region to remote Consumers. For a transport
11937 Implementation that does not provide Memory Region bounds checking with byte-level
11938 granularity, the remote bounds are obtained by rounding the bounds requested by the Consumer
11939 to 4K byte boundaries, and the local bounds are obtained by rounding the bounds requested by
11940 the Consumer to page boundaries. The Actual Remote Protection Bounds will be contained
11941 within the Actual Local Protection Bounds for any IB transport Implementation. This means the
11942 Consumer may safely use the returned *actual_addr* and *actual_length* as both local and remote
11943 bounds.

11944 For iWARP and any transport supporting Relative Addressing, LMR creation and bounds
11945 checking must be performed with byte-level granularity. For InfiniBand 1.2, LMR creation and
11946 bounds checking should be performed with byte-level granularity. In case of byte-level
11947 granularity, the actual starting address and actual length must equal the requested starting
11948 address and requested length, respectively.

11949 **it_lmr_flush_to_mem**
11950 **it_lmr_refresh_from_mem**

11951 There is no defined error code for the case where some portion of the *local_segments* array lies
11952 outside the Consumer's valid address space. It is expected that the Implementation will signal the
11953 application with the appropriate platform-dependent signal in this case, as would happen for any
11954 dereference of an invalid pointer.

11955 **it_lmr_unlink**

11956 For InfiniBand and iWARP, *it_lmr_unlink* corresponds to the Local Invalidate verb and the
11957 Invalidate Local STag verb, respectively.

11958 **it_post_rdma_read**

11959 The Implementation should avoid resource allocation as part of *it_post_rdma_read* to ensure that
11960 this operation is non-blocking and thread-safe. This operation cannot fail due to insufficient
11961 resources. All resource allocation required must be done at Endpoint creation time to ensure that
11962 all necessary resources are available at post time.

11963 The Implementation should support zero-copy data transfers and kernel bypass for the RDMA
11964 Read operation.

11965 For iWARP, an RNIC may provide advanced RDMA Read error handling support; i.e.,
11966 RDMAP/DDP support for mapping an error due to an inbound RDMA Read Response to a
11967 Completion Error, support for on-the-fly conversion of an RDMAP Terminate message into a
11968 Completion Error and, optionally, early detection of a local access violation due to an RDMA
11969 Read Request.

11970 RDMAP/DDP support for mapping an error due to an inbound RDMA Read Response to a
11971 Completion Error is optional in DDP, but useful. On-the-fly conversion of an RDMAP
11972 Terminate message into a Completion Error is useful in case of a remotely detected access
11973 violation due to an RDMA Read Request, which results in a Terminate message containing the
11974 RDMA Read Request header. Early detection of a local access violation due to an RDMA Read
11975 Request is useful since certain access violations pertaining to the local sink buffer or Endpoint
11976 can be predicted before an RDMA Read Request is issued via the underlying transport protocol;
11977 issuing an RDMA Read Request can be avoided in this case, but incoming RDMA Writes/Read
11978 Responses are checked anyway regarding access rights.

11979 **it_post_rdma_read_to_rmr**

11980 This call is available for iWARP only.

11981 The Implementation should avoid resource allocation as part of *it_post_rdma_read_to_rmr* to
11982 ensure that this operation is non-blocking and thread-safe. This operation cannot fail due to
11983 insufficient resources. All resource allocation required must be done at Endpoint creation time to
11984 ensure that all necessary resources are available at post time.

11985 The Implementation should support zero-copy data transfers and kernel bypass for the RDMA
11986 Read operation.

11987 An RNIC may provide advanced RDMA Read error handling support; i.e., RDMAP/DDP
11988 support for mapping an error due to an inbound RDMA Read Response to a Completion Error,
11989 support for on-the-fly conversion of an RDMAP Terminate message into a Completion Error
11990 and, optionally, early detection of a local access violation due to an RDMA Read Request.

11991 RDMAP/DDP support for mapping an error due to an inbound RDMA Read Response to a
11992 Completion Error is optional in DDP, but useful. On-the-fly conversion of an RDMAP
11993 Terminate message into a Completion Error is useful in case of a remotely detected access
11994 violation due to an RDMA Read Request, which results in a Terminate message containing the
11995 RDMA Read Request header. Early detection of a local access violation due to an RDMA Read
11996 Request is useful since certain access violations pertaining to the local sink buffer or Endpoint
11997 can be predicted before an RDMA Read Request is issued via the underlying transport protocol;
11998 issuing an RDMA Read Request can be avoided in this case, but incoming RDMA Writes/Read
11999 Responses are checked anyway regarding access rights.

12000 **it_post_rdma_write**

12001 The Implementation should avoid resource allocation as part of *it_post_rdma_write* to ensure
12002 that this operation is non-blocking and thread-safe. This operation cannot fail due to insufficient
12003 resources. All resource allocation required must be done at Endpoint creation time to ensure that
12004 all necessary resources are available at post time.

12005 The Implementation should support zero-copy data transfers and kernel bypass for the RDMA
12006 Write operation.

12007 **it_post_recv**

12008 The Implementation should avoid resource allocation as part of *it_post_recv* to ensure that this
12009 operation is non-blocking and thread-safe. This operation cannot fail due to insufficient
12010 resources. All resource allocation required must be done at Endpoint or Shared Receive Queue
12011 creation time to ensure that all necessary resources are available at post time.

12012 The Implementation should support zero-copy data transfers and kernel bypass for the Receive
12013 operation.

12014 **it_post_recvfrom**

12015 The Implementation should avoid resource allocation as part of *it_post_recvfrom* to ensure that
12016 this operation is non-blocking and thread-safe. This operation cannot fail due to insufficient
12017 resources. All resource allocation required must be done at Endpoint creation time to ensure that
12018 all necessary resources are available at post time.

12019 The Implementation should support zero-copy data transfers and kernel bypass for the
12020 ReceiveFrom operation.

12021 **it_post_send**

12022 The Implementation should avoid resource allocation as part of *it_post_send* to ensure that this
12023 operation is non-blocking and thread-safe. This operation cannot fail due to insufficient

12024 resources. All resource allocation required must be done at Endpoint creation time to ensure that
12025 all necessary resources are available at post time.

12026 The Implementation should support zero-copy data transfers and kernel bypass for the Send
12027 operation.

12028 **it_post_sendto**

12029 The Implementation should avoid resource allocation as part of *it_post_sendto* to ensure that this
12030 operation is non-blocking and thread-safe. This operation cannot fail due to insufficient
12031 resources. All resource allocation required must be done at Endpoint creation time to ensure that
12032 all necessary resources are available at post time.

12033 For details on handling IT_ERR_INVALID_AH under InfiniBand see compliance statement
12034 o10-2.1.1 in [IB-R1.2].

12035 The Implementation should support zero-copy data transfers and kernel bypass for the *SendTo*
12036 operation.

12037 **it_rmr_create**

12038 For both InfiniBand and iWARP, *it_rmr_create* corresponds to the Allocate Memory Window
12039 verb.

12040 **it_rmr_free**

12041 For InfiniBand and iWARP, *it_rmr_free* corresponds to the Deallocate Memory Window verb
12042 and the Deallocate STag verb, respectively.

12043 **it_rmr_link**

12044 On the InfiniBand Transport, *it_rmr_link* must immediately select the appropriate verb for
12045 linking an RMR, namely Bind MW for a Wide RMR and Post Send Bind MW for a Narrow
12046 RMR.

12047 On the iWARP Transport, *it_rmr_link* translates to the PostSQ verb's Bind Memory Window
12048 operation. The Implementation should modify the 8-bit STag Key when binding a Memory
12049 Window to reduce the potential for problems with stale STags.

12050 The *rmr_context* returned by *it_rmr_link* is defined to be returned in network byte order; i.e., in
12051 big endian format. The intent is that no further reordering of the bytes of the *rmr_context* will be
12052 performed by either the local or remote Consumer, or the local or remote Implementation. When
12053 the remote Consumer passes the *rmr_context* to a call such as *it_post_rdma_write*, the
12054 Implementation of *it_post_rdma_write* will put the first byte of *rmr_context* on the network wire
12055 first, the second byte of *rmr_context* second, and so on. Thus, the Implementation of *it_rmr_link*
12056 should return the *rmr_context* in the byte order that the IA expects to see on the wire, so that the
12057 IA may correctly interpret the incoming *rmr_context*.

12058

it_rmr_query

12059

For InfiniBand and iWARP, *it_rmr_query* can be implemented with the Query Memory Window verb.

12060

12061

Alternatively, in order to return correct values for any RMR attributes that may change after RMR creation, the Implementation may track the RMR Create/Free operations and the completion status of RMR Link and RMR Unlink operations, including the parameters passed to these operations. To do so, the Implementation can replace the Consumer's cookie passed to *it_rmr_link* or *it_rmr_unlink* with an Implementation cookie that points to a structure. This structure stores the original Consumer cookie and all relevant parameters to the Link or Unlink operation. In *it_evd_dequeue*, the Implementation would peek at the operation type of the dequeued Completion Event. In addition to tracking RMR Create/Free operations, an Implementation would track the Bind MW, Post Send Type-2 MW Bind, and Post Send Local Invalidate operations for InfiniBand (note that InfiniBand does not provide an Unlink MW operation), and the PostSQ Bind Memory Window and PostSQ Invalidate Local STag operations for iWARP. If such an operation completes successfully, then extract the Implementation cookie, and restore the Consumer's cookie. Read the structure, and copy the parameters to storage associated with the RMR object. The next *it_rmr_query* call can obtain its parameters from this storage. This alternative requires that the Implementation always requests signaled completions, regardless of whether the Consumer includes IT_COMPLETION_FLAG in the *dto_flags* of an RMR Link/Unlink operation. For RMR Link/Unlink operations with IT_COMPLETION_FLAG cleared, the Implementation must dequeue the completion as if none was generated.

12062

12063

12064

12065

12066

12067

12068

12069

12070

12071

12072

12073

12074

12075

12076

12077

12078

12079

12080

it_rmr_unlink

12081

For InfiniBand, *it_rmr_unlink* must immediately select the appropriate verb for unlinking an RMR, namely Bind MW (with a length of zero) for a Wide RMR and Post Send Invalidate MW for a Narrow RMR.

12082

12083

12084

For iWARP, *it_rmr_unlink* corresponds to the PostSQ verb's Invalidate Local STag operation.

12085

it_ud_service_request_handle_create

12086

The Implementation only needs to validate the components of the *it_path_t* structure that refer to local entities, specifically *spigot_id*, *local_port_lid*, and *local_port_gid*.

12087

12088

When running over the InfiniBand Transport, if the Consumer provides a Path to *it_ud_service_request_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.

12089

12090

12091

it_unaffiliated_event_t

12092

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 Unaffiliated Event EVD occurs, hardware resources should still be dequeued and discarded.

12093

12094

12095

12096
12097
12098

In the case of Unaffiliated Events, the underlying Implementation should copy every Event into each process-specific queue.

12099 **B** **Implementer’s Guide to Connection Management for** 12100 **iWARP**

12101 **B.1** **Overview**

12102 The IT-API provides RDMA services for several underlying RDMA transports, including
12103 InfiniBand and iWARP¹⁹. The specifics of these RDMA transports as well as different
12104 application requirements led to the design of two different Connection management interfaces
12105 for Reliable Connected (RC) RDMA transports, as introduced below.

12106 The Transport-Independent Interface (TII) provides a unified RDMA Connection management
12107 service abstraction for any RC transport supported by the IT-API. The TII includes the
12108 *it_ep_connect*, *it_listen_create*, *it_ep_accept*, *it_reject*, and *it_ep_disconnect* calls. The TII
12109 allows Consumers to negotiate the IRD and ORD parameters and to exchange Private Data
12110 during Connection establishment. Consumers are expected to preconfigure the IRD and ORD
12111 parameters of their Endpoints prior to calling *it_ep_connect* or *it_ep_accept*. From the
12112 Consumer’s viewpoint, the RDMA service is available immediately after Connection
12113 establishment. For iWARP, Connection establishment with the TII involves creating and
12114 connecting a socket within the API Implementation, as described in Section 5.2. This socket is
12115 not exposed through a Lower Layer Protocol (LLP) handle to the Consumer.

12116 The Transport-Dependent Interface (TDI) is available for iWARP only and allows the
12117 Conversion of an unconnected Endpoint of the RC type and a connected socket to a connected
12118 Endpoint. The TDI includes the *it_socket_convert* and *it_ep_disconnect* calls. It enables
12119 Consumers to exchange an arbitrary (but non-zero) amount of streaming-mode data via a TCP
12120 socket (and possibly via an SCTP socket in the future) prior to performing a Conversion, which
12121 is a requirement for several TCP/IP-based applications. This prior exchange of streaming-mode
12122 data occurs outside the IT-API. In contrast to Connection establishment with the TII, the
12123 *it_socket_convert* call neither supports the negotiation of IRD and ORD parameters, nor the
12124 exchange of Private Data. The main reason for this simplification is that the peers have an
12125 opportunity to negotiate IRD and ORD and to exchange Private Data prior to Conversion;
12126 moreover, they have an opportunity to modify ORD after the Conversion²⁰; i.e., while already in
12127 RDMA mode. The Implementation of the TDI is described in Section 5.3.

12128 Additional details are found in Chapter 5.

¹⁹ IT-API Version 2 will support TCP-based iWARP. SCTP support may be added in a later issue.

²⁰ Unfortunately, there is no opportunity to modify IRD after the Conversion because this is an optional capability according to [VERBS-RDMAC].

12129 **B.2 Miscellaneous**

12130 **Private Data**

12131 The IT-API must report Private Data length supported for the *it_ep_connect*, *it_ep_accept*, and
12132 *it_reject* calls as 256 bytes and report the Private Data length supported for the *it_ep_disconnect*
12133 routine as zero.

12134 The IT-API must report Private Data length supported for the Unreliable Datagram service as
12135 zero as well even though the iWARP Transport does not support the IT_UD_SERVICE service
12136 type.

12137 The Private Data fields are fixed-size and it is the Consumer's responsibility to define which
12138 portion of the fields contain their own valid data. Such a requirement is legacy behavior from
12139 other RDMA transports (specifically InfiniBand).

12140 **Handshake**

12141 The IT-API supports only two-way Connection establishment for iWARP. The
12142 *three_way_handshake_support* boolean found in the *it_ia_info_t* structure must be reported as
12143 IT_FALSE.

12144 **B.3 Interoperability Considerations**

12145 The IETF has released a draft document describing modifications to the MPA startup wire
12146 protocol [INTEROP-IETF] intended to aid in achieving interoperability between the RDMAC
12147 and IETF versions of iWARP.

12148 Where interoperability is not possible – i.e., between Non-permissive AV-RNIC/IETFs and
12149 RDMAC AV-RNICS – Connection attempts via the TII or TDI should result in
12150 IT_CM_MSG_CONN_NONPEER_REJECT_EVENT Events with the reject reason code
12151 IT_CN_REJ_BAD_CONN_PARMS or IT_CM_MSG_CONN_BROKEN_EVENT Events. See
12152 the Interoperability Ladder Diagrams found in both the TII and TDI sections of this document.

12153 Where the MPA startup protocol is not used – i.e., between RDMAC AV-RNICS –
12154 interoperability concerns are left to the ULP.

12155 **B.4 MPA Marker Control**

12156 The RDMAC version of MPA [MPA-RDMAC] mandated the use of MPA framing (Markers)
12157 for the use of recovering placement information for DDP frames. The IETF version of MPA
12158 [MPA-IETF] made the use of Markers optional and as specified by the receiver.

12159 The IT-API implementations should implement the recommendations of [INTEROP-IETF].

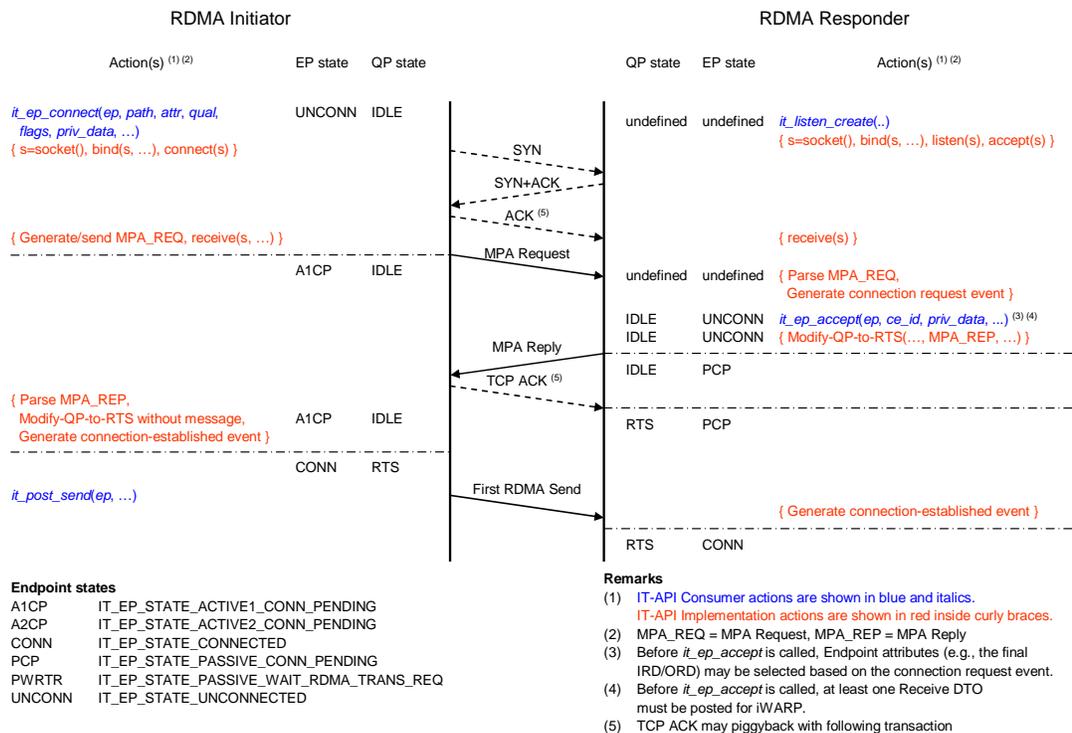
12160 B.5 Transport-Independent Interface

12161 Overview

12162 For the *Transport-Independent Interface (TII)*, the Consumer calling *it_ep_connect* and the
 12163 corresponding side of the Connection is referred to as the *RDMA Initiator*, while the Consumer
 12164 calling *it_listen_create* followed by *it_ep_accept* (or *it_reject*) and the corresponding side of the
 12165 Connection is referred to as the *RDMA Responder*.

12166 The Endpoint finite-state machine uses the active (passive) Endpoint states for the RDMA
 12167 Initiator (Responder).

12168 Connection establishment for the TII in case of an RNIC without RTR state (see below) is
 12169 illustrated in Figure 13.



12170
 12171 **Figure 13: Connection Establishment with MPA Startup (TII): RNIC without RTR State**

12172 The RDMA Responder uses *it_listen_create* to create a Listen Point, which causes the (API)
 12173 Implementation to create and bind a socket, to listen for an incoming LLP connection request,
 12174 and to accept such a request. The RDMA Initiator invokes *it_ep_connect* for connecting an
 12175 Endpoint of the RC type, which causes the Implementation to create and bind a socket and to
 12176 attempt a connection. As soon as the LLP connection is established, the Implementation
 12177 completes TII Connection establishment by performing an Immediate RDMA Transition. For

12178 TCP, the transition to RDMA mode corresponds to the *MPA Startup phase*, during which an
12179 MPA Request/Reply handshake takes place in streaming mode [MPA-IETF].

12180 The RDMA Transition is visible to the Consumers only through the exchange of Private Data; in
12181 particular, the Implementation handles the details of the MPA Startup.

12182 The RDMA Initiator generates and sends an MPA Request message with private data containing
12183 the IRD/ORD Header (IOH) and the private data provided by the Consumer with *it_ep_connect*.
12184 After receiving a valid MPA Reply message with the MPA Reject bit not set, the RDMA
12185 Initiator invokes the Modify-QP-to-RTS verb with a message length of zero and generates a
12186 Connection established event for the Consumer providing the remote Endpoint's IRD and ORD
12187 parameters as well as the remote Consumer's private data.

12188 The RDMA Responder's Listen Point expects an MPA Request. Upon receiving a valid MPA
12189 Request message, it generates a Connection Request event for the Consumer containing the
12190 remote Endpoint's IRD and ORD parameters as well as the remote Consumer's Private Data.
12191 The Consumer on the RDMA Responder side now invokes either *it_ep_accept* or *it_reject*,
12192 optionally with Private Data.

12193 Calling *it_ep_accept* causes the Implementation to associate the Connection establishment ID
12194 (and the corresponding LLP connection) with an Endpoint. It then causes the Implementation to
12195 generate an MPA Reply message with Private Data containing the IOH and the Private Data
12196 provided by the Consumer. Next, the Implementation invokes the Modify-QP-to-RTS verb,
12197 providing the MPA Reply message as the Last Streaming-Mode Message. Finally, upon
12198 detection of the first iWARP payload, the Implementation of *it_ep_accept* generates a
12199 Connection established event for the Consumer providing the remote Endpoint's IRD and ORD
12200 parameters as well as the remote Consumer's Private Data, and transitions the Endpoint into
12201 IT_EP_STATE_CONNECTED state.

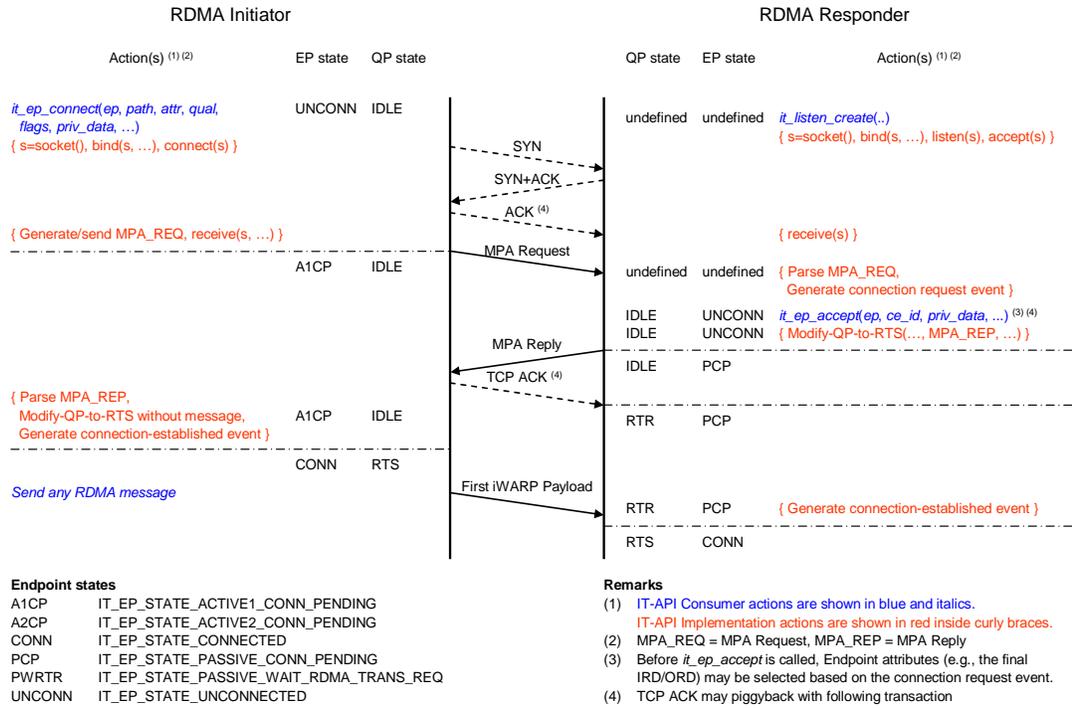
12202 For Figure 13, the Modify-QP-to-RTS verb is assumed to transition the QP to RTS state without
12203 waiting for the reception of the first iWARP payload (FPDU or RDMA message). While this
12204 behavior complies with [VERBS-RDMAC], it will not guarantee that the underlying iWARP
12205 Implementation stalls the sending of iWARP payload before the first iWARP payload has been
12206 received, as required by [MPA-IETF]. The fact that premature sending of the iWARP payload
12207 could result in synchronization problems and that receiving a Completion for an RDMAP Post
12208 Receive work request is the only way to detect the arrival of the first iWARP payload on an
12209 RNIC complying with [VERBS-RDMAC] has the following, far-reaching consequences:

- 12210 • For the ULP, after reaping the Connection established event, the RDMA Initiator must
12211 post a Send DTO to allow the RDMA Responder to detect the first iWARP payload.
- 12212 • For the ULP, the RDMA Responder must post at least one Receive DTO prior to calling
12213 *it_ep_accept*.
- 12214 • For the Implementation of *it_ep_accept*, the reception of the first iWARP payload must be
12215 detected by noting the Completion of the Consumer's first Receive DTO.

12216 In order to minimize transport-dependent ULP requirements, the IT-API offers an alternative for
12217 RNICs that implement an additional *Ready-To-Receive (RTR)* QP state to wait for the first
12218 iWARP payload, as well as an additional asynchronous event to indicate the RTR-to-RTS
12219 transition when the first iWARP payload is received on the RDMA Responder side. Connection

12220
12221

establishment for the TII in case of an RNIC with an RTR state (extended QP state machine) is depicted in Figure 14.



12222

Figure 14: Connection Establishment with MPA Startup (TII): RNIC with RTR State

12223

12224

This alternative approach has the following consequences:

12225

12226

12227

- 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.
- 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.

12228

12229

12230

IT-API ULP and IRD/ORD Header

12231

12232

12233

12234

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.

12235

12236

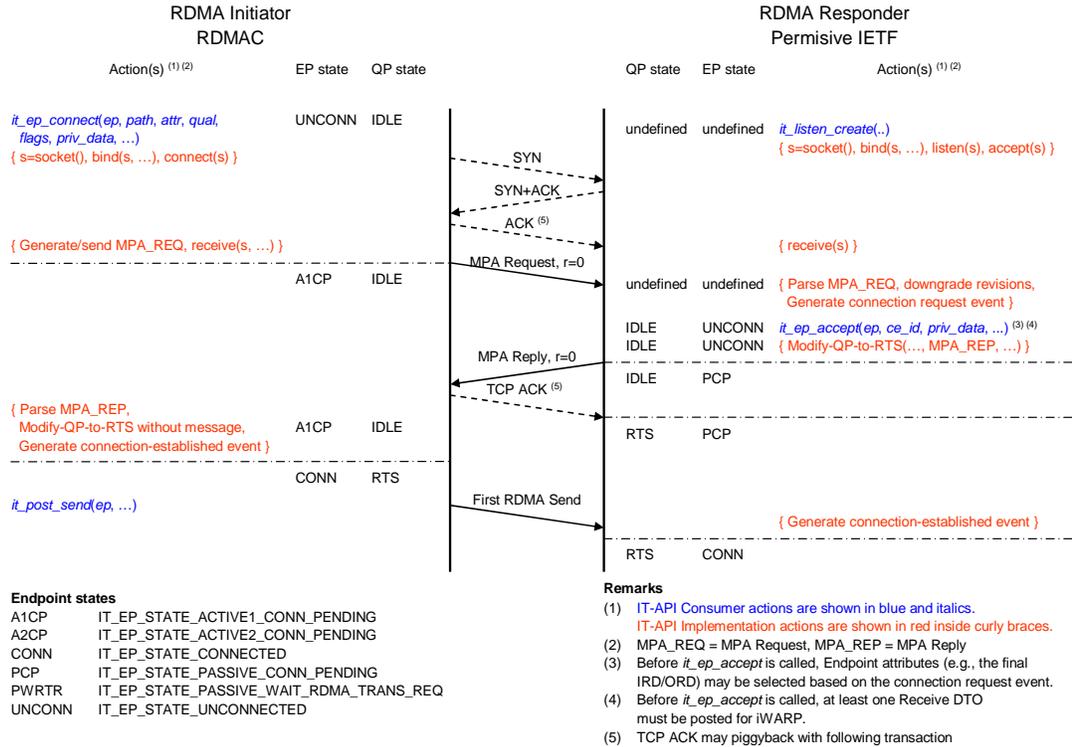
The TII of the IT-API fills this gap by specifying an *Immediate RDMA Transition* and thereby defines a (minimal) ULP with the following properties:

12285
12286
12287

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.

12288

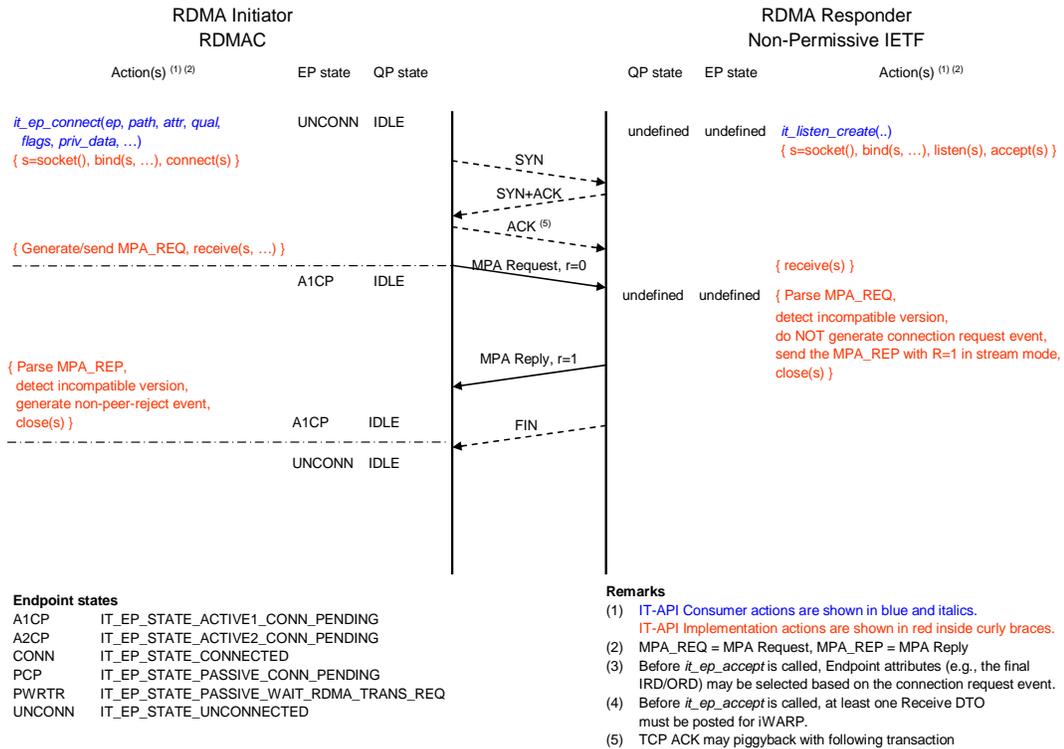
TII Interoperability Ladder Diagrams



12289

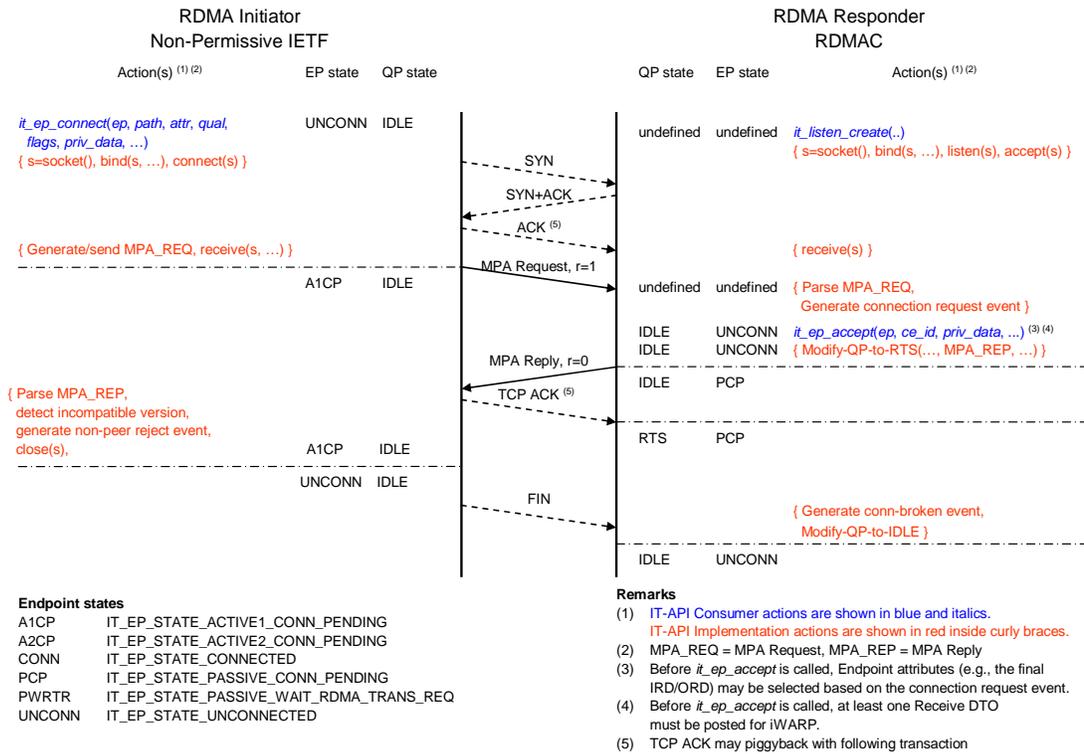
12290

Figure 16: RDMAC Initiator to Permissive IETF Responder



12293
12294

Figure 18: RDMAC Initiator to Non-Permissive IETF Responder



12295
12296

Figure 19: Non-Permissive IETF Initiator to RDMAC Responder

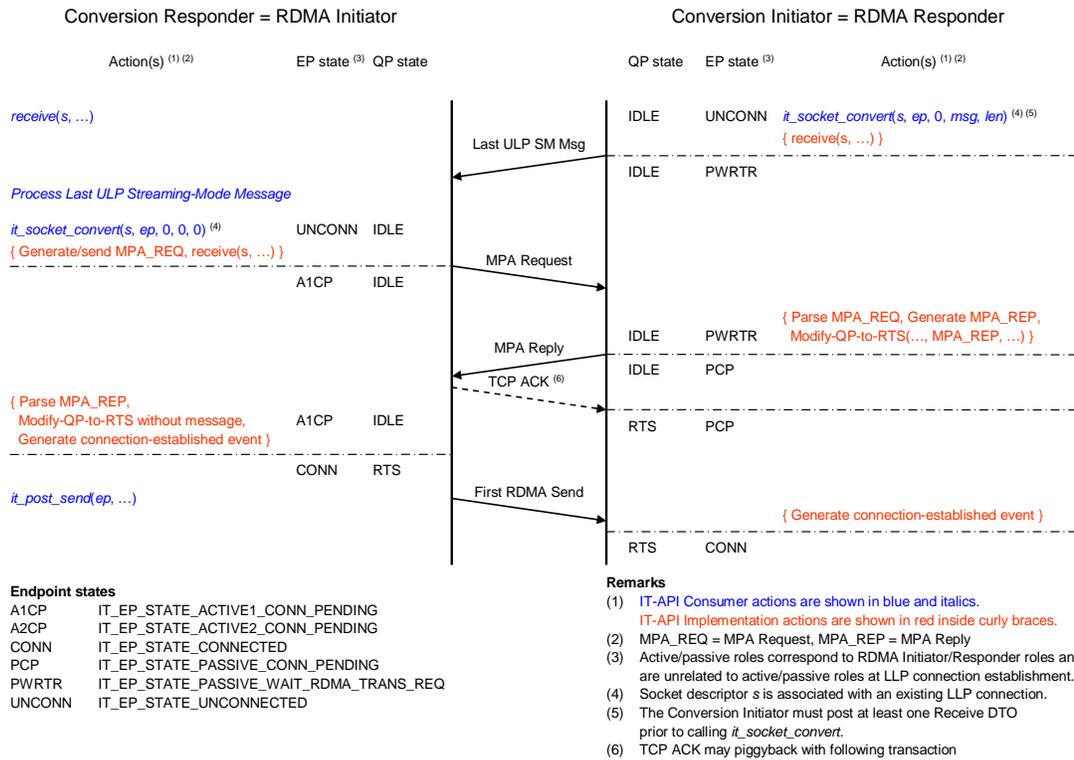
12297 **B.6 Transport-Dependent Interface**

12298 **Overview**

12299 For the *Transport-Dependent Interface (TDI)*, the Consumer calling *it_socket_convert* with a
 12300 *Last ULP Streaming-Mode (SM) Message* of length greater than zero is referred to as the
 12301 *Conversion Initiator*, while the Consumer calling *it_socket_convert* with a message length of
 12302 zero is referred to as the *Conversion Responder*.

12303 As for the TII, the Endpoint finite-state machine uses the active (passive) Endpoint states for the
 12304 RDMA Initiator (Responder). In order to avoid an inconsistency in the meaning of Endpoint
 12305 states between TII and TDI, the TDI uses the convention that the Conversion Initiator and the
 12306 corresponding side of the connection is in the *RDMA Responder* role, while the Conversion
 12307 Responder and the corresponding side of the connection is in the *RDMA Initiator* role.

12308 The Conversion process in case of an RNIC without RTR state is illustrated in Figure 20.



12309

12310

Figure 20: Conversion Process with MPA Startup (TDI) - RNIC without RTR State

12311

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.

12312

12313

12314

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.

12315

12316

12317

12318

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.

12319

12320

12321

12322

12323

12324

12325

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

12326

12327

12328

12329

12330
12331

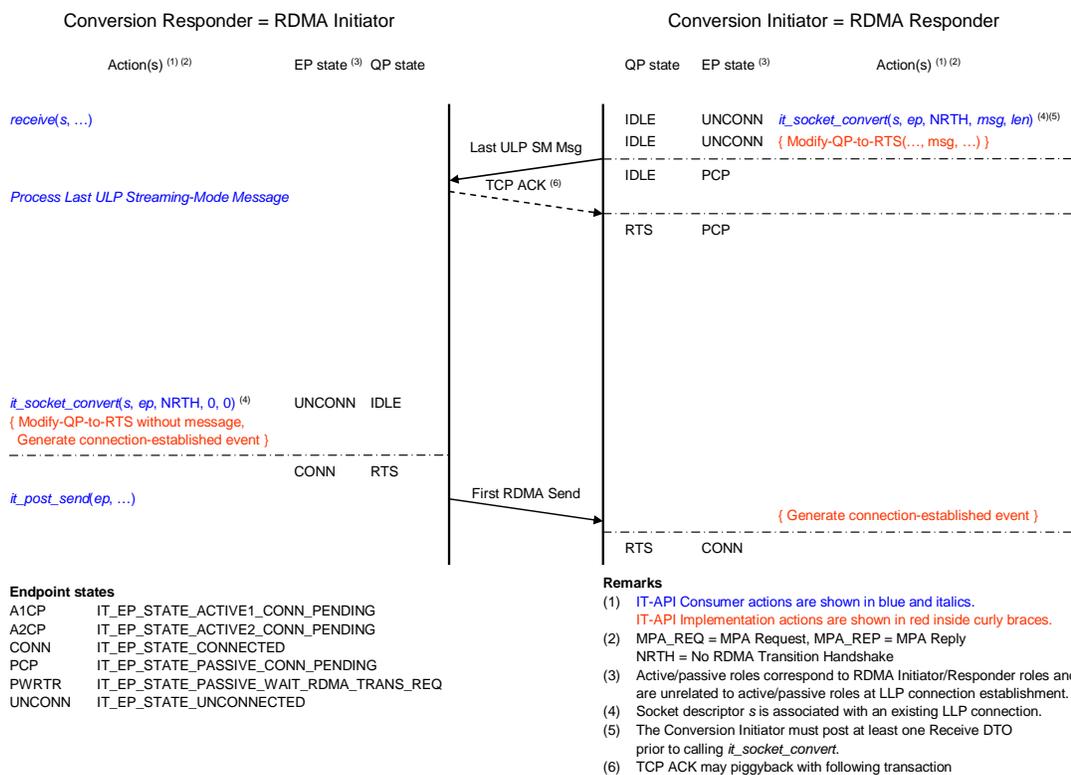
Endpoint into IT_EP_STATE_CONNECTED state. The IRD/ORD values shall appear as zero in the Event and the *private_data_present* flag shall be in the cleared state.

12332
12333
12334
12335
12336
12337
12338
12339
12340
12341
12342

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.

12343
12344
12345
12346

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.



12347
12348

Figure 21: Conversion Process with MPA Startup Suppression (TDI): RNIC without RTR State

TDI Interoperability Ladder Diagrams

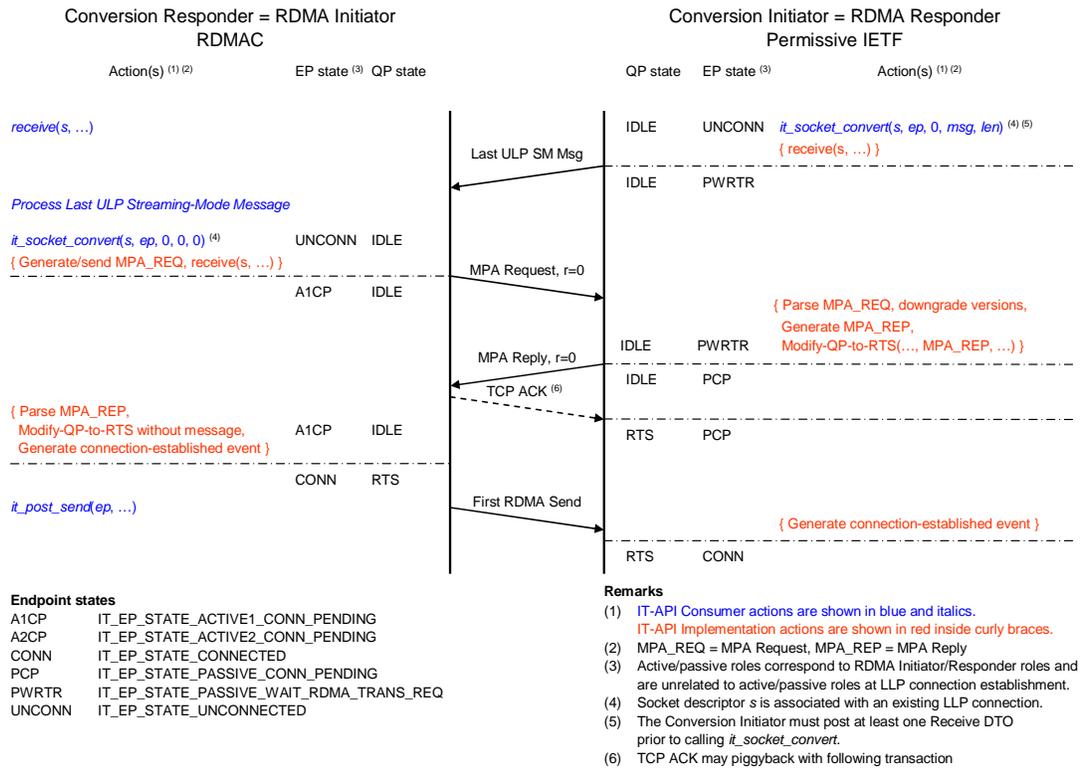
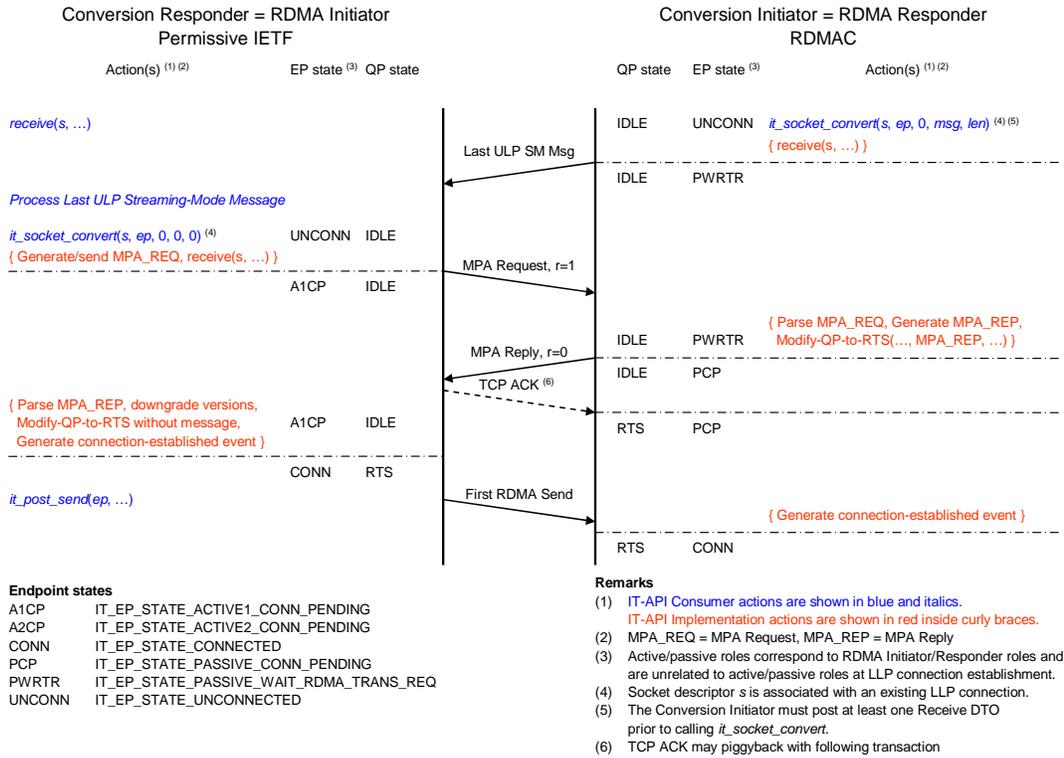


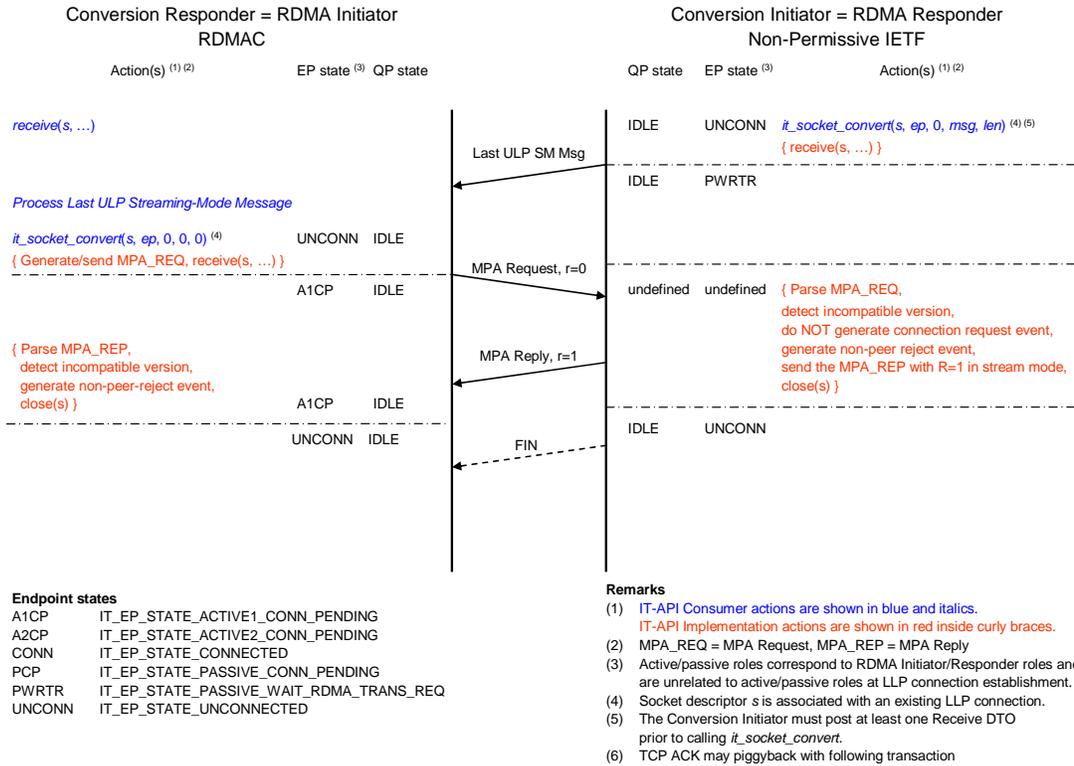
Figure 22: Permissive IETF Conversion Initiator to RDMAC Conversion Responder



12352

12353

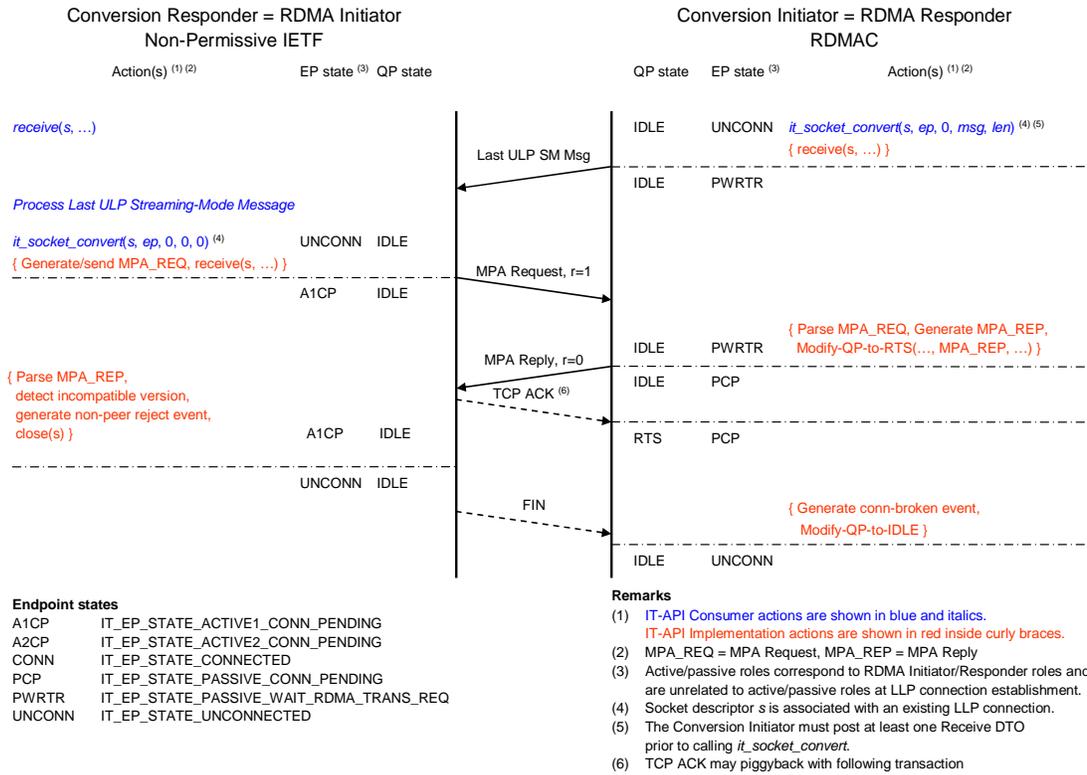
Figure 23: RDMAC Conversion Initiator to Permissive IETF Conversion Responder



12354

12355

Figure 24: Non-Permissive IETF Conversion Initiator to RDMAC Conversion Responder



12356
12357

Figure 25: RDMAC Conversion Initiator to Non-Permissive IETF Conversion Responder

12358

C Backwards Compatibility with Earlier IT-API Versions

12359
12360
12361
12362
12363
12364

Like IT-API Version 2.0, IT-API Version 2.1 is designed to minimize source-level incompatibility with IT-API Version 1.0 (as well as with Version 2.0). For several Version 1.0 calls, the functionality extensions for Version 2.1 required a modification of the function signature from Version 1.0 to Version 2.1; 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.

12365

IT-API Version 2.1 Implementation Requirements

12366
12367

A Version 2.1 Implementation with Version 1.0 backwards-compatibility must provide the following:

12368
12369

- A set of macros in the **<it_api.h>** header file as shown below for mapping the names of deprecated structure fields

12370
12371

- Additional enumeration entries in *it_mem_priv_t* and *it_rmr_param_mask_t* as identified below

12372
12373

- A set of macros in the **<it_api.h>** header file as shown below for mapping the names of modified functions

12374
12375
12376
12377

For each modified function *it_fun*, implementations of the Version 1.0 and the Version 2.1 functions are explicitly called *it_fun10* and *it_fun21*, respectively. The former can be a simple inline call to the latter, with default arguments as necessary. For *it_rmr_bind* (v1.0), Version 2.1 uses the new function name *it_rmr_link*; an *it_rmr_bind21* shall not be provided.

12378

Version 2.0 backwards-compatibility is also provided similarly.

12379

Application Requirements

12380
12381

The compile-time flag `ITAPI_ENABLE_V21_BINDINGS` (see **<it_api.h>**) controls the backwards-compatibility mode.

12382
12383
12384
12385
12386

A Consumer wishing to compile a program with Version 1.0 bindings need not change anything and does not define the `ITAPI_ENABLE_V21_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.

12387
12388
12389
12390
12391

A Consumer wishing to take advantage of Version 2.1 functionality, particularly in the area of memory management, must define the flag `ITAPI_ENABLE_V21_BINDINGS` before including **<it_api.h>**. In this case, modified functions must be used with the new Version 2.1 signatures. The **<it_api.h>** header file converts function names for modified functions to the *explicit* Version 2.1 function names *it_fun20* provided by the Implementation.

12392 In any case, it is not recommended to use explicit Version 1.0 function names directly, as the
12393 functionality may be incomplete for certain transports and explicit Version 1.0 function names
12394 may be removed in a future IT-API version. Similarly, it is not recommended to use explicit
12395 Version 2.1 function names directly, as these may be renamed in a future IT-API version by
12396 dropping the suffix “21”.

12397 A Consumer wishing to take advantage of Version 2.0 functionality, particularly in the area of
12398 memory management, must define the flag `ITAPI_ENABLE_V20_BINDINGS` before including
12399 `<it_api.h>`. The flag `ITAPI_ENABLE_V20_BINDINGS` should be used mutually exclusively
12400 with the flag `ITAPI_ENABLE_V21_BINDINGS`.

12401 **Backwards-Compatibility Support in Header File**

```
12402 /* default for IT-API is to maintain 1.0 bindings */
12403 #define ITAPI_ENABLE_V10_BINDINGS
12404 #if (defined(ITAPI_ENABLE_V20_BINDINGS) ||
12405     defined(ITAPI_ENABLE_V21_BINDINGS))
12406 #undef ITAPI_ENABLE_V10_BINDINGS
12407 #endif
12408
12409 /*
12410  * Macros restoring some IT-API 1.0 field names:
12411  */
12412
12413 /* define IT-API 1.0 variable name mappings for
12414  it_rc_only_attributes_t */
12415 #define rdma_read_inflight_incoming rdma_read_ird
12416 #define rdma_read_inflight_outgoing rdma_read_ord
12417
12418 /* define IT-API 1.0 variable name mappings for it_ia_info_t */
12419 #define ird_support ird_ord_ia_support
12420 #define ord_support ird_ord_ia_support
12421
12422 /* define IT-API 1.0 name mappings for it_make_rdma_addr */
12423 #define it_make_rdma_addr it_make_rdma_addr_absolute
12424
12425 /* define IT-API 1.0 name mappings for it_lmr_sync_* calls */
12426 #define it_lmr_sync_rdma_read it_lmr_flush_to_mem
12427 #define it_lmr_sync_rdma_write it_lmr_refresh_from_mem
12428
12429 /* define IT-API 1.0 name mapping for it_rmr_unbind call */
12430 #define it_rmr_unbind it_rmr_unlink
12431
12432 /* Need to use "bound" rather than "linked" in IT-API 1.0 */
12433 #ifdef ITAPI_ENABLE_V10_BINDINGS
12434
12435 typedef struct {
12436     it_ia_handle_t    ia;           /* IT_RMR_PARAM_IA */
12437     it_pz_handle_t    pz;           /* IT_RMR_PARAM_PZ */
12438     it_boolean_t      bound;        /* IT_RMR_PARAM_LINKED */
12439     it_lmr_handle_t    lmr;         /* IT_RMR_PARAM_LMR */
12440     void *             addr;         /* IT_RMR_PARAM_ADDR */
12441     it_length_t        length;      /* IT_RMR_PARAM_LENGTH */
12442 }
```

```

12442         it_mem_priv_t      privs;          /* IT_RMR_PARAM_MEM_PRIV */
12443         it_rmr_context_t   rmr_context; /* IT_RMR_PARAM_RMR_CONTEXT */
12444         it_rmr_type_t      type;          /* IT_RMR_PARAM_TYPE */
12445         it_addr_mode_t     addr_mode;     /* IT_RMR_PARAM_ADDR_MODE */
12446     } it_rmr_param_t;
12447
12448     #else /* #ifdef ITAPI_ENABLE_V10_BINDINGS */
12449
12450     typedef struct {
12451         it_ia_handle_t      ia;            /* IT_RMR_PARAM_IA */
12452         it_pz_handle_t     pz;            /* IT_RMR_PARAM_PZ */
12453         it_boolean_t       linked;       /* IT_RMR_PARAM_LINKED */
12454         it_lmr_handle_t    lmr;          /* IT_RMR_PARAM_LMR */
12455         void *              addr;         /* IT_RMR_PARAM_ADDR */
12456         it_length_t        length;       /* IT_RMR_PARAM_LENGTH */
12457         it_mem_priv_t      privs;        /* IT_RMR_PARAM_MEM_PRIV */
12458         it_rmr_context_t   rmr_context; /* IT_RMR_PARAM_RMR_CONTEXT */
12459         it_rmr_type_t      type;        /* IT_RMR_PARAM_TYPE */
12460         it_addr_mode_t     addr_mode;     /* IT_RMR_PARAM_ADDR_MODE */
12461     } it_rmr_param_t;
12462
12463     #endif /* #ifdef ITAPI_ENABLE_V10_BINDINGS */
12464
12465     /* prototypes */
12466
12467     #ifdef ITAPI_ENABLE_V10_BINDINGS
12468     /*
12469      * Backwards compatibility mode:
12470      * For functions whose signature changed from v1.0 to v2.0,
12471      * <it_api.h> converts v1.0 function names to explicit v1.0
12472      * function names. Functions such as it_lmr_create continue
12473      * to use v1.0 signatures.
12474      */
12475     #define it_lmr_create   it_lmr_create10
12476     #define it_rmr_create   it_rmr_create10
12477     #define it_rmr_bind     it_rmr_bind10
12478
12479     #elif (defined(ITAPI_ENABLE_V20_BINDINGS))
12480
12481     /*
12482      * Full v2.0 functionality:
12483      * For functions whose signature changed from v1.0 to v2.0,
12484      * <it_api.h> converts v1.0 function names to explicit v2.0
12485      * function names. Functions such as it_lmr_create use the
12486      * new v2.0 signatures.
12487      */
12488     #define it_lmr_create   it_lmr_create20
12489     #define it_rmr_create   it_rmr_create20
12490     #define it_rmr_bind     it_rmr_link
12491
12492     #elif (defined(ITAPI_ENABLE_V21_BINDINGS))
12493
12494     /*
12495      * Full v2.1 functionality:

```

```
12496         For functions whose signature changed relative to an earlier
12497         Version, <it_api.h> converts the function name to an explicit
12498         function name carrying as a suffix the IT-API major and minor
12499         version number in which the signature changed most recently.
12500     */
12501     #define it_lmr_create    it_lmr_create21
12502     #define it_rmr_create    it_rmr_create20
12503     #define it_rmr_bind      it_rmr_link
12504
12505     #endif      /* ifdef ITAPI_ENABLE_V10_BINDINGS */
12506
```

12507 **D** **Functional Changes (IT-API Version 2.1 Relative to IT-**
12508 **API Version 2.0)**

12509 This Appendix contains a list of all *functional changes* for every reference page in IT-API
12510 Version 2.0 and for every new reference page in IT-API Version 2.1, including:

- 12511
- Changes in call signatures
 - 12512 • New functionality in previously existing calls
 - 12513 • New calls
 - 12514 • Changes in data structures
 - 12515 • New data structures
 - 12516 • Changes and clarifications in the programming model such as:
 - 12517 — Additional or modified restrictions and rules
 - 12518 — Clarified restrictions and rules that were previously ambiguous
 - 12519 • Errata from Version 2.0 that have been resolved

12520 **Connection Management for iWARP (Section 5.3)**

12521 Erroneous text indicating use of MPA Reject for Conversion has been removed resolving
12522 Erratum “Bug 180”.

12523 **it_dto_events**

12524 A new field, representing a Memory Region handle that has been invalidated, has been added to
12525 both the DTO completion event structures.

12526 **it_dto_flags**

12527 New flags have been added:

- 12528
- IT_UNLINK_FENCE_FLAG
 - 12529 • IT_UNLINK_LOCAL_SINK
 - 12530 • IT_COALESCE_WR_FLAG

- 12531 **it_ep_attributes_t**
- 12532 A new constant, IT_EP_PARAM_ATOMICS_ENABLE, has been added to the
12533 *it_ep_param_mask_t* enumeration, giving the Consumer the option to allow incoming Atomic
12534 operations on the Endpoint.
- 12535 A new constant, IT_EP_PARAM_PRIV_OPS_ENABLE, has been added to the
12536 *it_ep_param_mask_t* enumeration, giving the Consumer the option to create a Privileged Mode
12537 Endpoint.
- 12538 **it_ep_ud_create**
- 12539 A typo in the NAME section has been fixed, changing “*ep_ud_create*” to “*it_ep_ud_create*” in
12540 IT-API Version 2.1.
- 12541 **it_evd_callback_attach**
- 12542 This is a new routine added for IT-API Version 2.1.
- 12543 **it_evd_callback_detach**
- 12544 This is a new routine added for IT-API Version 2.1.
- 12545 **it_evd_create**
- 12546 A typo in the SYNOPSIS section has been fixed, changing “*evd_create*” to “*it_evd_create*” in
12547 IT-API Version 2.1.
- 12548 **it_ia_info_t**
- 12549 Numerous new booleans have been added in IT-API Version 2.1:
- 12550
 - *atomic_support*
- 12551
 - *lmr_link_support*
- 12552
 - *direct_lmr_handle_support*
- 12553
 - *post_send_unlink_remote_support*
- 12554
 - *narrow_rmr_unlink_support*
- 12555
 - *wide_rmr_unlink_support*
- 12556
 - *unlink_fence_support*
- 12557 New data structures have been added:
- 12558
 - *direct_lmr_handle*
- 12559
 - *iobl_types_supported*

- 12560 **it_io_addr_t**
- 12561 This is a new data type added for IT-API Version 2.1.
- 12562 **it_iobl_t**
- 12563 This is a new data type added for IT-API Version 2.1.
- 12564 **it_lmr_create**
- 12565 A new formal parameter, *iobl*, has been added for IT-API Version 2.1 to allow Privileged
12566 Consumers to register memory using buffer lists.
- 12567 A new flag, IT_LMR_FLAG_NON_SHAREABLE, has been added. This, along with a change
12568 in *it_lmr_query*, resolves Erratum “Bug 166”.
- 12569 **it_lmr_create_unlinked**
- 12570 This is a new routine added for IT-API Version 2.1.
- 12571 **it_lmr_link**
- 12572 This is a new routine added for IT-API Version 2.1.
- 12573 New text describing the hazard of overflowing an EVD has been added to Application Usage
12574 resolving Erratum “Bug 168”.
- 12575 **it_lmr_query**
- 12576 The IT_LMR_FLAG_SHARED flag is returned as the set value if the LMR is in the shared state
12577 and cleared otherwise. It no longer merely reports the state of the flag at *it_lmr_create time*,
12578 rather it reports the actual sharing status of the LMR. This resolves Erratum “Bug 166”.
- 12579 New constants have been added to *it_lmr_param_mask_t*:
- 12580 • IT_LMR_PARAM_LINKED
- 12581 • IT_LMR_PARAM_IOBL_NUM_ELTS
- 12582 • IT_LMR_PARAM_IOBL_TYPE
- 12583 New fields have been added to *it_lmr_param_t*:
- 12584 • *linked*
- 12585 • *iobl_num_elts*
- 12586 • *iobl_type*

- 12587 **it_lmr_sync_rdma_read, it_lmr_sync_rdma_write**
- 12588 These have been renamed *it_lmr_flush_to_mem* and *it_lmr_refresh_from_mem* respectively,
12589 resolving Erratum “Bug 167”. A renaming macro has been added to <**it_api.h**>.
- 12590 **it_lmr_triplet_t**
- 12591 Added new field, *io*, of type *it_io_addr_t*.
- 12592 **it_lmr_unlink**
- 12593 This is a new routine added for IT-API Version 2.1.
- 12594 New text describing the hazard of overflowing an EVD has been added to Application Usage,
12595 resolving Erratum “Bug 168”.
- 12596 **it_mem_map**
- 12597 This is a new routine added for IT-API Version 2.1.
- 12598 **it_mem_priv_t**
- 12599 A new enable flag, *IT_PRIV_ATOMIC*, for Atomic operations is added.
- 12600 **it_mem_unmap**
- 12601 This is a new routine added for IT-API Version 2.1.
- 12602 **it_post_atomic**
- 12603 This is a new routine added for IT-API Version 2.1.
- 12604 New text describing the hazard of overflowing an EVD has been added to Application Usage,
12605 resolving Erratum “Bug 168”.
- 12606 **it_post_rdma_read**
- 12607 New text describing the hazard of overflowing an EVD has been added to Application Usage,
12608 resolving Erratum “Bug 168”.
- 12609 **it_post_rdma_read_to_rmr**
- 12610 New text describing the hazard of overflowing an EVD has been added to Application Usage,
12611 resolving Erratum “Bug 168”.

- 12612 **it_post_rdma_write**
- 12613 New text describing the hazard of overflowing an EVD has been added to Application Usage,
12614 resolving Erratum “Bug 168”.
- 12615 **it_post_rcv**
- 12616 New text describing the hazard of overflowing an EVD has been added to Application Usage,
12617 resolving Erratum “Bug 168”.
- 12618 **it_post_rcv_from**
- 12619 New text describing the hazard of overflowing an EVD has been added to Application Usage,
12620 resolving Erratum “Bug 168”.
- 12621 **it_post_send**
- 12622 New text describing the hazard of overflowing an EVD has been added to Application Usage,
12623 resolving Erratum “Bug 168”.
- 12624 **it_post_send_and_unlink**
- 12625 This is a new routine added for IT-API Version 2.1.
- 12626 New text describing the hazard of overflowing an EVD has been added to Application Usage,
12627 resolving Erratum “Bug 168”.
- 12628 **it_post_sendto**
- 12629 New text describing the hazard of overflowing an EVD has been added to Application Usage,
12630 resolving Erratum “Bug 168”.
- 12631 **it_rmr_link**
- 12632 The IT_ERR_RESOURCES immediate return has been deprecated.
- 12633 New text describing the hazard of overflowing an EVD has been added to Application Usage,
12634 resolving Erratum “Bug 168”.
- 12635 **it_rmr_unlink**
- 12636 New text describing the hazard of overflowing an EVD has been added to Application Usage,
12637 resolving Erratum “Bug 168”.

12638 **E** **Functional Changes (IT-API Version 2.0 Relative to IT-**
12639 **API Version 1.0)**

12640 This Appendix contains a list of all *functional changes* for every reference page in IT-API
12641 Version 1.0 and for every new reference page in IT-API Version 2.0, including:

- 12642 • Changes in call signatures
- 12643 • New functionality in previously existing calls
- 12644 • New calls
- 12645 • Changes in data structures
- 12646 • New data structures
- 12647 • Changes and clarifications in the programming model such as:
 - 12648 — Additional or modified restrictions and rules
 - 12649 — Clarified restrictions and rules that were previously ambiguous
- 12650 • Errata from Version 1.0 that have been resolved

12651 **it_addr_mode_t**

12652 This is a new data type added for IT-API Version 2.0.

12653 **it_address_handle_create**

12654 Flags now identified as bitwise rather than logical, resolving Erratum GN63=IBM1.

12655 **it_address_handle_free**

12656 Removed bad advice regarding free handles, resolving Erratum GN50=HP10.

12657 **it_address_handle_modify**

12658 Flags now identified as bitwise rather than logically ORed value, resolving Erratum
12659 GN63=IBM1.

12660 Replaced use of IT_ADDR_PATH with IT_ADDR_PARAM_PATH, resolving Erratum
12661 GN32=OG12.

12662 Replaced use of IT_ERR_INVALID_SLID and IT_ERR_INVALID_SGID with
12663 IT_ERR_INVALID_SOURCE_PATH, resolving Erratum GN13=SUN1. Also see *it_status_t*
12664 change listing for related change.

12665 **it_address_handle_query**

12666 Flags now identified as bitwise rather than logically ORed value, resolving Erratum
12667 GN63=IBM1.

12668 Added additional cross-reference text about relevant fields in *it_path_t*, resolving Erratum
12669 GN14=SUN2.

12670 Resolved GN21=SUN9 as defined in Global Behavior section *Output Parameters*.

12671 **it_affiliated_event_t**

12672 Added Affiliated Asynchronous Events for a finer discrimination of locally detected errors.

12673 Added Affiliated Asynchronous Events for a finer discrimination of remotely detected errors,
12674 supporting interpretation of iWARP Terminate messages.

12675 Added Affiliated Asynchronous Events supporting the use of S-RQs.

12676 Provided mappings between iWARP Affiliated Asynchronous Events and IT-API Affiliated
12677 Asynchronous Events.

12678 **it_cm_msg_events**

12679 Added new reject reason code for iWARP (IT_CN_REJ_BAD_CONN_PARMS).

12680 Added text describing manifestation of capability to disable IRD/ORD.

12681 Replaced broken cross-references, resolving Erratum GN60=HP17.

12682 Added text clarifying that Events apply both to the TII (two-way and three-way) as well as the
12683 TDI.

12684 Revised text describing non-peer reject Events to reflect that they can apply both to remotely
12685 detected and locally detected events, and also that a possible cause of a non-peer reject Event is a
12686 non-interoperable RNIC combination.

12687 Removed text specifying that IRD/ORD values in an accept arrival Event are valid only on the
12688 active side in three-way Connection establishment – they are now valid also on the passive side.

12689 Added text specifying that IRD/ORD values are valid in the connection established Event and
12690 may differ from the actual IRD/ORD in use on the connection under certain circumstances.

12691 Added table specifying the mapping of reject reason codes on the iWARP Transport.

12692 Added text defining the lack of ordering relationship between
12693 IT_CM_MSG_EVENT_STREAM Events and IT_DTO_EVENT_STREAM Events, resolving
12694 Erratum GN62=HP19.

12695 **it_cm_req_events**

12696 Fixed typo, resolving Erratum GN38=OG18.

12697 **it_convert_net_addr**

12698 Added text to the IT_ERR_INVALID_CONVERSION error description stating this can also be
12699 returned when the Implementation cannot convert the particular source address that was input,
12700 resolving Erratum GN22=SUN10.

12701 Clarified the text of IT_ERR_INVALID_NETADDR to mean that the type of the Network
12702 Address specified in *source_addr* was not recognized (as opposed to
12703 IT_ERR_INVALID_ADDRESS which means the address was a valid type but an invalid value),
12704 resolving Erratum GN23=SUN11.

12705 Text added to reflect that this call may block.

12706 **it_conn_qual_t**

12707 Removed assigned enumerated values to *it_conn_qual_type_t*. IT-API Version 1.0 specified
12708 these as bit values but they are all mutually-exclusive.

12709 Added a new Connection Qualifier type for iWARP that allows specification of local and remote
12710 IANA ports.

12711 **it_dto_events**

12712 Modified the definition for the *ep* member of *it_dto_cmpl_event_t* so that for Receive
12713 completions it refers to the Endpoint on which the Receive completed, rather than to the
12714 Endpoint on which the Receive was posted. The only time this is different from the IT-API
12715 Version 1.0 behavior is when the Receive operation is posted to an S-RQ rather than to an
12716 Endpoint.

12717 Typos fixed resolving Erratum GN37=OG17 and GN40=OG20.

12718 Added text defining the lack of ordering relationship between
12719 IT_CM_MSG_EVENT_STREAM Events and IT_DTO_EVENT_STREAM Events, resolving
12720 Erratum GN62=HP19.

12721 **it_dto_flags_t**

12722 Moved a sentence in the IT_COMPLETION_FLAG description to instead be in the
12723 IT_NOTIFY_FLAG description. This was not a functional change; it was done only to enhance
12724 readability. This resolves Erratum GN20=SUN8.

- 12725 Fixed typo. IT_SOLICITED_WAIT_FLAG was changed to IT_SOLICITED_WAIT_FLAG.
12726 This resolves Erratum GN36=OG16.
- 12727 **it_dto_status_t**
- 12728 Added a new DTO status for memory management operation errors.
- 12729 Added additional causes for most of the existing DTO status values. A DTO status other than
12730 IT_DTO_SUCCSS represents a category of Completion Errors.
- 12731 Provided mappings between iWARP Completion Errors and IT-API DTO status values.
- 12732 **it_ep_accept**
- 12733 Added text describing use in two-way and three-way Connection Establishment, resolving
12734 Erratum GN6 = HP5.
- 12735 Removed text erroneously indicating that Private Data delivery is always unreliable, clarifying a
12736 corner case.
- 12737 Asynchronous Errors:
- 12738 • Specified how lack of interoperability between RDMAC RNIC and Non-permissive IETF
12739 RNIC is manifested.
- 12740 **it_ep_attributes_t**
- 12741 Added the following RC attributes for Shared Receive Queue (S-RQ) support:
- 12742 • *srq*
 - 12743 • *soft_hi_watermark*
 - 12744 • *hard_hi_watermark*
- 12745 Modified description of *max_recv_dtos* and *max_recv_segments* attributes for S-RQ support.
- 12746 Modified description of *rdma_read_ord* to reflect new capability of IT-API Version 2.0 to allow
12747 Consumer to change ORD after Endpoint is connected.
- 12748 **it_ep_connect**
- 12749 Replaced erroneous description implying that Private Data is delivered in an
12750 IT_CM_MSG_CONN_PEER_REJECT_EVENT and corrected stating that it appears in the
12751 IT_CM_REQ_CONN_REQUEST_EVENT, resolving Erratum GN7=HP6.
- 12752 Removed text erroneously indicating that Private Data delivery is unreliable, resolving Erratum
12753 GN8=HP7.
- 12754 Added text explaining that the connected event occurs both on Active and Passive sides,
12755 resolving Erratum GN9=HP8.

- 12756 Added dummy entries to attributes structures to aid in ANSI compilation.
- 12757 Added IT_CONNECT_SUPPRESS_IRD_ORD flags to *it_cn_est_flags_t*. This flag allows the
12758 Consumer to suppress use of IRD/ORD negotiation by the Implementation.
- 12759 Flags now identified as bitwise rather than logically ORed value, resolving Erratum
12760 GN63=IBM1.
- 12761 Clarified the mutual exclusivity of two-way *versus* three-way connection flags.
- 12762 Asynchronous Errors:
- 12763 • Specified how lack of interoperability between RDMAC RNIC and Non-permissive IETF
12764 RNIC is manifested.
- 12765 **it_ep_disconnect**
- 12766 Added text clarifying the relationship between Private Data delivery and generation of
12767 disconnect events.
- 12768 Fixed bad grammar, resolving Erratum GN61=HP18.
- 12769 **it_ep_free**
- 12770 Clarified that when *it_ep_free* returns, the handle can no longer be used; the previous statement
12771 was that an error was guaranteed to be returned if an attempt was made to use a handle that had
12772 been freed, which is not necessarily true. This resolves Erratum GN52=HP12.
- 12773 Reorganized and expanded the paragraph in the Application Usage section that talks about
12774 marker operations to make it clearer how they are used. This resolves Erratum GN54=HP14.
- 12775 Clarified that the Events that can be lost from the EVDs associated with the Endpoint are only
12776 those Events that pertain to the Endpoint. This resolves Erratum GN55=HP15.
- 12777 **it_ep_modify**
- 12778 Added the Endpoint Hard High Watermark and the Endpoint Soft High Watermark to the set of
12779 Endpoint parameters that can be modified. These new parameters are only used if the Endpoint
12780 is associated with a Shared Receive Queue.
- 12781 Made it illegal to use this routine to modify the total number of entries in the Receive Queue if
12782 the Endpoint is associated with a Shared Receive Queue.
- 12783 **it_ep_query**
- 12784 Reworded description for the *spigot_id* field to help clarify that this field is not valid if the
12785 Endpoint is in the IT_EP_STATE_UNCONNECTED state. This resolves Erratum GN25=OG5.

- 12786 Reworded description for the *dst_path* field to help clarify that this field is not valid if the
12787 Endpoint is in the IT_EP_STATE_UNCONNECTED state, and that it is not valid if this
12788 Endpoint is of the UD Service Type. This resolves Erratum GN26=OG6.
- 12789 Resolved GN21=SUN9 as defined in Global Behavior section *Output Parameters*.
- 12790 **it_ep_rc_create**
- 12791 Added the IT_EP_SRQ bit to the set of flags passed to this routine to allow the Consumer to
12792 specify that they want to associate a Shared Receive Queue with the Endpoint that they are
12793 creating.
- 12794 **it_ep_reset**
- 12795 No functional changes.
- 12796 **it_ep_state_t**
- 12797 Added the IT_EP_STATE_PASSIVE_WAIT_RDMA_TRANS_REQ state. This new state is
12798 used by the TDI (*it_socket_convert*).
- 12799 Removed text that erroneously implies Private Data delivery is always unreliable.
- 12800 **it_ep_ud_create**
- 12801 Fixed typo in description text. IT_EP_STATE_OPERATIONAL was changed to
12802 IT_EP_STATE_UD_OPERATIONAL. This resolves Erratum GN33=OG13.
- 12803 Fixed typo resolving Erratum GN40=OG20.
- 12804 **it_evd_create**
- 12805 Flags now identified as bitwise rather than logical, resolving Erratum GN63=IBM1.
- 12806 IT_SOLICITED_WAIT changed to IT_SOLICITED_WAIT_FLAG, resolving Erratum
12807 GN35=OG15.
- 12808 Invalid threshold value defined (must be strictly less than or equal to the actual Implementation-
12809 allocated queue size), resolving Erratum GN27=OG7.
- 12810 Revised text describing unblocking, resolving Erratum GN57=SUN16.
- 12811 Numerous grammatical fixes applied to text.
- 12812 **it_evd_dequeue**
- 12813 Typo fixed resolving Erratum GN28=OG8.

- 12814 **it_evd_free**
- 12815 Removed bad advice regarding freed handles, resolving Erratum GN53=HP13.
- 12816 **it_evd_modify**
- 12817 Clarified text regarding conditions under which the AEVD associated with an SEVD may be
12818 modified.
- 12819 Flags now identified as bitwise rather than logical, resolving Erratum GN63=IBM1.
- 12820 IT_EVD_CREAT_FD changed to IT_EVD_CREATE_FD, resolving Erratum GN34=OG14.
- 12821 **it_evd_post_se**
- 12822 No functional changes.
- 12823 **it_evd_wait**
- 12824 Corrected grammar, resolving Erratum GN29=OG9.
- 12825 Text added to reflect that this call may block.
- 12826 **it_event_t**
- 12827 The Event Stream for RMR Link completions is now called IT_RMR_LINK_CMPL_EVENT.
12828 The old name IT_RMR_BIND_CMPL_EVENT remains valid for source backwards-
12829 compatibility.
- 12830 Added Event Streams for the new Affiliated Asynchronous Events defined in
12831 *it_affiliated_event_t*.
- 12832 Removed (and marked deprecated) IT_ASYNC_UNAFF_IA_CATASTROPHIC_ERROR
12833 resolving Erratum GN4=HP4.
- 12834 **it_get_consumer_context**
- 12835 Added text documenting that an error is returned if the Consumer Context was never set,
12836 resolving Erratum GN15=SUN3.
- 12837 **it_get_pathinfo**
- 12838 Added IT_ERR_INTERRUPT as a return value, resolving Erratum GN1=HP1.
- 12839 Clarified the text of IT_ERR_INVALID_NETADDR to mean the type of *it_net_addr_t* passed
12840 in is not recognized.
- 12841 Added IT_ERR_INVALID_ADDRESS to mean the address was of a legal type but was invalid.
- 12842 Text added to reflect that this call may block.

12843	it_handle_t
12844	No functional changes.
12845	it_handoff
12846	Typos fixed resolving Erratum GN10=OG2 and GN11=OG3.
12847	Clarifying text added resolving Erratum GN56=HP16.
12848	it_ia_create
12849	Documented that the second release of the IT-API shall have a major version number of 2 and a minor version number of 0.
12850	
12851	Typo fixed resolving Erratum GN5=OG1.
12852	it_ia_free
12853	No functional changes.
12854	it_ia_info_t
12855	Added iWARP Transport type to <i>it_transport_type_t</i> .
12856	Added an iWARP vendor data structure definition.
12857	Added booleans indicating support or not for features as follows:
12858	• S-RQ
12859	• Hard high watermark (S-RQ)
12860	• Soft high watermark (S-RQ)
12861	• Resizable S-RQ
12862	• Extended iWARP state machine
12863	• Support for socket conversion (TDI)
12864	• IRD modifiable
12865	• ORD increasable
12866	• DTO Overflow detection
12867	• RDMA Read local extensions

- 12868 **it_listen_create**
- 12869 Flags now identified as bitwise rather than logically ORed value, resolving Erratum
12870 GN63=IBM1.
- 12871 Added functional capability to suppress use of IRD/ORD.
- 12872 **it_listen_free**
- 12873 Added text constraining use of *listen_handle* after call returns, resolving Erratum GN51 (HP11).
- 12874 **it_lmr_create**
- 12875 Added input modifier to select addressing mode (absolute addressing or relative addressing).
- 12876 *privs* and *flags* now identified as bitwise rather than logically ORed values, resolving Erratum
12877 GN63=IBM1.
- 12878 *it_mem_priv_t* is now specified on a separate reference page.
- 12879 Use of 0 for *privs* is no longer permitted.
- 12880 Clarified restrictions on and use of access privileges, resolving Erratum GN49=SUN15.
- 12881 Deprecated the use of IT_PRIV_DEFAULT (value remains in header file only), resolving
12882 Erratum GN3=HP3.
- 12883 Clarified the semantics of IT_LMR_FLAG_SHARED, resolving Erratum GN16=SUN4.
- 12884 Clarified potential issue with stale RMR Contexts, resolving Erratum GN46=SUN12.
- 12885 New return value IT_ERR_INVALID_ADDR_MODE.
- 12886 **it_lmr_free**
- 12887 Specified that any RMR Context associated with the LMR may no longer be used after freeing
12888 an LMR, resolving Erratum GN46=SUN12.
- 12889 Added missing statement that a failing call due to linked RMR(s) will neither affect an
12890 associated RMR Context, nor any linked RMR, resolving Erratum GN47=SUN13.
- 12891 While not a functional change, the discussion on stale RMR Contexts was moved to the
12892 Application Usage section, resolving Erratum GN18=SUN6.
- 12893 **it_lmr_modify**
- 12894 Added input modifier to select addressing mode (absolute addressing or relative addressing).
- 12895 *mask* now identified as bitwise rather than logically ORed value, resolving Erratum
12896 GN63=IBM1.

- 12897 Clarified potential issue with stale RMR Contexts, resolving Erratum GN46=SUN12.
- 12898 Clarified in which cases an LMR remains unaffected when the operation fails and that the
12899 operation fails if the LMR has any RMRs linked to it, resolving Erratum GN48=SUN14.
- 12900 **it_lmr_query**
- 12901 *mask* now identified as bitwise rather than logically ORed value, resolving Erratum
12902 GN63=IBM1.
- 12903 Added addressing mode (*addr_mode*) LMR attribute.
- 12904 Specified in which cases the LMR must have byte-level granularity. In particular,
12905 Implementations supporting Relative Addressing must provide byte-level granularity.
- 12906 Resolved GN21=SUN9 as defined in Global Behavior section *Output Parameters*.
- 12907 **it_lmr_flush_to_mem**
- 12908 Added statement that when both normal and non-coherent LMRs are input to this routine that
12909 only the non-coherent segments are affected, resolving Erratum GN17=SUN5.
- 12910 **it_lmr_refresh_from_mem**
- 12911 Added statement that when both normal and non-coherent LMRs are input to this routine that
12912 only the non-coherent segments are affected, resolving Erratum GN17=SUN5.
- 12913 **it_lmr_triplet_t**
- 12914 The *addr* member of LMR Triplets must now be interpreted depending on the addressing mode
12915 (Absolute Addressing or Relative Addressing) of the LMR.
- 12916 Additionally, the *addr* member has been changed from a simple void pointer to a union of
12917 absolute (void pointer) and relative (*it_length_t*) elements.
- 12918 **it_make_rdma_addr**
- 12919 The IT-API 1.0 routine *it_make_rdma_addr* has been replaced with two new routines,
12920 *it_make_rdma_addr_absolute* and *it_make_rdma_addr_relative*. A backward-compatibility
12921 macro is recommended to define *it_make_rdma_addr* as *it_make_rdma_addr_absolute*.
- 12922 **it_mem_priv_t**
- 12923 This is a new reference page for a data type that was specified under *it_lmr_create* in IT-API
12924 Version 1.0.
- 12925 *privs* now identified as bitwise rather than logically ORed value, resolving Erratum
12926 GN63=IBM1.

- 12927 Use of 0 for *privs* is no longer permitted.
- 12928 Clarified restrictions on and use of access privileges, resolving Erratum GN49=SUN15.
- 12929 Deprecated the use of IT_PRIV_DEFAULT (value remains in header file only), resolving
12930 Erratum GN3=HP3.
- 12931 **it_path_t**
- 12932 Removed erroneous assertion that *it_ib_net_endpoint* is the remote component of an InfiniBand
12933 path.
- 12934 Added Path data structures for the iWARP Transport.
- 12935 **it_post_rdma_read**
- 12936 The starting address of the remote source buffer, *rdma_addr*, must be chosen according to the
12937 addressing mode of the remote buffer.
- 12938 Specified required access privileges for remote source buffer.
- 12939 Specified required access privileges for local sink buffer, which are transport-dependent.
- 12940 Clarified that Consumer retains ownership of the array specified by *local_segments* and
12941 *num_segments*, resolving Erratum GN31=OG11.
- 12942 Added ordering rules for subsequent DTOs targeting overlapping sections of local sink buffers.
- 12943 Added ordering rule for Work Requests posted after RMR Link/Unlink, resolving Erratum
12944 GN58=SUN17.
- 12945 Asynchronous Errors:
- 12946 • Described scenarios causing an access violation or transport error at the remote/local
12947 Endpoint.
 - 12948 • Specified that zero-sized data transfers never cause an access violation.
 - 12949 • Specified how remotely detected errors will manifest remotely and locally.
 - 12950 • Specified how locally detected errors will manifest.
- 12951 **it_post_rdma_write**
- 12952 The starting address of the remote sink buffer, *rdma_addr*, must be chosen according to the
12953 addressing mode of the remote buffer.
- 12954 Specified required access privileges for source and sink buffer.
- 12955 Clarified that the Consumer retains ownership of the array specified by *local_segments* and
12956 *num_segments*, resolving Erratum GN31=OG11.

- 12957 Clarified the lack of end-to-end completions.
- 12958 Clarified the difference between operation ordering and local/remote completion ordering.
- 12959 Added ordering rules for subsequent DTOs targeting overlapping sections of remote sink
12960 buffers.
- 12961 Added ordering rule for Work Requests posted after RMR Link/Unlink, resolving Erratum
12962 GN58=SUN17.
- 12963 Asynchronous Errors:
- 12964 • Described scenarios causing an access violation or transport error at the remote Endpoint.
 - 12965 • Specified that zero-sized data transfers never cause an access violation.
 - 12966 • Change in how remotely detected errors will manifest remotely and locally (transport-
12967 dependent).
- 12968 **it_post_rcv**
- 12969 Receive DTOs can now be posted to an Endpoint or Shared Receive Queue.
- 12970 Specified required access privileges for the sink buffer.
- 12971 Clarified that the Consumer retains ownership of the array specified by *local_segments* and
12972 *num_segments*, resolving Erratum GN31=OG11.
- 12973 Added ordering rules for subsequent incoming Send DTOs matching with Receive DTOs whose
12974 Receive buffers overlap.
- 12975 Asynchronous Errors:
- 12976 • Described scenarios causing an access violation, length error, or transport error at the local
12977 Endpoint.
 - 12978 • Change in how locally detected errors will manifest locally and remotely (transport-
12979 dependent).
- 12980 Clarified that a Receive DTO can be posted to an EP prior to reaching the
12981 IT_EP_STATE_CONNECTED state.
- 12982 **it_post_rcvfrom**
- 12983 Specified required access privileges for the sink buffer.
- 12984 Clarified that the Consumer retains ownership of the array specified by *local_segments* and
12985 *num_segments*, resolving Erratum GN31=OG11.
- 12986 Asynchronous Errors:
- 12987 • Described scenarios causing an access violation or length error at the local Endpoint.

12988	it_post_send
12989	Specified required access privileges for the source buffer.
12990	Clarified that the Consumer retains ownership of the array specified by <i>local_segments</i> and
12991	<i>num_segments</i> , resolving Erratum GN31=OG11.
12992	Clarified the lack of end-to-end completions.
12993	Clarified the difference between operation ordering and local/remote completion ordering.
12994	Added ordering rules for subsequent Send DTOs matching with Receive DTOs whose Receive
12995	buffers overlap.
12996	Added ordering rule for Work Requests posted after RMR Link/Unlink, resolving Erratum
12997	GN58=SUN17.
12998	Asynchronous Errors:
12999	<ul style="list-style-type: none"> • Described scenarios causing an access violation, length error, or transport error at the
13000	remote Endpoint.
13001	<ul style="list-style-type: none"> • Change in how remotely detected errors will manifest remotely and locally (transport-
13002	dependent).
13003	it_post_sendto
13004	Specified required access privileges for the source buffer.
13005	Clarified that the Consumer retains ownership of the array specified by <i>local_segments</i> and
13006	<i>num_segments</i> , resolving Erratum GN31=OG11.
13007	Asynchronous Errors:
13008	<ul style="list-style-type: none"> • Described scenarios causing an access violation or length error at the remote Endpoint.
13009	it_pz_query
13010	Flags now identified as bitwise rather than logical, resolving Erratum GN63=IBM1.
13011	Resolved GN21=SUN9 as defined in Global Behavior section <i>Output Parameters</i> .
13012	it_reject
13013	Removed text that erroneously implies Private Data delivery is always unreliable and replaced it
13014	with a clarification, resolving Erratum GN12=HP9.
13015	it_rmr_bind
13016	This routine was replaced by <i>it_rmr_link</i> . See also Appendix C.

- 13017 **it_rmr_create**
- 13018 Added input modifier to select RMR type (Narrow RMR or Wide RMR).
- 13019 New return value IT_ERR_INVALID_RMR_TYPE.
- 13020 **it_rmr_free**
- 13021 No functional changes.
- 13022 **it_rmr_link**
- 13023 This routine replaces *it_rmr_bind* of IT-API Version 1.0. See also Appendix C.
- 13024 Added input modifier to select addressing mode (absolute addressing or relative addressing).
- 13025 *privs* and *dto_flags* now identified as bitwise rather than logically ORed values, resolving
13026 Erratum GN63=IBM1.
- 13027 Specified additional restrictions for linking a Narrow RMR.
- 13028 Clarified that any local access privileges specified in *privs* are ignored.
- 13029 Added ordering rule for Work Requests posted after RMR Link, resolving Erratum
13030 GN58=SUN17.
- 13031 New return value IT_ERR_INVALID_ADDR_MODE.
- 13032 Detailed description of Asynchronous Errors.
- 13033 Deprecated the use of 0 for *privs* and recommended using *it_rmr_unlink* instead, resolving
13034 Erratum GN19=SUN7.
- 13035 Erratum GN2=HP2 (reference to a inexistant IT.DTO_DEFAULT_FLAG) is resolved – in fact
13036 it was already resolved in Version 1.0.
- 13037 Clarified ways to infer completion of an RMR Link operation, resolving Erratum
13038 GN59=SUN18.
- 13039 Clarified potential issue with stale RMR Contexts.
- 13040 **it_rmr_query**
- 13041 *mask* now identified as bitwise rather than logically ORed value, resolving Erratum
13042 GN63=IBM1.
- 13043 Added RMR type (*type*) RMR attribute.
- 13044 Added addressing mode (*addr_mode*) RMR attribute.
- 13045 Replaced the RMR attribute *bound* by *linked*. If ITAPI_ENABLE_V20_BINDINGS is not
13046 defined, the old attribute name remains valid. See also Appendix C.

- 13047 Resolved GN21=SUN9 as defined in Global Behavior section *Output Parameters*.
- 13048 **it_rmr_triplet_t**
- 13049 This is a new data type added for IT-API Version 2.0.
- 13050 **it_rmr_type_t**
- 13051 This is a new data type added for IT-API Version 2.0.
- 13052 **it_rmr_unbind**
- 13053 This routine was replaced by *it_rmr_unlink*. See also Appendix C.
- 13054 **it_rmr_unlink**
- 13055 This routine replaces *it_rmr_unbind* of IT-API Version 1.0. See also Appendix C.
- 13056 *dto_flags* now identified as bitwise rather than logically ORed value, resolving Erratum
13057 GN63=IBM1.
- 13058 Specified additional restrictions for unlinking a Narrow RMR.
- 13059 Added ordering rule for Work Requests posted after RMR Unlink, resolving Erratum
13060 GN58=SUN17.
- 13061 Detailed description of Asynchronous Errors.
- 13062 Improved guidance for dealing with a linked RMR after a disconnect.
- 13063 Clarified ways to infer completion of an RMR Unlink operation, resolving Erratum
13064 GN59=SUN18.
- 13065 Eliminated typo, resolving Erratum GN39=OG19.
- 13066 **it_socket_convert**
- 13067 This is a new routine added for IT-API Version 2.0.
- 13068 **it_srq_create**
- 13069 This is a new routine added for IT-API Version 2.0.
- 13070 **it_srq_free**
- 13071 This is a new routine added for IT-API Version 2.0.

- 13072 **it_srq_modify**
- 13073 This is a new routine added for IT-API Version 2.0.
- 13074 **it_srq_query**
- 13075 This is a new routine added for IT-API Version 2.0.
- 13076 **it_status_t**
- 13077 Removed IT_ERR_INVALID_SLID and IT_ERR_INVALID_SGID, resolving Erratum
13078 GN13=SUN1. Also see *it_address_handle_modify* Functional Change listing (above) for related
13079 change. Code using IT_ERR_INVALID_SLID or IT_ERR_INVALID_SGID will no longer
13080 compile and should be updated to use IT_ERR_INVALID_SOURCE_PATH.
- 13081 **it_ud_service_reply**
- 13082 Removed text erroneously indicating that Private Data delivery is unreliable.
- 13083 **it_ud_service_request_handle_create**
- 13084 Typo corrected, resolving Erratum GN42=OG22.
- 13085 Typo corrected, resolving Erratum GN41=OG21.
- 13086 **it_ud_service_request_handle_query**
- 13087 Flags now identified as bitwise rather than logically ORed value, resolving Erratum
13088 GN63=IBM1.
- 13089 Typo corrected, resolving Erratum GN24=OG4.
- 13090 Resolved GN21=SUN9 as defined in Global Behavior section *Output Parameters*.
- 13091 **it_unaffiliated_event_t**
- 13092 Typo corrected, resolving Erratum GN30=OG10.

F 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

13094

13095

G Header Files

13096

G.1 it_api.h

```
13097 /*
13098  * it_api.h
13099  *
13100  * Copyright © 2005 The Open Group
13101  * All rights reserved.
13102  * The copyright owner hereby grants permission for all or part of this
13103  * publication to be reproduced, stored in a retrieval system, or
13104  * transmitted, in any form or by any means, electronic, mechanical,
13105  * photocopying, recording, or otherwise, provided that it remains
13106  * unchanged and that this copyright statement is included in all
13107  * copies or substantial portions of the publication.
13108  *
13109  * For any software code contained within this specification,
13110  * permission is hereby granted, free-of-charge, to any person
13111  * obtaining a copy of this specification (the "Software"), to deal in
13112  * the Software without restriction, including without limitation the
13113  * rights to use, copy, modify, merge, publish, distribute, sublicense,
13114  * and/or sell copies of the Software, and to permit persons to whom
13115  * the Software is furnished to do so, subject to the above copyright
13116  * notice and this permission notice being included in all copies or
13117  * substantial portions of the Software.
13118  *
13119  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
13120  * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
13121  * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND
13122  * NON-INFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS
13123  * BE LIABLE FOR ANY CLAIM, DAMAGES, OR OTHER LIABILITY, WHETHER IN AN
13124  * ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF, OR IN
13125  * CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
13126  * SOFTWARE.
13127  *
13128  * Permission is granted for implementers to use the names, labels,
13129  * etc. contained within the specification. The intent of publication
13130  * of the specification is to encourage implementations of the
13131  * specification. This specification has not been verified for
13132  * avoidance of possible third-party proprietary rights. In
13133  * implementing this specification, usual procedures to ensure the
13134  * respect of possible third-party intellectual property rights should
13135  * be followed.
13136  */
13137
13138 #ifndef _ITAPI_H_
13139 #define _ITAPI_H_
13140 #include "it_api_os_specific.h"
```

```

13141
13142      /* default for IT-API is to maintain 1.0 bindings */
13143      #define ITAPI_ENABLE_V10_BINDINGS
13144      #if (defined(ITAPI_ENABLE_V20_BINDINGS) ||
13145          defined(ITAPI_ENABLE_V21_BINDINGS))
13146          #undef ITAPI_ENABLE_V10_BINDINGS
13147      #endif
13148
13149      #define IN
13150      #define OUT
13151
13152      /* define IT-API 1.0 variable name mappings for
13153         it_rc_only_attributes_t */
13154      #define rdma_read_inflight_incoming rdma_read_ird
13155      #define rdma_read_inflight_outgoing rdma_read_ord
13156
13157      /* define IT-API 1.0 variable name mappings for it_ia_info_t */
13158      #define ird_support ird_ord_ia_support
13159      #define ord_support ird_ord_ia_support
13160
13161      /* define IT-API 1.0 name mappings for it_make_rdma_addr */
13162      #define it_make_rdma_addr it_make_rdma_addr_absolute
13163
13164      /* define IT-API 1.0 name mappings for it_lmr_sync_* calls */
13165      #define it_lmr_sync_rdma_read it_lmr_flush_to_mem
13166      #define it_lmr_sync_rdma_write it_lmr_refresh_from_mem
13167
13168      /* define IT-API 1.0 name mapping for it_rmr_unbind call */
13169      #define it_rmr_unbind it_rmr_unlink
13170
13171      /* typedefs */
13172      typedef enum
13173      {
13174          IT_SUCCESS = 0,
13175          IT_ERR_ABORT,
13176          IT_ERR_ACCESS,
13177          IT_ERR_ADDRESS,
13178          IT_ERR_AEVD_NOT_ALLOWED,
13179          IT_ERR_ASYNC_AFF_EVD_EXISTS,
13180          IT_ERR_ASYNC_UNAFF_EVD_EXISTS,
13181          IT_ERR_CALLBACK_EXISTS,
13182          IT_ERR_CANNOT_RESET,
13183          IT_ERR_CONN_QUAL_BUSY,
13184          IT_ERR_EP_BUSY,
13185          IT_ERR_EP_TIMEOUT,
13186          IT_ERR_EVD_BUSY,
13187          IT_ERR_EVD_WAIT,
13188          IT_ERR_EVD_QUEUE_FULL,
13189          IT_ERR_FAULT,
13190          IT_ERR_HARD_HI_WATERMARK,
13191          IT_ERR_IA_CATASTROPHE,
13192          IT_ERR_INTERRUPT,
13193          IT_ERR_INVALID_ADDR_MODE,
13194          IT_ERR_INVALID_ADDRESS,

```

13195	IT_ERR_INVALID_AEVD,
13196	IT_ERR_INVALID_AH,
13197	IT_ERR_INVALID_ETIMEOUT,
13198	IT_ERR_INVALID_ATOMIC_OP,
13199	IT_ERR_INVALID_CM_RETRY,
13200	IT_ERR_INVALID_CN_EST_FLAGS,
13201	IT_ERR_INVALID_CN_EST_ID,
13202	IT_ERR_INVALID_CONN_EVD,
13203	IT_ERR_INVALID_CONN_QUAL,
13204	IT_ERR_INVALID_CONVERSION,
13205	IT_ERR_INVALID_DTO_FLAGS,
13206	IT_ERR_INVALID_EP,
13207	IT_ERR_INVALID_EP_ATTR,
13208	IT_ERR_INVALID_EP_KEY,
13209	IT_ERR_INVALID_EP_STATE,
13210	IT_ERR_INVALID_EP_TYPE,
13211	IT_ERR_INVALID_EVD,
13212	IT_ERR_INVALID_EVD_STATE,
13213	IT_ERR_INVALID_EVD_TYPE,
13214	IT_ERR_INVALID_FLAGS,
13215	IT_ERR_INVALID_HANDLE,
13216	IT_ERR_INVALID_IA,
13217	IT_ERR_INVALID_IOBL,
13218	IT_ERR_INVALID_LENGTH,
13219	IT_ERR_INVALID_LISTEN,
13220	IT_ERR_INVALID_LMR,
13221	IT_ERR_INVALID_LTIMEOUT,
13222	IT_ERR_INVALID_MAJOR_VERSION,
13223	IT_ERR_INVALID_MASK,
13224	IT_ERR_INVALID_MINOR_VERSION,
13225	IT_ERR_INVALID_NAME,
13226	IT_ERR_INVALID_NETADDR,
13227	IT_ERR_INVALID_NUM_SEGMENTS,
13228	IT_ERR_INVALID_PDATA_LENGTH,
13229	IT_ERR_INVALID_PRIVS,
13230	IT_ERR_INVALID_PZ,
13231	IT_ERR_INVALID_QUEUE_SIZE,
13232	IT_ERR_INVALID_RECV_DTO,
13233	IT_ERR_INVALID_RECV_EVD,
13234	IT_ERR_INVALID_RECV_EVD_STATE,
13235	IT_ERR_INVALID_REQ_EVD,
13236	IT_ERR_INVALID_REQ_EVD_STATE,
13237	IT_ERR_INVALID_RETRY,
13238	IT_ERR_INVALID_RMR,
13239	IT_ERR_INVALID_RNR_RETRY,
13240	IT_ERR_INVALID_RMR_TYPE,
13241	IT_ERR_INVALID_RTIMEOUT,
13242	IT_ERR_INVALID_SOFT_EVD,
13243	IT_ERR_INVALID_SOURCE_PATH,
13244	IT_ERR_INVALID_SPIGOT,
13245	IT_ERR_INVALID_SRQ,
13246	IT_ERR_INVALID_SRQ_SIZE,
13247	IT_ERR_INVALID_THRESHOLD,
13248	IT_ERR_INVALID_UD_STATUS,

```

13249     IT_ERR_INVALID_UD_SVC,
13250     IT_ERR_INVALID_UD_SVC_REQ_ID,
13251     IT_ERR_INVALID_WATERMARK,
13252     IT_ERR_LMR_BUSY,
13253     IT_ERR_MISMATCH_FD,
13254     IT_ERR_NO_CONTEXT,
13255     IT_ERR_NO_PERMISSION,
13256     IT_ERR_OP_NOT_SUPPORTED,
13257     IT_ERR_PAYLOAD_SIZE,
13258     IT_ERR_PDATA_NOT_SUPPORTED,
13259     IT_ERR_PRIV_OPS_UNAVAILABLE,
13260     IT_ERR_PZ_BUSY,
13261     IT_ERR_QUEUE_EMPTY,
13262     IT_ERR_RANGE,
13263     IT_ERR_RESOURCES,
13264     IT_ERR_RESOURCE_IRD,
13265     IT_ERR_RESOURCE_LMR_LENGTH,
13266     IT_ERR_RESOURCE_ORD,
13267     IT_ERR_RESOURCE_QUEUE_SIZE,
13268     IT_ERR_RESOURCE_RECV DTO,
13269     IT_ERR_RESOURCE_REQ DTO,
13270     IT_ERR_RESOURCE_RRSEG,
13271     IT_ERR_RESOURCE_RSEG,
13272     IT_ERR_RESOURCE_RWSEG,
13273     IT_ERR_RESOURCE_SSEG,
13274     IT_ERR_SOFT_HI_WATERMARK,
13275     IT_ERR_SRQ_BUSY,
13276     IT_ERR_SRQ_LOW_WATERMARK,
13277     IT_ERR_SRQ_NOT_SUPPORTED,
13278     IT_ERR_TIMEOUT_EXPIRED,
13279     IT_ERR_TOO_MANY_POSTS,
13280     IT_ERR_WAITER_LIMIT
13281 } it_status_t;
13282
13283 typedef uint32_t it_rmr_context_t;
13284
13285 #ifdef IT_32BIT
13286 typedef uint32_t it_length_t; /* a 32-bit platform */
13287 #else
13288 typedef uint64_t it_length_t; /* a 64-bit platform */
13289 #endif
13290
13291 typedef enum
13292 {
13293     IT_PRIV_NONE = 0x0000, /* needed for IT-API 1.0 compat */
13294     IT_PRIV_LOCAL_READ = 0x0001,
13295     IT_PRIV_LOCAL_WRITE = 0x0002,
13296     IT_PRIV_LOCAL = 0x0003,
13297     IT_PRIV_DEFAULT = 0x0003, /* deprecated by IT_PRIV_LOCAL */
13298     IT_PRIV_REMOTE_READ = 0x0004,
13299     IT_PRIV_REMOTE_WRITE = 0x0008,
13300     IT_PRIV_REMOTE = 0x000c,
13301     IT_PRIV_ATOMIC = 0x0010,
13302     IT_PRIV_ALL = 0x001f

```

```

13303     } it_mem_priv_t;
13304
13305     typedef enum
13306     {
13307         IT_LMR_FLAG_NONE = 0x0001,
13308         IT_LMR_FLAG_SHARED = 0x0002,
13309         IT_LMR_FLAG_NONCOHERENT = 0x0004,
13310         IT_LMR_FLAG_NON_SHAREABLE = 0x0008
13311     } it_lmr_flag_t;
13312
13313     typedef enum
13314     {
13315         IT_RMR_TYPE_DEFAULT = 0,
13316         IT_RMR_TYPE_NARROW = 1,
13317         IT_RMR_TYPE_WIDE = 2
13318     } it_rmr_type_t;
13319
13320     typedef enum
13321     {
13322         IT_ADDR_MODE_ABSOLUTE = 0,
13323         IT_ADDR_MODE_RELATIVE = 1
13324     } it_addr_mode_t;
13325
13326     typedef uint64_t it_ud_svc_req_identifier_t;
13327
13328     typedef uint64_t it_cn_est_identifier_t;
13329
13330     typedef enum
13331     {
13332         IT_FALSE = 0,
13333         IT_TRUE = 1
13334     } it_boolean_t;
13335
13336     typedef enum
13337     {
13338         IT_HANDLE_TYPE_ADDR,
13339         IT_HANDLE_TYPE_EP,
13340         IT_HANDLE_TYPE_EVD,
13341         IT_HANDLE_TYPE_IA,
13342         IT_HANDLE_TYPE_LISTEN,
13343         IT_HANDLE_TYPE_LMR,
13344         IT_HANDLE_TYPE_PZ,
13345         IT_HANDLE_TYPE_RMR,
13346         IT_HANDLE_TYPE_UD_SVC_REQ,
13347         IT_HANDLE_TYPE_SRQ
13348     } it_handle_type_enum_t;
13349
13350     typedef void *it_handle_t;
13351
13352     #define IT_NULL_HANDLE ((it_handle_t) NULL)
13353
13354     typedef struct it_addr_handle_s *it_addr_handle_t;
13355     typedef struct it_ep_handle_s *it_ep_handle_t;
13356     typedef struct it_evd_handle_s *it_evd_handle_t;

```

```

13357 typedef struct it_ia_handle_s *it_ia_handle_t;
13358 typedef struct it_listen_handle_s *it_listen_handle_t;
13359 typedef struct it_lmr_handle_s *it_lmr_handle_t;
13360 typedef struct it_pz_handle_s *it_pz_handle_t;
13361 typedef struct it_rmr_handle_s *it_rmr_handle_t;
13362 typedef struct it_ud_svc_req_handle_s *it_ud_svc_req_handle_t;
13363 typedef struct it_srq_handle_s *it_srq_handle_t;
13364
13365 typedef enum
13366 {
13367
13368     /* IANA (TCP/UDP) Port Number */
13369     IT_IANA_PORT = 0x01,
13370
13371     /* InfiniBand Service ID, as described in Section 12.7.3 of
13372      Volume 1 of the InfiniBand specification. */
13373     IT_IB_SERVICEID = 0x02,
13374
13375     /* VIA Connection Discriminator */
13376     IT_VIA_DISCRIMINATOR = 0x04,
13377
13378     /* iWARP local and remote IP (IANA) port object */
13379     IT_IANA_LR_PORT = 0x08
13380 } it_conn_qual_type_t;
13381
13382 #define IT_MAX_VIA_DISC_LEN 64
13383
13384 typedef struct
13385 {
13386
13387     /* The total number of bytes in the array below */
13388     /* that are significant. */
13389     uint16_t len;
13390
13391     /* VIA connection discriminator, which is an array of bytes. */
13392     unsigned char discriminator[IT_MAX_VIA_DISC_LEN];
13393
13394 } it_via_discriminator_t;
13395
13396 typedef uint64_t it_ib_serviceid_t;
13397
13398 typedef struct
13399 {
13400     uint16_t local;
13401     uint16_t remote;
13402 } it_iana_lr_port_t;
13403
13404 typedef struct
13405 {
13406
13407     /* The discriminator for the union below. */
13408     it_conn_qual_type_t type;
13409
13410     union

```

```

13411     {
13412
13413         /* IANA Port Number, in network byte order. */
13414         uint16_t port;
13415
13416         /* InfiniBand Service ID, in network byte order. */
13417         it_ib_serviceid_t serviceid;
13418
13419         /* VIA connection discriminator. */
13420         it_via_discriminator_t discriminator;
13421
13422         /* IANA local/remote Port numbers, in network byte order. */
13423         it_iana_lr_port_t lr_port;
13424
13425     } conn_qual;
13426 } it_conn_qual_t;
13427
13428 typedef union
13429 {
13430     void *ptr;
13431     uint64_t index;
13432 } it_context_t;
13433
13434 typedef uint64_t it_dto_cookie_t;
13435
13436 typedef enum
13437 {
13438     IT_DTO_SUCCESS = 0,
13439     IT_DTO_ERR_LOCAL_LENGTH = 1,
13440     IT_DTO_ERR_LOCAL_EP = 2,
13441     IT_DTO_ERR_LOCAL_PROTECTION = 3,
13442     IT_DTO_ERR_FLUSHED = 4,
13443     IT_RMR_OPERATION_FAILED = 5,
13444     IT_DTO_ERR_BAD_RESPONSE = 6,
13445     IT_DTO_ERR_REMOTE_ACCESS = 7,
13446     IT_DTO_ERR_REMOTE_RESPONDER = 8,
13447     IT_DTO_ERR_TRANSPORT = 9,
13448     IT_DTO_ERR_RECEIVER_NOT_READY = 10,
13449     IT_DTO_ERR_PARTIAL_PACKET = 11,
13450     IT_DTO_ERR_LOCAL_MM_OPERATION = 12
13451 } it_dto_status_t;
13452
13453 typedef enum
13454 {
13455     /* If flag set, completion generates a local event. */
13456     IT_COMPLETION_FLAG = 0x01,
13457
13458     /* If flag set, completion causes local Notification. */
13459     IT_NOTIFY_FLAG = 0x02,
13460
13461     /* If flag set, receipt of DTO at remote will cause Notification
13462        at remote. */
13463     IT_SOLICITED_WAIT_FLAG = 0x04,
13464

```

```

13465
13466      /* If flag set, a WR posted to an EP's SQ will not start if
13467         RDMA Reads posted previously to the EP are not complete. */
13468      IT_BARRIER_FENCE_FLAG = 0x08,
13469
13470      /* If flag set, an LMR Unlink or RMR Unlink WR posted to an EP
13471         will not start if WRs posted previously to the EP's SQ are
13472         not complete. */
13473      IT_UNLINK_FENCE_FLAG = 0x10,
13474
13475      /* If flag set, the local data sink is unlinked upon WR
13476         completion. */
13477      IT_UNLINK_LOCAL_SINK = 0x20,
13478
13479      /* If flag set, attempt to enqueue this WR for later processing. */
13480      IT_COALESCE_WR_FLAG = 0x40
13481  } it_dto_flags_t;
13482
13483  typedef enum
13484  {
13485
13486      /* IPv4 address */
13487      IT_IPV4 = 0x1,
13488
13489      /* IPv6 address */
13490      IT_IPV6 = 0x2,
13491
13492      /* InfiniBand GID */
13493      IT_IB_GID = 0x3,
13494
13495      /* VIA Network Address */
13496      IT_VIA_HOSTADDR = 0x4
13497  } it_net_addr_type_t;
13498
13499  #define IT_MAX_VIA_ADDR_LEN 64
13500
13501  typedef struct
13502  {
13503
13504      /* The number of bytes in the array below that are
13505         significant. */
13506      uint16_t len;
13507
13508      /* VIA host address, which is an array of bytes. */
13509      unsigned char hostaddr[IT_MAX_VIA_ADDR_LEN];
13510
13511  } it_via_net_addr_t;
13512
13513  typedef struct in6_addr it_ib_gid_t;
13514
13515  typedef struct
13516  {
13517
13518      /* The discriminator for the union below. */

```

```

13519     it_net_addr_type_t addr_type;
13520
13521     union
13522     {
13523
13524         /* IPv4 address, in network byte order. */
13525         struct in_addr ipv4;
13526
13527         /* IPv6 address, in network byte order. */
13528         struct in6_addr ipv6;
13529
13530         /* InfiniBand GID, in network byte order. */
13531         it_ib_gid_t gid;
13532
13533         /* VIA Network Address. */
13534         it_via_net_addr_t via;
13535
13536     } addr;
13537
13538 } it_net_addr_t;
13539
13540 typedef enum
13541 {
13542
13543     /* InfiniBand Transport */
13544     IT_IB_TRANSPORT = 1,
13545
13546     /* VIA host Interface using IP transport, supporting
13547        only the Reliable Delivery reliability level */
13548     IT_VIA_IP_TRANSPORT = 2,
13549
13550     /* VIA host Interface, using Fibre Channel transport, supporting
13551        only the Reliable Delivery reliability level */
13552     IT_VIA_FC_TRANSPORT = 3,
13553
13554     /* iWARP over TCP transport */
13555     IT_IWARP_TCP_TRANSPORT = 4,
13556
13557     /* Vendor-proprietary Transport */
13558     IT_VENDOR_TRANSPORT = 1000
13559 } it_transport_type_t;
13560
13561 typedef enum
13562 {
13563
13564     /* Reliable Connected Transport Service Type */
13565     IT_RC_SERVICE = 0x1,
13566
13567     /* Unreliable Datagram Transport Service Type */
13568     IT_UD_SERVICE = 0x2,
13569
13570 } it_transport_service_type_t;
13571
13572 typedef struct

```

```

13573     {
13574
13575         /* Spigot identifier */
13576         size_t spigot_id;
13577
13578         /* Maximum sized Send operation for the RC service
13579            on this Spigot. */
13580         size_t max_rc_send_len;
13581
13582         /* Maximum sized RDMA Read/Write operation for the RC service on
13583            this Spigot. */
13584         size_t max_rc_rdma_len;
13585
13586         /* Maximum sized Send operation for the UD service
13587            on this Spigot. */
13588         size_t max_ud_send_len;
13589
13590         /* Indicates whether the Spigot is online or offline.
13591            An IT_TRUE value means online. */
13592         it_boolean_t spigot_online;
13593
13594         /* A mask indicating which Connection Qualifier types this
13595            IA supports for input to it_ep_connect and
13596            it_ud_service_request_handle_create. The bits in the mask are
13597            an inclusive OR of the values for Connection Qualifier types
13598            that this IA supports. */
13599         it_conn_qual_type_t active_side_conn_qual;
13600
13601         /* A mask indicating which Connection Qualifier types this to
13602            it_listen_create. The bits in the mask are an inclusive OR of
13603            the values for Connection Qualifier types that this IA
13604            supports. */
13605         it_conn_qual_type_t passive_side_conn_qual;
13606
13607         /* The number of Network Addresses associated with Spigot. */
13608         size_t num_net_addr;
13609
13610         /* Pointer to array of Network Address addresses. */
13611         it_net_addr_t *net_addr;
13612     } it_spigot_info_t;
13613
13614 typedef struct
13615 {
13616     /* The NodeInfo:VendorID as described in chapter 14 of the
13617        IB spec. */
13618     uint32_t vendor:24;
13619
13620     /* The NodeInfo:DeviceID as described in chapter 14 of the
13621        IB spec. */
13622     uint16_t device;
13623
13624     /* The NodeInfo:Revision as described in chapter 14 of the
13625
13626

```

```

13627         IB spec. */
13628     uint32_t revision;
13629 } it_vendor_ib_t;
13630
13631 typedef struct
13632 {
13633     /* The "Name" member of the VIP_NIC_ATTRIBUTES structure, as
13634     described in the VIA spec. */
13635     char name[64];
13636
13637     /* The "HardwareVersion" member of the VIP_NIC_ATTRIBUTES
13638     structure, as described in the VIA spec. */
13639     unsigned long hardware;
13640
13641     /* The "ProviderVersion" member of the VIP_NIC_ATTRIBUTES
13642     structure, as described in the VIA spec. */
13643     unsigned long provider;
13644 } it_vendor_via_t;
13645
13646 typedef struct
13647 {
13648     /* Indicates whether or not vid field contains valid data. */
13649     it_boolean_t valid_vid;
13650
13651     /* Vendor Identification field - strictly vendor-specific (only
13652     valid if valid_vid field is IT_TRUE). */
13653     unsigned char vid[64];
13654 } it_vendor_iwarp_tcp_t;
13655
13656 typedef struct it_io_addr_s *it_io_addr_t;
13657
13658 typedef enum
13659 {
13660     IT_IOBL_PAGE_LIST = 0x0001,
13661     IT_IOBL_BLOCK_LIST = 0x0002,
13662     IT_IOBL_VE_PAGE_LIST = 0x0004
13663 } it_iobl_type_t;
13664
13665 typedef struct
13666 {
13667     /* I/O Address - bus address or physical address.
13668     it_io_addr_t is an OS-dependent type. */
13669     it_io_addr_t io_addr;
13670
13671     /* IOBL element length for this element */
13672     it_length_t elt_len;
13673 } it_iobl_ve_elt_t; /* Variable-element-length list element */
13674
13675 typedef struct
13676 {
13677     it_iobl_type_t iobl_type; /* Union discriminator */
13678     union
13679     {
13680         struct

```

```

13681     {
13682         it_io_addr_t *list_elt; /* Head of list */
13683         it_length_t elt_len; /* Constant IOBL element length */
13684     } ce_list; /* Constant-element-length list */
13685     struct
13686     {
13687         it_iobl_ve_elt_t *list_elt; /* Head of list */
13688     } ve_list; /* Variable-element-length list */
13689     } list;
13690     size_t num_elts; /* Number of elements in list */
13691     it_length_t fbo; /* First-Byte Offset */
13692 } it_iobl_t;
13693
13694 typedef struct
13695 {
13696     /* Interface Adapter name, as specified in it_ia_create. */
13697     char *ia_name;
13698
13699     /* The major version number of the latest version of the
13700     IT-API that this IA supports. */
13701     uint32_t api_major_version;
13702
13703     /* The minor version number of the latest version of the
13704     IT-API that this IA supports. */
13705     uint32_t api_minor_version;
13706
13707     /* The major version number for the software being used to control
13708     this IA. The IT-API imposes no structure whatsoever on this
13709     number; its meaning is completely IA-dependent. */
13710     uint32_t sw_major_version;
13711
13712     /* The minor version number for the software being used to control
13713     this IA. The IT-API imposes no structure whatsoever on this
13714     number; its meaning is completely IA-dependent. */
13715     uint32_t sw_minor_version;
13716
13717     /* The vendor associated with the IA. This information is useful
13718     if the Consumer wishes to do device-specific programming. This
13719     union is discriminated by transport_type. No vendor
13720     identification is provided for transports not listed below. */
13721     union
13722     {
13723         /* Used if transport_type is IT_IB_TRANSPORT. */
13724         it_vendor_ib_t ib;
13725
13726         /* Used if transport_type is IT_VIA_IP_TRANSPORT or
13727         IT_VIA_FC_TRANSPORT. */
13728         it_vendor_via_t via;
13729
13730         /* Used if transport_type is IT_IWARP_TCP_TRANSPORT. */
13731         it_vendor_iwarp_tcp_t iwarp;
13732     };
13733 }
13734

```

```

13735     } vendor;
13736
13737     /* The Interface Adapter and platform provide a data alignment hint
13738        to the Consumer to help the Consumer align their data transfer
13739        buffers in a way that is optimal for the performance of the IA.
13740        For example, if the best throughput is obtained by aligning
13741        buffers to 128-byte boundaries, dto_alignment_hint will have the
13742        value 128. The Consumer may choose to ignore the alignment hint
13743        without any adverse functional impact. (There may be an adverse
13744        performance impact.) */
13745     uint32_t dto_alignment_hint;
13746
13747     /* The transport type (e.g., InfiniBand) supported by Interface
13748        Adapter. An Interface Adapter supports precisely one transport
13749        type. */
13750     it_transport_type_t transport_type;
13751
13752     /* The Transport Service Types supported by this IA. This is
13753        constructed by doing an inclusive OR of the Transport Service
13754        Type values. */
13755     it_transport_service_type_t supported_service_types;
13756
13757     /* Indicates whether Work Queues are resizable. */
13758     it_boolean_t ep_work_queues_resizable;
13759
13760     /* Indicates whether the underlying transport used by this IA uses
13761        a three-way handshake for doing Connection establishment. Note
13762        that if the underlying transport supports a three-way handshake
13763        the Consumer can choose whether to use two handshakes or three
13764        when establishing the Connection. If the underlying transport
13765        supports a two-way handshake for establishing a Connection, the
13766        Consumer can only use two handshakes when establishing the
13767        Connection. */
13768     it_boolean_t three_way_handshake_support;
13769
13770     /* Indicates whether Private Data is supported on Connection
13771        establishment or UD service resolution operations. */
13772     it_boolean_t private_data_support;
13773
13774     /* Indicates whether the max_message_size field in the
13775        IT_CM_REQ_CONN_REQUEST_EVENT is valid for this IA. */
13776     it_boolean_t max_message_size_support;
13777
13778     /* Indicates whether or not the IA supports IRD/ORD. Affects
13779        whether the rdma_read_ird or rdma_read_ord fields in the
13780        IT_CM_REQ_CONN_REQUEST_EVENT or the
13781        IT_CM_MSG_CONN_ESTABLISHED_EVENT are valid for this IA.
13782        Also affects whether IRD/ORD suppression is an option.
13783        Deprecates IT-API 1.0 values "ird_support" and "ord_support". */
13784     it_boolean_t ird_ord_ia_support;
13785
13786     /* Indicates whether IRD/ORD suppression is supported
13787        for this IA. If this member has a value of IT_TRUE, the
13788        Consumer can control IRD/ORD suppression in it_ep_connect

```

```

13789         and it_listen_create. Otherwise they cannot. */
13790 it_boolean_t ird_ord_suppressible;
13791
13792 /* Indicates whether the IA generates IT_ASYNC_UNAFF_SPIGOT_ONLINE
13793 Events. See it_unaffiliated_event_t for details. */
13794 it_boolean_t spigot_online_support;
13795
13796 /* Indicates whether the IA generates IT_ASYNC_UNAFF_SPIGOT_OFFLINE
13797 Events. See it_unaffiliated_event_t for details. */
13798 it_boolean_t spigot_offline_support;
13799
13800 /* The maximum number of bytes of Private Data supported for the
13801 it_ep_connect routine. This will be less than or equal to
13802 IT_MAX_PRIV_DATA. */
13803 size_t connect_private_data_len;
13804
13805 /* The maximum number of bytes of Private Data supported for the
13806 it_ep_accept routine. This will be less than or equal to
13807 IT_MAX_PRIV_DATA. */
13808 size_t accept_private_data_len;
13809
13810 /* The maximum number of bytes of Private Data supported for the
13811 it_reject routine. This will be less than or equal to
13812 IT_MAX_PRIV_DATA. */
13813 size_t reject_private_data_len;
13814
13815 /* The maximum number of bytes of Private Data supported for the
13816 it_ep_disconnect routine. This will be less than or equal to
13817 IT_MAX_PRIV_DATA. */
13818 size_t disconnect_private_data_len;
13819
13820 /* The maximum number of bytes of Private Data supported for the
13821 it_ud_service_request_handle_create routine. This will be
13822 less than or equal to IT_MAX_PRIV_DATA. */
13823 size_t ud_req_private_data_len;
13824
13825 /* The maximum number of bytes of Private Data supported for the
13826 it_ud_service_reply routine. This will be less than or equal to
13827 IT_MAX_PRIV_DATA. */
13828 size_t ud_rep_private_data_len;
13829
13830 /* Specifies the number of Spigots associated with this Interface
13831 Adapter. */
13832 size_t num_spigots;
13833
13834 /* An array of Spigot information data structures. The array
13835 contains num_spigots elements. */
13836 it_spigot_info_t *spigot_info;
13837
13838 /* The Handle for the EVD that contains the affiliated async Event
13839 Stream. If no EVD contains the Affiliated Async Event Stream,
13840 this member will have the distinguished value IT_NULL_HANDLE. */
13841 it_evd_handle_t affiliated_err_evd;
13842

```

```

13843     /* The Handle for the EVD that contains the Unaffiliated Async
13844        Event Stream. If no EVD contains the Unaffiliated Async Event
13845        Stream, this member will have the distinguished value
13846        IT_NULL_HANDLE. */
13847     it_evd_handle_t unaffiliated_err_evd;
13848
13849     /* Indicates whether the IA supports the S-RQ feature. */
13850     it_boolean_t srq_support;
13851
13852     /* Indicates whether the IA supports the Endpoint Hard
13853        High Watermark mechanism for limiting the number of Receive
13854        DTOs that can be in progress on an Endpoint that has an
13855        associated S-RQ. */
13856     it_boolean_t hard_hi_watermark_support;
13857
13858     /* Indicates whether the IA supports the Endpoint Soft
13859        High Watermark mechanism for generating an Affiliated
13860        Asynchronous Event when the number of Receive DTOs in
13861        progress on an Endpoint that has an associated S-RQ exceeds
13862        the Endpoint Soft High Watermark. */
13863     it_boolean_t soft_hi_watermark_support;
13864
13865     /* Indicates whether an S-RQ can be resized after it is created. */
13866     it_boolean_t srq_resizable;
13867
13868     /* Indicates that an iWARP V-RNIC supports a modified qp state
13869        diagram (outside the RDMAC verbs). */
13870     it_boolean_t extended_iwarp_qp_states;
13871
13872     /* Indicates that Implementation supports socket conversion (TDI).
13873        This attribute is IT_TRUE if and only if transport_type is
13874        IT_IWARP_TCP_TRANSPORT. */
13875     it_boolean_t socket_conversion_support;
13876
13877     /* Indicates that IA supports increasing ORD
13878        (decreasing ORD is mandatory for all RNICs). */
13879     it_boolean_t rdma_read_ord_increasable;
13880
13881     /* Indicates that IA supports modifying IRD. */
13882     it_boolean_t rdma_read_ird_modifiable;
13883
13884     /* Bit set indicating which RMR types are supported.
13885        Possible values are IT_RMR_TYPE_NARROW, IT_RMR_TYPE_WIDE, and
13886        (IT_RMR_TYPE_NARROW|IT_RMR_TYPE_WIDE). See also it_rmr_type_t. */
13887     it_rmr_type_t rmr_types_supported;
13888
13889     /* Indicates whether the IA supports Relative Addressing. See also
13890        it_addr_mode_t. */
13891     it_boolean_t addr_mode_relative_support;
13892
13893     /* Indicates whether the Destination buffer for an RDMA Read DTO
13894        must have remote or local write permission, and whether or not
13895        the Endpoint to which an RDMA Read DTO is posted must have RDMA
13896        Write access enabled. See also it_post_rdma_read. */

```

```

13897     it_boolean_t rdma_read_requires_remote_write;
13898
13899     /* Indicates whether the IA supports changing the RDMA enables
13900        after EP creation. */
13901     it_boolean_t ep_rdma_enables_modifiable;
13902
13903     /* Indicates whether the IA supports it_post_rdma_read_to_rmr and
13904        also whether the IT_UNLINK_LOCAL_SINK flag may be used on
13905        it_post_rdma_read or it_post_rdma_read_to_rmr operations. */
13906     it_boolean_t rdma_read_local_extensions;
13907
13908     /* Indicates whether the IA supports DTO EVD overflow detection. */
13909     it_boolean_t dto_evd_overflow_detection;
13910
13911     /* Indicates that the transport protocol used by the IA supports
13912        Atomics, and that the IA itself implements that support. If
13913        the underlying transport protocol used by the IA does not
13914        support Atomics, this attribute will have the value IT_FALSE. */
13915     it_boolean_t atomic_support;
13916
13917     /* Indicates whether the IA supports use of LMR Link capabilities
13918        by Privileged Consumers. */
13919     it_boolean_t lmr_link_support;
13920
13921     /* Indicates whether the IA supports use of the Direct LMR Handle
13922        by Privileged Consumers. */
13923     it_boolean_t direct_lmr_handle_support;
13924
13925     /* The Direct LMR Handle. If direct_lmr_handle_support
13926        is IT_FALSE, then direct_lmr_handle is reported as
13927        IT_NULL_HANDLE. */
13928     it_lmr_handle_t direct_lmr_handle;
13929
13930     /* Indicates whether the IA supports use of posting remote unlink
13931        requests in Send calls. */
13932     it_boolean_t post_send_unlink_remote_support;
13933
13934     /* Indicates whether the IA supports unlinking narrow RMRs. */
13935     it_boolean_t narrow_rmr_unlink_support;
13936
13937     /* Indicates whether the IA supports unlinking wide RMRs. */
13938     it_boolean_t wide_rmr_unlink_support;
13939
13940     /* Indicates whether the IA supports local fencing operations. */
13941     it_boolean_t unlink_fence_support;
13942
13943     /* Defines types of IOBL supported by the IA. */
13944     it_ioibl_type_t ioibl_types_supported;
13945 } it_ia_info_t;
13946
13947 typedef struct
13948 {
13949     it_lmr_handle_t lmr;
13950     union

```

```

13951     {
13952         void *abs;
13953         it_length_t rel;
13954         it_io_addr_t io;
13955     } addr;
13956     it_length_t length;
13957 } it_lmr_triplet_t;
13958
13959 typedef struct
13960 {
13961     it_rmr_handle_t rmr;
13962     union
13963     {
13964         void *abs;
13965         it_length_t rel;
13966     } addr;
13967     it_length_t length;
13968 } it_rmr_triplet_t;
13969
13970 typedef struct
13971 {
13972     /* Partition Key, as defined in the REQ message for the IB
13973        CM protocol. */
13974     uint16_t partition_key;
13975
13976     /* Path Packet Payload MTU, as defined in the REQ message
13977        for the IB CM protocol. */
13978     uint8_t path_mtu:4;
13979
13980     /* PacketLifeTime, as defined in the PathRecord in IB
13981        specification. This field is useful for Consumers that
13982        wish to use timeout values other than the default ones
13983        for doing Connection establishment. */
13984     uint8_t packet_lifetime:6;
13985
13986     /* Local Port LID, as defined in the REQ message for the IB
13987        CM protocol. The low-order bits of this value also
13988        constitute the Source Path Bits that are used to
13989        create an Address Handle. */
13990     uint16_t local_port_lid;
13991
13992     /* Remote Port LID, as defined in the REQ message for the
13993        IB CM protocol. This is also the Destination LID used
13994        to create an Address Handle. */
13995     uint16_t remote_port_lid;
13996
13997     /* Local Port GID in network byte order, as defined in the
13998        REQ message for the IB CM protocol. This is also used to
13999        determine the appropriate Source GID Index to be used
14000        when creating an Address Handle. */
14001     it_ib_gid_t local_port_gid;
14002
14003     /* Remote Port GID in network byte order, as defined in the

```

```

14005         REQ message for the IB CM protocol. This is also the
14006         Destination GID or MGID used to create an Address
14007         Handle. */
14008     it_ib_gid_t remote_port_gid;
14009
14010     /* Packet Rate, as defined in the REQ message for the IB CM
14011         protocol. This is also the Maximum Static Rate to be
14012         used when creating an Address Handle. */
14013     uint8_t packet_rate:6;
14014
14015     /* SL, as defined in the REQ message for the IB CM
14016         protocol. This is also the Service Level to be used
14017         when creating an Address Handle. */
14018     uint8_t sl:4;
14019
14020     /* Subnet Local, as defined in the REQ message for the IB
14021         CM protocol. When creating an Address Handle, setting
14022         this bit causes a GRH to be included as part of any
14023         Unreliable Datagram sent using the Address Handle. */
14024     uint8_t subnet_local:1;
14025
14026     /* Flow Label, as defined in the REQ message for the IB CM
14027         protocol. This is also the Flow Label to be used when
14028         creating an Address Handle. This is only valid if
14029         subnet_local is clear. */
14030     uint32_t flow_label:20;
14031
14032     /* Traffic Class, as defined in the REQ message for the IB
14033         CM protocol. This is also the Traffic Class to be
14034         used when creating an Address Handle. This is only
14035         valid if subnet_local is clear. */
14036     uint8_t traffic_class;
14037
14038     /* Hop Limit, as defined in the REQ message for the IB CM
14039         protocol. This is also the Hop Limit to be used when
14040         creating an Address Handle. This is only valid if
14041         subnet_local is clear. */
14042     uint8_t hop_limit;
14043
14044 } it_ib_net_endpoint_t;
14045
14046 typedef it_via_net_addr_t it_via_net_endpoint_t;
14047
14048 typedef enum
14049 {
14050     IT_IP_VERS_IPV4 = 0x1,
14051     IT_IP_VERS_IPV6 = 0x2
14052 } it_ip_vers_t;
14053
14054 typedef struct
14055 {
14056     /* Designates the type of IP address that is found in
14057         both the laddr and raddr unions below. */
14058     it_ip_vers_t ip_vers;

```

```

14059
14060     /* Local path element */
14061     union
14062     {
14063         struct in_addr ipv4;
14064         struct in6_addr ipv6;
14065     } laddr;
14066
14067     /* Remote path element */
14068     union
14069     {
14070         struct in_addr ipv4;
14071         struct in6_addr ipv6;
14072     } raddr;
14073 } it_iwarp_net_endpoint_t;
14074
14075 typedef struct
14076 {
14077
14078     /* Identifier for the Spigot to be used on the local IA.
14079     Note that this data structure is always used in a
14080     Context where the IA associated with the Spigot can be
14081     deduced. */
14082     size_t spigot_id;
14083
14084     /* The transport-independent timeout parameter for how long
14085     to wait, in microseconds, before timing out a Connection
14086     establishment attempt using this Path. The timeout
14087     period for establishing a Connection
14088     can only be specified on the Active side; the timeout
14089     period cannot be changed on the Passive side. */
14090     uint64_t timeout;
14091
14092     /* The remote component of the Path. */
14093     union
14094     {
14095
14096         /* For use with InfiniBand. */
14097         it_ib_net_endpoint_t ib;
14098
14099         /* For use with VIA. */
14100         it_via_net_endpoint_t via;
14101
14102         /* For use with iWARP. */
14103         it_iwarp_net_endpoint_t iwarp;
14104
14105     } remote;
14106
14107 } it_path_t;
14108
14109 typedef uint32_t it_ud_ep_id_t;
14110 typedef uint32_t it_ud_ep_key_t;
14111
14112 typedef enum

```

```

14113 {
14114     IT_EP_PARAM_ALL = 0x00000001,
14115     IT_EP_PARAM_IA = 0x00000002,
14116     IT_EP_PARAM_SPIGOT = 0x00000004,
14117     IT_EP_PARAM_STATE = 0x00000008,
14118     IT_EP_PARAM_SERV_TYPE = 0x00000010,
14119     IT_EP_PARAM_PATH = 0x00000020,
14120     IT_EP_PARAM_PZ = 0x00000040,
14121     IT_EP_PARAM_REQ_SEVD = 0x00000080,
14122     IT_EP_PARAM_RECV_SEVD = 0x00000100,
14123     IT_EP_PARAM_CONN_SEVD = 0x00000200,
14124     IT_EP_PARAM_RDMA_RD_ENABLE = 0x00000400,
14125     IT_EP_PARAM_RDMA_WR_ENABLE = 0x00000800,
14126     IT_EP_PARAM_MAX_RDMA_READ_SEG = 0x00001000,
14127     IT_EP_PARAM_MAX_RDMA_WRITE_SEG = 0x00002000,
14128     IT_EP_PARAM_MAX_IRD = 0x00004000,
14129     IT_EP_PARAM_MAX_ORD = 0x00008000,
14130     IT_EP_PARAM_EP_ID = 0x00010000,
14131     IT_EP_PARAM_EP_KEY = 0x00020000,
14132     IT_EP_PARAM_MAX_PAYLOAD = 0x00040000,
14133     IT_EP_PARAM_MAX_REQ_DTO = 0x00080000,
14134     IT_EP_PARAM_MAX_RECV_DTO = 0x00100000,
14135     IT_EP_PARAM_MAX_SEND_SEG = 0x00200000,
14136     IT_EP_PARAM_MAX_RECV_SEG = 0x00400000,
14137     IT_EP_PARAM_SRQ = 0x00800000,
14138     IT_EP_PARAM_SOFT_HI_WATERMARK = 0x01000000,
14139     IT_EP_PARAM_HARD_HI_WATERMARK = 0x02000000,
14140     IT_EP_PARAM_ATOMICS_ENABLE = 0x04000000,
14141     IT_EP_PARAM_PRIV_OPS_ENABLE = 0x08000000
14142 } it_ep_param_mask_t;
14143
14144 typedef struct
14145 {
14146     it_boolean_t rdma_read_enable;
14147     /* IT_EP_PARAM_RDMA_RD_ENABLE */
14148     it_boolean_t rdma_write_enable;
14149     /* IT_EP_PARAM_RDMA_WR_ENABLE */
14150     size_t max_rdma_read_segments;
14151     /* IT_EP_PARAM_MAX_RDMA_READ_SEG */
14152     size_t max_rdma_write_segments;
14153     /* IT_EP_PARAM_MAX_RDMA_WRITE_SEG */
14154     uint32_t rdma_read_ird;
14155     /* IT_EP_PARAM_MAX_IRD */
14156     uint32_t rdma_read_ord;
14157     /* IT_EP_PARAM_MAX_ORD */
14158     it_srq_handle_t srq;
14159     /* IT_EP_PARAM_SRQ */
14160     size_t soft_hi_watermark;
14161     /* IT_EP_PARAM_SOFT_HI_WATERMARK */
14162     size_t hard_hi_watermark;
14163     /* IT_EP_PARAM_HARD_HI_WATERMARK */
14164     it_boolean_t atomics_enable;
14165     /* IT_EP_PARAM_ATOMICS_ENABLE */
14166

```

```

14167     } it_rc_only_attributes_t;
14168
14169     #define IT_HARD_HI_WATERMARK_DISABLE ((size_t) -1)
14170
14171     typedef struct
14172     {
14173         it_ud_ep_id_t ud_ep_id; /* IT_EP_PARAM_EP_ID */
14174         it_ud_ep_key_t ud_ep_key; /* IT_EP_PARAM_EP_KEY */
14175     } it_remote_ep_info_t;
14176
14177     typedef struct
14178     {
14179         it_remote_ep_info_t ep_info;
14180     } it_ud_only_attributes_t;
14181
14182     typedef union
14183     {
14184         it_rc_only_attributes_t rc;
14185         it_ud_only_attributes_t ud;
14186     } it_service_attributes_t;
14187
14188     typedef struct
14189     {
14190         size_t max_dto_payload_size; /* IT_EP_PARAM_MAX_PAYLOAD */
14191         size_t max_request_dtos; /* IT_EP_PARAM_MAX_REQ_DTO */
14192         size_t max_rcv_dtos; /* IT_EP_PARAM_MAX_RECV_DTO */
14193         size_t max_send_segments; /* IT_EP_PARAM_MAX_SEND_SEG */
14194         size_t max_rcv_segments; /* IT_EP_PARAM_MAX_RECV_SEG */
14195
14196         it_service_attributes_t srv;
14197
14198         it_boolean_t priv_ops_enable; /* IT_EP_PARAM_PRIV_OPS_ENABLE */
14199     } it_ep_attributes_t;
14200
14201     #define IT_EVENT_STREAM_MASK 0xff000
14202
14203     #define IT_TIMEOUT_INFINITE ((uint64_t)(-1))
14204
14205     typedef enum
14206     {
14207         /*
14208          * Event Stream for WR/DTO completions
14209          */
14210         IT_DTO_EVENT_STREAM = 0x00000,
14211         IT_DTO_SEND_CMPL_EVENT = 0x00001,
14212         IT_DTO_RC_RECV_CMPL_EVENT = 0x00002,
14213         IT_DTO_UD_RECV_CMPL_EVENT = 0x00003,
14214         IT_DTO_RDMA_WRITE_CMPL_EVENT = 0x00004,
14215         IT_DTO_RDMA_READ_CMPL_EVENT = 0x00005,
14216         IT_RMR_BIND_CMPL_EVENT = 0x00006,
14217         IT_RMR_LINK_CMPL_EVENT = 0x00006,
14218         IT_DTO_FETCH_ADD_CMPL_EVENT = 0x00007,
14219         IT_DTO_CMP_SWAP_CMPL_EVENT = 0x00008,
14220         IT_LMR_LINK_CMPL_EVENT = 0x00009,

```

```

14221
14222      /*
14223      * Event Stream for Communication Management Request Events
14224      */
14225      IT_CM_REQ_EVENT_STREAM = 0x01000,
14226      IT_CM_REQ_CONN_REQUEST_EVENT = 0x01001,
14227      IT_CM_REQ_UD_SERVICE_REQUEST_EVENT = 0x01002,
14228
14229      /*
14230      * Event Stream for Communication Management Message Events
14231      */
14232      IT_CM_MSG_EVENT_STREAM = 0x02000,
14233      IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT = 0x02001,
14234      IT_CM_MSG_CONN_ESTABLISHED_EVENT = 0x02002,
14235      IT_CM_MSG_CONN_DISCONNECT_EVENT = 0x02003,
14236      IT_CM_MSG_CONN_PEER_REJECT_EVENT = 0x02004,
14237      IT_CM_MSG_CONN_NONPEER_REJECT_EVENT = 0x02005,
14238      IT_CM_MSG_CONN_BROKEN_EVENT = 0x02006,
14239      IT_CM_MSG_UD_SERVICE_REPLY_EVENT = 0x02007,
14240
14241      /* Event Stream for Affiliated Asynchronous Events */
14242      IT_ASYNC_AFF_EVENT_STREAM = 0x04000,
14243      IT_ASYNC_AFF_SEVD_ENQUEUE_FAILURE = 0x04001,
14244      IT_ASYNC_AFF_EP_FAILURE = 0x04002,
14245      IT_ASYNC_AFF_EP_BAD_TRANSPORT_OPCODE = 0x04003,
14246      IT_ASYNC_AFF_EP_REQ_DROPPED = 0x04005,
14247      IT_ASYNC_AFF_EP_RDMAW_ACCESS_VIOLATION = 0x04006,
14248      IT_ASYNC_AFF_EP_RDMAW_CORRUPT_DATA = 0x04007,
14249      IT_ASYNC_AFF_EP_RDMAR_ACCESS_VIOLATION = 0x04008,
14250      IT_ASYNC_AFF_EP_LOCAL_ACCESS_VIOLATION = 0x04020,
14251      IT_ASYNC_AFF_EP_L_ACCESS_VIOLATION = 0x04020,
14252      IT_ASYNC_AFF_EP_L_RECV_ACCESS_VIOLATION = 0x04021,
14253      IT_ASYNC_AFF_EP_L_IRRQ_ACCESS_VIOLATION = 0x04022,
14254      IT_ASYNC_AFF_EP_L_TRANSPORT_ERROR = 0x04023,
14255      IT_ASYNC_AFF_EP_L_LLP_ERROR = 0x04024,
14256      IT_ASYNC_AFF_EP_R_ERROR = 0x04040,
14257      IT_ASYNC_AFF_EP_R_ACCESS_VIOLATION = 0x04041,
14258      IT_ASYNC_AFF_EP_R_RECV_ACCESS_VIOLATION = 0x04042,
14259      IT_ASYNC_AFF_EP_R_RECV_LENGTH_ERROR = 0x04043,
14260      IT_ASYNC_AFF_EP_SOFT_HI_WATERMARK = 0x04060,
14261      IT_ASYNC_AFF_SRQ_LOW_WATERMARK = 0x04100,
14262
14263      /* Event Stream for Unaffiliated Asynchronous Events */
14264      IT_ASYNC_UNAFF_EVENT_STREAM = 0x08000,
14265      /* 0x08001 is deprecated */
14266      IT_ASYNC_UNAFF_SPIGOT_ONLINE = 0x08002,
14267      IT_ASYNC_UNAFF_SPIGOT_OFFLINE = 0x08003,
14268      IT_ASYNC_UNAFF_SEVD_ENQUEUE_FAILURE = 0x08004,
14269
14270      /* Event Stream for Software Events */
14271      IT_SOFTWARE_EVENT_STREAM = 0x10000,
14272      IT_SOFTWARE_EVENT = 0x10001,
14273
14274      /* Event Stream for AEVD Notifications */

```

```

14275     IT_AEVD_NOTIFICATION_EVENT_STREAM = 0x20000,
14276     IT_AEVD_NOTIFICATION_EVENT = 0x20001
14277 } it_event_type_t;
14278
14279 typedef struct
14280 {
14281     it_event_type_t event_number;
14282     it_evd_handle_t aevd;
14283     it_evd_handle_t sevd;
14284 } it_aevd_notification_event_t;
14285
14286 typedef struct
14287 {
14288     it_event_type_t event_number;
14289     it_evd_handle_t evd;
14290
14291     union
14292     {
14293         it_evd_handle_t sevd;
14294         it_ep_handle_t ep;
14295         it_srq_handle_t srq;
14296     } cause;
14297 } it_affiliated_event_t;
14298
14299 #define IT_MAX_PRIV_DATA 256
14300
14301 typedef enum
14302 {
14303     IT_CN_REJ_OTHER = 0,
14304     IT_CN_REJ_TIMEOUT = 1,
14305     IT_CN_REJ_BAD_PATH = 2,
14306     IT_CN_REJ_STALE_CONN = 3,
14307     IT_CN_REJ_BAD_ORD = 4,
14308     IT_CN_REJ_RESOURCES = 5,
14309     IT_CN_REJ_BAD_CONN_PARMS = 6
14310 } it_conn_reject_code_t;
14311
14312 typedef struct
14313 {
14314     it_event_type_t event_number;
14315     it_evd_handle_t evd;
14316     it_cn_est_identifier_t cn_est_id;
14317     it_ep_handle_t ep;
14318     uint32_t rdma_read_ird;
14319     uint32_t rdma_read_ord;
14320     it_path_t dst_path;
14321     it_conn_reject_code_t reject_reason_code;
14322     unsigned char private_data[IT_MAX_PRIV_DATA];
14323     it_boolean_t private_data_present;
14324 } it_connection_event_t;
14325
14326 typedef enum
14327 {
14328     IT_UD_SVC_EP_INFO_VALID = 0,

```

```

14329         IT_UD_SVC_ID_NOT_SUPPORTED = 1,
14330         IT_UD_SVC_REQ_REJECTED = 2,
14331         IT_UD_NO_EP_AVAILABLE = 3,
14332         IT_UD_REQ_REDIRECTED = 4
14333     } it_ud_svc_req_status_t;
14334
14335     typedef struct
14336     {
14337         it_event_type_t event_number;
14338         it_evd_handle_t evd;
14339         it_ud_svc_req_handle_t ud_svc;
14340         it_ud_svc_req_status_t status;
14341         it_remote_ep_info_t ep_info;
14342         it_path_t dst_path;
14343         unsigned char private_data[IT_MAX_PRIV_DATA];
14344         it_boolean_t private_data_present;
14345     } it_ud_svc_reply_event_t;
14346
14347     typedef struct
14348     {
14349         it_event_type_t event_number;
14350         it_evd_handle_t evd;
14351         it_cn_est_identifier_t cn_est_id;
14352         it_conn_qual_t conn_qual;
14353         it_net_addr_t source_addr;
14354         size_t spigot_id;
14355         uint32_t max_message_size;
14356         uint32_t rdma_read_ird;
14357         uint32_t rdma_read_ord;
14358         unsigned char private_data[IT_MAX_PRIV_DATA];
14359         it_boolean_t private_data_present;
14360     } it_conn_request_event_t;
14361
14362     typedef struct
14363     {
14364         it_event_type_t event_number;
14365         it_evd_handle_t evd;
14366         it_ud_svc_req_identifier_t ud_svc_req_id;
14367         it_conn_qual_t conn_qual;
14368         it_net_addr_t source_addr;
14369         size_t spigot_id;
14370         unsigned char private_data[IT_MAX_PRIV_DATA];
14371         it_boolean_t private_data_present;
14372     } it_ud_svc_request_event_t;
14373
14374     typedef enum
14375     {
14376         IT_UD_IB_GRH_PRESENT = 0x01
14377     } it_dto_ud_flags_t;
14378
14379     typedef struct
14380     {
14381         it_event_type_t event_number;
14382         it_evd_handle_t evd;

```

```

14383         it_ep_handle_t ep;
14384         it_dto_cookie_t cookie;
14385         it_dto_status_t dto_status;
14386         uint32_t transferred_length;
14387         it_handle_t unlinked_mr_handle;
14388     } it_dto_cmpl_event_t;
14389
14390     typedef struct
14391     {
14392         it_event_type_t event_number;
14393         it_evd_handle_t evd;
14394         it_ep_handle_t ep;
14395         it_dto_cookie_t cookie;
14396         it_dto_status_t dto_status;
14397         uint32_t transferred_length;
14398         it_handle_t unlinked_mr_handle;
14399         it_dto_ud_flags_t flags;
14400         it_ud_ep_id_t ud_ep_id;
14401         it_path_t src_path;
14402     } it_all_dto_cmpl_event_t;
14403
14404     typedef struct
14405     {
14406         it_event_type_t event_number;
14407         it_evd_handle_t evd;
14408         void *data;
14409     } it_software_event_t;
14410
14411     typedef struct
14412     {
14413         it_event_type_t event_number;
14414         it_evd_handle_t evd;
14415         it_ia_handle_t ia;
14416
14417         size_t spigot_id;
14418     } it_unaffiliated_event_t;
14419
14420     typedef struct
14421     {
14422         it_event_type_t event_number;
14423         it_evd_handle_t evd;
14424     } it_any_event_t;
14425
14426     typedef union
14427     {
14428         /*
14429         * The following two union elements are
14430         * available for programming convenience.
14431         *
14432         * The event_number may be used to determine the
14433         * it_event_type_t of any Event. it_any_event_t
14434         * allows the EVD to be determined as well.
14435         */
14436         it_event_type_t event_number;

```

```

14437     it_any_event_t any;
14438
14439     /*
14440     * The remaining union elements correspond to
14441     * the various it_event_type_t types.
14442     */
14443
14444     /*
14445     * The following two Event structures
14446     * support the IT_DTO_EVENT_STREAM Event Stream.
14447     *
14448     * it_dto_cmpl_event_t supports
14449     * only the following events:
14450     *     IT_DTO_SEND_CMPL_EVENT
14451     *     IT_DTO_RC_RECV_CMPL_EVENT
14452     *     IT_DTO_RDMA_WRITE_CMPL_EVENT
14453     *     IT_DTO_RDMA_READ_CMPL_EVENT
14454     *     IT_RMR_BIND_CMPL_EVENT = IT_RMR_LINK_CMPL_EVENT
14455     *     IT_DTO_FETCH_ADD_CMPL_EVENT
14456     *     IT_DTO_CMP_SWAP_CMPL_EVENT
14457     *     IT_LMR_LINK_CMPL_EVENT
14458     *
14459     * it_all_dto_cmpl_event_t supports all
14460     * possible DTO and RMR events:
14461     *     IT_DTO_SEND_CMPL_EVENT
14462     *     IT_DTO_RC_RECV_CMPL_EVENT
14463     *     IT_DTO_UD_RECV_CMPL_EVENT
14464     *     IT_DTO_RDMA_WRITE_CMPL_EVENT
14465     *     IT_DTO_RDMA_READ_CMPL_EVENT
14466     *     IT_RMR_BIND_CMPL_EVENT = IT_RMR_LINK_CMPL_EVENT
14467     *     IT_DTO_FETCH_ADD_CMPL_EVENT
14468     *     IT_DTO_CMP_SWAP_CMPL_EVENT
14469     *     IT_LMR_LINK_CMPL_EVENT
14470     */
14471     it_dto_cmpl_event_t dto_cmpl;
14472     it_all_dto_cmpl_event_t all_dto_cmpl;
14473
14474     /*
14475     * The following two Event structures
14476     * support the IT_CM_REQ_EVENT_STREAM Event
14477     * stream:
14478     *
14479     * it_conn_request_event_t supports:
14480     *     IT_CM_REQ_CONN_REQUEST_EVENT
14481     *
14482     * it_ud_svc_request_event_t supports:
14483     *     IT_CM_REQ_UD_SERVICE_REQUEST_EVENT
14484     */
14485     it_conn_request_event_t conn_req;
14486     it_ud_svc_request_event_t ud_svc_request;
14487
14488     /*
14489     * The following two Event structures
14490     * support the IT_CM_MSG_EVENT_STREAM Event

```

```

14491     * stream:
14492     *
14493     * it_connection_event_t supports:
14494     *     IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT
14495     *     IT_CM_MSG_CONN_ESTABLISHED_EVENT
14496     *     IT_CM_MSG_CONN_PEER_REJECT_EVENT
14497     *     IT_CM_MSG_CONN_NONPEER_REJECT_EVENT
14498     *     IT_CM_MSG_CONN_DISCONNECT_EVENT
14499     *     IT_CM_MSG_CONN_BROKEN_EVENT
14500     *
14501     * it_ud_svc_reply_event_t supports:
14502     *     IT_CM_MSG_UD_SERVICE_REPLY_EVENT
14503     */
14504 it_connection_event_t conn;
14505 it_ud_svc_reply_event_t ud_svc_reply;
14506
14507 /*
14508  * it_affiliated_event_t supports
14509  * the following Event Stream:
14510  *     IT_ASYNC_AFF_EVENT_STREAM
14511  */
14512 it_affiliated_event_t aff_async;
14513
14514 /*
14515  * it_unaffiliated_event_t supports
14516  * the following Event Stream:
14517  *     IT_ASYNC_UNAFF_EVENT_STREAM
14518  */
14519 it_unaffiliated_event_t unaff_async;
14520
14521 /*
14522  * it_software_event_t supports
14523  * the following Event Stream:
14524  *     IT_SOFTWARE_EVENT_STREAM
14525  */
14526 it_software_event_t sw;
14527
14528 /*
14529  * it_aevd_notification_event_t supports
14530  * the following Event Stream:
14531  *     IT_AEVD_NOTIFICATION_EVENT_STREAM
14532  */
14533 it_aevd_notification_event_t aevd_notify;
14534 } it_event_t;
14535
14536 typedef enum
14537 {
14538     IT_EP_STATE_UNCONNECTED = 0,
14539     IT_EP_STATE_ACTIVE1_CONNECTION_PENDING = 1,
14540     IT_EP_STATE_ACTIVE2_CONNECTION_PENDING = 2,
14541     IT_EP_STATE_PASSIVE_CONNECTION_PENDING = 3,
14542     IT_EP_STATE_CONNECTED = 4,
14543     IT_EP_STATE_NONOPERATIONAL = 5,
14544     IT_EP_STATE_PASSIVE_WAIT_RDMA_TRANS_REQ = 6

```

```

14545     } it_ep_state_rc_t;
14546
14547     typedef enum
14548     {
14549         IT_EP_STATE_UD_NONOPERATIONAL = 0,
14550         IT_EP_STATE_UD_OPERATIONAL = 1
14551     } it_ep_state_ud_t;
14552
14553     typedef union
14554     {
14555         it_ep_state_rc_t rc;
14556         it_ep_state_ud_t ud;
14557     } it_ep_state_t;
14558
14559     typedef struct
14560     {
14561         it_addr_handle_t addr;
14562         it_remote_ep_info_t ep_info;
14563     } it_ib_ud_addr_t;
14564
14565     typedef enum
14566     {
14567         IT_DG_TYPE_IB_UD
14568     } it_dg_type_t;
14569
14570     typedef struct
14571     {
14572         it_dg_type_t type; /* IT_DG_TYPE_IB_UD */
14573         union
14574         {
14575             it_ib_ud_addr_t ud;
14576         } addr;
14577     } it_dg_remote_ep_addr_t;
14578
14579     typedef enum
14580     {
14581         IT_AH_PATH_COMPLETE = 0x1
14582     } it_ah_flags_t;
14583
14584     typedef enum
14585     {
14586         IT_ADDR_PARAM_ALL = 0x0001,
14587         IT_ADDR_PARAM_IA = 0x0002,
14588         IT_ADDR_PARAM_PZ = 0x0004,
14589         IT_ADDR_PARAM_PATH = 0x0008
14590     } it_addr_param_mask_t;
14591
14592     typedef struct
14593     {
14594         it_ia_handle_t ia; /* IT_ADDR_PARAM_IA */
14595         it_pz_handle_t pz; /* IT_ADDR_PARAM_PZ */
14596         it_path_t path; /* IT_ADDR_PARAM_PATH */
14597     } it_addr_param_t;
14598

```

```

14599     typedef struct
14600     {
14601
14602         /* Remote CM Response Timeout, as defined in the REQ
14603            message for the IB CM protocol */
14604         uint8_t remote_cm_timeout:5;
14605
14606         /* Local CM Response Timeout, as defined in the REQ
14607            message for the IB CM protocol */
14608         uint8_t local_cm_timeout:5;
14609
14610         /* Retry Count, as defined in the REQ message for the
14611            IB CM protocol */
14612         uint8_t retry_count:3;
14613
14614         /* RNR Retry Count, as defined in the REQ message for
14615            the IB CM protocol */
14616         uint8_t rnr_retry_count:3;
14617
14618         /* Max CM retries, as defined in the REQ message for
14619            the IB CM protocol */
14620         uint8_t max_cm_retries:4;
14621
14622         /* Local ACK Timeout, as defined in the REQ message
14623            for the IB CM protocol */
14624         uint8_t local_ack_timeout:5;
14625
14626     } it_ib_conn_attributes_t;
14627
14628     typedef struct
14629     {
14630
14631         /* VIA currently has no transport-specific connection
14632            attributes. A dummy entry is defined to allow ANSI
14633            compilation. */
14634         void *unused;
14635
14636     } it_via_conn_attributes_t;
14637
14638     typedef struct
14639     {
14640
14641         /* iWARP currently has no transport-specific connection
14642            attributes. A dummy entry is defined to allow ANSI
14643            compilation. */
14644         void *unused;
14645
14646     } it_iwarp_conn_attributes_t;
14647
14648     typedef union
14649     {
14650         it_ib_conn_attributes_t ib;
14651         it_via_conn_attributes_t via;
14652         it_iwarp_conn_attributes_t iwarp;

```

```

14653     } it_conn_attributes_t;
14654
14655     typedef enum
14656     {
14657         IT_CONNECT_FLAG_TWO_WAY = 0x0001,
14658         IT_CONNECT_FLAG_THREE_WAY = 0x0002,
14659         IT_CONNECT_SUPPRESS_IRD_ORD = 0x0004
14660     } it_cn_est_flags_t;
14661
14662     typedef struct
14663     {
14664         it_ia_handle_t ia; /* IT_EP_PARAM_IA */
14665         size_t spigot_id; /* IT_EP_PARAM_SPIGOT */
14666         it_ep_state_t ep_state; /* IT_EP_PARAM_STATE */
14667         it_transport_service_type_t service_type;
14668         /* IT_EP_PARAM_SERV_TYPE */
14669         it_path_t dst_path; /* IT_EP_PARAM_PATH */
14670         it_pz_handle_t pz; /* IT_EP_PARAM_PZ */
14671         it_evd_handle_t request_sevd; /* IT_EP_PARAM_REQ_SEVD */
14672         it_evd_handle_t recv_sevd; /* IT_EP_PARAM_RECV_SEVD */
14673         it_evd_handle_t connect_sevd; /* IT_EP_PARAM_CONN_SEVD */
14674         it_ep_attributes_t attr;
14675         /* See it_ep_attributes_t for mask flags for attr. */
14676     } it_ep_param_t;
14677
14678     typedef enum
14679     {
14680         IT_EP_NO_FLAG = 0x00,
14681         IT_EP_REUSEADDR = 0x01,
14682         IT_EP_SRQ = 0x02
14683     } it_ep_rc_creation_flags_t;
14684
14685     #define IT_THRESHOLD_DISABLE 0
14686
14687     typedef enum
14688     {
14689         IT_EVD_DEQUEUE_NOTIFICATIONS = 0x01,
14690         IT_EVD_CREATE_FD = 0x02,
14691         IT_EVD_OVERFLOW_DEFAULT = 0x04,
14692         IT_EVD_OVERFLOW_NOTIFY = 0x08,
14693         IT_EVD_OVERFLOW_AUTO_RESET = 0x10
14694     } it_evd_flags_t;
14695
14696     typedef enum
14697     {
14698         IT_EVD_PARAM_ALL = 0x000001,
14699         IT_EVD_PARAM_IA = 0x000002,
14700         IT_EVD_PARAM_EVENT_NUMBER = 0x000004,
14701         IT_EVD_PARAM_FLAG = 0x000008,
14702         IT_EVD_PARAM_QUEUE_SIZE = 0x000010,
14703         IT_EVD_PARAM_THRESHOLD = 0x000020,
14704         IT_EVD_PARAM_AEVD_HANDLE = 0x000040,
14705         IT_EVD_PARAM_FD = 0x000080,
14706         IT_EVD_PARAM_BOUND = 0x000100,

```

```

14707     IT_EVD_PARAM_ENABLED = 0x000200,
14708     IT_EVD_PARAM_OVERFLOWED = 0x000400,
14709     IT_EVD_PARAM_CALLBACK = 0x000800
14710 } it_evd_param_mask_t;
14711
14712 typedef struct
14713 {
14714     it_ia_handle_t ia; /* IT_EVD_PARAM_IA */
14715     it_event_type_t event_number; /* IT_EVD_PARAM_EVENT_NUMBER */
14716     it_evd_flags_t evd_flag; /* IT_EVD_PARAM_FLAG */
14717     size_t sev_d_queue_size; /* IT_EVD_PARAM_QUEUE_SIZE */
14718     size_t sev_d_threshold; /* IT_EVD_PARAM_THRESHOLD */
14719     it_evd_handle_t aevd; /* IT_EVD_PARAM_AEVD_HANDLE */
14720     int fd; /* IT_EVD_PARAM_FD */
14721     it_boolean_t evd_bound; /* IT_EVD_PARAM_BOUND */
14722     it_boolean_t evd_enabled; /* IT_EVD_PARAM_ENABLED */
14723     it_boolean_t evd_overflowed; /* IT_EVD_PARAM_OVERFLOWED */
14724     it_boolean_t evd_callback; /* IT_EVD_PARAM_CALLBACK */
14725 } it_evd_param_t;
14726
14727 typedef struct
14728 {
14729     /* Most recent major version number of the IT-API supported by the
14730      * Interface. */
14731     uint32_t major_version;
14732
14733     /* Most recent minor version number of the IT-API supported by the
14734      * Interface. */
14735     uint32_t minor_version;
14736
14737     /* The transport that the Interface uses, as defined in
14738      * it_ia_info_t. */
14739     it_transport_type_t transport_type;
14740
14741     /* The name of the Interface, suitable for input to it_ia_create.
14742      * The name is a string of maximum length IT_INTERFACE_NAME_SIZE,
14743      * including the terminating NULL character. */
14744     char name[IT_INTERFACE_NAME_SIZE];
14745 } it_interface_t;
14746
14747 typedef enum
14748 {
14749     IT_LISTEN_NO_FLAG = 0x0000,
14750     IT_LISTEN_CONN_QUAL_INPUT = 0x0001,
14751     IT_LISTEN_SUPPRESS_IRD_ORD = 0x0002
14752 } it_listen_flags_t;
14753
14754 typedef enum
14755 {
14756     IT_LISTEN_PARAM_ALL = 0x0001,
14757     IT_LISTEN_PARAM_IA_HANDLE = 0x0002,
14758     IT_LISTEN_PARAM_SPIGOT_ID = 0x0004,
14759     IT_LISTEN_PARAM_CONNECT_EVD = 0x0008,
14760

```

```

14761     IT_LISTEN_PARAM_CONN_QUAL = 0x0010
14762 } it_listen_param_mask_t;
14763
14764 typedef struct
14765 {
14766     it_ia_handle_t ia_handle; /* IT_LISTEN_PARAM_IA_HANDLE */
14767     size_t spigot_id; /* IT_LISTEN_PARAM_SPIGOT_ID */
14768     it_evd_handle_t connect_evd; /* IT_LISTEN_PARAM_CONNECT_EVD */
14769     it_conn_qual_t connect_qual; /* IT_LISTEN_PARAM_CONN_QUAL */
14770 } it_listen_param_t;
14771
14772 typedef enum
14773 {
14774     IT_LMR_PARAM_ALL = 0x000001,
14775     IT_LMR_PARAM_IA = 0x000002,
14776     IT_LMR_PARAM_PZ = 0x000004,
14777     IT_LMR_PARAM_ADDR = 0x000008,
14778     IT_LMR_PARAM_LENGTH = 0x000010,
14779     IT_LMR_PARAM_MEM_PRIV = 0x000020,
14780     IT_LMR_PARAM_FLAG = 0x000040,
14781     IT_LMR_PARAM_SHARED_ID = 0x000080,
14782     IT_LMR_PARAM_RMR_CONTEXT = 0x000100,
14783     IT_LMR_PARAM_ACTUAL_ADDR = 0x000200,
14784     IT_LMR_PARAM_ACTUAL_LENGTH = 0x000400,
14785     IT_LMR_PARAM_ADDR_MODE = 0x000800,
14786     IT_LMR_PARAM_LINKED = 0x001000,
14787     IT_LMR_PARAM_IOBL_NUM_ELTS = 0x002000,
14788     IT_LMR_PARAM_IOBL_TYPE = 0x004000
14789 } it_lmr_param_mask_t;
14790
14791 typedef struct
14792 {
14793     it_ia_handle_t ia; /* IT_LMR_PARAM_IA */
14794     it_pz_handle_t pz; /* IT_LMR_PARAM_PZ */
14795     void *addr; /* IT_LMR_PARAM_ADDR */
14796     it_length_t length; /* IT_LMR_PARAM_LENGTH */
14797     it_mem_priv_t privs; /* IT_LMR_PARAM_MEM_PRIV */
14798     it_lmr_flag_t flags; /* IT_LMR_PARAM_FLAG */
14799     uint32_t shared_id; /* IT_LMR_PARAM_SHARED_ID */
14800     it_rmr_context_t rmr_context; /* IT_LMR_PARAM_RMR_CONTEXT */
14801     void *actual_addr; /* IT_LMR_PARAM_ACTUAL_ADDR */
14802     it_length_t actual_length; /* IT_LMR_PARAM_ACTUAL_LENGTH */
14803     it_addr_mode_t addr_mode; /* IT_LMR_PARAM_ADDR_MODE */
14804     it_boolean_t linked; /* IT_LMR_PARAM_LINKED */
14805     size_t iobl_num_elts; /* IT_LMR_PARAM_IOBL_NUM_ELTS */
14806     it_iobl_type_t iobl_type; /* IT_LMR_PARAM_IOBL_TYPE */
14807 } it_lmr_param_t;
14808
14809 typedef uint64_t it_rdma_addr_t;
14810
14811 typedef enum
14812 {
14813     IT_PZ_PARAM_ALL = 0x01,
14814     IT_PZ_PARAM_IA = 0x02

```

```

14815     } it_pz_param_mask_t;
14816
14817     typedef struct
14818     {
14819         it_ia_handle_t ia; /* IT_PZ_PARAM_IA */
14820     } it_pz_param_t;
14821
14822     typedef enum
14823     {
14824         IT_RMR_PARAM_ALL = 0x000001,
14825         IT_RMR_PARAM_IA = 0x000002,
14826         IT_RMR_PARAM_PZ = 0x000004,
14827         IT_RMR_PARAM_LINKED = 0x000008,
14828         IT_RMR_PARAM_BOUND = 0x000008,
14829         /* deprecated by IT_RMR_PARAM_LINKED */
14830         IT_RMR_PARAM_LMR = 0x000010,
14831         IT_RMR_PARAM_ADDR = 0x000020,
14832         IT_RMR_PARAM_LENGTH = 0x000040,
14833         IT_RMR_PARAM_MEM_PRIV = 0x000080,
14834         IT_RMR_PARAM_RMR_CONTEXT = 0x000100,
14835         IT_RMR_PARAM_TYPE = 0x000200,
14836         IT_RMR_PARAM_ADDR_MODE = 0x000400
14837     } it_rmr_param_mask_t;
14838
14839     /* Need to use "bound" rather than "linked" in IT-API 1.0. */
14840     #ifdef ITAPI_ENABLE_V10_BINDINGS
14841
14842     typedef struct
14843     {
14844         it_ia_handle_t ia; /* IT_RMR_PARAM_IA */
14845         it_pz_handle_t pz; /* IT_RMR_PARAM_PZ */
14846         it_boolean_t bound; /* IT_RMR_PARAM_BOUND */
14847         it_lmr_handle_t lmr; /* IT_RMR_PARAM_LMR */
14848         void *addr; /* IT_RMR_PARAM_ADDR */
14849         it_length_t length; /* IT_RMR_PARAM_LENGTH */
14850         it_mem_priv_t privs; /* IT_RMR_PARAM_MEM_PRIV */
14851         it_rmr_context_t rmr_context; /* IT_RMR_PARAM_RMR_CONTEXT */
14852         it_rmr_type_t type; /* IT_RMR_PARAM_TYPE */
14853         it_addr_mode_t addr_mode; /* IT_RMR_PARAM_ADDR_MODE */
14854     } it_rmr_param_t;
14855
14856     #else /* #ifdef ITAPI_ENABLE_V10_BINDINGS */
14857
14858     typedef struct
14859     {
14860         it_ia_handle_t ia; /* IT_RMR_PARAM_IA */
14861         it_pz_handle_t pz; /* IT_RMR_PARAM_PZ */
14862         it_boolean_t linked; /* IT_RMR_PARAM_LINKED */
14863         it_lmr_handle_t lmr; /* IT_RMR_PARAM_LMR */
14864         void *addr; /* IT_RMR_PARAM_ADDR */
14865         it_length_t length; /* IT_RMR_PARAM_LENGTH */
14866         it_mem_priv_t privs; /* IT_RMR_PARAM_MEM_PRIV */
14867         it_rmr_context_t rmr_context; /* IT_RMR_PARAM_RMR_CONTEXT */
14868         it_rmr_type_t type; /* IT_RMR_PARAM_TYPE */

```

```

14869     it_addr_mode_t addr_mode; /* IT_RMR_PARAM_ADDR_MODE */
14870 } it_rmr_param_t;
14871
14872 #endif /* #ifdef ITAPI_ENABLE_V10_BINDINGS */
14873
14874 typedef enum
14875 {
14876     IT_SC_DEFAULT = 0x0000,
14877     IT_SC_NO_REQ_REP = 0x0001,
14878 } it_sc_flags_t;
14879
14880 typedef enum
14881 {
14882     IT_SRQ_PARAM_ALL = 0x000001,
14883     IT_SRQ_PARAM_IA = 0x000002,
14884     IT_SRQ_PARAM_PZ = 0x000004,
14885     IT_SRQ_PARAM_MAX_RECV_DTO = 0x000008,
14886     IT_SRQ_PARAM_MAX_RECV_SEG = 0x000010,
14887     IT_SRQ_PARAM_LOW_WATERMARK = 0x000020
14888 } it_srq_param_mask_t;
14889
14890 typedef struct
14891 {
14892     it_ia_handle_t ia; /* IT_SRQ_PARAM_IA */
14893     it_pz_handle_t pz; /* IT_SRQ_PARAM_PZ */
14894     size_t max_recv_dtos; /* IT_SRQ_PARAM_MAX_RECV_DTO */
14895     size_t max_recv_segs; /* IT_SRQ_PARAM_MAX_RECV_SEG */
14896     size_t low_watermark; /* IT_SRQ_PARAM_LOW_WATERMARK */
14897 } it_srq_param_t;
14898
14899 typedef enum
14900 {
14901     IT_UD_PARAM_ALL = 0x00000001,
14902     IT_UD_PARAM_IA_HANDLE = 0x00000002,
14903     IT_UD_PARAM_REQ_ID = 0x00000004,
14904     IT_UD_PARAM_REPLY_EVD = 0x00000008,
14905     IT_UD_PARAM_CONN_QUAL = 0x00000010,
14906     IT_UD_PARAM_DEST_PATH = 0x00000020,
14907     IT_UD_PARAM_PRIV_DATA = 0x00000040,
14908     IT_UD_PARAM_PRIV_DATA_LENGTH = 0x00000080
14909 } it_ud_svc_req_param_mask_t;
14910
14911 typedef struct
14912 {
14913     it_ia_handle_t ia; /* IT_UD_PARAM_IA_HANDLE */
14914     uint32_t request_id; /* IT_UD_PARAM_REQ_ID */
14915     it_evd_handle_t reply_evd; /* IT_UD_PARAM_REPLY_EVD */
14916     it_conn_qual_t conn_qual; /* IT_UD_PARAM_CONN_QUAL */
14917     it_path_t destination_path; /* IT_UD_PARAM_DEST_PATH */
14918     unsigned char private_data[IT_MAX_PRIV_DATA];
14919     /* IT_UD_PARAM_PRIV_DATA */
14920     size_t private_data_length; /* IT_UD_PARAM_PRIV_DATA_LENGTH */
14921 } it_ud_svc_req_param_t;
14922

```

```

14923     typedef void (*it_evd_callback_rtn_t)(
14924         IN it_evd_handle_t evd,
14925         IN void *arg
14926     );
14927
14928     typedef enum
14929     {
14930         IT_COMPARE_AND_SWAP,
14931         IT_FETCH_AND_ADD
14932     } it_atomic_op_t;
14933
14934     /* prototypes */
14935
14936     #ifdef ITAPI_ENABLE_V10_BINDINGS
14937     /*
14938         Backwards compatibility mode:
14939         For functions whose signature changed from v1.0 to v2.0,
14940         <it_api.h> converts v1.0 function names to explicit v1.0
14941         function names. Functions such as it_lmr_create continue
14942         to use v1.0 signatures.
14943     */
14944     #define it_lmr_create    it_lmr_create10
14945     #define it_rmr_create    it_rmr_create10
14946     #define it_rmr_bind     it_rmr_bind10
14947
14948     #elif (defined(ITAPI_ENABLE_V20_BINDINGS))
14949
14950     /*
14951         Full v2.0 functionality:
14952         For functions whose signature changed from v1.0 to v2.0,
14953         <it_api.h> converts v1.0 function names to explicit v2.0
14954         function names. Functions such as it_lmr_create use the
14955         new v2.0 signatures.
14956     */
14957     #define it_lmr_create    it_lmr_create20
14958     #define it_rmr_create    it_rmr_create20
14959     #define it_rmr_bind     it_rmr_link
14960
14961     #elif (defined(ITAPI_ENABLE_V21_BINDINGS))
14962
14963     /*
14964         Full v2.1 functionality:
14965         For functions whose signature changed relative to an earlier
14966         Version, <it_api.h> converts the function name to an explicit
14967         function name carrying as a suffix the IT-API major and minor
14968         version number in which the signature changed most recently.
14969     */
14970     #define it_lmr_create    it_lmr_create21
14971     #define it_rmr_create    it_rmr_create20
14972     #define it_rmr_bind     it_rmr_link
14973
14974     #endif /* ifdef ITAPI_ENABLE_V10_BINDINGS */
14975
14976     it_status_t it_address_handle_create (

```

```

14977         IN          it_pz_handle_t    pz_handle,
14978         IN  const  it_path_t          *destination_path,
14979         IN          it_ah_flags_t     ah_flags,
14980         OUT         it_addr_handle_t  *addr_handle
14981     );
14982     it_status_t it_address_handle_free (
14983         IN  it_addr_handle_t  addr_handle
14984     );
14985     it_status_t it_address_handle_modify (
14986         IN          it_addr_handle_t    addr_handle,
14987         IN          it_addr_param_mask_t mask,
14988         IN  const  it_addr_param_t     *params
14989     );
14990     it_status_t it_address_handle_query (
14991         IN  it_addr_handle_t    addr_handle,
14992         IN  it_addr_param_mask_t mask,
14993         OUT it_addr_param_t     *params
14994     );
14995     it_status_t it_convert_net_addr (
14996         IN  const  it_net_addr_t      *source_addr,
14997         IN          it_net_addr_type_t addr_type,
14998         OUT         it_net_addr_t      *destination_addr
14999     );
15000     it_status_t it_ep_accept (
15001         IN          it_ep_handle_t     ep_handle,
15002         IN          it_cn_est_identifrier_t cn_est_id,
15003         IN  const  unsigned char       *private_data,
15004         IN          size_t              private_data_length
15005     );
15006     it_status_t it_ep_connect (
15007         IN          it_ep_handle_t     ep_handle,
15008         IN  const  it_path_t           *path,
15009         IN  const  it_conn_attributes_t *conn_attr,
15010         IN  const  it_conn_qual_t      *connect_qual,
15011         IN          it_cn_est_flags_t   cn_est_flags,
15012         IN  const  unsigned char       *private_data,
15013         IN          size_t              private_data_length
15014     );
15015     it_status_t it_ep_disconnect (
15016         IN          it_ep_handle_t     ep_handle,
15017         IN  const  unsigned char       *private_data,
15018         IN          size_t              private_data_length
15019     );
15020     it_status_t it_ep_free (
15021         IN  it_ep_handle_t  ep_handle
15022     );
15023     it_status_t it_ep_modify (
15024         IN          it_ep_handle_t     ep_handle,
15025         IN          it_ep_param_mask_t mask,
15026         IN  const  it_ep_attributes_t *ep_attr
15027     );
15028     it_status_t it_ep_query (
15029         IN  it_ep_handle_t    ep_handle,
15030         IN  it_ep_param_mask_t mask,

```

```

15031         OUT it_ep_param_t      *params
15032     );
15033     it_status_t it_ep_rc_create (
15034         IN         it_pz_handle_t      pz_handle,
15035         IN         it_evd_handle_t     request_sevd_handle,
15036         IN         it_evd_handle_t     recv_sevd_handle,
15037         IN         it_evd_handle_t     connect_sevd_handle,
15038         IN         it_ep_rc_creation_flags_t flags,
15039         IN  const  it_ep_attributes_t   *ep_attr,
15040         OUT        it_ep_handle_t      *ep_handle
15041     );
15042     it_status_t it_ep_reset (
15043         IN  it_ep_handle_t ep_handle
15044     );
15045     it_status_t it_ep_ud_create (
15046         IN         it_pz_handle_t      pz_handle,
15047         IN         it_evd_handle_t     request_sevd_handle,
15048         IN         it_evd_handle_t     recv_sevd_handle,
15049         IN  const  it_ep_attributes_t *ep_attr,
15050         IN         size_t              spigot_id,
15051         OUT        it_ep_handle_t      *ep_handle
15052     );
15053     it_status_t it_evd_callback_attach (
15054         IN  it_evd_handle_t      evd_handle,
15055         IN  it_evd_callback_rtn_t callback,
15056         IN  void                 *arg
15057     );
15058     it_status_t it_evd_callback_detach (
15059         IN  it_evd_handle_t evd_handle
15060     );
15061     it_status_t it_evd_create (
15062         IN  it_ia_handle_t      ia_handle,
15063         IN  it_event_type_t     event_number,
15064         IN  it_evd_flags_t      evd_flag,
15065         IN  size_t              sevd_queue_size,
15066         IN  size_t              sevd_threshold,
15067         IN  it_evd_handle_t     aeved_handle,
15068         OUT it_evd_handle_t     *evd_handle,
15069         OUT int                 *fd
15070     );
15071     it_status_t it_evd_dequeue (
15072         IN  it_evd_handle_t evd_handle,
15073         OUT it_event_t      *event
15074     );
15075     it_status_t it_evd_free (
15076         IN  it_evd_handle_t evd_handle
15077     );
15078     it_status_t it_evd_modify (
15079         IN         it_evd_handle_t      evd_handle,
15080         IN         it_evd_param_mask_t mask,
15081         IN  const  it_evd_param_t      *params
15082     );
15083     it_status_t it_evd_post_se (
15084         IN         it_evd_handle_t evd_handle,

```

```

15085         IN const void          *event
15086     );
15087 it_status_t it_evd_query (
15088     IN it_evd_handle_t      evd_handle,
15089     IN it_evd_param_mask_t mask,
15090     OUT it_evd_param_t      *params
15091 );
15092 it_status_t it_evd_wait (
15093     IN it_evd_handle_t      evd_handle,
15094     IN uint64_t             timeout,
15095     OUT it_event_t          *event,
15096     OUT size_t              *nmore
15097 );
15098 it_status_t it_get_consumer_context (
15099     IN it_handle_t          handle,
15100     OUT it_context_t        *context
15101 );
15102 it_status_t it_get_handle_type (
15103     IN it_handle_t          handle,
15104     OUT it_handle_type_enum_t *type_of_handle
15105 );
15106 it_status_t it_get_pathinfo (
15107     IN it_ia_handle_t        ia_handle,
15108     IN size_t                spigot_id,
15109     IN const it_net_addr_t   *net_addr,
15110     IN OUT size_t            *num_paths,
15111     OUT size_t               *total_paths,
15112     OUT it_path_t           *paths
15113 );
15114 it_status_t it_handoff (
15115     IN const it_conn_qual_t   *conn_qual,
15116     IN size_t                spigot_id,
15117     IN it_cn_est_identifier_t cn_est_id
15118 );
15119 uint64_t it_hton64 (
15120     uint64_t hostint
15121 );
15122 uint64_t it_ntoh64 (
15123     uint64_t netint
15124 );
15125 it_status_t it_ia_create (
15126     IN const char            *name,
15127     IN uint32_t              major_version,
15128     IN uint32_t              minor_version,
15129     OUT it_ia_handle_t       *ia_handle
15130 );
15131 it_status_t it_ia_free (
15132     IN it_ia_handle_t        ia_handle
15133 );
15134 void it_ia_info_free (
15135     IN it_ia_info_t          *ia_info
15136 );
15137 it_status_t it_ia_query (
15138     IN it_ia_handle_t        ia_handle,

```

```

15139     OUT it_ia_info_t **ia_info
15140 );
15141 void it_interface_list (
15142     OUT it_interface_t *interfaces,
15143     IN  OUT size_t      *num_interfaces,
15144     IN  OUT size_t      *total_interfaces
15145 );
15146 it_status_t it_listen_create (
15147     IN  it_ia_handle_t   ia_handle,
15148     IN  size_t           spigot_id,
15149     IN  it_evd_handle_t  connect_evd,
15150     IN  it_listen_flags_t flags,
15151     IN  OUT it_conn_qual_t *conn_qual,
15152     OUT it_listen_handle_t *listen_handle
15153 );
15154 it_status_t it_listen_free (
15155     IN  it_listen_handle_t listen_handle
15156 );
15157 it_status_t it_listen_query (
15158     IN  it_listen_handle_t listen_handle,
15159     IN  it_listen_param_mask_t mask,
15160     OUT it_listen_param_t *params
15161 );
15162 /*
15163     it_lmr_create10 is provided for backwards-compatibility
15164     and may be dropped in a future IT-API version.
15165
15166     Calls with a suffix "10" can be typically implemented through
15167     direct inlining to the corresponding calls with suffix "20",
15168     providing default arguments as necessary.
15169 */
15170 it_status_t it_lmr_create10 (
15171     IN  it_pz_handle_t   pz_handle,
15172     IN  void              *addr,
15173     IN  it_length_t      length,
15174     IN  it_mem_priv_t     privs,
15175     IN  it_lmr_flag_t     flags,
15176     IN  uint32_t          shared_id,
15177     OUT it_lmr_handle_t   *lmr_handle,
15178     IN  OUT it_rmr_context_t *rmr_context
15179 );
15180 /*
15181     it_lmr_create20 provides the v2.0 functionality
15182     and may be renamed to it_lmr_create in a future IT-API version.
15183 */
15184 it_status_t it_lmr_create20 (
15185     IN  it_pz_handle_t   pz_handle,
15186     IN  void              *addr,
15187     IN  it_length_t      length,
15188     IN  it_addr_mode_t   addr_mode,
15189     IN  it_mem_priv_t     privs,
15190     IN  it_lmr_flag_t     flags,
15191     IN  uint32_t          shared_id,
15192

```

```

15193         OUT it_lmr_handle_t      *lmr_handle,
15194         IN  OUT it_rmr_context_t *rmr_context
15195     );
15196
15197     /*
15198         it_lmr_create21 provides the v2.1 functionality
15199         and may be renamed to it_lmr_create in a future IT-API version.
15200     */
15201     it_status_t it_lmr_create21 (
15202         IN  it_pz_handle_t      pz_handle,
15203         IN  void                *addr,
15204         IN  it_iobl_t          *iobl,
15205         IN  it_length_t        length,
15206         IN  it_addr_mode_t     addr_mode,
15207         IN  it_mem_priv_t      privs,
15208         IN  it_lmr_flag_t      flags,
15209         IN  uint32_t           shared_id,
15210         OUT it_lmr_handle_t     *lmr_handle,
15211         IN  OUT it_rmr_context_t *rmr_context
15212     );
15213
15214     it_status_t it_lmr_create_unlinked (
15215         IN  it_pz_handle_t      pz_handle,
15216         IN  size_t              iobl_num_elts,
15217         IN  it_lmr_flag_t      flags,
15218         IN  uint32_t           shared_id,
15219         OUT it_lmr_handle_t     *lmr_handle
15220     );
15221     it_status_t it_lmr_flush_to_mem (
15222         IN  const it_lmr_triplet_t *local_segments,
15223         IN  size_t                num_segments
15224     );
15225     it_status_t it_lmr_free (
15226         IN  it_lmr_handle_t     lmr_handle
15227     );
15228     it_status_t it_lmr_link (
15229         IN  it_lmr_handle_t     lmr_handle,
15230         IN  const void          *addr,
15231         IN  const it_iobl_t     *iobl,
15232         IN  it_length_t        length,
15233         IN  it_addr_mode_t     addr_mode,
15234         IN  it_mem_priv_t      privs,
15235         IN  it_ep_handle_t     ep_handle,
15236         IN  it_dto_cookie_t     cookie,
15237         IN  it_dto_flags_t     dto_flags,
15238         OUT it_rmr_context_t    *rmr_context
15239     );
15240     it_status_t it_lmr_modify (
15241         IN  it_lmr_handle_t     lmr_handle,
15242         IN  it_lmr_param_mask_t mask,
15243         IN  const it_lmr_param_t *params
15244     );
15245     it_status_t it_lmr_query (
15246         IN  it_lmr_handle_t     lmr_handle,

```

```

15247         IN  it_lmr_param_mask_t mask,
15248         OUT it_lmr_param_t      *params
15249     );
15250     it_status_t it_lmr_refresh_from_mem (
15251         IN  const it_lmr_triplet_t *local_segments,
15252         IN      size_t              num_segments
15253     );
15254     it_status_t it_lmr_unlink (
15255         IN  it_lmr_handle_t lmr_handle,
15256         IN  it_ep_handle_t  ep_handle,
15257         IN  it_dto_cookie_t cookie,
15258         IN  it_dto_flags_t  dto_flags
15259     );
15260     it_rdma_addr_t it_make_rdma_addr_absolute (
15261         void *addr
15262     );
15263     it_rdma_addr_t it_make_rdma_addr_relative (
15264         it_length_t offset
15265     );
15266     it_status_t it_mem_map (
15267         IN  it_ia_handle_t ia_handle,
15268         IN  void          *vaddr,
15269         IN  uint64_t      aspace_id,
15270         IN  size_t        len,
15271         OUT it_io_addr_t  *io_addr,
15272         OUT size_t        *mapped_len
15273     );
15274     it_status_t it_mem_unmap (
15275         IN  it_ia_handle_t ia_handle,
15276         IN  it_io_addr_t  io_addr,
15277         IN  size_t        mapped_len
15278     );
15279     it_status_t it_post_atomic (
15280         IN      it_ep_handle_t  ep_handle,
15281         IN      it_atomic_op_t  op,
15282         IN      uint64_t        swap_or_add,
15283         IN      uint64_t        compare,
15284         IN  const it_lmr_triplet_t *result,
15285         IN      it_dto_cookie_t  cookie,
15286         IN      it_dto_flags_t    dto_flags,
15287         IN      it_rdma_addr_t    rdma_addr,
15288         IN      it_rmr_context_t  rmr_context
15289     );
15290     it_status_t it_post_rdma_read (
15291         IN      it_ep_handle_t  ep_handle,
15292         IN  const it_lmr_triplet_t *local_segments,
15293         IN      size_t          num_segments,
15294         IN      it_dto_cookie_t  cookie,
15295         IN      it_dto_flags_t    dto_flags,
15296         IN      it_rdma_addr_t    rdma_addr,
15297         IN      it_rmr_context_t  rmr_context
15298     );
15299     it_status_t it_post_rdma_read_to_rmr (
15300         IN      it_ep_handle_t  ep_handle,

```

```

15301         IN  const it_rmr_triplet_t *local_segments,
15302         IN      size_t             num_segments,
15303         IN      it_dto_cookie_t    cookie,
15304         IN      it_dto_flags_t     dto_flags,
15305         IN      it_rdma_addr_t     rdma_addr,
15306         IN      it_rmr_context_t   rmr_context
15307     );
15308     it_status_t it_post_rdma_write (
15309         IN      it_ep_handle_t     ep_handle,
15310         IN  const it_lmr_triplet_t *local_segments,
15311         IN      size_t             num_segments,
15312         IN      it_dto_cookie_t    cookie,
15313         IN      it_dto_flags_t     dto_flags,
15314         IN      it_rdma_addr_t     rdma_addr,
15315         IN      it_rmr_context_t   rmr_context
15316     );
15317     it_status_t it_post_recv (
15318         IN      it_handle_t        handle,
15319         IN  const it_lmr_triplet_t *local_segments,
15320         IN      size_t             num_segments,
15321         IN      it_dto_cookie_t    cookie,
15322         IN      it_dto_flags_t     dto_flags
15323     );
15324     it_status_t it_post_recvfrom (
15325         IN      it_ep_handle_t     ep_handle,
15326         IN  const it_lmr_triplet_t *local_segments,
15327         IN      size_t             num_segments,
15328         IN      it_dto_cookie_t    cookie,
15329         IN      it_dto_flags_t     dto_flags
15330     );
15331     it_status_t it_post_send (
15332         IN      it_ep_handle_t     ep_handle,
15333         IN  const it_lmr_triplet_t *local_segments,
15334         IN      size_t             num_segments,
15335         IN      it_dto_cookie_t    cookie,
15336         IN      it_dto_flags_t     dto_flags
15337     );
15338     it_status_t it_post_send_and_unlink (
15339         IN      it_ep_handle_t     ep_handle,
15340         IN  const it_lmr_triplet_t *local_segments,
15341         IN      size_t             num_segments,
15342         IN      it_dto_cookie_t    cookie,
15343         IN      it_dto_flags_t     dto_flags,
15344         IN      it_rmr_context_t   rmr_context
15345     );
15346     it_status_t it_post_sendto (
15347         IN      it_ep_handle_t     ep_handle,
15348         IN  const it_lmr_triplet_t *local_segments,
15349         IN      size_t             num_segments,
15350         IN      it_dto_cookie_t    cookie,
15351         IN      it_dto_flags_t     dto_flags,
15352         IN  const it_dg_remote_ep_addr_t *remote_ep_addr
15353     );
15354     it_status_t it_pz_create (

```

```

15355         IN it_ia_handle_t ia_handle,
15356         OUT it_pz_handle_t *pz_handle
15357     );
15358     it_status_t it_pz_free (
15359         IN it_pz_handle_t pz_handle
15360     );
15361     it_status_t it_pz_query (
15362         IN it_pz_handle_t pz_handle,
15363         IN it_pz_param_mask_t mask,
15364         OUT it_pz_param_t *params
15365     );
15366     it_status_t it_reject (
15367         IN it_cn_est_identifier_t cn_est_id,
15368         IN const unsigned char *private_data,
15369         IN size_t private_data_length
15370     );
15371
15372     /*IT-API 1.0 prototype for B/W compatibility */
15373     it_status_t it_rmr_bind10 (
15374         IN it_rmr_handle_t rmr_handle,
15375         IN it_lmr_handle_t lmr_handle,
15376         IN void *addr,
15377         IN it_length_t length,
15378         IN it_mem_priv_t privs,
15379         IN it_ep_handle_t ep_handle,
15380         IN it_dto_cookie_t cookie,
15381         IN it_dto_flags_t dto_flags,
15382         OUT it_rmr_context_t *rmr_context
15383     );
15384
15385     /*
15386         it_rmr_create10 is provided for backwards-compatibility
15387         and may be dropped in a future IT-API version.
15388
15389         Calls with a suffix "10" can be typically implemented through
15390         direct inlining to the corresponding calls with suffix "20",
15391         providing default arguments as necessary.
15392     */
15393
15394     it_status_t it_rmr_create10 (
15395         IN it_pz_handle_t pz_handle,
15396         OUT it_rmr_handle_t *rmr_handle
15397     );
15398
15399     /*
15400         it_rmr_create20 provides the v2.0 functionality
15401         and may be renamed to it_rmr_create in a future IT-API version.
15402     */
15403     it_status_t it_rmr_create20 (
15404         IN it_pz_handle_t pz_handle,
15405         IN it_rmr_type_t rmr_type,
15406         OUT it_rmr_handle_t *rmr_handle
15407     );
15408

```

```

15409     it_status_t it_rmr_free (
15410         IN  it_rmr_handle_t rmr_handle
15411     );
15412     it_status_t it_rmr_link (
15413         IN  it_rmr_handle_t  rmr_handle,
15414         IN  it_lmr_handle_t  lmr_handle,
15415         IN  void              *addr,
15416         IN  it_length_t      length,
15417         IN  it_addr_mode_t   addr_mode,
15418         IN  it_mem_priv_t    privs,
15419         IN  it_ep_handle_t   ep_handle,
15420         IN  it_dto_cookie_t  cookie,
15421         IN  it_dto_flags_t   dto_flags,
15422         OUT it_rmr_context_t *rmr_context
15423     );
15424     it_status_t it_rmr_query (
15425         IN  it_rmr_handle_t  rmr_handle,
15426         IN  it_rmr_param_mask_t mask,
15427         OUT it_rmr_param_t   *params
15428     );
15429     it_status_t it_rmr_unlink (
15430         IN  it_rmr_handle_t rmr_handle,
15431         IN  it_ep_handle_t  ep_handle,
15432         IN  it_dto_cookie_t cookie,
15433         IN  it_dto_flags_t  dto_flags
15434     );
15435     it_status_t it_set_consumer_context (
15436         IN  it_handle_t handle,
15437         IN  it_context_t context
15438     );
15439     it_status_t it_socket_convert (
15440         IN  int          sd,
15441         IN  it_ep_handle_t ep_handle,
15442         IN  it_sc_flags_t flags,
15443         IN  const unsigned char *msg,
15444         IN  size_t          len
15445     );
15446     it_status_t it_srq_create (
15447         IN  it_pz_handle_t  pz_handle,
15448         IN  size_t          max_recv_segments,
15449         IN  size_t          max_recv_dtos,
15450         OUT it_srq_handle_t *srq_handle
15451     );
15452     it_status_t it_srq_free (
15453         IN  it_srq_handle_t srq_handle
15454     );
15455     it_status_t it_srq_modify (
15456         IN  it_srq_handle_t  srq_handle,
15457         IN  it_srq_param_mask_t mask,
15458         IN  const it_srq_param_t *params
15459     );
15460     it_status_t it_srq_query (
15461         IN  it_srq_handle_t  srq_handle,
15462         IN  it_srq_param_mask_t mask,

```

```

15463         OUT it_srq_param_t      *params
15464     );
15465     it_status_t it_ud_service_reply (
15466         IN         it_ud_svc_req_identifier_t ud_svc_req_id,
15467         IN         it_ud_svc_req_status_t    status,
15468         IN         it_remote_ep_info_t       ep_info,
15469         IN  const  unsigned char             *private_data,
15470         IN         size_t                    private_data_length
15471     );
15472     it_status_t it_ud_service_request (
15473         IN  it_ud_svc_req_handle_t ud_svc_handle
15474     );
15475     it_status_t it_ud_service_request_handle_create (
15476         IN  const it_conn_qual_t      *conn_qual,
15477         IN         it_evd_handle_t     reply_evd,
15478         IN  const it_path_t           *destination_path,
15479         IN  const unsigned char       *private_data,
15480         IN         size_t              private_data_length,
15481         OUT         it_ud_svc_req_handle_t *ud_svc_handle
15482     );
15483     it_status_t it_ud_service_request_handle_free (
15484         IN  it_ud_svc_req_handle_t ud_svc_handle
15485     );
15486     it_status_t it_ud_service_request_handle_query (
15487         IN  it_ud_svc_req_handle_t ud_svc_handle,
15488         IN  it_ud_svc_req_param_mask_t mask,
15489         OUT it_ud_svc_req_param_t *ud_svc_handle_info
15490     );
15491     #endif /* _ITAPI_H_ */

```

15492 **G.2 it_api_os_specific.h**

```

15493     /*
15494     * it_api_os_specific.h
15495     *
15496     * Copyright © 2005 The Open Group
15497     * All rights reserved.
15498     * The copyright owner hereby grants permission for all or part of this
15499     * publication to be reproduced, stored in a retrieval system, or
15500     * transmitted, in any form or by any means, electronic, mechanical,
15501     * photocopying, recording, or otherwise, provided that it remains
15502     * unchanged and that this copyright statement is included in all
15503     * copies or substantial portions of the publication.
15504     *
15505     * For any software code contained within this specification,
15506     * permission is hereby granted, free-of-charge, to any person
15507     * obtaining a copy of this specification (the "Software"), to deal in
15508     * the Software without restriction, including without limitation the
15509     * rights to use, copy, modify, merge, publish, distribute, sublicense,
15510     * and/or sell copies of the Software, and to permit persons to whom
15511     * the Software is furnished to do so, subject to the above copyright
15512     * notice and this permission notice being included in all copies or
15513     * substantial portions of the Software.

```

```

15514 *
15515 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
15516 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
15517 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND
15518 * NON-INFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS
15519 * BE LIABLE FOR ANY CLAIM, DAMAGES, OR OTHER LIABILITY, WHETHER IN AN
15520 * ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF, OR IN
15521 * CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
15522 * SOFTWARE.
15523 *
15524 * Permission is granted for implementers to use the names, labels,
15525 * etc. contained within the specification. The intent of publication
15526 * of the specification is to encourage implementations of the
15527 * specification. This specification has not been verified for
15528 * avoidance of possible third-party proprietary rights. In
15529 * implementing this specification, usual procedures to ensure the
15530 * respect of possible third-party intellectual property rights should
15531 * be followed.
15532 */
15533
15534 #ifndef _ITAPI_OS_SPECIFIC_H_
15535 #define _ITAPI_OS_SPECIFIC_H_
15536 #include "/usr/include/netinet/in.h"
15537 #include "/usr/include/sys/types.h"
15538
15539 #define IT_NO_ADDR NULL
15540 #define IT_INTERFACE_NAME_SIZE 128
15541 #endif /* _ITAPI_OS_SPECIFIC_H_ */

```