*Consortium Specification*

# Interconnect Transport API (IT-API)
# Version 2.1

## The Interconnect Software Consortium

in association with

Consortium Specification

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

ISBN: 1-931624-59-3

Document Number: C055

Published by The Open Group, November, 2005.

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

ogspecs@opengroup.org

# Contents

## List of Figures



## List of Tables

# Preface

## The Interconnect Software Consortium

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.

## The Open Group

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.

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

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.

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

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.

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

## This Document

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.

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

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

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

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

### Typographical Conventions

The following typographical conventions are used throughout this document:

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

### Reference Page Conventions

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

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

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

# Trademarks

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

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

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

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

# Acknowledgements

The Interconnect Software Consortium gratefully acknowledges the contribution of:

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

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

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

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

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

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

# 1 Introduction

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

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

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

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

## 1.1 Interface Adapters

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

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

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

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

## 1.2 Memory Management

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

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

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

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

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

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

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

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

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

## 1.3 Communication Endpoints

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

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

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

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

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

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

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

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

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

## 1.4 Work Requests and Data Transfer Operations

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

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

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

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

## 1.5 Events

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

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

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

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

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

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

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

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

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

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

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

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

# 2 Referenced Documents

The following documents are referenced in this specification:


[DDP-IETF] Direct Data Placement over Reliable Transports, H. Shah et al, IETF Internet Draft, February 2005. (www.ietf.org/internet-drafts/draft-ietf-rddp-ddp-04.txt)

[DDP-RDMAC] Direct Data Placement over Reliable Transports (Version 1.0), H. Shah et al, RDMA Consortium, October 2002. (www.rdmaconsortium.org/home/draft-shah-iwarp-ddp-v1.0.pdf)

[IB-R1.1] InfiniBand Architecture Specification Volume 1, Release 1.1, InfiniBand Trade Association, November 2002.

[IB-R1.2] InfiniBand Architecture Specification Volume 1, Release 1.2, InfiniBand Trade Association, October 2004.

[INTEROP-IETF] RNIC Interoperability, J. Carrier & J. Pinkerton, IETF Internet Draft, Novemer 2004. (www.ietf.org/internet-drafts/draft-carrier-rddp-rnic-interop-00.txt)

[IT-API-V1.0] Interconnect Transport API (IT-API), Issue 1.0, The Interconnect Software Consortium, February 2004.

[IT-API-V2.0] Interconnect Transport API (IT-API), Version 2.0, The Interconnect Software Consortium, March 2005.

[MPA-IETF] Marker PDU Aligned Framing for TCP Specification, P. Culley et al, IETF Internet Draft, February 2005. (www.ietf.org/internet-drafts/draft-ietf-rddp-mpa-02.pdf)

[MPA-RDMAC] Marker PDU Aligned Framing for TCP Specification (Version 1.0), P. Culley et al, RDMA Consortium, October 2002. (www.rdmaconsortium.org/home/draft-culley-iwarp-mpa-v1.0.pdf)

[RDMAP-IETF] An RDMA Protocol Specification, R. Recio et al, IETF Internet Draft, February 2005. (www.ietf.org/internet-drafts/draft-ietf-rddp-rdmap-03.txt)

[RDMAP-RDMAC] An RDMA Protocol Specification (Version 1.0), R. Recio et al, RDMA Consortium, October 2002. (www.rdmaconsortium.org/home/draft-recio-iwarp-rdmap-v1.0.pdf)

[VERBS-RDMAC] RDMA Protocol Verbs Specification (Version 1.0), J. Hilland et al, RDMA Consortium, April 2003. (www.rdmaconsortium.org/home/draft-hilland-iwarp-verbs-v1.0-RDMAC.pdf)

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

# 3 Definitions

### Absolute Addressing

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

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.

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.

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.

See also **Relative Addressing**.

### Address Handle

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.)

### AEVD

See **Aggregation Event Dispatcher**.

### Affiliated Asynchronous Error

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.

### Affiliated Asynchronous Event

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

### Affiliated Event

See **Affiliated Asynchronous Event**.

### Aggregation Event Dispatcher (AEVD)

An IT Object that conceptually merges Event completion Notifications from one or more Simple Event Dispatchers. This provides the Consumer with a single point to receive Notification of Event completions across multiple Event Streams.

### Asynchronous Error

An error that could not be reported as an immediate error. Asynchronous Errors include Completion Errors, Affiliated Asynchronous Errors, and Unaffiliated Asynchronous Errors.[1]

### Atomic

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

### AV-RNIC

API +Verbs-enabled RNIC. See **V-RNIC**. An implementation of an RDMA API (such as IT-API) over a V-RNIC.

### AV-RNIC/IETF

AV-RNIC based on a V-RNIC/IETF with API support for handling the MPA Startup. An AV-RNIC/IETF based on a Permissive (Non-permissive) V-RNIC/IETF is called Permissive (Non-permissive).

### AV-RNIC/IETF with MPA Startup Suppression

An AV-RNIC/IETF, whose API is instructed to suppress MPA Startup for transitioning a particular streaming-mode connection to RDMA mode. API support for suppressing the MPA Startup is outside of [MPA-IETF] but is expected to be supported for interoperability with the device class “AV-RNIC/RDMAC without MPA Startup”. The IT-API provides this capability for the TDI (Conversion process), but not for the TII.

### AV-RNIC/RDMAC with MPA Startup

AV-RNIC based on a V-RNIC/RDMAC and with API support for handling the MPA Startup. according to [INTEROP-IETF], wherein such an AV-RNIC is designated a (Software) Upgraded 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].

### AV-RNIC/RDMAC without MPA Startup

AV-RNIC based on a V-RNIC/RDMAC without API support for handling the MPA Startup or with suppressed MPA Startup. An AV-RNIC/RDMAC without MPA Startup also has MPA Marker/CRC restrictions.

### Base Address

The address corresponding to the first byte of an LMR or RMR in a given linear address space. The Base Address of a linked RMR is within the address range of the underlying LMR.

### Bind

See **RMR Link**.

### Block List

A list of memory blocks with arbitrary alignment and constant block size.

### Bus Address

An address exposed on an I/O interconnect bus. On some system architectures, bus addresses and physical addresses are identical.

### Communication Management Message Events

The set of Event types related to the sequence of messages involved in RC Connection establishment, normal disconnect, Connection error conditions, and Unreliable Datagram Service Resolution Replies.

### Communication Management Request Events

The set of Event types that result from messages received requesting RC Connection establishment or Unreliable Datagram service. Normally these Events trigger state changes at the receiving Endpoint.

### Completion Error

A processing error that could not be reported through an immediate error and which is associated with a specific Work Request. Completion Errors are reported through an Event Stream for WR/DTO completions.

### Completion Event

An Event indicating that a previously posted Work Request has completed.

### Completion Suppression

An optional Work Request behavior specifying that no Event is to be generated upon successful completion of the requested operation.

### Connection

An association between a pair of Endpoints such that data posted via Data Transfer Operations of either Endpoint arrives at the other Endpoint of the Connection.

### Connection Qualifier

A value that allows an incoming Connection Request or Unreliable Datagram Service Resolution Request to be associated with an entity that can provide that service.

### Connection Reply

A message in response to a Connection Request message.

### Connection Request

A message that requests RC Connection establishment.

### Consumer

An application that utilizes the IT-API.

### Context

A Consumer-supplied value that can be associated with an instance of an IT Object.

### Conversion

The actions performed by *it_socket_convert* for converting an unconnected Endpoint of the RC type and a connected socket to a connected, RDMA-enabled Endpoint. For the Conversion Initiator, the Conversion includes the transmission of the Last ULP Streaming-Mode Message.

### Conversion Initiator

A Consumer calling *it_socket_convert* with a Last ULP Streaming-Mode Message of length greater than zero.

### Conversion Responder

A Consumer calling *it_socket_convert* with a Last ULP Streaming-Mode Message of length zero.

#### Data Transfer Operation (DTO)

A request submitted by the Consumer to the Implementation to move data between two Endpoints. See also **Work Request**.

#### Deferred RDMA Transition

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.

#### Destination

The Endpoint where a message is received.

#### Direct LMR Handle

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”.

#### DTO

See **Data Transfer Operation**.

#### DTO Cookie

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.

#### Endpoint (EP)

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.

#### Endpoint Hard High Watermark

The maximum number of Receive DTOs that an Endpoint with an associated Shared Receive Queue can have in progress.

#### Endpoint ID

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 *it_evd_post_se* 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

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

#### FBO

See **First-Byte-Offset**.

#### First-Byte-Offset

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

#### Handle

An opaque data type used to reference an object.

#### IA

See **Interface Adapter**.

#### IANA

See **Internet Address Naming Authority**.

#### IANA Port Number

A specific port address as defined by IANA.

#### IB

See **InfiniBand**.

#### ICSC

See **Interconnect Software Consortium**.

#### IETF

See **Internet Engineering Task Force**.

#### Immediate RDMA Transition

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

**Implementation**

The collection of software and hardware that combine to provide the service exported by the IT-API.

**Implementer's Guide**

A non-normative section of the IT-API documentation set that contains information provided to assist implementers of the IT-API.

**Inbound RDMA Read Queue**

An internal queue of an Endpoint that handles incoming RDMA Read Requests. The processing of an incoming RDMA Read Request involves generating an RDMA Read Response.

**InfiniBand (IB)**

One of the transports that the IT-API supports. The host interface portion of InfiniBand is defined in [IB-R1.1] and [IB-R1.2] for Releases 1.1 and 1.2, respectively.

**InfiniBand Global Routing Header**

A routing header that may be present in the first 40 bytes of a completed Unreliable Datagram Receive operation. See the InfiniBand specification for a description of the format of this routing header.

**InfiniBand Transport**

Transport services defined by the InfiniBand Architecture.

**Interconnect Software Consortium (ICSC)**

Standards organization that includes the ITWG. The *Interconnect Software Consortium* is affiliated with *The Open Group*.

**Interconnect Transport Working Group (ITWG)**

The *ICSC Working Group* that created the IT-API.

**Internet Engineering Task Force (IETF)**

*Internet Engineering Task Force*.

**Internet Address Naming Authority (IANA)**

IETF Network Address naming authority.

### Interface

A host resident device that transfers data to and from the host memory to which it is attached.

### Interface Adapter (IA)

An instance of an Interface that is created by the *it_ia_create* call. An Interface Adapter may contain one or more Spigots.

### I/O Address

An address accessible by an IA. Depending on the system architecture, this may be a **Physical Address** or a **Bus Address**. In IT-API, the I/O Address object is an opaque, Implementation-dependent object.

### IOBL

See **I/O Buffer List**.

### I/O Buffer List (IOBL)

An IT-API object that specifies a list of buffers using **I/O Addresses**. Elements of an I/O Buffer List can have constant length or variable length.

### IP

The IETF Internet Protocol.

### IPv4

The IETF Internet Protocol Version 4.

### IPv6

The IETF Internet Protocol Version 6.

### IRD

Inbound RDMA Read Queue Depth (IRD). The maximum number of inbound RDMA Read requests that can be outstanding on an Endpoint.

### IRRQ

See **Inbound RDMA Read Queue**.

### IT-API

The data structures and routines that make up the Interconnect Transport Application Programming Interface.

### IT Handle

An opaque reference to an IT Object. An IT Handle is returned to the Consumer whenever an IT Object is created for the Consumer's use. The IT Handle can be used to reference the IT Object in subsequent calls into the IT-API.

### IT Object

A software object created by the IT-API Implementation as a result of a Consumer call into the IT-API, used to satisfy subsequent Consumer requests. When the IT Object is created, an opaque reference to the object, called a Handle, is returned to the Consumer for use in subsequent calls into the IT-API.

### ITWG

See **Interconnect Transport Working Group**.

### iWARP

The protocol suite enabling RDMA on top of TCP/IP or SCTP/IP, comprising MPA [MPA-IETF] [MPA-RDMAC], DDP [DDP-IETF] [DDP-RDMAC], and RDMAP [RDMAP-IETF] [RDMAP-RDMAC].

### iWARP Transport

The iWARP protocol suite on top of TCP/IP or SCTP/IP.

### Last Streaming-Mode Message

The last message sent in streaming mode prior to enabling RDMA mode on a connection. In the presence of an MPA Startup, this message is the MPA Reply message.

### Last ULP Streaming-Mode Message

The last ULP message sent in TCP streaming mode prior to enabling RDMA mode on a connection, sent by an IT-API Consumer acting as the Conversion Initiator. Need not be the Last Streaming-Mode Message since the Conversion of a TCP connection typically involves an MPA Startup handshake that is invisible to the Consumers.

### Link

See **LMR Link** and **RMR Link**.

#### Listen Point

An object capable of listening for incoming Connection Requests and of passing such requests to a Simple Event Dispatcher. A Listen Point is associated with a single Spigot.

#### LLP

See **Lower Layer Protocol**.

#### LMR

See **Local Memory Region**.

#### LMR Link

The memory management operation initiated by *it_lmr_link*, which associates an LMR with an IOBL and thereby enables local and optional remote access to that LMR.

#### LMR Triplet

A type used to specify a section of a Local Memory Region. An LMR Triplet specifies an LMR Handle, an address, and a length.

#### LMR Unlink

The memory management operation initiated by *it_lmr_unlink*, which destroys the association of an LMR with an IOBL.

#### Local Memory Region (LMR)

A contiguously addressable area of arbitrary size within a linear address space enabling local access and optional remote access.

#### Lower Layer Protocol (LLP)

A protocol used in a layer of the underlying RDMA transport.

#### Marker PDU Aligned Framing for TCP (MPA)

The framing protocol used to support messaging over TCP streams. See [MPA-IETF] and [MPA-RDMAC].

#### MPA

See **Marker PDU Aligned Framing for TCP**.

### MPA Startup

Exchange of MPA Request/Reply messages according to [MPA-IETF], preferably with additional support for interoperability according to [INTEROP-IETF].

### Narrow RMR

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**.

### Network Address

An identifier that can be used to reach a particular Spigot attached to a network.

### Non-Permissive AV-RNIC/IETF

An AV-RNIC/IETF that cannot support downgrading versions of RDMAP, DDP, and MPA. See [INTEROP-IETF].

### Notification

An asynchronous mechanism for providing the Consumer with information about the completion of a previously posted operation.

### Notification Event

An Event in an Event Stream whose arrival triggers the Notification of the Event to a waiting Consumer via either a wakeup from *it_evd_wait*, or via a higher-level Notification mechanism.

### Notification Suppression

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.

### ORD

Outbound RDMA Read Queue Depth (ORD). The maximum number of outbound RDMA Read Work Requests a Consumer can have outstanding on an Endpoint.

### Organization Unique Identifier (OUI)

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 *it_path_t* 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

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

#### Privileged Mode Endpoint

An Endpoint created with Privilege.

#### Protection Zone (PZ)

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

#### PZ

See **Protection Zone**.

#### RC

See **Reliable Connected**.

#### RDMA

See **Remote Direct Memory Access**.

#### RDMA Initiator

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

#### RDMA Responder

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

#### RDMA Read

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

#### RDMA Read Request

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

### RDMA Read Response

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

### RDMA Write

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

### RDMA Write Request

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

### Receive

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

### Receive Queue (RQ)

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

### Relative Addressing

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

The *addr* 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 *addr.rel* member of an LMR Triplet or the *rdma_addr* 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 *addr* parameter of the RMR Link operation and the *addr* 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 *addr* 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 *addr.rel* member of an RMR Triplet or the *rdma_addr* parameter passed to an RDMA DTO is interpreted as a byte offset relative to the first byte of the RMR. The total offset relative to the first byte of the underlying LMR is obtained by adding to this byte offset the difference 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.

#### RMR Triplet

A type used to specify a section of a Remote Memory Region. An RMR Triplet specifies an RMR Handle, an address, and a length.

#### RNIC

Hardware or software implementation of RDMAP/DDP over MPA/TCP/IP or RDMAP/DDP over SCTP/IP.

#### RQ

See **Receive Queue**.

#### SE

See **Software Event**.

#### Send

The Data Transfer Operation (DTO) that is initiated by the *it_post_send* or *it_post_sendto* routine.

#### Send Queue (SQ)

An internal queue of an Endpoint on which Send, RDMA Write, RDMA Read, and RMR (Un)Link Work Requests are posted.

#### Service Reply

See **UD Service Reply**.

#### Service Request

See **UD Service Request**.

#### Service Request Handle

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.

#### Service Type

A class of transport service defining basic attributes of the communication; e.g., connected or unconnected, reliable or unreliable.

### SEVD

See **Simple Event Dispatcher**.

### Shared Receive Queue (S-RQ)

A Work Queue that can be shared by multiple Endpoints and to which Receive DTOs (Work Requests) can be posted.

### Simple Event Dispatcher (SEVD)

An IT Object that conceptually merges Events from one or more Event Streams. These Events can be dequeued by the Consumer directly. The Consumer is notified that Events are available through the *it_evd_wait* interface, or through higher-level Notification mechanisms, such as the Aggregation Event Dispatcher. The Simple Event Dispatcher is responsible for completion of transport-specific fetching and handshaking for the Events it collects. Each Event is delivered to the Consumer exactly once.

### Socket Conversion

See **Conversion**.

### Software Event (SE)

An Event generated for a Simple Event Dispatcher by the Consumer, as opposed to those generated by the Interface Adapter.

### Solicited Wait

A modifier for Send DTOs submitted to an Endpoint of the Connection. It specifies that the completion of matching Receive DTOs on the remote side of the Connection generate Notification Receive DTO Completion Events.

### Source

The Endpoint where a message originates.

### Spigot

A host resident device that transfers data to and from the host memory to which it is attached. A Spigot is associated with a single Interface. One or more Spigots may be associated with the same Interface.

### SQ

See **Send Queue**.

### S-RQ

See **Shared Receive Queue**.

### S-RQ Low Watermark

The minimum number of Receive DTOs that can be sitting idle in a Shared Receive Queue without causing an S-RQ Low Watermark Event to be generated.

### S-RQ Low Watermark Event

An Affiliated Asynchronous Event that is generated when the Consumer has armed the S-RQ Low Watermark mechanism for an S-RQ and the total number of Receive DTOs sitting idle in the S-RQ drops below the S-RQ Low Watermark.

### TCP

See **Transmission Control Protocol**.

### Transmission Control Protocol (TCP)

The IP-based, reliable, connection-oriented transport protocol.

### The Open Group

*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.

### Transport-Dependent Interface (TDI)

RDMA connection management service for iWARP only, allowing the Conversion of an unconnected Endpoint of the RC type and a connected socket to a connected Endpoint. The TDI includes the *it_socket_convert* and *it_ep_disconnect* calls.

### Transport-Independent Interface (TII)

Unified RDMA connection management service abstraction for any RC transport supported by the IT-API. The TII includes the *it_ep_connect*, *it_listen_create*, *it_ep_accept*, *it_reject*, and *it_ep_disconnect* calls.

### Transport Service Type

See **Service Type**.

### UD

See **Unreliable Datagram**.

#### UD Service Reply

A reply message sent via the Unreliable Datagram service in response to a UD Service Request.

#### UD Service Request

A request message sent via the Unreliable Datagram service requesting service resolution.

#### ULP

See **Upper Layer Protocol**.

#### Unaffiliated Asynchronous Error

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

#### Unaffiliated Asynchronous Event

An Event that is associated only with an Interface Adapter, rather than a specific Endpoint, EVD, S-RQ, or Work Request.

#### Unaffiliated Event

See **Unaffiliated Asynchronous Event**.

#### Unbind

See **RMR Unlink**.

#### Unlink

See **LMR Unlink** and **RMR Unlink**.

#### Unreliable Datagram (UD)

A Transport Service Type in which an Endpoint may transmit and Receive single-packet messages to/from any other Endpoint that supports that Service Type. Ordering and delivery are not guaranteed, and the Receiver may drop delivered packets.

#### Upper Layer Protocol (ULP)

A protocol taking advantage of RDMA services as provided by the IT-API.

#### VIA

See **Virtual Interface Architecture**.

#### Virtual Interface Architecture (VIA)

One of the transports that the IT-API supports [VIA-V1.0].

#### V-RNIC

Verbs-enabled RNIC (equivalent to RI defined in [VERBS-RDMAC]).

#### V-RNIC/IETF

V-RNIC compliant with [MPA-IETF] and exposing Verbs that are slightly extended relative to [VERBS-RDMAC] for MPA Marker/CRC control according to [MPA-IETF]. A V-RNIC/IETF can be Permissive or Non-permissive [INTEROP-IETF].

#### V-RNIC/RDMAC

V-RNIC compliant with [MPA-RDMAC] and [VERBS-RDMAC].

#### Wide RMR

An RMR that can be referenced or accessed via all Endpoints in the Protection Zone of the RMR. See also **Narrow RMR**.

#### Work Queue (WQ)

A queue on which Work Requests are posted. Endpoints commonly have two internal Work Queues: a Send Queue and a Receive Queue. Alternatively, an Endpoint may be associated with a Shared Receive Queue, in which case the Endpoint has no internal Receive Queue. Send, RDMA Read, RDMA Write, RMR Link, and RMR Unlink Work Requests are posted to an Endpoint's Send Queue. Receive Work Requests are posted either to an Endpoint's Receive Queue or to a Shared Receive Queue.

#### Work Request (WR)

A request to perform an operation, posted by the Consumer to an Endpoint. Work Requests include DTOs, RMR Link, and RMR Unlink operations.

#### WQ

See **Work Queue**.

#### WR

See **Work Request**.

# 4 Global Behavior

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

## 4.1 Asynchronous versus Synchronous APIs

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

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

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

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

## 4.2 Thread Safety

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

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

- Strongly Thread-Safe
- Efficiently Thread-Safe
- Not Thread-Safe

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

Implementations of the IT-API may support one or more thread safety models. Which model or models a particular Implementation supports, and how that thread safety support is communicated to the Consumer, is beyond the scope of the IT-API. One potential mechanism would have the Implementation vendor associate a specific thread safety model with a particular library Implementation of the IT-API. In this scheme, different libraries would support different 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.<br>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.<br>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.<br>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. |

**Table 1: Thread Safety Models**

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

Thread-safety means that the routine:

1. Provides well-defined results without imposing any restrictions on other IT-API routines called by other threads in the system
2. Provides well-defined results without regard to which other IT-API routines currently have threads of execution within them

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

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

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

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

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

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

1271

| **Non-Performance Critical Routines** | **Strongly Thread-Safe Model** | **Efficiently Thread-Safe Model** | **Not Thread-Safe Model** |
|---|---|---|---|
| *it_address_handle_create*<br>*it_convert_net_addr*<br>*it_ep_rc_create*<br>*it_ep_ud_create*<br>*it_evd_create*<br>*it_evd_callback_attach*<br>*it_evd_callback_detach*<br>*it_get_handle_type*<br>*it_get_pathinfo*<br>*it_hton64*, *it_ntoh64*<br>*it_ia_create*<br>*it_ia_query*<br>*it_interface_list*<br>*it_listen_create*<br>*it_listen_query*<br>*it_lmr_create*<br>*it_lmr_create_unlinked*<br>*it_lmr_link*<br>*it_make_rdma_addr_absolute*<br>*it_make_rdma_addr_relative*<br>*it_mem_map*<br>*it_pz_create*<br>*it_pz_query*<br>*it_rmr_create*<br>*it_srq_create*<br>*it_ud_service_request_handle_create* | Thread-safe | | Not thread-safe |
| **Event Harvesting and Posting Routines** | **Strongly Thread-Safe Model** | **Efficiently Thread-Safe Model** | **Not Thread-Safe Model** |
| *it_evd_dequeue*<br>*it_evd_post_se*<br>*it_evd_wait* | Thread-safe | | Not thread-safe |
| **Performance Critical Routines** | **Strongly Thread-Safe Model** | **Efficiently Thread-Safe Model** | **Not Thread-Safe Model** |
| *it_post_atomic*<br>*it_post_rdma_read*<br>*it_post_rdma_read_to_rmr*<br>*it_post_rdma_write*<br>*it_post_recv*<br>*it_post_recvfrom*<br>*it_post_send*<br>*it_post_send_and_unlink*<br>*it_post_sendto*<br>*it_rmr_link* | Thread-safe | Not thread-safe | |

| | | | |
|---|---|---|---|
| *it_rmr_unlink*<br>*it_ud_service_reply*<br>*it_ud_service_request* | | | |
| **Object Management Routines** | **Strongly Thread-Safe Model** | **Efficiently Thread-Safe Model** | **Not Thread-Safe Model** |
| *it_address_handle_modify*<br>*it_address_handle_query*<br>*it_ep_connect*<br>*it_ep_modify*<br>*it_ep_query*<br>*it_ep_reset*<br>*it_evd_modify*<br>*it_evd_query*<br>*it_get_consumer_context*<br>*it_lmr_modify*<br>*it_lmr_query*<br>*it_lmr_flush_to_mem*<br>*it_lmr_refresh_from_mem*<br>*it_rmr_query*<br>*it_set_consumer_context*<br>*it_socket_convert*<br>*it_srq_modify*<br>*it_srq_query*<br>*it_ud_service_request_handle_query* | Thread-safe | Not thread-safe | |
| **Object Destructor Routines** | **Strongly Thread-Safe Model** | **Efficiently Thread-Safe Model** | **Not Thread-Safe Model** |
| *it_address_handle_free*<br>*it_ep_accept*<br>*it_ep_disconnect*<br>*it_ep_free*<br>*it_evd_free*<br>*it_handoff*<br>*it_ia_free*<br>*it_ia_info_free*<br>*it_listen_free*<br>*it_lmr_free*<br>*it_lmr_unlink*<br>*it_mem_unmap*<br>*it_reject*<br>*it_ud_service_request_handle_free*<br>*it_pz_free*<br>*it_rmr_free*<br>*it_srq_free* | Not thread-safe | | |

1272 **Table 2: Thread-Safety Models Applied to IT-APIs**

## 4.3 Signal Handlers

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

## 4.4 Fork Semantics

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

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

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

## 4.5 Exec Semantics

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

## 4.6 Exit Semantics

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

## 4.7 Error Handling

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

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

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

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

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

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

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

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

## 4.8 IT Handle Management

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

## 4.9 Output Parameters

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

## 4.10 Privileged Consumer

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

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

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

## 4.11 Memory Management

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

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

# 5 Connection Management

## 5.1 Overview

The IT-API provides RDMA services for several underlying RDMA transports, including InfiniBand and iWARP[2]. The specifics of these RDMA transports as well as different application requirements led to the design of two different Connection management interfaces for Reliable Connected (RC) RDMA transports, as introduced below.

The Transport-Independent Interface (TII) provides a unified RDMA Connection management service abstraction for any RC transport supported by the IT-API and is the legacy Connection management API from IT-API Version 1.0. The TII includes the *it_ep_connect*, *it_listen_create*, *it_ep_accept*, *it_reject*, and *it_ep_disconnect* calls[3]. The TII allows Consumers to negotiate the *Inbound RDMA Read Queue Depth* (IRD) and *Outbound RDMA Read Queue Depth* (ORD) parameters and to exchange Private Data during Connection establishment. Consumers are expected to preconfigure IRD and ORD prior to calling *it_ep_connect* or *it_ep_accept* by setting the corresponding Endpoint attributes *rdma_read_ird* and *rdma_read_ord*, respectively. From the Consumer's viewpoint, the RDMA service is available immediately after Connection establishment. No Lower Layer Protocol (LLP) handle is exposed to the Consumer. The usage of the TII is described in *Transport-Independent Interface*. Implementation aspects for the TII on iWARP are described in Appendix B.

The Transport-Dependent Interface (TDI) is available for iWARP only and allows the Conversion of an unconnected Endpoint of the RC type and a connected socket into a connected Endpoint. The TDI includes the *it_socket_convert* and *it_ep_disconnect* calls. It enables Consumers to exchange an arbitrary (but non-zero) amount of streaming-mode data via a TCP socket (and possibly via an SCTP socket in the future) prior to performing a Conversion, which is a requirement for several iWARP-specific ULPs. This prior exchange of streaming-mode data occurs outside the IT-API. In contrast to Connection establishment with the TII, the *it_socket_convert* call neither supports the negotiation of IRD and ORD parameters, nor the exchange of Private Data. The main reason for this simplification is that the peers have an opportunity to negotiate IRD and ORD, to select the corresponding Endpoint attributes *rdma_read_ird* and *rdma_read_ord*, and to exchange Private Data prior to Conversion; moreover, they have an opportunity to modify ORD after the Conversion[4]; i.e., while already in RDMA mode. The TDI also allows the Consumer more detailed control of the Connection establishment process, including whether or not the MPA Request/Reply startup sequence is 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].

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

### 5.1.1 IRD/ORD Negotiation

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

IRD and ORD must be negotiated such that each side of a Connection uses compatible values. If ORD on one side of a Connection exceeds IRD on the other, there is a potential for the Connection to be terminated if more RDMA Reads are issued than the advertised IRD can tolerate[5].

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

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

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

The RDMA Initiator's IRD/ORD values (IRDi/ORDi) must be conveyed to and examined by the RDMA Responder.

The RDMA Responder's IRD (IRDr) can be reduced from its preset value to a value not exceeding ORDi.

The RDMA Responder's ORD (ORDr) should be set to the minimum of its preset value and IRDi.

Finally, the RDMA Initiator adjusts ORDi and optionally IRDi after receiving the RDMA Responder's IRDr and ORDr.

---

[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.

## 5.2 Transport-Independent Interface

### 5.2.1 Overview

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

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

Connection establishment for the TII is illustrated in Figure 1.

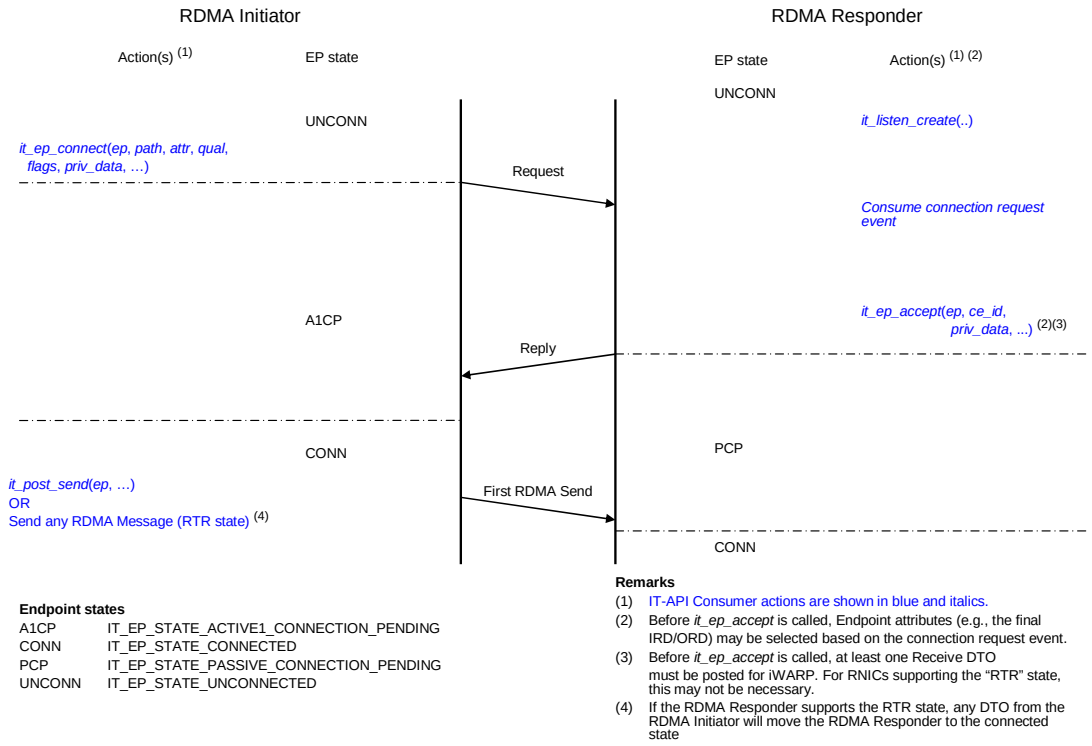



**Figure 1: Connection Establishment with the TII**

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

The Implementation of the RDMA Initiator generates and sends an RDMA Connection Request containing the local Endpoint IRD and ORD parameters[7] and, amongst other things, the Private Data provided by the Consumer with *it_ep_connect*. After receiving a valid RDMA Connection reply from the Responding side, the Implementation generates a Connection established event for the RDMA Initiator providing the remote Consumer's Private Data.

The Responding side Listen Point expects an incoming RDMA Connection Request. Upon receiving a valid request, it generates a Connection Request event for the Consumer containing the remote Endpoint's IRD and ORD parameters[8] as well as the remote Consumer's Private Data. The Connection Request event is identified by the Connection Establishment ID (*cn_est_id*). The Consumer on the RDMA Responder side now invokes either *it_ep_accept* or *it_reject*.

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

### 5.2.2 ULP Constraints for iWARP

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

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

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

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

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

### 5.2.3 API-Supported IRD/ORD Negotiation

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

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

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

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

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

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

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

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

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

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

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

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

## 5.2.4 Transport-Dependent Attributes

### *5.2.4.1 iWARP*

#### 5.2.4.1.1 Connection Qualifiers

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

#### 5.2.4.1.2 Service Types

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

#### 5.2.4.1.3 Handshake

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

## 5.3 Transport-Dependent Interface (iWARP-Only)

### 5.3.1 Overview

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

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

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

The Conversion process is illustrated below.

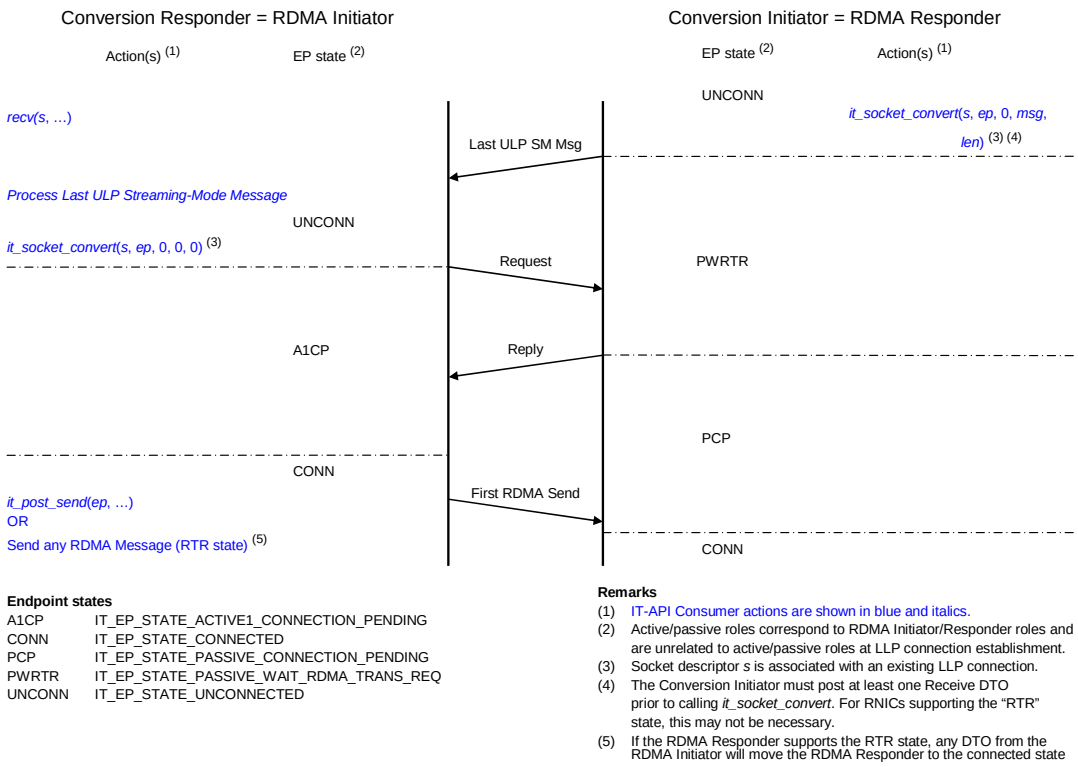



**Figure 2: Conversion Process with MPA Startup (TDI)**

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.

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.

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

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.

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.

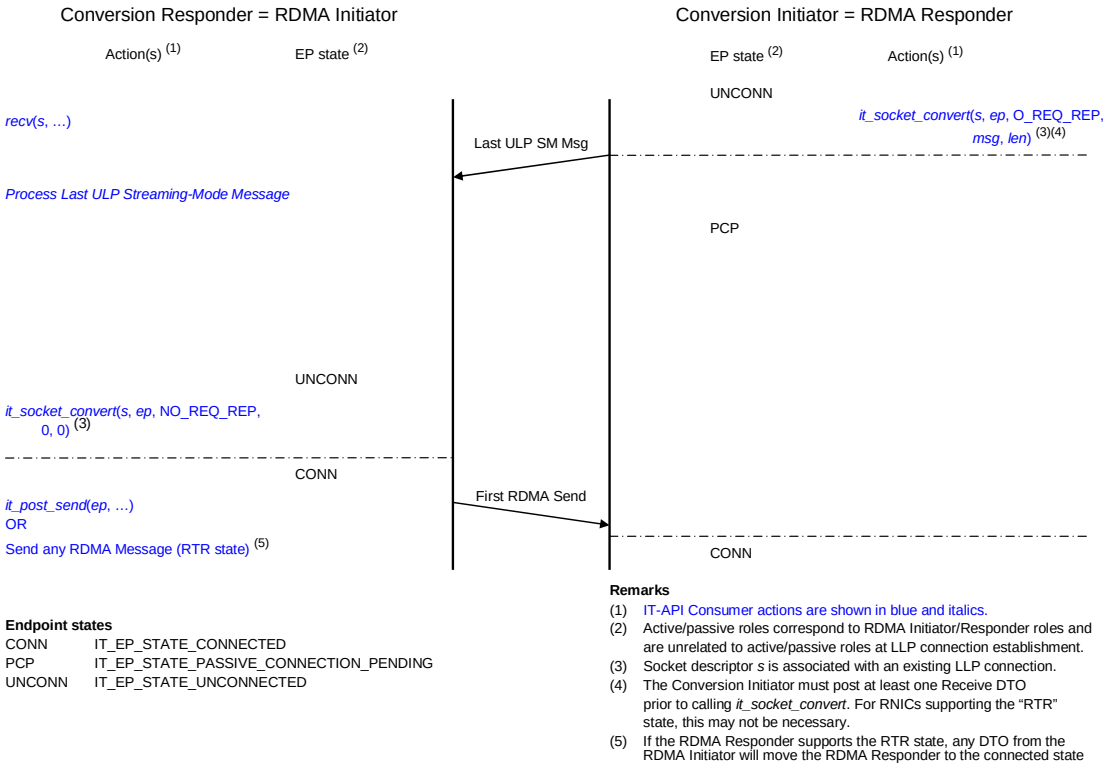



**Figure 3: Conversion Process with MPA Startup Suppression (TDI)**

### 5.3.2 IRD/ORD Negotiation

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

1548 # 6 API Reference Pages

| | |
|---|---|
| *it_address_handle_create* | Create an Address Handle. |
| *it_address_handle_free* | Free an Address Handle. |
| *it_address_handle_modify* | Modify an Address Handle. |
| *it_address_handle_query* | Query an Address Handle. |
| *it_convert_net_addr* | Convert a Network Address from one format to another. |
| *it_ep_accept* | Accept an incoming Connection establishment request or reply. |
| *it_ep_connect* | Initiate an Endpoint Connection establishment request. |
| *it_ep_disconnect* | Disconnect an existing Endpoint–to-Endpoint Connection. |
| *it_ep_free* | Destroy an RC or UD Endpoint. |
| *it_ep_modify* | Modify parameters of an existing Endpoint. |
| *it_ep_query* | Query an existing Endpoint. |
| *it_ep_rc_create* | Create an Endpoint for Reliable Connection. |
| *it_ep_reset* | Reset a Reliable Connected Endpoint into the initial state. |
| *it_ep_ud_create* | Create an Endpoint for Unreliable Datagram. |
| *it_evd_callback_attach* | Attach a callback routine to an EVD. |
| *it_evd_callback_detach* | Detach a callback routine from an EVD. |
| *it_evd_create* | Create Simple or Aggregate Event Dispatcher. |
| *it_evd_dequeue* | Dequeue Events from Event Dispatcher. |
| *it_evd_free* | Destroy an Event Dispatcher. |
| *it_evd_modify* | Modify an existing Event Dispatcher. |
| *it_evd_post_se* | Post Software Event on Simple Event Dispatcher. |
| *it_evd_query* | Query an existing Simple or Aggregate Event Dispatcher. |
| *it_evd_wait* | Wait for Events on Event Dispatcher. |
| *it_get_consumer_context* | Get Consumer Context associated with an IT Object Handle. |
| *it_get_handle_type* | Return the Handle type value associated with an IT Object Handle. |
| *it_get_pathinfo* | Retrieve Paths used to communicate with a remote Network Address. |
| *it_handoff* | Hand-off an incoming Connection Request to another Connection Qualifier. |

| | |
|---|---|
| *it_hton64*, *it_ntoh64* | Convert 64-bit integers between host and network byte order. |
| *it_ia_create* | Create an Interface Adapter. |
| *it_ia_free* | Destroy an Interface Adapter Handle. |
| *it_ia_info_free* | Free an *it_ia_info_t* structure that was returned by *it_ia_query*. |
| *it_ia_query* | Retrieve attributes of given Interface Adapter and its Spigots. |
| *it_interface_list* | Retrieve information about the available Interface Adapters. |
| *it_listen_create* | Listen for an incoming Connection Request for a Connection Qualifier. |
| *it_listen_free* | Destroy a listening point for a Connection Qualifier. |
| *it_listen_query* | Query parameters associated with a listening point. |
| *it_lmr_create* | Create a Local Memory Region and register with an Interface Adapter. |
| *it_lmr_create_unlinked* | Create an unlinked Local Memory Region. |
| *it_lmr_flush_to_mem* | Make memory changes visible to an incoming RDMA Read or Atomic operation. |
| *it_lmr_free* | Destroy a Local Memory Region. |
| *it_lmr_link* | Link an unlinked Local Memory Region. |
| *it_lmr_modify* | Modify selected attributes of a Local Memory Region. |
| *it_lmr_query* | Get attributes of a Local Memory Region. |
| *it_lmr_refresh_from_mem* | Make effects of an incoming RDMA Write or Atomic operation visible. |
| *it_lmr_unlink* | Unlink a Local Memory Region. |
| *it_make_rdma_addr_absolute* | Make a platform-independent RDMA address for Absolute Addressing. |
| *it_make_rdma_addr_relative* | Make a platform-independent RDMA address for Relative Addressing. |
| *it_mem_map* | Map a Virtual Address to I/O Address Space. |
| *it_mem_unmap* | Undo a Virtual to I/O Address mapping. |
| *it_post_atomic* | Post an Atomic Operation DTO to an RC-type Endpoint. |
| *it_post_rdma_read* | Post an RDMA Read DTO to an RC-type Endpoint. |
| *it_post_rdma_read_to_rmr* | Post an RDMA Read DTO to an RC-type Endpoint. |
| *it_post_rdma_write* | Post an RDMA Write DTO to an RC-type Endpoint. |
| *it_post_recv* | Post a Receive DTO to an RC-type Endpoint. |
| *it_post_recvfrom* | Post a Receive DTO to a datagram Endpoint. |
| *it_post_send* | Post a Send DTO to an RC-type Endpoint. |

| | |
|---|---|
| *it_post_send_and_unlink* | Post a Send DTO to an RC-type Endpoint and Unlink a remote RMR. |
| *it_post_sendto* | Post a Send DTO to a datagram Endpoint. |
| *it_pz_create* | Create a new Protection Zone. |
| *it_pz_free* | Destroy a Protection Zone. |
| *it_pz_query* | Get attributes of a Protection Zone. |
| *it_reject* | Reject an incoming Connection establishment request or reply. |
| *it_rmr_create* | Create a Remote Memory Region (RMR). |
| *it_rmr_free* | Destroy a Remote Memory Region. |
| *it_rmr_link* | Post request to Link a Remote Memory Region to a memory range. |
| *it_rmr_query* | Get attributes of a Remote Memory Region. |
| *it_rmr_unlink* | Post operation to Unlink a Remote Memory Region from its memory range. |
| *it_set_consumer_context* | Associate a Consumer Context with an IT Object Handle. |
| *it_socket_convert* | Convert a connected socket into a connected IT-API Endpoint. |
| *it_srq_create* | Create a Shared Receive Queue (S-RQ). |
| *it_srq_free* | Destroy a Shared Receive Queue. |
| *it_srq_modify* | Modify selected attributes of a Shared Receive Queue. |
| *it_srq_query* | Get attributes of a Shared Receive Queue. |
| *it_ud_service_reply* | Return UD communication information. |
| *it_ud_service_request* | Request the recipient to return UD communication information. |
| *it_ud_service_request_handle_create* | Create a UD Service Request Handle. |
| *it_ud_service_request_handle_free* | Free a previously created *it_ud_svc_req_handle_t*. |
| *it_ud_service_request_handle_query* | Return *it_ud_svc_req_handle_t* information. |

1549

1550

# it_address_handle_create()

**NAME**

it_address_handle_create – create an Address Handle

**SYNOPSIS**

```
#include <it_api.h>

it_status_t it_address_handle_create(
    IN         it_pz_handle_t     pz_handle,
    IN  const  it_path_t         *destination_path,
    IN         it_ah_flags_t      ah_flags,
    OUT        it_addr_handle_t  *addr_handle
);

typedef enum {
    IT_AH_PATH_COMPLETE = 0x1
} it_ah_flags_t;
```

**APPLICABILITY**

*it_address_handle_create* is applicable only to the UD service type.

**DESCRIPTION**

| | |
|---|---|
| *pz_handle* | Handle for the Protection Zone to be associated with the created Address Handle. This implicitly identifies the Interface Adapter that the Address Handle will be associated with. |
| *destination_path* | The Path to use to create the Address Handle. |
| *ah_flags* | The bitwise OR of the set of operation modifier flags specified below. |
| *addr_handle* | Returned datagram Address Handle. |

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

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

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

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

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

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

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

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

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

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_RESOURCES | The requested operation failed due to insufficient resources. |
| IT_ERR_INVALID_PZ | The Protection Zone Handle (*pz_handle*) was invalid. |
| IT_ERR_INVALID_FLAGS | The flags (*ah_flags*) value was invalid. |
| IT_ERR_NO_PERMISSION | The Consumer did not have the proper permissions to perform the requested operation. |
| IT_ERR_INVALID_SOURCE_PATH | One of the components of the Source portion of the supplied Path was invalid. |
| IT_ERR_INVALID_SPIGOT | An invalid Spigot ID was specified. |
| 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. |

**SEE ALSO**

*it_get_pathinfo*(), *it_path_t*, *it_address_handle_query*(), *it_address_handle_modify*(), *it_address_handle_free*()

# it_address_handle_free()

**NAME**

it_address_handle_free – free an Address Handle

**SYNOPSIS**

```
#include <it_api.h>

it_status_t it_address_handle_free(
    IN  it_addr_handle_t  addr_handle
);
```

**APPLICABILITY**

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

**DESCRIPTION**

| | |
|---|---|
| *addr_handle* | Address Handle to free. |

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

**RETURN VALUE**

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

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

**SEE ALSO**

*it_address_handle_create*(), *it_address_handle_query*(), *it_address_handle_modify*()

# it_address_handle_modify()

## NAME

it_address_handle_modify – modify an Address Handle

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_address_handle_modify(
    IN         it_addr_handle_t      addr_handle,
    IN         it_addr_param_mask_t  mask,
    IN  const  it_addr_param_t      *params
);
```

## APPLICABILITY

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

## DESCRIPTION

| | |
|---|---|
| *addr_handle* | Address Handle to modify. |
| *mask* | Bitwise OR of flags for desired parameters to be modified. |
| *params* | Structure whose members contain the new parameter values. |

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

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

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

```
typedef struct {
    ...
    it_path_t  path;  /* IT_ADDR_PARAM_PATH */
    ...
} it_addr_param_t;
```

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

| it_addr_param_t Member | Meaning |
|---|---|
| *path* | 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 *it_address_handle_query* to retrieve the current Path, modify the Path attributes as desired, and then call |

| it_addr_param_t Member | Meaning |
|---|---|
| | *it_address_handle_modify* with the resulting Path. See the *it_address_handle_create* reference page for details about which portions of the Path are relevant for Address Handles. |

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.

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_INVALID_AH | The Address Handle (*addr_handle*) was invalid. |
| IT_ERR_INVALID_MASK | The *mask* contained invalid flag values. |
| IT_ERR_NO_PERMISSION | The Consumer did not have the proper permissions to perform the requested operation. |
| IT_ERR_INVALID_SOURCE_PATH | One of the components of the Source portion of the supplied Path was invalid. |
| IT_ERR_INVALID_SPIGOT | An invalid Spigot ID was specified. |
| 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. |

**SEE ALSO**

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

# it_address_handle_query()

## NAME

it_address_handle_query – query an Address Handle

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_address_handle_query(
    IN   it_addr_handle_t      addr_handle,
    IN   it_addr_param_mask_t  mask,
    OUT  it_addr_param_t      *params
);

typedef enum {
    IT_ADDR_PARAM_ALL    = 0x0001,
    IT_ADDR_PARAM_IA     = 0x0002,
    IT_ADDR_PARAM_PZ     = 0x0004,
    IT_ADDR_PARAM_PATH   = 0x0008
} it_addr_param_mask_t;

typedef struct {
it_ia_handle_t  ia;    /* IT_ADDR_PARAM_IA */
it_pz_handle_t  pz;    /* IT_ADDR_PARAM_PZ */
it_path_t       path;  /* IT_ADDR_PARAM_PATH */
} it_addr_param_t;
```

## APPLICABILITY

*it_address_handle_query* is applicable only to the UD service type.

## DESCRIPTION

*addr_handle* Address Handle to query.

*mask* Bitwise OR of flags for desired parameters.

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

*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.

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

| it_addr_param_t Member | Meaning |
|---|---|
| *ia* | The Handle for the IA with which this Address Handle is associated. |
| *pz* | The Handle for the Protection Zone with which this Address Handle is associated. |
| *path* | 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*. |

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_INVALID_AH | The Address Handle (*addr_handle*) was invalid. |
| IT_ERR_INVALID_MASK | The *mask* contained invalid flag values. |
| 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. |

**SEE ALSO**

*it_address_handle_create*(), *it_address_handle_modify*(), *it_address_handle_free*()

# it_convert_net_addr()

**NAME**

it_convert_net_addr – convert a Network Address from one format to another

**SYNOPSIS**

```
#include <it_api.h>

it_status_t it_convert_net_addr(
    IN  const  it_net_addr_t      *source_addr,
    IN         it_net_addr_type_t  addr_type,
    OUT        it_net_addr_t      *destination_addr
);
```

**DESCRIPTION**

| | |
|---|---|
| *source_addr* | The input Network Address that is to be converted. |
| *addr_type* | The new type of address to convert the *source_addr* address to. |
| *destination_addr* | The returned Network Address. |

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

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

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

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

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_INVALID_ADDRESS | The Network Address specified in *source_addr* was invalid. |
| IT_ERR_INVALID_NETADDR | The type of the Network Address specified in *source_addr* was not recognized. |
| IT_ERR_INVALID_CONVERSION | The requested Network Address conversion either was not supported by the Implementation, or was supported but could not be performed by the Implementation for the particular *source_addr* that was input. |

| | |
|---|---|
| IT_ERR_INTERRUPT | The call was unblocked by a signal. |
| 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. |

**APPLICATION USAGE**

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

**SEE ALSO**

*it_listen_create*(), *it_net_addr_t*

# it_ep_accept()

## NAME

it_ep_accept – accept an incoming Connection Request or Connection Reply

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_ep_accept(
    IN         it_ep_handle_t          ep_handle,
    IN         it_cn_est_identifier_t  cn_est_id,
    IN  const  unsigned char          *private_data,
    IN         size_t                  private_data_length
);

typedef uint64_t it_cn_est_identifier_t;
```

## APPLICABILITY

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

## DESCRIPTION

| | |
|---|---|
| *ep_handle* | Local Endpoint to be bound to the Connection Request being accepted. |
| *cn_est_id* | Connection Establishment Identifier associated with the Connection Request being accepted. The *cn_est_id* is obtained from the data delivered in the Connection Request Event (IT_CM_REQ_CONN_REQUEST_EVENT). See the *it_cm_req_events* reference page for details. |
| *private_data* | Opaque Private Data provided by the IT_CM_MSG_CONN_PEER_REJECT_EVENT Event delivered to the Remote Consumer that will be sent to the Remote Endpoint. If the Interface Adapter does not support Private Data, *private_data_length* must be zero. When *it_ep_accept* is called on the active side of the Connection (i.e., in three-way Connection Establishment), delivery of Private Data is unreliable. |
| *private_data_length* | Length of *private_data*. This field must be 0 if the IA does not support Private Data. |

*it_ep_accept* accepts an incoming Connection Request Event (IT_CM_REQ_CONN_REQUEST_EVENT) or Connection accept arrival Event (IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT). Calling *it_ep_accept* 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 Endpoint. The Event is generated on both the active and passive side of the Connection establishment. See the Communication Management Message Event (*it_cm_msg_events*) reference page for details.

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

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

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

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

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

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

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_PDATA_NOT_SUPPORTED | Private Data was supplied by the Consumer but this Interface Adapter does not support Private Data. |
| IT_ERR_INVALID_PDATA_LENGTH | The Interface Adapter supports Private Data, but the length specified exceeded the Interface Adapter's capabilities. |
| IT_ERR_INVALID_EP | The Endpoint Handle (*ep_handle*) was invalid. |
| IT_ERR_INVALID_EP_STATE | The Endpoint was not in the proper state for the attempted operation. See *it_ep_state_t* reference page. |
| IT_ERR_INVALID_EP_TYPE | The attempted operation was invalid for the Service Type of the Endpoint. |
| IT_ERR_INVALID_CN_EST_ID | The Connection Establishment Identifier (*cn_est_id*) was invalid. |
| IT_ERR_INVALID_EP_ATTR | The Local and Remote Endpoint attributes conflicted. Either the *max_message_size*, the number of *rdma_read_ird*, or the number of *rdma_read_ord* conflicted between the two Endpoints. This error will not be reported on the Passive-side accept of a three-way Connection Establishment. |
| IT_ERR_EP_TIMEWAIT | The Endpoint provided to *it_ep_accept* was in the TimeWait condition, therefore the Connection could not be established. See *it_ep_rc_create* for details of the TimeWait condition. |
| 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.

## ASYNCHRONOUS ERRORS

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

## APPLICATION USAGE

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

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

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

## SEE ALSO

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

# it_ep_connect()

**NAME**

it_ep_connect – initiate an Endpoint Connection establishment request

**SYNOPSIS**

```
#include <it_api.h>

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
);

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

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

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

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

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

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

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

} it_ib_conn_attributes_t;

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

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

```
        void  *unused;

    } it_via_conn_attributes_t;

    /* Transport-specific connection attributes for iWARP */
    typedef struct {

        /* iWARP currently has no transport-specific connection
           attributes. A dummy entry is defined to allow ANSI
           compilation. */
        void  *unused;

    } it_iwarp_conn_attributes_t;

    /* Transport-specific connection attributes. This union is
       discriminated by the transport type being used to form the
       connection. This can be determined by examining
       the transport_type member in the it_ia_info_t that is
       associated with the IA that contains ep_handle. */

    typedef union {
        it_ib_conn_attributes_t     ib;
        it_via_conn_attributes_t    via;
        it_iwarp_conn_attributes_t  iwarp;
    } it_conn_attributes_t;

    typedef enum {
    IT_CONNECT_FLAG_TWO_WAY      = 0x0001,
    IT_CONNECT_FLAG_THREE_WAY    = 0x0002,
    IT_CONNECT_SUPPRESS_IRD_ORD  = 0x0004
    } it_cn_est_flags_t;
```

**APPLICABILITY**

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

**DESCRIPTION**

| | |
|---|---|
| *ep_handle* | Handle of the local Endpoint. |
| *path* | Path for Connection establishment request. |
| *conn_attr* | The transport-specific attributes for the Connection establishment attempt. |
| *connect_qual*: | The Connection Qualifier for which the Consumer is initiating a Connection establishment request. See *it_conn_qual_t* for more information on Connection Qualifiers. |
| *cn_est_flags* | Bitwise OR of flags for the Connection establishment request. |

| 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 *it_ep_accept*. |
| Three-way Connection establishment | IT_CONNECT_FLAG_THREE_WAY | 0x0002 | The Connection is established once the active side of the Connection establishment calls *it_ep_accept*. |
| 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.<br>The IT_CONNECT_ SUPPRESS_IRD_ORD bit is ignored if the *it_ia_info_t* value *ird_ord_ia_support* is IT_FALSE. |

*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.

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

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.

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.

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

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

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

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

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

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

**EXTENDED DESCRIPTION**

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

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_NO_PERMISSION | The Consumer did not have the proper permissions to perform the requested operation. |
| IT_ERR_RESOURCES | The operation failed due to resource limitations. |
| IT_ERR_INVALID_CONN_QUAL | The Connection Qualifier is invalid. |
| IT_ERR_PDATA_NOT_SUPPORTED | Private Data was supplied by the Consumer but this IA does not support Private Data. See *it_ia_query* for the IA's capabilities to support Private Data. |
| IT_ERR_INVALID_PDATA_LENGTH | The IA supports Private Data, but the length specified exceeds the IA's capabilities. |
| IT_ERR_INVALID_SOURCE_PATH | The Source Path specified in the *it_path_t* was invalid. |
| IT_ERR_INVALID_SPIGOT | The Spigot specified in the *it_path_t* was invalid. |
| IT_ERR_INVALID_EP | The *ep_handle* was invalid. |
| IT_ERR_INVALID_EP_STATE | The Endpoint is not in the proper state to be connected. |
| IT_ERR_INVALID_EP_TYPE | The Endpoint Service Type does not support this operation. |
| IT_ERR_INVALID_CN_EST_FLAGS | The Connection establishment flags are invalid. |
| IT_ERR_INVALID_RTIMEOUT | The *conn_attr.ib.remote_cm_timeout* value was invalid. The criteria for determining what constitutes an invalid value are Implementation-dependent. |
| IT_ERR_INVALID_LTIMEOUT | The *conn_attr.ib.local_cm_timeout* value was invalid. The criteria for determining what constitutes an invalid value are Implementation-dependent. |
| IT_ERR_INVALID_RETRY | The *conn_attr.ib.retry_count* value was invalid. The criteria for determining what constitutes an invalid value are Implementation-dependent. |
| IT_ERR_INVALID_RNR_RETRY | The *conn_attr.ib.rnr_retry_count* value was invalid. The criteria for determining what constitutes an invalid value are Implementation-dependent. |
| IT_ERR_INVALID_CM_RETRY | The *conn_attr.ib.max_cm_retries* value was invalid. The criteria for determining what constitutes an invalid value are Implementation-dependent. |
| IT_ERR_INVALID_ATIMEOUT | The *conn_attr.ib.local_ack_timeout* value was invalid. The criteria for determining what constitutes an invalid value are Implementation-dependent. |
| IT_ERR_INVALID_IANA_LR_PORT | The local port in *connect_qual* is already in use. |

| | |
|---|---|
| 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. |

**ASYNCHRONOUS ERRORS**

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

**APPLICATION USAGE**

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

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

**SEE ALSO**

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

# it_ep_disconnect()

## NAME

it_ep_disconnect – disconnect an existing Endpoint-to-Endpoint Connection

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_ep_disconnect (
    IN         it_ep_handle_t  ep_handle,
    IN  const  unsigned char  *private_data,
    IN         size_t          private_data_length
);
```

## APPLICABILITY

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

## DESCRIPTION

| | |
|---|---|
| *ep_handle* | Endpoint to be disconnected. |
| *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. |
| *private_data_length*: | Length of *private_data.* This field must be 0 if the IA does not support Private Data. |

*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.

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

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.

See the *it_ep_state_t* reference page for a complete description of the Endpoint state behavior and transitions.

## RETURN VALUE

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

| | |
|---|---|
| IT_ERR_INVALID_EP_STATE | The Endpoint was not in the proper state for the attempted operation. |
| IT_ERR_INVALID_EP | The Endpoint Handle (*ep_handle*) was invalid. |
| IT_ERR_INVALID_EP_TYPE | The attempted operation was invalid for the Service Type of the Endpoint. |
| 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. |
| IT_ERR_INVALID_PDATA_LENGTH | The Interface Adapter supports Private Data, but the length specified exceeded the Interface Adapter's capabilities. |
| 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. |

**APPLICATION USAGE**

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 Identifiers is indeterminate.

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

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

**SEE ALSO**

*it_ep_accept*(), *it_reject*(), *it_ep_connect*(), *it_ep_state_t*, *it_cm_msg_events*, *it_ep_reset*(), *it_ia_query*()

# it_ep_free()

## NAME

it_ep_free – destroy an RC or UD Endpoint

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_ep_free(
    IN  it_ep_handle_t  ep_handle
);
```

## DESCRIPTION

*ep_handle* Endpoint.

*it_ep_free* destroys an Endpoint.

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

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

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

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

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

## RETURN VALUE

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

| | |
|---|---|
| IT_ERR_INVALID_EP | The Endpoint Handle (*ep_handle*) was invalid. |
| IT_ERR_EP_BUSY | An RMR of type IT_RMR_TYPE_NARROW is still bound to the Endpoint. |
| 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. |

**ASYNCHRONOUS ERRORS**

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

**APPLICATION USAGE**

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

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

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

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

**SEE ALSO**

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

# it_ep_modify()

**NAME**

it_ep_modify – modify attributes of an existing Endpoint

**SYNOPSIS**

```
#include <it_api.h>

it_status_t it_ep_modify(
    IN         it_ep_handle_t      ep_handle,
    IN         it_ep_param_mask_t  mask,
    IN  const  it_ep_attributes_t *ep_attr
);
```

**DESCRIPTION**

| | |
|---|---|
| ep_handle | Endpoint. |
| mask | Bitwise OR of flags for desired attributes to be modified. |
| ep_attr | Pointer to Consumer-allocated structure that contains new Consumer-requested Endpoint attributes. |

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

Attributes to be modified are specified by flags in *mask*. New values for the attributes are specified by the corresponding fields in the structure pointed to by *ep_attr*. See *it_ep_attributes_t* for the definition of the structure.

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

Flag values for attributes that may be potentially modified:

IT_EP_PARAM_MAX_PAYLOAD
IT_EP_PARAM_MAX_REQ_DTO
IT_EP_PARAM_MAX_RECV_DTO
IT_EP_PARAM_RDMA_RD_ENABLE
IT_EP_PARAM_RDMA_WR_ENABLE
IT_EP_PARAM_MAX_IRD
IT_EP_PARAM_MAX_ORD
IT_EP_PARAM_EP_KEY
IT_EP_PARAM_SOFT_HI_WATERMARK
IT_EP_PARAM_HARD_HI_WATERMARK

See *it_ep_attributes_t* for the definition of valid states in which each of the above attributes may be modified.

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

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_INVALID_EP | The Endpoint Handle (*ep_handle*) was invalid. |
| IT_ERR_INVALID_MASK | The mask contained invalid flag values. |
| IT_ERR_INVALID_EP_STATE | The Endpoint was not in the proper state for the attempted operation. See *it_ep_attributes_t*. |
| IT_ERR_INVALID_EP_TYPE | The attempted operation was invalid for the Service Type of the Endpoint. |
| IT_ERR_PAYLOAD_SIZE | The requested *max_dto_payload_size* exceeds the maximum payload size supported by the underlying transport. |
| IT_ERR_RESOURCE_REQ_DTO | The underlying transport could not allocate the requested *max_req_dtos* resources at this time. |
| IT_ERR_RESOURCE_RECV_DTO | The underlying transport could not allocate the requested *max_recv_dtos* resources at this time. |
| IT_ERR_RESOURCE_IRD | The underlying transport could not allocate the requested *rdma_read_ird* resources at this time. |
| IT_ERR_RESOURCE_ORD | The underlying transport could not allocate the requested *rdma_read_ord* resources at this time. |
| IT_ERR_INVALID_EP_KEY | Invalid Endpoint Key value. The Consumer does not have local permissions to use the specified Endpoint Key. |
| 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. |
| 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. |
| 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. |
| IT_ERR_INVALID_RECV_DTO | An attempt was made to modify the *max_recv_dtos* attribute of an Endpoint that has an associated S-RQ. |
| 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. |

**SEE ALSO**

*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()

## NAME

it_ep_query – query an existing Endpoint

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_ep_query(
    IN   it_ep_handle_t      ep_handle,
    IN   it_ep_param_mask_t  mask,
    OUT  it_ep_param_t      *params
);

typedef struct {
it_ia_handle_t              ia;           /* IT_EP_PARAM_IA */
size_t                      spigot_id;    /* IT_EP_PARAM_SPIGOT */
it_ep_state_t               ep_state;     /* IT_EP_PARAM_STATE */
it_transport_service_type_t service_type; /* IT_EP_PARAM_SERV_TYPE */
it_path_t                   dst_path;     /* IT_EP_PARAM_PATH */
it_pz_handle_t              pz;           /* IT_EP_PARAM_PZ */
it_evd_handle_t             request_sevd; /* IT_EP_PARAM_REQ_SEVD */
it_evd_handle_t             recv_sevd;    /* IT_EP_PARAM_RECV_SEVD */
it_evd_handle_t             connect_sevd; /* IT_EP_PARAM_CONN_SEVD */
it_ep_attributes_t          attr;         /* See it_ep_attributes_t
                                             for mask flags for attr */
} it_ep_param_t;
```

## DESCRIPTION

*ep_handle* — Endpoint.

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

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

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

The definition of each field follows:

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

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

*ep_state* — State of the Endpoint.

| | |
|---|---|
| *service_type* | Endpoint Service Type. |
| *dst_path* | 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 *it_ep_connect*; 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. |
| *pz* | Handle for the Protection Zone specified while creating the EP. |
| *request_sevd* | Handle for the IT_DTO_EVENT_STREAM Simple Event Dispatcher for DTO request Completion Events of the created Endpoint. |
| *recv_sevd* | Handle for the IT_DTO_EVENT_STREAM Simple Event Dispatcher for DTO Receive Completion Events of the created Endpoint. |
| *connect_sevd* | Handle for the IT_CM_MSG_EVENT_STREAM Simple Event Dispatcher for Connection Events of the created Endpoint. Invalid for UD Endpoint. |
| *attr* | Attributes of Endpoint – definitions and mask values found in *it_ep_attributes_t*. Consumer ORs the appropriate mask values for each attribute field desired into the *mask* parameter to *it_ep_query*. |

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_INVALID_EP | The Endpoint Handle (*ep_handle*) was invalid. |
| IT_ERR_INVALID_MASK | The mask contained invalid flag values. |
| IT_ERR_INVALID_EP_TYPE | The attempted operation was invalid for the Service Type of the Endpoint. |
| 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. |

**SEE ALSO**

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

# it_ep_rc_create()

## NAME

it_ep_rc_create – create an Endpoint for Reliable Connection

## SYNOPSIS

```
#include <it_api.h>

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
);

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

## APPLICABILITY

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

## DESCRIPTION

| | |
|---|---|
| *pz_handle* | Handle for the Protection Zone of the created Endpoint. Implicitly identifies the Interface Adapter to be used. |
| *request_sevd_handle* | Handle for the IT_DTO_EVENT_STREAM Simple Event Dispatcher for DTO request Completion Events of the created Endpoint. |
| *recv_sevd_handle* | Handle for the IT_DTO_EVENT_STREAM Simple Event Dispatcher for DTO Receive Completion Events of the created Endpoint. |
| *connect_sevd_handle* | Handle for the IT_CM_MSG_EVENT_STREAM Simple Event Dispatcher for Connection Events of the created Endpoint. |
| *flags* | Flags allowing the Consumer optionally to control behavior of the Implementation on Endpoint creation. Default is IT_EP_NO_FLAG. |
| *ep_attr* | Pointer to a structure that contains Consumer-requested Endpoint attributes. |
| *ep_handle* | Handle for the created Endpoint. |

*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.

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

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

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

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

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

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

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

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

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

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

- *srq*
- *soft_hi_watermark*
- *hard_hi_watermark*

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

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

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_INVALID_PZ | The Protection Zone Handle (*pz_handle*) was invalid. |
| IT_ERR_INVALID_REQ_EVD | The Simple Event Dispatcher Handle for Data Transfer Operation request completions (*request_sevd_handle*) was invalid. |
| IT_ERR_INVALID_RECV_EVD | The Simple Event Dispatcher Handle for Data Transfer Operation Receive completions (*recv_sevd_handle*) was invalid. |
| IT_ERR_INVALID_CONN_EVD | The Connection Simple Event Dispatcher Handle was invalid. |
| IT_ERR_INVALID_EVD_TYPE | The Event Stream type for the Event Dispatcher was invalid. |
| IT_ERR_INVALID_REQ_EVD_STATE | The Simple Event Dispatcher for Data Transfer Operation request completions was in an unusable state. |
| IT_ERR_INVALID_RECV_EVD_STATE | The Simple Event Dispatcher for Data Transfer Operation Receive completions was in an unusable state. |
| IT_ERR_INVALID_FLAGS | The *flags* value was invalid. |
| IT_ERR_RESOURCES | The requested operation failed due to insufficient resources. |
| IT_ERR_PAYLOAD_SIZE | The requested *max_dto_payload_size* exceeds the maximum payload size supported by the underlying transport. |

| | |
|---|---|
| IT_ERR_RESOURCE_REQ_DTO | The underlying transport could not allocate the requested *max_req_dtos* resources at this time. |
| IT_ERR_RESOURCE_RECV_DTO | The underlying transport could not allocate the requested *max_recv_dtos* resources at this time. |
| IT_ERR_RESOURCE_SSEG | The underlying transport could not allocate the requested *max_send_segments* resources at this time. |
| IT_ERR_RESOURCE_RSEG | The underlying transport could not allocate the requested *max_recv_segments* resources at this time. |
| IT_ERR_RESOURCE_RRSEG | The underlying transport could not allocate the requested *max_rdma_read_segments* resources at this time. |
| IT_ERR_RESOURCE_RWSEG | The underlying transport could not allocate the requested *max_rdma_write_segments* resources at this time. |
| IT_ERR_RESOURCE_IRD | The underlying transport could not allocate the requested *rdma_read_ird* resources at this time. |
| IT_ERR_RESOURCE_ORD | The underlying transport could not allocate the requested *rdma_read_ord* resources at this time. |
| IT_ERR_INVALID_SRQ | The S-RQ handle was invalid. |
| 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. |
| 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. |
| 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. |
| IT_ERR_PRIV_OPS_UNAVAILABLE | The IA does not support Privileged Mode operations. |
| IT_ERR_NO_PERMISSION | The Consumer is not Privileged. |
| 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. |

**APPLICATION USAGE**

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

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

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

**SEE ALSO**

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

# it_ep_reset()

## NAME

it_ep_reset – reset a Reliable Connected Endpoint to the initial state

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_ep_reset(
    IN  it_ep_handle_t  ep_handle
);
```

## APPLICABILITY

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

## DESCRIPTION

*ep_handle* Reliable Connected Endpoint.

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

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

## RETURN VALUE

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

| | |
|---|---|
| IT_ERR_INVALID_EP | The Endpoint Handle (*ep_handle*) was invalid. |
| IT_ERR_INVALID_EP_STATE | The Endpoint was not in the proper state for the attempted operation. |
| IT_ERR_INVALID_EP_TYPE | The attempted operation was invalid for the Service Type of the Endpoint. |
| IT_ERR_CANNOT_RESET | The Endpoint could not be reset due to an overflow of one of its Data Transfer Operation Event Stream Event Dispatchers. |
| 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. |

## SEE ALSO

*it_ep_rc_create*(), *it_ep_disconnect*(), *it_ep_free*(), *it_ia_info_t*

# it_ep_ud_create()

## NAME

it_ep_ud_create – create an Endpoint for Unreliable Datagram

## SYNOPSIS

```
#include <it_api.h>

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

## APPLICABILITY

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

## DESCRIPTION

| | |
|---|---|
| *pz_handle* | Handle for the Protection Zone of the created Endpoint. Implicitly identifies the Interface Adapter. |
| *request_sevd_handle*: | Handle for the IT_DTO_EVENT_STREAM Simple Event Dispatcher for DTO request Completion Events of the created Endpoint. |
| *recv_sevd_handle*: | Handle for the IT_DTO_EVENT_STREAM Simple Event Dispatcher for DTO Receive Completion Events of the created Endpoint. |
| *ep_attr* | Pointer to a structure that contains Consumer-requested Endpoint Attributes. |
| *spigot_id* | Interface Adapter Spigot identifier to use when creating Endpoint. |
| *ep_handle* | Handle for the created Endpoint. |

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

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

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

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

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

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

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_INVALID_PZ | The Protection Zone Handle (*pz_handle*) was invalid. |
| IT_ERR_INVALID_REQ_EVD | The Simple Event Dispatcher Handle for Data Transfer Operation request completions (*request_sevd_handle*) was invalid. |
| IT_ERR_INVALID_RECV_EVD | The Simple Event Dispatcher Handle for Data Transfer Operation Receive completions (*recv_sevd_handle*) was invalid. |
| IT_ERR_INVALID_EVD_TYPE - | The Event Stream Type for the Event Dispatcher was invalid. |
| IT_ERR_INVALID_REQ_EVD_STATE | The Simple Event Dispatcher for Data Transfer Operation request completions was in an unusable state. |
| IT_ERR_INVALID_RECV_EVD_STATE | The Simple Event Dispatcher for Data Transfer Operation Receive completions was in an unusable state. |
| IT_ERR_INVALID_SPIGOT | An invalid Spigot ID was specified. |
| IT_ERR_RESOURCES | The requested operation failed due to insufficient resources. |
| IT_ERR_PAYLOAD_SIZE | The requested *max_dto_payload_size* exceeds the maximum payload size supported by the underlying transport. |
| IT_ERR_RESOURCE_REQ_DTO | The underlying transport could not allocate the requested *max_req_dtos* resources at this time. |
| IT_ERR_RESOURCE_RECV_DTO | The underlying transport could not allocate the requested *max_recv_dtos* resources at this time. |
| IT_ERR_RESOURCE_SSEG | The underlying transport could not allocate the requested *max_send_segments* resources at this time. |

| | |
|---|---|
| IT_ERR_RESOURCE_RSEG | The underlying transport could not allocate the requested *max_recv_segments* resources at this time. |
| IT_ERR_INVALID_EP_KEY | Invalid Endpoint Key value. The Consumer doesn't have local permissions to use the specified Endpoint Key. |
| IT_ERR_PRIV_OPS_UNAVAILABLE | The IA does not support Privileged Mode operations. |
| IT_ERR_NO_PERMISSION | The Consumer is not Privileged. |
| 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. |

**SEE ALSO**

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

# it_evd_callback_attach()

## NAME

it_evd_callback_attach – attach a callback routine to an Event Dispatcher

## SYNOPSIS

```
#include <it_api.h>

typedef void (*it_evd_callback_rtn_t)(
    IN  it_evd_handle_t  evd,
    IN  void            *arg
);

it_status_t it_evd_callback_attach(
    IN  it_evd_handle_t        evd_handle,
    IN  it_evd_callback_rtn_t  callback,
    IN  void                  *arg
);
```

## DESCRIPTION

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

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

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

*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 an an EVD, an error will be returned.

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.

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:

- 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.
- Calling *it_evd_wait* for an EVD that has a callback routine attached.
- 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).
- Attempting to use *it_evd_callback_attach* to attach a callback routine to an SEVD that is associated with an AEVD.
- Attempting to associate an AEVD with an SEVD that has a callback routine attached.

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

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

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

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

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

- *it_address_handle_create*
- *it_address_handle_modify*
- *it_convert_net_addr*
- *it_ep_connect*
- *it_ep_modify*
- *it_ep_rc_create*
- *it_ep_ud_create*
- *it_evd_create*
- *it_evd_modify*
- *it_evd_wait*
- *it_get_pathinfo*
- *it_ia_create*
- *it_ia_query*
- *it_listen_create*
- *it_lmr_create*
- *it_lmr_modify*
- *it_pz_create*
- *it_rmr_create*
- *it_socket_convert*
- *it_srq_create*
- *it_srq_modify*
- *it_ud_service_request_handle_create*

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

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

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_INVALID_EVD | The Event Dispatcher Handle (*evd_handle*) was invalid. |
| IT_ERR_CALLBACK_EXISTS | A callback routine has already been attached to the input EVD. |
| IT_ERR_EVD_WAIT | A callback routine could not be attached to the EVD because a call to *it_evd_wait* is currently in progress for the EVD. |
| IT_ERR_NO_PERMISSION | The caller was not a Privileged Consumer. |
| IT_ERR_INVALID_EVD_STATE | A callback routine could not be attached to the EVD because another notification mechanism is already enabled. |
| 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. |

**SEE ALSO**

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

# it_evd_callback_detach()

## NAME

it_evd_callback_detach – detach a callback routine from an Event Dispatcher

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_evd_callback_detach(
    IN  it_evd_handle_t  evd_handle
);
```

## DESCRIPTION

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

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

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

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

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

## RETURN VALUE

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

| | |
|---|---|
| IT_ERR_INVALID_EVD | The Event Dispatcher Handle (*evd_handle*) was invalid. |
| IT_ERR_INVALID_EVD_STATE | The attempted operation was invalid for the current state of the Event Dispatcher. |
| IT_ERR_NO_PERMISSION | The caller was not a Privileged Consumer. |
| 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. |

## SEE ALSO

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

# it_evd_create()

## NAME

it_evd_create – create Simple or Aggregate Event Dispatcher

## SYNOPSIS

```
#include <it_api.h>

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
);

#define IT_THRESHOLD_DISABLE 0

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;
```

## DESCRIPTION

| | |
|---|---|
| *ia_handle* | Handle for the Interface Adapter to which the created Event Dispatcher belongs. |
| *event_number* | Identifier for Event Stream type that can be enqueued to the EVD. |
| *evd_flag* | Bitwise OR of flag values for creation operation. |
| *sevd_queue_size* | Minimum size of the Simple EVD Event queue. This parameter is ignored for an Aggregate EVD. |
| *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. |
| *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. |
| *evd_handle* | Handle for the created Event Dispatcher. |

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

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

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

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

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

To create an Aggregate EVD, the *event_number* must be set to IT_AEVD_NOTIFICATION_EVENT_STREAM; a Simple EVD (SEVD) is created otherwise. To create a Simple EVD the *event_number* can be any one of IT_DTO_EVENT_STREAM, IT_CM_REQ_EVENT_STREAM, IT_CM_MSG_EVENT_STREAM, IT_ASYNC_AFF_EVENT_STREAM, IT_ASYNC_UNAFF_EVENT_STREAM, or IT_SOFTWARE_EVENT_STREAM.

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

1. Only one IT_ASYNC_AFF_EVENT_STREAM Simple EVD may be created per Interface Adapter instance. Subsequent calls to *it_evd_create* for the IT_ASYNC_AFF_EVENT_STREAM Event Stream, without intervening calls to *it_evd_free* the EVD, will fail with the error return IT_ERR_ASYNC_AFF_EVD_EXISTS.

2. Only one IT_ASYNC_UNAFF_EVENT_STREAM Simple EVD may be created per Interface Adapter instance. Subsequent calls to *it_evd_create* for the IT_ASYNC_UNAFF_EVENT_STREAM Event Stream, without intervening calls to *it_evd_free* the EVD, will fail with the error return IT_ERR_ASYNC_UNAFF_EVD_EXISTS.

For all Event Stream types except IT_SOFTWARE_EVENT_STREAM, IT_ASYNC_AFF_EVENT_STREAM, and IT_ASYNC_UNAFF_EVENT_STREAM, upon creation there is no Event Stream of *event_number* feeding Events to the created EVD. For an Aggregate EVD this means that there are no Simple EVDs associated with *evd_handle*. No Events are fed to *evd_handle* until *evd_handle* is associated with an object that feeds Events to it. For an Aggregate EVD this means that no Events are fed to *evd_handle* until *evd_handle* is associated with a Simple EVD. For a Simple EVD this means that no Events are fed to *evd_handl*e until *evd_handle* is associated with an Endpoint, Listen Point, or UD Service Request Handle depending on the stream type.

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

Multiple Event Streams of the same Event Stream type can be associated with the same EVD, with the exception of IT_ASYNC_AFF_EVENT_STREAM, IT_ASYNC_UNAFF_EVENT_STREAM, and IT_SOFTWARE_EVENT_STREAM Event Stream types. For the IT_AEVD_NOTIFICATION_EVENT_STREAM Event Stream type, multiple SEVDs can be associated with the same AEVD. For IT_DTO_EVENT_STREAM, multiple EPs can be associated with the same SEVD. For IT_CM_REQ_EVENT_STREAM, multiple Listens can be associated with the same SEVD. For IT_CM_MSG_EVENT_STREAM, multiple RC EPs and/or UD Service Requests can be associated with the same SEVD. For the IT_ASYNC_AFF_EVENT_STREAM, IT_ASYNC_UNAFF_EVENT_STREAM, and IT_SOFTWARE_EVENT_STREAM Event Stream types, only a single Event Stream feeds each EVD, respectively, and the corresponding Event Stream is created upon EVD creation. For IT_SOFTWARE_EVENT_STREAM, the Events are generated explicitly by the Consumer calling *it_evd_post_se*.

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

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

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

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

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

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

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

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

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

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

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

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

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

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 *evd_flag* 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 *evd_flag* 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 *it_evd_modify*). Attempting to wait on the SEVD when disallowed will return IT_ERR_INVALID_EVD_STATE.

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

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

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 *event_number* 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 *event_number* 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 *event_number* is IT_ASYNC_AFF_EVENT_STREAM. It is invalid to set this flag if *event_number* 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 *evd_overflowed* state of the EVD (see *it_evd_modify*) 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. |

*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.

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.

A "Notification Event" is one of the following:

- 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
- Any Event of Event Stream other than IT_DTO_ EVENT_STREAM

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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 *fd*, then Notification will be generated for that AEVD or *fd*. In the AEVD case, as many Notifications may be generated as there are Events available on all SEVDs of the AEVD.

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

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_INVALID_IA | The Interface Adapter Handle (*ia_handle*) was invalid. |
| IT_ERR_INVALID_EVD_TYPE | The Event Stream Type for the Event Dispatcher was invalid. |

| | |
|---|---|
| IT_ERR_INVALID_FLAGS | The flags value was invalid. |
| IT_ERR_RESOURCE_QUEUE_SIZE | The underlying transport could not allocate the requested *sevd_queue_size* resources at this time. |
| IT_ERR_INVALID_THRESHOLD | An invalid value for the Simple Event Dispatcher threshold was specified. |
| IT_ERR_INVALID_AEVD | The Aggregation Event Dispatcher Handle (*aevd_handle*) was invalid. |
| IT_ERR_RESOURCES | The requested operation failed due to insufficient resources. |
| IT_ERR_MISMATCH_FD | An illegal request was made for both the File Descriptor and the Aggregation Event Dispatcher. |
| IT_ERR_AEVD_NOT_ALLOWED | The *aevd_handle* was non-NULL and the *event_number* was IT_AEVD_NOTIFICATION_ EVENT_STREAM. |
| IT_ERR_ASYNC_AFF_EVD_EXISTS | The Asynchronous Affiliated Event Dispatcher already exists. |
| IT_ERR_ASYNC_UNAFF_EVD_EXISTS | The Asynchronous Unaffiliated Event Dispatcher already exists. |
| 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. |

**APPLICATION USAGE**

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

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

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

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

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

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

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

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

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

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

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

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

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

**FUTURE DIRECTIONS**

IT-API support for a callback routine being invoked when an Event is enqueued on an SEVD may be added in the future.

Aggregate EVD support for multiple IAs may be added in the future.

**SEE ALSO**

*it_evd_post_se*(), *it_ep_rc_create*(), *it_ep_ud_create*(), *it_listen_create*(), *it_ud_service_request_handle_create*(), *it_evd_query*(), *it_evd_modify*(), *it_evd_wait*(), *it_evd_dequeue*(), *it_evd_free*(), *it_event_t*, *it_dto_flags_t*, *it_unaffiliated_event_t*, *it_ia_create*(), *it_ia_info_t*

# it_evd_dequeue()

**NAME**

it_evd_dequeue – dequeue for Events from Event Dispatcher

**SYNOPSIS**

```
#include <it_api.h>

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

**DESCRIPTION**

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

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

*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.

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

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

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

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.

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

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.

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.

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

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.

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

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

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

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_QUEUE_EMPTY | There were no entries on the Event Dispatcher queue. |
| IT_ERR_INVALID_EVD | The Event Dispatcher Handle (*evd_handle*) was invalid. |
| IT_ERR_INVALID_EVD_STATE | The attempted operation was invalid for the current state of the Event Dispatcher. |
| 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. |

**APPLICATION USAGE**

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

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

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

**SEE ALSO**

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

# it_evd_free()

**NAME**

it_evd_free – destroy an Event Dispatcher

**SYNOPSIS**

```
#include <it_api.h>

it_status_t it_evd_free(
    IN  it_evd_handle_t  evd_handle
);
```

**DESCRIPTION**

| | |
|---|---|
| *evd_handle* | Handle to Simple or Aggregate Event Dispatcher. |
| *it_evd_free* | Destroys an Event Dispatcher. |

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

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

An AEVD with IT_EVD_DEQUEUE_NOTIFICATIONS set may be dissociated from its SEVDs through use of *it_evd_modify* on each SEVD or through use of *it_evd_free* on each SEVD. An AEVD with the IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* bit clear may be dissociated from SEVDs through use of *it_evd_free* on each SEVD. DTO SEVDs may be disassociated from their DTO Event Streams through use of *it_ep_free* on each associated Endpoint. Communication Management Request SEVDs may be disassociated from their Event Streams through use of *it_listen_free* on each associated listen Handle. Communication Management Message SEVDs may be disassociated from their Event Streams through use of *it_ep_free* on each associated Endpoint.

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

Once *it_evd_free* returns, *evd_handle* may no longer be used.

This operation is applicable to both AEVD and SEVD Handles.

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_INVALID_EVD | The Event Dispatcher Handle (*evd_handle*) was invalid. |
| IT_ERR_EVD_BUSY | The Event Dispatcher was still associated with active Event Streams. |

| | |
|---|---|
| IT_ERR_INVALID_EVD_STATE | The attempted operation was invalid for the current state of the Event Dispatcher. |
| 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. |

**APPLICATION USAGE**

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

**SEE ALSO**

*it_evd_create*(), *it_evd_modify*(), *it_evd_query*(), *it_ep_free*(), *it_listen_free*()

# it_evd_modify()

## NAME

it_evd_modify – modify an existing Event Dispatcher

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_evd_modify(
    IN         it_evd_handle_t      evd_handle,
    IN         it_evd_param_mask_t  mask,
    IN  const  it_evd_param_t      *params
);
```

## DESCRIPTION

*evd_handle* — Simple or Aggregate Event Dispatcher.

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

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

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

```
typedef struct {
    ...
    size_t           sevd_queue_size;  /* IT_EVD_PARAM_QUEUE_SIZE */
    size_t           sevd_threshold;   /* IT_EVD_PARAM_THRESHOLD */
    it_evd_handle_t  aevd;             /* IT_EVD_PARAM_AEVD_HANDLE */
    ...
    it_boolean_t     evd_enabled;      /* IT_EVD_PARAM_ENABLED */
    it_boolean_t     evd_overflowed;   /* IT_EVD_PARAM_OVERFLOWED */
} it_evd_param_t;
```

The definition of each field follows:

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

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

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

| | | |
|---|---|---|
| | | Attempting to associate an Aggregate EVD with an SEVD that has a callback routine attached will return an IT_ERR_CALLBACK_EXISTS error code. |
| | *evd_enabled* | Consumer may set this *it_boolean_t* to the value IT_TRUE to indicate that an EVD should notify an associated AEVD or *fd* when Notification criteria are reached. Clearing *evd_enabled* (making it equal to IT_FALSE) will disable this capability. May be done at any time. |
| | *evd_overflowed* | Consumer may clear this *it_boolean_t* (make it equal to IT_FALSE) to reset an overflow condition on the EVD. See *it_evd_create* for more details. |

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

The new AEVD may have IT_EVD_DEQUEUE_NOTIFICATIONS set or cleared.

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

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

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

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

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

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

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

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

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

Enabling the enabled EVD has no effect.

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

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

Disabling the disabled EVD has no effect.

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_INVALID_EVD | The Event Dispatcher Handle (*evd_handle*) was invalid. |
| IT_ERR_INVALID_MASK | The mask contained invalid flag values. |
| IT_ERR_INVALID_EVD_STATE | The attempted operation was invalid for the current state of the Event Dispatcher. |
| IT_ERR_RESOURCE_QUEUE_SIZE | The underlying transport could not allocate the requested *sevd_queue_size* resources at this time. |
| IT_ERR_INVALID_QUEUE_SIZE | The requested Simple Event Dispatcher queue size (*sevd_queue_size*) was less than the outstanding Events on the Event queue. |
| IT_ERR_INVALID_THRESHOLD | An invalid value for the Simple Event Dispatcher threshold was specified. |
| IT_ERR_INVALID_AEVD | The Aggregation Event Dispatcher Handle (*aevd*) was invalid. |
| IT_ERR_RESOURCES | The requested operation failed due to insufficient resources. |
| IT_ERR_CALLBACK_EXISTS | An attempt was made to associate an AEVD with an SEVD that has a callback routine attached. |
| 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. |

**SEE ALSO**

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

# it_evd_post_se()

**NAME**

it_evd_post_se – post software Event on Simple Event Dispatcher

**SYNOPSIS**

```
#include <it_api.h>

it_status_t it_evd_post_se(
    IN         it_evd_handle_t  evd_handle,
    IN  const  void            *event
);
```

**DESCRIPTION**

| | |
|---|---|
| *evd_handle* | Simple Event Dispatcher of IT_SOFTWARE_EVENT_STREAM Event Stream type. |
| *event* | Pointer to the Consumer-created Software Event. |

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

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

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

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_EVD_QUEUE_FULL | The Simple Event Dispatcher queue was full. |
| IT_ERR_INVALID_EVD | The Event Dispatcher Handle (*evd_handle*) was invalid. |
| IT_ERR_INVALID_SOFT_EVD | The Simple Event Dispatcher Handle (*evd_handle*) was not an IT_SOFTWARE_EVENT_STREAM Event Dispatcher. |
| 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. |

**APPLICATION USAGE**

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

**SEE ALSO**

*it_evd_create*(), *it_software_event_t*, *it_evd_wait*()

# it_evd_query()

**NAME**

it_evd_query – query an existing Simple or Aggregate Event Dispatcher

**SYNOPSIS**

```
#include <it_api.h>

it_status_t it_evd_query(
    IN   it_evd_handle_t      evd_handle,
    IN   it_evd_param_mask_t  mask,
    OUT  it_evd_param_t      *params
);

typedef enum {
    IT_EVD_PARAM_ALL           = 0x000001,
    IT_EVD_PARAM_IA            = 0x000002,
    IT_EVD_PARAM_EVENT_NUMBER  = 0x000004,
    IT_EVD_PARAM_FLAG          = 0x000008,
    IT_EVD_PARAM_QUEUE_SIZE    = 0x000010,
    IT_EVD_PARAM_THRESHOLD     = 0x000020,
    IT_EVD_PARAM_AEVD_HANDLE   = 0x000040,
    IT_EVD_PARAM_FD            = 0x000080,
    IT_EVD_PARAM_BOUND         = 0x000100,
    IT_EVD_PARAM_ENABLED       = 0x000200,
    IT_EVD_PARAM_OVERFLOWED    = 0x000400,
    IT_EVD_PARAM_CALLBACK      = 0x000800
} it_evd_param_mask_t;

typedef struct {
    it_ia_handle_t   ia;               /* IT_EVD_PARAM_IA */
    it_event_type_t  event_number;     /* IT_EVD_PARAM_EVENT_NUMBER*/
    it_evd_flags_t   evd_flag;         /* IT_EVD_PARAM_FLAG */
    size_t           sevd_queue_size;  /* IT_EVD_PARAM_QUEUE_SIZE */
    size_t           sevd_threshold;   /* IT_EVD_PARAM_THRESHOLD */
    it_evd_handle_t  aevd;             /* IT_EVD_PARAM_AEVD_HANDLE*/
    int              fd;               /* IT_EVD_PARAM_FD */
    it_boolean_t     evd_bound;        /* IT_EVD_PARAM_BOUND */
    it_boolean_t     evd_enabled;      /* IT_EVD_PARAM_ENABLED */
    it_boolean_t     evd_overflowed;   /* IT_EVD_PARAM_OVERFLOWED */
    it_boolean_t     evd_callback;     /* IT_EVD_PARAM_CALLBACK */
} it_evd_param_t;
```

**DESCRIPTION**

| | |
|---|---|
| *evd_handle* | Event Dispatcher. |
| *mask* | Bitwise OR of flags for requested EVD parameters. |
| *params* | Pointer to Consumer-allocated structure that the Implementation fills with Consumer-requested Event Dispatcher parameters. |

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

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

The definition of each field follows:

| | |
|---|---|
| *ia* | Handle for the Interface Adapter. |
| *event_number* | Identifier for Event Stream type that can be enqueued to the EVD. |
| *evd_flag* | Flags for Event Dispatcher. See *it_evd_create* for definitions and use of *evd_flag*. |
| *sevd_queue_size* | Minimum size of the SEVD Event queue or zero for an AEVD. |
| *sevd_threshold* | The number of non-notification Events on the Simple Event Dispatcher queue for Notification, unblocking. |
| *aevd* | Handle for Aggregate EVD associated with SEVD or IT_NULL_HANDLE if none. |
| *fd* | *File descriptor* corresponding to Event Dispatcher or -1 if none. |
| *evd_bound* | When it has the value IT_TRUE, indicates that the EVD is tied to an Event Stream so Events can be queued on EVD. For an AEVD, indicates that SEVDs are tied to the AEVD. |
| *evd_enabled* | When it has the value IT_TRUE, indicates: for an SEVD that it has been configured to notify an associated AEVD or *fd* when Notification criteria is reached; for an AEVD that it has been configured to notify an associated *fd* when it is notified by one of its associated SEVDs. See *it_evd_modify*. |
| *evd_overflowed* | When it has the value IT_TRUE, indicates that the EVD has overflowed. See *it_evd_create* for more details. |
| *evd_callback* | When it has the value IT_TRUE, indicates that a callback routine has been attached to the EVD. See *it_evd_callback_attach*. |

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_INVALID_EVD | The Event Dispatcher Handle (*evd_handle*) was invalid. |
| IT_ERR_INVALID_MASK | The mask contained invalid flag values. |
| 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. |

**SEE ALSO**

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

# it_evd_wait()

**NAME**

it_evd_wait – wait for Events on Event Dispatcher

**SYNOPSIS**

```
#include <it_api.h>

it_status_t it_evd_wait(
    IN   it_evd_handle_t  evd_handle,
    IN   uint64_t         timeout,
    OUT  it_event_t      *event,
    OUT  size_t          *nmore
);
```

**DESCRIPTION**

| | |
|---|---|
| *evd_handle* | Handle for Simple or Aggregate Event Dispatcher. |
| *timeout* | The duration of time, in microseconds, that the Consumer is willing to wait for an Event. |
| *event* | Pointer to the Consumer-allocated structure that the Implementation fills with the Event information. |
| *nmore* | The snapshot of the number of Events queued on the EVD at the time of *it_evd_wait* return. Only applicable for SEVD. |

*it_evd_wait* 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.

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

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

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

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

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

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

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

An SEVD waiter will return immediately if there is a Notification Event (see *it_evd_create* for the definition of the Notification Event) on the queue or if the number of Events on the SEVD is larger than or equal to *sevd_threshold*. An AEVD waiter with the IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* bit cleared will return immediately if there is a Notification Event on any of the associated SEVDs or any of the associated SEVDs has a number of Events larger than or equal to its *sevd_threshold*. An AEVD waiter with the IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* bit set will return immediately if there is an IT_AEVD_NOTIFICATION_EVENT available.

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

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

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

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

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

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

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

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_WAITER_LIMIT | No more waiters are permitted for the Event Dispatcher. |
| IT_ERR_INVALID_EVD | The Event Dispatcher Handle (*evd_handle*) was invalid. |
| IT_ERR_INVALID_EVD_STATE | The attempted operation was invalid for the current state of the Event Dispatcher. |
| IT_ERR_ABORT | The Event Dispatcher has been destroyed. |
| IT_ERR_INTERRUPT | The Event Dispatcher waiter was unblocked by a signal. |
| IT_ERR_TIMEOUT_EXPIRED | The operation timed out. |
| IT_ERR_CALLBACK_EXISTS | The wait for Notification was disallowed because the EVD has a callback routine attached. |
| 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. |

**APPLICATION USAGE**

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

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

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

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

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

- The underlying Implementation does not support thresholding.
- The underlying Implementation does not support IT_NOTIFY_FLAG.

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

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

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

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

**SEE ALSO**

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

# it_get_consumer_context()

## NAME

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

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_get_consumer_context(
    IN   it_handle_t   handle,
    OUT  it_context_t *context
);
```

## DESCRIPTION

| | |
|---|---|
| *handle* | Handle of the IT-API object associated with the Consumer Context to be retrieved. |
| *context* | The address of the location where the retrieved Consumer Context is returned. |

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

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

## RETURN VALUE

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

| | |
|---|---|
| IT_ERR_INVALID_HANDLE | The *handle* was invalid. |
| IT_ERR_NO_CONTEXT | The *handle* does not have an associated Context. |
| 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. |

## EXAMPLES

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

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

## SEE ALSO

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

# it_get_handle_type()

## NAME

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

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_get_handle_type(
    IN   it_handle_t             handle,
    OUT  it_handle_type_enum_t  *type_of_handle
);
```

## DESCRIPTION

| | |
|---|---|
| *handle* | Handle of an IT-API object. |
| *type_of_handle* | Type of the Handle of *handle*. |

The *it_get_handle_type* interface allows the Consumer to retrieve the type of an IT Object using its Handle. See *it_handle_t* for a description of the Handle types and associated enumeration values returned.

## RETURN VALUE

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

| | |
|---|---|
| IT_ERR_INVALID_HANDLE | The *handle* was invalid. |
| 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. |

## SEE ALSO

*it_handle_t*

# it_get_pathinfo()

## NAME

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

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_get_pathinfo(
    IN      it_ia_handle_t        ia_handle,
    IN      size_t                spigot_id,
    IN      const it_net_addr_t  *net_addr,
    IN OUT  size_t               *num_paths,
    OUT     size_t               *total_paths,
    OUT     it_path_t            *paths
);
```

## DESCRIPTION

| | |
|---|---|
| *ia_handle* | The Handle for the IA that the caller wishes to use for communicating with the remote Network Address. |
| *spigot_id* | The Spigot on the IA that the caller wishes to use for communicating with the remote Network Address. |
| *net_addr* | The remote Network Address with which to communicate. |
| *num_paths* | On input, points to the count of the maximum number of Paths that the Consumer wishes to have returned. On output, points to the count of the total number of Paths that were actually returned, which is guaranteed to be less than or equal to the number that the Consumer requested. This is only valid on output if the call returns IT_SUCCESS. |
| *total_paths* | The total number of Paths that were available to access the remote Network Address. This may be greater than the number of Paths returned via *num_paths* if there were more Paths available than the maximum the Consumer wished to have returned. This is only valid if the call returns IT_SUCCESS. |
| *paths* | An array allocated by the Consumer that holds the returned Path(s). This only contains valid information if the call returns IT_SUCCESS. |

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

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

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

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

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

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

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

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

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_INVALID_IA | The Interface Adapter Handle (*ia_handle*) was invalid. |
| IT_ERR_INVALID_SPIGOT | An invalid Spigot ID was specified. |
| IT_ERR_INVALID_ADDRESS | The Network Address specified in *net_addr* was invalid. |
| IT_ERR_INVALID_NETADDR | The type of the Network Address specified in *net_addr* was not recognized. |
| IT_ERR_INTERRUPT | The call was unblocked by a signal. |
| 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. |

**SEE ALSO**

*it_interface_list*(), *it_ia_create*(), *it_ia_query*(), *it_ep_connect*(), *it_listen_create*()

# it_handoff()

## NAME

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

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_handoff(
    IN  const  it_conn_qual_t         *conn_qual,
    IN         size_t                  spigot_id,
    IN         it_cn_est_identifier_t  cn_est_id
);

typedef uint64_t it_cn_est_identifier_t;
```

## DESCRIPTION

| | |
|---|---|
| *conn_qual* | The Connection Qualifier to which the Connection Request should be forwarded. |
| *spigot_id* | Interface Adapter Spigot to which the Connection Request should be forwarded. |
| *cn_est_id* | Connection establishment identifier associated with the Connection Request to be forwarded. |

*it_handoff* forwards a Connection Request to the specified Spigot and Connection Qualifier of the IA on which the Connection Request originally arrived. Specifying a *conn_qual* or *spigot_id* 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. The forwarded Connection Request generates an IT_CM_REQ_CONN_REQUEST_EVENT Event at the listen point to which the request was forwarded. Forwarded Connection Request Events look identical to the original Events, therefore the Consumer cannot distinguish them. The connection establishment identifier, *cn_est_id*, is destroyed by this function.

## RETURN VALUE

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

| | |
|---|---|
| ERR_INVALID_CN_EST_ID | The connection establishment identifier (*cn_est_id*) was invalid. |
| IT_ERR_INVALID_CONN_QUAL | The Connection Qualifier (*conn_qual*) was invalid. |
| IT_ERR_INVALID_SPIGOT | An invalid Spigot ID was specified. |
| 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. |

**APPLICATION USAGE**

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

**SEE ALSO**

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

# it_hton64()

## NAME

it_hton64, it_ntoh64 – convert 64-bit integers between host and network byte order

## SYNOPSIS

```
#include <it_api.h>

uint64_t it_hton64(
    uint64_t  hostint
);

uint64_t it_ntoh64(
    uint64_t  netint
);
```

## DESCRIPTION

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

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

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

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

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

## RETURN VALUE

Both functions always succeed and return their converted input argument.

## APPLICATION USAGE

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

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

## SEE ALSO

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

# it_ia_create()

**NAME**

it_ia_create – create an Interface Adapter

**SYNOPSIS**

```
#include <it_api.h>

it_status_t it_ia_create(
    IN   const char      *name,
    IN   uint32_t         major_version,
    IN   uint32_t         minor_version,
    OUT  it_ia_handle_t  *ia_handle
);
```

**DESCRIPTION**

| | |
|---|---|
| *name* | The name of the Interface for which to create an Interface Adapter. |
| *major_version* | The IT-API major version that the Consumer will use in subsequent calls to the IA. |
| *minor_version* | The IT-API minor version that the Consumer will use in subsequent calls to the IA. |
| *ia_handle* | Upon successful return, points to an Interface Adapter Handle for the created Interface Adapter. |

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

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

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

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

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_RESOURCES | The requested operation failed due to insufficient resources. |
| IT_ERR_INVALID_NAME | The specified *name* was invalid. |
| IT_ERR_INVALID_MAJOR_VERSION | The requested IT-API major version number was not supported for this Interface Adapter. |
| IT_ERR_INVALID_MINOR_VERSION | The requested IT-API minor version number was not supported for this Interface Adapter. |

**SEE ALSO**

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

# it_ia_free()

## NAME

it_ia_free – free Interface Adapter Handle

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_ia_free(
    IN  it_ia_handle_t  ia_handle
);
```

## DESCRIPTION

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

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

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

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

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

## RETURN VALUE

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

| | |
|---|---|
| IT_ERR_INVALID_IA | The Interface Adapter Handle (*ia_handle*) was invalid. |
| IT_ERR_INVALID_EVD_STATE | The attempted operation was invalid for the current state of the Event Dispatcher. |

## SEE ALSO

*it_ia_create*(), *it_ia_query*()

# it_ia_info_free()

## NAME

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

## SYNOPSIS

```
#include <it_api.h>

void it_ia_info_free(
    IN  it_ia_info_t  *ia_info
);
```

## DESCRIPTION

| | |
|---|---|
| ia_info | Points to an *it_ia_info_t* data structure that was previously returned from a call to *it_ia_query*. |

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

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

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

## SEE ALSO

*it_ia_info_t*(), *it_ia_query*()

# it_ia_query()

## NAME

it_ia_query – retrieve attributes of given Interface Adapter and its Spigots

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_ia_query(
    IN   it_ia_handle_t  ia_handle,
    OUT  it_ia_info_t  **ia_info
);
```

## DESCRIPTION

| | |
|---|---|
| *ia_handle* | Identifies the Interface Adapter to be queried. |
| *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. |

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

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

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

## RETURN VALUE

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

| | |
|---|---|
| IT_ERR_INVALID_IA | The Interface Adapter Handle (*ia_handle*) was invalid. |
| IT_ERR_RESOURCES | The requested operation failed due to insufficient resources. |
| 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. |

## SEE ALSO

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

# it_interface_list()

## NAME

it_interface_list – retrieve information about the available Interfaces

## SYNOPSIS

```
#include <it_api.h>

void it_interface_list(
    OUT     it_interface_t  *interfaces,
    IN OUT  size_t          *num_interfaces,
    IN OUT  size_t          *total_interfaces
);

typedef struct {

    /* Most recent major version number of the IT-API supported by the
      Interface. */
    uint32_t  major_version;

    /* Most recent minor version number of the IT-API supported by the
       Interface. */
    uint32_t  minor_version;

    /* The transport that the Interface uses, as defined in
       it_ia_info_t. */
    it_transport_type_t  transport_type;

    /* The name of the Interface, suitable for input to it_ia_create.
       The name is a string of maximum length IT_INTERFACE_NAME_SIZE,
       including the terminating NULL character. */
    char  name[IT_INTERFACE_NAME_SIZE];
} it_interface_t;
```

## DESCRIPTION

| | |
|---|---|
| *interfaces* | An array allocated by the Consumer that contains the information returned for the Interface(s). |
| *num_interface* | 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. |
| *total_interfaces* | 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. |

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

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

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

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

**EXAMPLES**

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

```
it_interface_t  interface;
size_t          num_interfaces;
it_ia_handle_t  ia;
it_ia_info_t   *infop;

num_interfaces = 1;
it_interface_list(&interface, &num_interfaces, IT_NO_ADDR);
if (num_interfaces != 1) {

/* Failed to find any IAs; */
}

if (it_ia_create( interface.name, interface.major_version,
interface.minor_version, &ia) != IT_SUCCESS) {

/* The IA wasn't found. Assuming sufficient resources were available,
   this can happen if the Interface that was retrieved by the
   it_interface_list call is no longer available. */
}

if (it_ia_query(ia, &infop) != IT_SUCCESS) {

/* This can happen if the Implementation didn't have sufficient
   resources to return the information for the IA. */
}

if ((infop->transport_type != interface.transport_type) {
```

```
        /* This can happen if the Interface that was retrieved by the
           it_interface_list call is no longer available. (A different
           Interface with the same name as was retrieved by it_interface_list
           is available as it turns out. The most likely reason this happened
           is that a new Interface was added to the system between the time
           it_interface_list was called and the time that it_ia_create was
           called.) */
        }

        /* Validation of the information returned by it_interface_list is
           complete. */
```

**APPLICATION USAGE**

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

**SEE ALSO**

*it_ia_create*(), *it_ia_info_t*

# it_listen_create()

**NAME**

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

**SYNOPSIS**

```
#include <it_api.h>

it_status_t it_listen_create(
    IN      it_ia_handle_t       ia_handle,
    IN      size_t               spigot_id,
    IN      it_evd_handle_t      connect_evd,
    IN      it_listen_flags_t    flags,
    IN OUT  it_conn_qual_t      *conn_qual,
    OUT     it_listen_handle_t  *listen_handle
);

typedef enum {
    IT_LISTEN_NO_FLAG           = 0x0000,
    IT_LISTEN_CONN_QUAL_INPUT   = 0x0001,
    IT_LISTEN_SUPPRESS_IRD_ORD  = 0x0002
} it_listen_flags_t;
```

**DESCRIPTION**

| | |
|---|---|
| *ia_handle*: | Interface Adapter Handle. |
| *spigot_id* | Interface Adapter Spigot identifier. |
| *connect_evd* | The Handle of the Simple Event Dispatcher where Connection Request Events for this Listen Point will be posted. The Event Stream Type of the Simple Event Dispatcher must be IT_CM_REQ_EVENT_STREAM. |
| *flags* | Bitwise OR of flag values. Can specify whether the Connection Qualifer is an input or output parameter and whether or not IRD/ORD values are to be used (if applicable). |
| *conn_qual* | The Connection Qualifier for which the Consumer wants to listen for Connection Requests. |
| *listen_handle* | Upon successful return points to a Handle to the created Listen Point. |

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

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

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

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

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

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_INVALID_CONN_QUAL | The Connection Qualifier (*conn_qual*) was invalid. |
| IT_ERR_CONN_QUAL_BUSY | The Connection Qualifier was already in use. |
| IT_ERR_NO_PERMISSION | The Consumer did not have the proper permissions to perform the requested operation. |
| IT_ERR_RESOURCES | The requested operation failed due to insufficient resources. |
| IT_ERR_INVALID_CONN_EVD | The Connection Simple Event Dispatcher Handle was invalid. |
| IT_ERR_INVALID_IA | The Interface Adapter Handle (*ia_handle*) was invalid. |
| IT_ERR_INVALID_SPIGOT | An invalid Spigot ID was specified. |
| IT_ERR_INVALID_FLAGS | The *flags* value was invalid. |
| IT_ERR_INVALID_EVD_TYPE | The Event Stream Type for the Event Dispatcher was invalid. |
| 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. |

**SEE ALSO**

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

# it_listen_free()

## NAME

it_listen_free – free a Listen Point

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_listen_free(
    IN  it_listen_handle_t  listen_handle
);
```

## DESCRIPTION

*listen_handle* Identifies the Listen Point to be destroyed.

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

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

## RETURN VALUE

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

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

## SEE ALSO

*it_listen_create*(), *it_listen_query*()

# it_listen_query()

## NAME

it_listen_query – query parameters associated with a Listen Point

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_listen_query(
    IN   it_listen_handle_t      listen_handle,
    IN   it_listen_param_mask_t  mask,
    OUT  it_listen_param_t      *params
);

typedef enum {
    IT_LISTEN_PARAM_ALL          = 0x0001,
    IT_LISTEN_PARAM_IA_HANDLE    = 0x0002,
    IT_LISTEN_PARAM_SPIGOT_ID    = 0x0004,
    IT_LISTEN_PARAM_CONNECT_EVD  = 0x0008,
    IT_LISTEN_PARAM_CONN_QUAL    = 0x0010
} it_listen_param_mask_t;

typedef struct {
    it_ia_handle_t   ia_handle;     /* IT_LISTEN_PARAM_IA_HANDLE */
    size_t           spigot_id;     /* IT_LISTEN_PARAM_SPIGOT_ID */
    it_evd_handle_t  connect_evd;   /* IT_LISTEN_PARAM_CONNECT_EVD */
    it_conn_qual_t   connect_qual;  /* IT_LISTEN_PARAM_CONN_QUAL */
} it_listen_param_t;
```

## DESCRIPTION

| | |
|---|---|
| *listen_handle* | Handle associated with the Listen Point being queried. |
| *mask* | Bitwise OR of flags for desired parameters. |
| *params* | Pointer to Consumer-allocated structure whose members are written with the desired Listen Point parameters and attributes. |

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

The definition of each field of *params* follows:

| | |
|---|---|
| *ia_handle*: | Interface Adapter Handle. |
| *spigot_id* | Interface Adapter Spigot identifier. |
| *connect_evd* | The Handle of the Simple Event Dispatcher where incoming Connection Request Events for this Listen Point are posted. |
| *connect_qual* | The Connection Qualifier associated with the Listen Point. |

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_INVALID_LISTEN | The Listen Point Handle (*listen_handle*) was invalid. |
| IT_ERR_INVALID_MASK | The mask contained invalid flag values. |
| 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. |

**SEE ALSO**

*it_listen_free*(), *it_listen_create*()

# it_lmr_create()

## NAME

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

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_lmr_create(
    IN      it_pz_handle_t     pz_handle,
    IN      void              *addr,
    IN      it_iobl_t         *iobl,
    IN      it_length_t        length,
    IN      it_addr_mode_t     addr_mode,
    IN      it_mem_priv_t      privs,
    IN      it_lmr_flag_t      flags,
    IN      uint32_t           shared_id,
    OUT     it_lmr_handle_t   *lmr_handle,
    IN OUT  it_rmr_context_t  *rmr_context
);

typedef uint32_t it_rmr_context_t;

#ifdef IT_32BIT
    typedef uint32_t it_length_t;  /* a 32-bit platform */
#else
    typedef uint64_t it_length_t;  /* a 64-bit platform */
#endif

typedef enum {
    IT_LMR_FLAG_NONE          = 0x0001,
    IT_LMR_FLAG_SHARED        = 0x0002,
    IT_LMR_FLAG_NONCOHERENT   = 0x0004,
    IT_LMR_FLAG_NON_SHAREABLE = 0x0008
} it_lmr_flag_t;
```

## DESCRIPTION

| | |
|---|---|
| *pz_handle* | Protection Zone in which to create LMR. |
| *addr* | Starting address of LMR. The *addr* argument is ignored if an IOBL is provided and *addr_mode* is IT_ADDR_MODE_RELATIVE. |
| *iobl* | I/O Buffer List (IOBL, Privileged Consumers only). |
| *length* | Length of LMR in bytes. |
| *addr_mode* | Addressing mode of LMR to be created. |
| *privs* | Bitwise OR of access privilege values for LMR, taken from *it_mem_priv_t*. |
| *flags* | Bitwise OR of modifier flags. |
| *shared_id* | Identifier for sharing Interface Adapter translation resources. |

*lmr_handle* Returned Handle for created LMR.

*rmr_context* Optionally returned Context allowing remote access to this LMR.

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

Non-Privileged Consumers must set *iobl* to IT_NO_ADDR.

Privileged Consumers may pass an I/O Buffer List in the *iobl* argument (*it_iobl_t*). Jointly with the *length* argument, the IOBL specifies the I/O Address range of the region to be linked. The first byte of the region is identified by the First-Byte Offset (*fbo*) member of the IOBL. The length of the IOBL is the number of bytes from the first byte of the region to the last byte of the IOBL. The *length* argument specifies the total size of the LMR in bytes, which is upper-bounded by the length of the IOBL.

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

If an IOBL is provided and Absolute Addressing is used, the *addr* parameter specifies the Base Address of the LMR; moreover, if the IOBL is a Page List (IOBL element length is guaranteed to be a power of two), the absolute address (*addr)* modulo the length (*elt_len*) of the first IOBL element must equal the First-Byte Offset (*fbo*) in the IOBL. If an IOBL is provided and Relative Addressing is used, the *addr* parameter is ignored.

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

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

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

| | |
|---|---|
| IT_LMR_FLAG_SHARED | This flag may be used to conserve finite Interface Adapter translation resources by sharing resources between multiple LMRs. The LMRs may have been created in the caller's process or in a different process. If set, then the Implementation will create a new LMR that re-uses resources from a matching LMR that has the IT_LMR_FLAG_NON_SHAREABLE flag cleared. Two LMRs are defined as matching if they have the same *shared_id* attribute, correspond to the same list of physical memory pages, have the same First-Byte Offset and the same length, correspond to the same I/O Address range, and have the same coherency mode. If any of these conditions are not met, the LMRs do not match. If a matching LMR is not found among the LMRs with the IT_LMR_FLAG_NON_SHAREABLE flag cleared, then a new LMR is created. |
| IT_LMR_FLAG_NON_SHAREABLE | Controls whether an LMR may be shared. If set, then the LMR is not eligible to be shared. Otherwise, the LMR may be shared. |
| IT_LMR_FLAG_NONCOHERENT | Controls whether an LMR is created in coherent or non-coherent mode. Coherent mode is the default and is supported by all Implementations. Non-coherent mode is not supported by all Implementations, and IT_LMR_FLAG_NONCOHERENT is silently ignored on such Implementations. |

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

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

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

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

After a memory range has been registered with the IA using *it_lmr_create*, the Consumer should not call routines outside of the IT-API that would invalidate any part of the memory referred to by the LMR or revoke access privileges that were granted to the IA at registration time.

Disallowed operations include but are not limited to unmapping part of the range using *munmap*, revoking privileges using *mprotect*, unlocking memory using *munlock*, truncating a file for file-backed regions, etc. Violation of this rule may result in DTO failures, data corruption in the Consumer's LMR, and/or program termination. These effects may extend to other Consumer processes if the LMR is in shared memory. However, the Implementation must prevent any adverse effect on unrelated processes that do not use this memory object.

**EXTENDED DESCRIPTION**

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

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

**BACKWARDS COMPATIBILITY**

*it_lmr_create* called with *iobl* set to IT_NO_ADDR behaves identically to *it_lmr_create* (v2.0).

*it_lmr_create* called with *iobl* set to IT_NO_ADDR and *addr_mode* set to IT_ADDR_MODE_ABSOLUTE behaves identically to *it_lmr_create* (v1.0).

See also Appendix C.

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_ACCESS | The Consumer was not allowed to have the requested memory privileges. |
| IT_ERR_FAULT | Part or all of the supplied address range was invalid. |
| IT_ERR_INVALID_ADDR_MODE | The addressing mode (*addr_mode*) was invalid or unsupported. |
| IT_ERR_INVALID_FLAGS | The *flags* value was invalid. |
| IT_ERR_INVALID_IOBL | The IOBL specified with *iobl* 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 (*iobl_num_elts* as reported via *it_lmr_query*), if it includes elements of an unsupported length, or if it has an invalid First-Byte Offset (*fbo*). An IOBL that is a Page List is invalid if an IOBL element has an invalid page |

| | |
|---|---|
| | size or is not page aligned or, for Absolute Addressing only, if the absolute address (*addr)* modulo the length (*elt_len*) of the first IOBL element does not equal the First-Byte Offset (*fbo*) in the IOBL. |
| IT_ERR_INVALID_LENGTH | An IOBL was provided and the requested LMR length (*length*) exceeded the length of the IOBL. |
| IT_ERR_INVALID_PRIVS | The requested memory privileges (*privs*) contained an invalid flag. |
| IT_ERR_INVALID_PZ | The Protection Zone Handle (*pz_handle*) was invalid. |
| IT_ERR_RESOURCES | The requested operation failed due to insufficient resources. |
| IT_ERR_RESOURCE_LMR_LENGTH | The underlying transport could not allocate an LMR of the requested *length* at this time. |
| IT_ERR_PRIV_OPS_UNAVAILABLE | The IA does not support Privileged Mode operations. |
| IT_ERR_NO_PERMISSION | The Consumer is not Privileged. |
| 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. |

**APPLICATION USAGE**

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

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

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

A Consumer wishing to prevent an LMR becoming shared as a side-effect of another Consumer creating a shared LMR should set the IT_LMR_NON_SHAREABLE flag.

A Consumer interested in (eventually) sharing a group of LMRs should clear the IT_LMR_NON_SHAREABLE flag, optionally set the IT_LMR_FLAG_SHARED flag, and specify a distinct value of *shared_id* for that group when creating these LMRs. The use of distinct values of *shared_id* for unrelated groups of shared LMRs improves the efficiency of the matching process, which searches only among LMRs having the same *shared_id* value. However, using distinct values of *shared_id* for unrelated LMRs is not necessary. For example, if uncoordinated Consumers supply the same value for *shared_id*, matches for an LMR will still be found if they exist, with no false matches, but the search may take longer on some

Implementations. It is recommended that uncoordinated Consumers wishing to take advantage of potential sharing opportunities should adopt the convention of setting *shared_id* to zero.

**SEE ALSO**

*it_lmr_free*(), *it_lmr_query*(), *it_lmr_modify*(), *it_lmr_flush_to_mem*(), *it_lmr_refresh_from_mem*(), *it_addr_mode_t*, *it_iobl_t*, *it_mem_priv_t*

# it_lmr_create_unlinked()

## NAME

it_lmr_create_unlinked – create an unlinked Local Memory Region (LMR)

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_lmr_create_unlinked(
    IN   it_pz_handle_t    pz_handle,
    IN   size_t            iobl_num_elts,
    IN   it_lmr_flag_t     flags,
    IN   uint32_t          shared_id,
    OUT  it_lmr_handle_t  *lmr_handle
);
```

## APPLICABILITY

*it_lmr_create_unlinked* is supported only if the Interface Adapter attribute *lmr_link_support* (see *it_ia_info_t*) is IT_TRUE.

## DESCRIPTION

| | |
|---|---|
| *pz_handle* | Protection Zone in which to create LMR. |
| *iobl_num_elts* | Number of entries allowed in corresponding IOBL. |
| *flags* | Bitwise OR of modifier flags. |
| *shared_id* | Identifier for sharing Interface Adapter translation resources. |
| *lmr_handle* | Returned Handle for created LMR. |

The *it_lmr_create_unlinked* routine allows a Privileged Consumer to request that an Interface Adapter allocate a set of Local Memory Region (LMR) resources without associating them with a range of local memory addresses. The LMR is returned in the unlinked state and is not fully usable until linked successfully using the *it_lmr_link* routine.

The Interface Adapter is implicitly identified by the *pz_handle* argument.

The *iobl_num_elts* argument identifies the requested number of IOBL elements to be allocated for this LMR handle. The Implementation may allocate more than the requested amount. The actual allocated will be at least that requested and may be queried using the *it_lmr_query* routine.

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

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

| | |
|---|---|
| IT_LMR_FLAG_NON_SHAREABLE | Controls whether an LMR may be shared. If set, then the LMR is not eligible to be shared. Otherwise, the LMR may be shared. |

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

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

The LMR is created in the unlinked state. For an unlinked LMR, the IT_LMR_FLAG_SHARED flag for the LMR will be reported as clear when queried.

The newly created LMR Handle is returned in the *lmr_handle* argument.

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_INVALID_FLAGS | The *flags* value was invalid. |
| IT_ERR_INVALID_PZ | The Protection Zone Handle (*pz_handle*) was invalid. |
| IT_ERR_OP_NOT_SUPPORTED | The IA has no LMR Link support (*lmr_link_support* in *it_ia_info_t* is IT_FALSE). |
| IT_ERR_NO_PERMISSION | The Consumer is not Privileged. |
| IT_ERR_RESOURCE_IOBL_LENGTH | The underlying transport could not allocate an LMR with the requested number of IOBLs (*iobl_num_elts*) at this time. |
| IT_ERR_RESOURCES | The requested operation failed due to insufficient resources. |
| 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. |

**APPLICATION USAGE**

Memory that is to be the Source or Destination of a DTO must first be registered. Using *it_lmr_create_unlinked*, the Consumer requests allocation of an unlinked LMR with a capacity for a number of IOBL entries. The Consumer then calls *it_lmr_link* with an IOBL with less than or equal to *iobl_num_elts* entries to register the memory with the Interface Adapter. If the LMR is linked with flags that enable remote access, then the Consumer typically passes the returned RMR Context to a remote peer using a DTO. The remote peer uses the RMR Context in RDMA calls such as *it_post_rdma_write* that access memory within the range of the LMR.

Having linked an LMR, the Consumer may unlink it locally using *it_lmr_unlink* or also by setting the IT_UNLINK_LOCAL_SINK flag on RDMA Read WRs or may unlink it remotely using *it_post_send_and_unlink*. Having unlinked an LMR, the Consumer may subsequently link it again.

If the LMR created with *it_lmr_create_unlinked* becomes shared, it will no longer be possible to unlink the LMR (via a local *it_lmr_link* or via a remote *it_post_send_and_unlink*) – instead the Consumer must free the LMR (*it_lmr_free*). A Consumer wishing to prevent an LMR becoming shared as a side-effect of another Consumer creating a shared LMR should set the IT_LMR_NON_SHAREABLE flag. For more details on sharing LMRs, see Application Usage of *it_lmr_create*.

**SEE ALSO**

*it_lmr_link*(), *it_lmr_unlink*(), *it_lmr_free*(), *it_lmr_query*(), *it_lmr_modify*(), *it_lmr_flush_to_mem*(), *it_lmr_refresh_from_mem*(), *it_lmr_create*(), *it_post_send_and_unlink*(), *it_addr_mode_t*, *it_mem_priv_t*

# it_lmr_flush_to_mem()

## NAME

it_lmr_flush_to_mem – make memory changes visible to an incoming RDMA Read or Atomic operation

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_lmr_flush_to_mem(
    IN  const  it_lmr_triplet_t  *local_segments,
    IN         size_t             num_segments
);
```

## DESCRIPTION

*local_segments* Array of buffer segments.

*num_segments* Number of segments in the array.

The *it_lmr_flush_to_mem* routine is needed if and only if an LMR was created in non-coherent mode using IT_LMR_FLAG_NONCOHERENT.

If a Local Memory Region is created in non-coherent mode, then the Consumer must call *it_lmr_flush_to_mem* after modifying data in a memory range in this region that will be the target of an *incoming* RDMA Read or Atomic operation. *it_lmr_flush_to_mem* must be called after the Consumer has modified the memory range but before the RDMA Read or Atomic operation starts, and the memory range that will be accessed by the RDMA Read or Atomic must be supplied by the caller in the *local_segments* array. After this call returns, the RDMA Read or Atomic operation may safely see the modified contents of the memory range. It is permissible to batch synchronizations for multiple RDMA Read operations in a single call, by passing a *local_segments* array that includes all modified memory ranges. The *local_segments* entries need not contain the same LMR, and need not be in the same Protection Zone.

If an RDMA Read or Atomic operation on an LMR created in non-coherent mode attempts to read from a memory range that is not properly synchronized using *it_lmr_flush_to_mem*, the returned contents are undefined.

If the *local_segments* specified by the Consumer contain both normal LMRs (i.e., those created with IT_LMR_FLAG_NONCOHERENT cleared) and non-coherent LMRs (i.e., those created with IT_LMR_FLAG_NONCOHERENT set), only the non-coherent LMRs will be synchronized by this routine; the normal LMRs will be unaffected.

## RETURN VALUE

This call is a no-op and always returns successfully if the Implementation does not support non-coherent mode, or if none of the LMRs in *local_segments* were created using the IT_LMR_FLAG_NONCOHERENT flag.

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

| | |
|---|---|
| IT_ERR_RANGE | The address range for a local segment fell outside the boundaries of the corresponding Local Memory Region and the Local Memory Region was created in non-coherent mode. |

| | |
|---|---|
| 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. |

**APPLICATION USAGE**

Determining when an RDMA Read or Atomic will start and what memory range it will read is the Consumer's responsibility. One possibility is to have the Consumer that is modifying memory call *it_lmr_flush_to_mem* and then post a Send DTO message that identifies the range in the body of the Send. The Consumer wishing to do the RDMA Read or Atomic can receive this message and thus know when it is safe to initiate the RDMA Read or Atomic operation.

**SEE ALSO**

*it_lmr_create*(), *it_lmr_refresh_from_mem*(), *it_lmr_triplet_t*

# it_lmr_free()

## NAME

it_lmr_free – destroy a Local Memory Region

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_lmr_free(
    IN  it_lmr_handle_t  lmr_handle
);
```

## DESCRIPTION

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

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

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

## RETURN VALUE

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

| | |
|---|---|
| IT_ERR_INVALID_LMR | The Local Memory Region Handle (*lmr_handle*) was invalid. |
| IT_ERR_LMR_BUSY | The Local Memory Region was still referenced by a Remote Memory Region. |
| 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. |

## SEE ALSO

*it_lmr_create*(), *it_lmr_create_unlinked*(), *it_lmr_query*(), *it_lmr_modify*()

# it_lmr_link()

## NAME

it_lmr_link – post operation to Link an unlinked Local Memory Region to an I/O Address range

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_lmr_link(
    IN   it_lmr_handle_t    lmr_handle,
    IN   const void        *addr,
    IN   const it_iobl_t   *iobl,
    IN   it_length_t        length,
    IN   it_addr_mode_t     addr_mode,
    IN   it_mem_priv_t      privs,
    IN   it_ep_handle_t     ep_handle,
    IN   it_dto_cookie_t    cookie,
    IN   it_dto_flags_t     dto_flags,
    OUT  it_rmr_context_t  *rmr_context
);
```

## APPLICABILITY

*it_lmr_link* is applicable only to Privileged Mode Endpoints created for the RC service type and is supported only if the Interface Adapter attribute *lmr_link_support* (see *it_ia_info_t*) is IT_TRUE.

## DESCRIPTION

| | |
|---|---|
| *lmr_handle* | Unlinked LMR to be linked. |
| *addr* | Absolute address when *addr_mode* is IT_ADDR_MODE_ABSOLUTE. The *addr* argument is ignored if *addr_mode* is IT_ADDR_MODE_RELATIVE. |
| *iobl* | I/O Buffer List specifying the I/O Address range of the LMR to be linked jointly with *length*. |
| *length* | Length of I/O Address range in bytes. Must not be 0. |
| *addr_mode* | Addressing mode of the LMR to be linked. |
| *privs* | Bitwise OR of remote access privilege flags for the linked LMR, taken from *it_mem_priv_t*. |
| *ep_handle* | Endpoint on which to post the Link operation. |
| *cookie* | Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the LMR Link operation. |
| *dto_flags* | Bitwise OR of options for operation handling. |
| *rmr_context* | Returned Context allowing remote access to the linked LMR. |

The *it_lmr_link* routine posts to Endpoint *ep_handle* an operation to Link the currently unlinked Local Memory Region *lmr_handle* (see *it_lmr_create_unlinked* and *it_lmr_unlink*) to the memory specified by the *iobl* and *length* arguments (and possibly the *addr* argument).

The Base Address of the LMR is given by *addr* for Absolute Addressing and by the I/O Address of the first byte of the region, identified by the *iobl* argument, for Relative Addressing. See also *addr_mode* below for addressing modes. The region extends for *length* bytes.

The *iobl* argument (*it_iobl_t*) provides an I/O Buffer List and, jointly with the *length* argument, specifies the I/O Address range of the region to be linked. The first byte of this region is identified by the First-Byte Offset (*fbo*) member of the IOBL. The length of the IOBL is the number of bytes from the first byte of the region to the last byte of the IOBL. The *length* argument specifies the total size of the LMR in bytes, which is upper-bounded by the length of the IOBL.

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

If Absolute Addressing is used and the IOBL is a Page List (IOBL element length is guaranteed to be a power of two), the absolute address (*addr)* modulo the length (*elt_len*) of the first IOBL element must equal the First-Byte Offset (*fbo*) in the IOBL. If Relative Addressing is used, the LMR is fully specified by the IOBL and the *addr* parameter is ignored.

An LMR can be linked only if the Endpoint *ep_handle* is in IT_EP_STATE_CONNECTED state and the Endpoint is Privileged.

An LMR can be linked only if it is not currently in linked state.

The LMR handle must be valid. The Protection Zones of the LMR and Endpoint must match.

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

The *cookie* (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work Request. This identifier is completely under Consumer control and opaque to the Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted Work Request.

Request handling is specified by the *dto_flags* argument and is the bitwise OR of zero or more of the following flags:

IT_COMPLETION_FLAG
IT_NOTIFY_FLAG
IT_BARRIER_FENCE_FLAG

For the definition of these flags, see *it_dto_flags_t*. In addition, *it_lmr_link* automatically fences all DTO, Link, and Unlink operations subsequently submitted on the Endpoint *ep_handle* such that none of these operations starts until the currently posted Link operation has completed.

An RMR Context allowing remote access to the memory region will be created if the *privs* argument includes either IT_PRIV_REMOTE_READ or IT_PRIV_REMOTE_WRITE. The Context will be returned in the location pointed to by *rmr_context* if the input value of *rmr_context* is not IT_NO_ADDR. The returned *rmr_context* is only valid if *privs* includes remote privileges and the call returns successfully; otherwise, it is undefined. The RMR Context may also be retrieved using *it_lmr_query*. The *rmr_context* is returned in network byte order, and may be passed by value to any remote process that wishes to use the Context in DTOs that target the corresponding LMR.

The value of *rmr_context* is immediately available when *it_lmr_link* returns, but it may not be used by a remote host for an RDMA operation until the Link Completion Event occurs. Violation of this rule may result in an error and a broken Connection for the reliable Connection Endpoint on which the RDMA operation is posted. See Application Usage for more details.

The completion of the posted Link operation is reported asynchronously to the Consumer according to the rules defined in *it_dto_flags_t*. An LMR Link Completion Event is of type *it_dto_cmpl_event_t*. Any LMR Link Completion Event generated manifests on the EVD associated with the Endpoint Send Queue. The Event type is IT_LMR_LINK_CMPL_EVENT.

The linking remains valid until the Unlink operation completes successfully, or until the RMR is destroyed. A Link operation will never be partially successful over a subset of the requested memory range.

If *it_lmr_link* returns successfully, but the Link Completion Event status indicates failure, then RMR Context is invalid.

Any Work Request posted to an Endpoint's Send Queue after a call to *it_lmr_link* will not begin execution until the Link operation has completed.

**EXTENDED DESCRIPTION**

For certain Implementations, not all LMRs may be unlinked. This may be the case for [IB-R1.2], if an LMR is created via a Verb that does not pass an IOBL (PBL); i.e., the IB Register Memory Region verb. Consumers may get a completion error when trying to unlink LMRs created with *it_lmr_create*. An LMR created with *it_lmr_create_unlinked* and subsequently linked may always be unlinked. Any successfully unlinked LMR may be linked again.

**RETURN VALUE**

A successful call returns IT_SUCCESS. Otherwise, an immediate error is returned and, if the RMR was previously linked, the previous linking and RMR Context remains valid. It is possible for *it_lmr_link* to return success but for the Completion Event to indicate failure.

Posting to an Endpoint that is not in the IT_EP_STATE_CONNECTED or IT_EP_STATE_NONOPERATIONAL state will return the IT_ERR_INVALID_EP_STATE immediate error.

The possible immediate errors for *it_lmr_link* are listed below:

| | |
|---|---|
| IT_ERR_INVALID_ADDR_MODE | The addressing mode (*addr_mode*) was invalid or unsupported. |
| IT_ERR_INVALID_EP | The Endpoint Handle (*ep_handle*) was invalid. |
| IT_ERR_INVALID_EP_STATE | The Endpoint was not in the proper state for the attempted operation. |
| IT_ERR_INVALID_EP_TYPE | The attempted operation was invalid for the Service Type of the Endpoint. |
| IT_ERR_INVALID_DTO_FLAGS | The Data Transfer Operation flags (*dto_flags*) value was invalid. |
| IT_ERR_INVALID_IOBL | The IOBL specified with *iobl* was invalid or of an unsupported type. |
| IT_ERR_INVALID_LMR | The Local Memory Region Handle (*lmr_handle*) was invalid. |
| IT_ERR_INVALID_PRIVS | The requested memory privileges (*privs*) contained an invalid flag. |
| IT_ERR_OP_NOT_SUPPORTED | The IA has no LMR Link support (*lmr_link_support* in *it_ia_info_t* is IT_FALSE). |
| IT_ERR_RESOURCES | The requested operation failed due to insufficient resources. |
| IT_ERR_TOO_MANY_POSTS | The operation failed due to an overflow of a Work Queue. |
| 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. |

**ASYNCHRONOUS ERRORS**

For Work Requests posted to an Endpoint in the IT_EP_STATE_CONNECTED state, a completion status (see *it_dto_status_t*) other than IT_DTO_SUCCESS will break the Connection by moving the Endpoint to the IT_EP_STATE_NONOPERATIONAL state and deliver an IT_CM_MSG_CONN_BROKEN_EVENT Event to the Connect EVD of *ep_handle*. Once the Connection is broken, all outstanding and in-progress operations on the Connection will complete with an error status.

Any posting to an Endpoint that is in the IT_EP_STATE_NONOPERATIONAL state will be flushed with completion status set to IT_DTO_ERR_FLUSHED.

An IT_DTO_ERR_LOCAL_MM_OPERATION completion status results in case of an invalid LMR handle, unless the Implementation handles this error as an immediate error.

An IT_DTO_ERR_LOCAL_MM_OPERATION completion status results if the Protection Zones of the LMR and the Endpoint do not match or if *privs* represents an invalid combination of access privileges.

If *lmr_handle* represents a linked LMR or if *lmr_handle* is the Direct LMR Handle, a Completion Error is generated with completion status set to IT_DTO_ERR_LOCAL_MM_OPERATION.

An IT_DTO_ERR_LOCAL_MM_OPERATION completion status results if the number of IOBL elements exceeds the *iobl_num_elts* LMR attribute, if the IOBL includes elements of an unsupported length, or if the IOBL has an invalid First-Byte Offset (*fbo*).

An IT_DTO_ERR_LOCAL_MM_OPERATION completion status also results for IOBLs that are Page Lists, if an IOBL element has an invalid page size or is not page aligned or, for Absolute Addressing only, if the absolute address (*addr)* modulo the length (*elt_len*) of the first IOBL element does not equal the First-Byte Offset (*fbo*) in the IOBL.

If the Endpoint to which the *it_lmr_link* Work Request is posted is not Privileged, then a Completion Error with the DTO status IT_DTO_ERR_LOCAL_MM_OPERATION shall be returned.

For the iWARP Transport only, if the *length* of the LMR exceeds the length of the IOBL or if *length* equals zero, a Completion Error is generated with completion status set to IT_DTO_ERR_MM_OPERATION_ERROR.

**APPLICATION USAGE**

The *it_lmr_link* operation is lightweight compared to creating an LMR. An application concerned with efficiency would typically create one or more unlinked LMRs at initialization time; later, it would link (unlink) LMRs as needed to enable (disable) local or remote access to regions of memory, possibly multiple times.

The Consumer should use unique identifiers for *cookie* if they desire to identify each DTO. If the Consumer does not require a unique DTO identifier, the value of zero or NULL (IT_NO_ADDR) can be used.

The local Consumer has several options for ensuring that the remote Consumer does not use *rmr_ context* before the Link Completion Event occurs. One is to wait for the Completion Event on the Send EVD of the specified Endpoint *ep_handle* before sending the *rmr_context* to a peer. Another option is to send the *rmr_context* to a peer by posting a DTO to the same Endpoint *ep_handle* that was used to Link the LMR. The barrier-fencing behavior of *it_lmr_link* ensures that the DTO does not start until the Link Completion Event has occurred. If the Link fails with a Completion Error, the Connection will be broken and the DTO flushed, so the *rmr_context* will not be sent.

For reasons already described, the completion of an LMR Link operation represents an important change that Consumers may need to monitor. One way to do this is to set IT_COMPLETION_FLAG in *dto_flags*, which will generate a Completion Event to indicate when the LMR has been linked. Consumers who do not set IT_COMPLETION_FLAG must rely on ordering semantics to infer when the LMR has been successfully linked. For example, if a subsequent DTO posted to the Send Queue of the same EP completes successfully, then the Link operation has completed, because DTOs posted to the Send Queue of an EP must wait for a Link operation to complete before processing. If the Link operation fails, a Link Completion Event is generated regardless of the use of IT_COMPLETION_FLAG.

When an EVD holding the DTO Event Stream associated with an Endpoint overflows, on some transports the behavior of the Endpoint and/or the EVD is undefined, making recovery from such an overflow either impossible or at the very least IA-dependent. Consumers should therefore take care to avoid this situation. Each Work Request that a Consumer posts to a Work Queue can result in a Completion Event being enqueued in an EVD, due to the possibility of an error occurring during Work Request processing. To avoid any possibility of overflow, the Consumer must therefore ensure that the total number of Outstanding Operations on all Work Queues that can enqueue Completion Events into a given EVD never exceeds the minimum queue size (as reported by *it_evd_query*) of that EVD.

**FUTURE DIRECTIONS**

None.

**SEE ALSO**

*it_lmr_create*(), *it_lmr_create_unlinked*(), *it_lmr_unlink*(), *it_dto_flags_t*, *it_dto_events*, *it_addr_mode_t*, *it_iobl_t*, *it_mem_priv_t*, *it_ia_info_t*

# it_lmr_modify()

## NAME

it_lmr_modify – modify selected attributes of a Local Memory Region

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_lmr_modify(
    IN         it_lmr_handle_t      lmr_handle,
    IN         it_lmr_param_mask_t  mask,
    IN  const  it_lmr_param_t      *params
);
```

## DESCRIPTION

*lmr_handle* — Local Memory Region.

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

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

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

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

*privs* — The new memory access privileges for the LMR. See *it_mem_priv_t*, *it_lmr_create*, and *it_lmr_link* for flag definitions and restrictions.

If remote access privileges are specified in *privs* and the routine returns successfully, then a new RMR Context has been created and associated with the LMR. The LMR's new associated RMR Context may be retrieved using *it_lmr_query*. The Implementation of *it_lmr_modify* strives to generate a new RMR Context that is different from any RMR Contexts generated previously. As long as the new RMR Context does not match a previously generated RMR Context, remote access using a previously generated RMR Context is revoked. If no remote access privileges are specified in *privs* and the routine returns successfully, then no RMR Context remains associated with the LMR, and remote access to the LMR is disabled. If the *lmr_handle* is invalidated due to an error, any associated RMR Context is invalidated, and any IA access to the LMR is disabled.

A Local Memory Region may not be modified if it is still referenced by bound Remote Memory Regions; an attempt to do so will fail with an error return, and the LMR will not be modified or affected.

The Consumer should not modify an LMR whose LMR Handle or RMR Context is used in outstanding DTOs, LMR Link, LMR Unlink, RMR Link, or RMR Unlink operations. The Consumer must dequeue the Completion Events for all such operations prior to modifying the LMR. If this rule is not followed, the Outstanding Operations may fail and complete with an error status.

If the *lmr_handle* or *mask* parameter is invalid, or the LMR has any RMRs linked to it, then the operation fails and, if *lmr_handle* was valid, the LMR is not modified or affected and any RMRs linked to the LMR are also unaffected.

For all other errors, the *lmr_handle* and any associated RMR Context are invalidated and may no longer be used. Any resources that were associated with the LMR are freed, as if the Consumer was calling *it_lmr_free* on the *lmr_handle*.

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_ACCESS | The Consumer was not allowed to have the requested memory privileges. |
| IT_ERR_INVALID_LMR | The Local Memory Region Handle (*lmr_handle*) was invalid. |
| IT_ERR_INVALID_MASK | The *mask* contained invalid flag values. |
| IT_ERR_INVALID_PRIVS | The requested memory privileges (*privs*) contained an invalid flag. |
| IT_ERR_INVALID_PZ | The Protection Zone Handle (*pz_handle*) was invalid. |
| IT_ERR_LMR_BUSY | The Local Memory Region was still referenced by a Remote Memory Region. |
| IT_ERR_RESOURCES | The requested operation failed due to insufficient resources. |
| 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. |

**APPLICATION USAGE**

Although *it_lmr_modify* can be used to change the remote access privileges for an LMR, this is a much more expensive operation than linking an RMR to an LMR using *it_rmr_link*.

**SEE ALSO**

*it_lmr_create*(), *it_lmr_free*(), *it_lmr_link*(), *it_lmr_query*(), *it_lmr_unlink*(), *it_rmr_link*(), *it_rmr_unlink*(), *it_addr_mode_t*, *it_lmr_param_t*, *it_lmr_param_mask_t*, *it_mem_priv_t*

# it_lmr_query()

**NAME**

it_lmr_query – get attributes of a Local Memory Region

**SYNOPSIS**

```
#include <it_api.h>

it_status_t it_lmr_query(
    IN   it_lmr_handle_t      lmr_handle,
    IN   it_lmr_param_mask_t  mask,
    OUT  it_lmr_param_t      *params
);

typedef enum {
    IT_LMR_PARAM_ALL            = 0x000001,
    IT_LMR_PARAM_IA             = 0x000002,
    IT_LMR_PARAM_PZ             = 0x000004,
    IT_LMR_PARAM_ADDR           = 0x000008,
    IT_LMR_PARAM_LENGTH         = 0x000010,
    IT_LMR_PARAM_MEM_PRIV       = 0x000020,
    IT_LMR_PARAM_FLAG           = 0x000040,
    IT_LMR_PARAM_SHARED_ID      = 0x000080,
    IT_LMR_PARAM_RMR_CONTEXT    = 0x000100,
    IT_LMR_PARAM_ACTUAL_ADDR    = 0x000200,
    IT_LMR_PARAM_ACTUAL_LENGTH  = 0x000400,
    IT_LMR_PARAM_ADDR_MODE      = 0x000800,
    IT_LMR_PARAM_LINKED         = 0x001000,
    IT_LMR_PARAM_IOBL_NUM_ELTS  = 0x002000,
    IT_LMR_PARAM_IOBL_TYPE      = 0x004000
} it_lmr_param_mask_t;

typedef struct {
    it_ia_handle_t    ia;            /* IT_LMR_PARAM_IA */
    it_pz_handle_t    pz;            /* IT_LMR_PARAM_PZ */
    void             *addr;          /* IT_LMR_PARAM_ADDR */
    it_length_t       length;        /* IT_LMR_PARAM_LENGTH */
    it_mem_priv_t     privs;         /* IT_LMR_PARAM_MEM_PRIV */
    it_lmr_flag_t     flags;         /* IT_LMR_PARAM_FLAG */
    uint32_t          shared_id;     /* IT_LMR_PARAM_SHARED_ID */
    it_rmr_context_t  rmr_context;   /* IT_LMR_PARAM_RMR_CONTEXT */
    void             *actual_addr;   /* IT_LMR_PARAM_ACTUAL_ADDR */
    it_length_t       actual_length; /* IT_LMR_PARAM_ACTUAL_LENGTH */
    it_addr_mode_t    addr_mode;     /* IT_LMR_PARAM_ADDR_MODE */
    it_boolean_t      linked;        /* IT_LMR_PARAM_LINKED */
    size_t            iobl_num_elts; /* IT_LMR_PARAM_IOBL_NUM_ELTS */
    it_iobl_type_t    iobl_type;     /* IT_LMR_PARAM_IOBL_TYPE */
} it_lmr_param_t;
```

**DESCRIPTION**

*lmr_handle* Local Memory Region.

| | |
|---|---|
| *mask* | Bitwise OR of flags for desired parameters. |
| *params* | Structure whose members are written with the desired parameters. |

The *it_lmr_query* routine returns the desired attributes of the Local Memory Region *lmr_handle* in the structure pointed to by *params*. On return, each field of *params* is only valid if the corresponding flag as shown in the Synopsis is set in the *mask* argument. The *mask* value IT_LMR_PARAM_ALL causes all fields to be returned.

The definition of each field of *params* follows:

| | |
|---|---|
| *ia* | The Interface Adapter Handle specified to create the LMR. |
| *pz* | The Protection Zone Handle specified to create the LMR. |
| *addr* | The requested starting address or, equivalently, Base Address of the LMR. To be interpreted as an absolute address, regardless of *addr_mode*. |
| *length* | The requested length in bytes of the LMR. |
| *privs* | The memory access privileges of the LMR. For the definition of bits, see *it_mem_priv_t*. |
| *flags* | Attributes of the LMR according to *it_lmr_flags_t*. The attribute corresponding to IT_LMR_FLAG_SHARED will be set if the LMR has been matched with another LMR (i.e., shared) or clear if the LMR is not. |
| *shared_id* | The *shared_id* specified to create the LMR. |
| *rmr_context* | The RMR Context associated with the LMR, or undefined if *privs* does not include remote access. Returned in network byte order. |
| *actual_addr* | The actual starting address for which bounds checking is done for data transfers. To be interpreted as an absolute address, regardless of *addr_mode*. |
| *actual_length* | The actual length for which bounds checking is done for data transfers. |
| *addr_mode* | The addressing mode of the LMR. |
| *linked* | Flag identifying whether or not the LMR is currently linked. |
| *iobl_num_elts* | The number of IOBL entries allocated by this LMR. Valid if LMR was created using *it_lmr_create_unlinked* (validity is Implementation-dependent otherwise). |
| *iobl_type* | The type of IOBL linked to this LMR. Valid if LMR was created using *it_lmr_create_unlinked* (validity is Implementation-dependent otherwise). |

**EXTENDED DESCRIPTION**

Implementations based on iWARP and Implementations on any transport supporting Relative Addressing will support LMR creation and bounds checking with byte-level granularity.

Implementations based on InfiniBand 1.2 can be expected to support LMR creation and bounds checking with byte-level granularity.

For Implementations with byte-level granularity for LMRs, the actual starting address and actual length will equal the requested starting address and requested length, respectively. For LMRs with Relative Addressing, LMR byte-level granularity ensures that the starting address of the LMR is unambiguous.

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_INVALID_LMR | The Local Memory Region Handle (*lmr_handle*) was invalid. |
| IT_ERR_INVALID_MASK | The *mask* contained invalid flag values. |
| 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. |

**SEE ALSO**

*it_lmr_create*(), *it_lmr_create_unlinked*(), *it_lmr_free*(), *it_lmr_link*(), *it_lmr_modify*(), *it_lmr_unlink*(), *it_addr_mode_t*, *it_mem_priv_t*

# it_lmr_refresh_from_mem()

**NAME**

it_lmr_refresh_from_mem – make effects of an incoming RDMA Write or Atomic operation visible to Consumer

**SYNOPSIS**

```
#include <it_api.h>

it_status_t it_lmr_refresh_from_mem(
    IN  const  it_lmr_triplet_t  *local_segments,
    IN         size_t             num_segments
);
```

**DESCRIPTION**

*local_segments* Array of buffer segments.

*num_segments* Number of segments in the array.

The *it_lmr_refresh_from_mem* routine is needed if and only if an LMR was created in non-coherent mode using IT_LMR_FLAG_NONCOHERENT.

If a Local Memory Region is created in non-coherent mode, then the Consumer must call *it_lmr_refresh_from_mem* before reading data from a memory range in this region that was the target of an *incoming* RDMA Write operation. *it_lmr_refresh_from_mem* must be called after the RDMA Write operation completes, and the memory range that was modified by the RDMA Write must be supplied by the caller in the *local_segments* array. After this call returns, the Consumer may safely see the modified contents of the memory range. It is permissible to batch synchronizations of multiple RDMA Write operations in a single call, by passing a *local_segments* array that includes all modified memory ranges. The *local_segments* entries need not contain the same LMR, and need not be in the same Protection Zone.

The Consumer must also use *it_lmr_refresh_from_mem* when performing local writes to a memory range that was or will be the target of incoming RDMA Writes. After performing the local write, the Consumer must call *it_lmr_refresh_from_mem* before the RDMA Write is initiated. Conversely, after an RDMA Write completes, the Consumer must call *it_lmr_refresh_from_mem* before performing a local write to the same range.

If the Consumer attempts to read from a memory range in an LMR that was created in non-coherent mode, without properly synchronizing using *it_lmr_refresh_from_mem*, the returned contents are undefined. If the Consumer attempts to write to a memory range without properly synchronizing, the contents of the memory range become undefined.

If the *local_segments* specified by the Consumer contain both normal LMRs (i.e., those created with IT_LMR_FLAG_NONCOHERENT cleared) and non-coherent LMRs (i.e., those created with IT_LMR_FLAG_NONCOHERENT set), only the non-coherent LMRs will be synchronized by this routine; the normal LMRs will be unaffected.

**RETURN VALUE**

This call is a no-op and always returns successfully if the Implementation does not support non-coherent mode, or if none of the LMRs in *local_segments* were created using the IT_LMR_FLAG_NONCOHERENT flag.

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

| | |
|---|---|
| IT_ERR_RANGE | The address range for a local segment fell outside the boundaries of the corresponding Local Memory Region and the Local Memory Region was created in non-coherent mode. |
| 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. |

**APPLICATION USAGE**

Determining when an RDMA Write completes and determining which memory range was modified is the Consumer's responsibility. One possibility is for the RDMA Write initiator to post a Send DTO message after each RDMA Write that identifies the range in the body of the Send. The Consumer at the target of the RDMA Write can receive the message and thus know when and how to call *it_lmr_refresh_from_mem*.

**SEE ALSO**

*it_lmr_create*(), *it_lmr_flush_to_mem*(), *it_lmr_triplet_t*

# it_lmr_unlink()

## NAME

it_lmr_unlink – post operation to Unlink a Local Memory Region from its memory range

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_lmr_unlink(
    IN  it_lmr_handle_t  lmr_handle,
    IN  it_ep_handle_t   ep_handle,
    IN  it_dto_cookie_t  cookie,
    IN  it_dto_flags_t   dto_flags
);
```

## APPLICABILITY

*it_lmr_unlink* is applicable only to Endpoints created for the RC service type and is supported only if the Interface Adapter attribute *lmr_link_support* (see *it_ia_info_t*) is IT_TRUE.

## DESCRIPTION

| | |
|---|---|
| *lmr_handle* | Handle of LMR that will be unlinked. |
| *ep_handle* | Endpoint on which to post the operation. |
| *cookie* | Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the operation. |
| *dto_flags* | Bitwise OR of options for operation handling. |

The *it_lmr_unlink* routine posts to Endpoint *ep_handle* an Unlink operation to unlink the Local Memory Region specified by *lmr_handle*. The *it_lmr_unlink* routine does not require Privilege (see Application Usage).

An LMR can be unlinked only if the Endpoint *ep_handle* is in IT_EP_STATE_CONNECTED state.

The Protection Zones of the LMR and Endpoint must match.

The *cookie* (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work Request. This identifier is completely under Consumer control and opaque to the Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted Work Request.

Request handling is specified by the *dto_flags* argument as the bitwise OR of zero or more of the following flags:

IT_COMPLETION_FLAG
IT_NOTIFY_FLAG
IT_BARRIER_FENCE_FLAG
IT_UNLINK_FENCE_FLAG

For the definition of these flags, see *it_dto_flags_t*.

The completion of the posted Unlink operation is reported asynchronously to the Consumer according to the rules defined in *it_dto_flags_t*. An LMR Unlink Completion Event is of type *it_dto_cmpl_event_t*. Any generated LMR Unlink Completion Event manifests on the EVD associated with the Endpoint Send Queue. The Event type is IT_LMR_LINK_CMPL_EVENT.

After a successful Unlink Completion Event, the previous linking for the LMR is invalidated, and any RMR Context for the LMR is no longer defined. Any RDMA operation that uses a previous RMR Context will fail with a protection violation; beware that this may include operations that are outstanding when *it_lmr_unlink* is called. The Consumer must ensure that such operations have completed prior to calling *it_lmr_unlink* if successful completions are desired. An Unlink operation will never be partially successful over a subset of the requested memory range; it either succeeds completely or fails without invalidating any portion of the previous linking.

If *it_lmr_unlink* returns successfully but the Completion Event status indicates failure, then the previous linking and RMR Context remains valid.

Any Work Request posted to an Endpoint's Send Queue after a call to *it_lmr_unlink* may begin execution before the Unlink operation has completed. To ensure deterministic behavior, the IT_UNLINK_FENCE_FLAG may be specified on subsequent Work Requests.

**EXTENDED DESCRIPTION**

The InfiniBand v1.2 Transport supports LMRs that may be linked and unlinked but also supports a separate class of LMRs that cannot be linked or unlinked. To guarantee that an LMR may be linked and unlinked, the Consumer should use the *it_lmr_create_unlinked* routine for LMR creation.

The InfiniBand v1.1 Transport does not support LMRs that may be linked and unlinked and reports *lmr_link_support* at IT_FALSE.

**RETURN VALUE**

A successful call returns IT_SUCCESS. Otherwise, an immediate error is returned and the previous linking of the LMR remains valid. It is possible for *it_lmr_unlink* to return success but for the Completion Event to indicate failure.

Posting to an Endpoint that is not in the IT_EP_STATE_CONNECTED or IT_EP_STATE_NONOPERATIONAL state will return the IT_ERR_INVALID_EP_STATE immediate error.

The possible immediate errors for *it_lmr_unlink* are listed below:

| | |
|---|---|
| IT_ERR_INVALID_DTO_FLAGS | The Data Transfer Operation flags (*dto_flags*) value was invalid. |
| IT_ERR_INVALID_EP | The Endpoint Handle (*ep_handle*) was invalid. |
| IT_ERR_INVALID_EP_STATE | The Endpoint was not in the proper state for the attempted operation. |
| IT_ERR_INVALID_EP_TYPE | The attempted operation was invalid for the Service Type of the Endpoint. |
| IT_ERR_INVALID_LMR | The Region Handle (*lmr_handle*) was invalid. |

| | |
|---|---|
| IT_ERR_OP_NOT_SUPPORTED | The IA has no LMR Link support (*lmr_link_support* in *it_ia_info_t* is IT_FALSE). |
| IT_ERR_TOO_MANY_POSTS | The operation failed due to an overflow of a Work Queue. |
| 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. |

**ASYNCHRONOUS ERRORS**

For Work Requests posted to an Endpoint in the IT_EP_STATE_CONNECTED state, a completion status (see *it_dto_status_t*) other than IT_DTO_SUCCESS will break the Connection by moving the Endpoint to the IT_EP_STATE_NONOPERATIONAL state and deliver an IT_CM_MSG_CONN_BROKEN_EVENT Event to the Connect EVD of *ep_handle*. Once the Connection is broken, all outstanding and in-progress operations on the Connection will complete with an error status.

Any posting to an Endpoint that is in the IT_EP_STATE_NONOPERATIONAL state will be flushed with completion status set to IT_DTO_ERR_FLUSHED.

An IT_DTO_ERR_LOCAL_MM_OPERATION completion status results in case of an invalid LMR handle, unless the Implementation handles this error as an immediate error.

An IT_DTO_ERR_LOCAL_MM_OPERATION completion status results if the Protection Zones of the LMR and the Endpoint do not match.

An IT_DTO_ERR_LOCAL_MM_OPERATION completion status results if *lmr_handle* is the Direct LMR Handle, or if the LMR is in the Shared state, or if the LMR has any RMR bound to it, or, for InfiniBand only, if the LMR could not be unlinked due to the way it was created.

**APPLICATION USAGE**

*it_lmr_unlink* is used to move an LMR from the linked state to the unlinked state, thereby revoking local Consumer access and remote Consumer access if that was previously granted.

The *it_lmr_unlink* routine does not require Privilege to invoke but the *it_lmr_link* routine does. Invoking *it_lmr_unlink* as a non-Privileged Consumer has the consequence that the unlinked LMR may not be re-linked (as the Consumer has no such Privilege) and may only be freed.

When an EVD holding the DTO Event Stream associated with an Endpoint overflows, on some transports the behavior of the Endpoint and/or the EVD is undefined, making recovery from such an overflow either impossible or at the very least IA-dependent. Consumers should therefore take care to avoid this situation. Each Work Request that a Consumer posts to a Work Queue can result in a Completion Event being enqueued in an EVD, due to the possibility of an error occurring during Work Request processing. To avoid any possibility of overflow, the Consumer must therefore ensure that the total number of Outstanding Operations on all Work Queues that can enqueue Completion Events into a given EVD never exceeds the minimum queue size (as reported by *it_evd_query*) of that EVD.

**SEE ALSO**

*it_lmr_create*(), *it_lmr_create_unlinked*(), *it_lmr_link*(), *it_dto_flags_t*

# it_make_rdma_addr_absolute()

## NAME

it_make_rdma_addr_absolute – make a platform-independent RDMA address for Absolute Addressing

## SYNOPSIS

```
#include <it_api.h>

typedef uint64_t it_rdma_addr_t;

it_rdma_addr_t it_make_rdma_addr_absolute(
    void  *addr
);
```

## DESCRIPTION

addr Local address.

The *it_make_rdma_addr_absolute* routine takes a local address *addr* that may be the target of a remote operation, and returns a 64-bit platform-independent representation of that address in network byte order and suitable for Absolute Addressings, called an RDMA address (see *it_addr_mode_t* for a description of Absolute Addressing). A network peer may use this RDMA address in RDMA Read and Write operations. *it_make_rdma_addr_absolute* performs no validity checking on *addr*, so *addr* is not required to lie within a currently registered LMR when *it_make_rdma_addr_absolute* is called.

## RETURN VALUE

This function always succeeds and returns a 64-bit RDMA address.

## APPLICATION USAGE

The returned RDMA address must be communicated to a network peer in order to be used in RDMA operations. The Consumer is responsible for performing this communication.

Because the RDMA address is in network byte order, a Consumer wishing to perform address arithmetic must first convert it to host byte order, which may be done using the *it_ntoh64* function. Derived addresses must be converted back to network byte order using *it_hton64* before being used in RDMA operations.

## SEE ALSO

*it_post_rdma_write*(), *it_ntoh64*, *it_hton64*, *it_make_rdma_addr_relative*, *it_addr_mode_t*

# it_make_rdma_addr_relative()

## NAME

it_make_rdma_addr_relative – make a platform-independent RDMA address for Relative Addressing

## SYNOPSIS

```
#include <it_api.h>

it_rdma_addr_t it_make_rdma_addr_relative(
    it_length_t  offset
);
```

## DESCRIPTION

offset — Local offset.

The *it_make_rdma_addr_relative* routine takes a local offset *offset* that may be the target of a remote operation, and returns a 64-bit platform-independent representation of that address in network byte order and suitable for Relative Addressing, called an RDMA address (see *it_addr_mode_t* for a description of Relative Addressing). A network peer may use this RDMA address in RDMA Read and Write operations. *it_make_rdma_addr_relative* performs no validity checking on *offset*, so *offset* is not required to lie within a currently registered LMR when *it_make_rdma_addr_relative* is called.

## RETURN VALUE

This function always succeeds and returns a 64-bit RDMA address.

## APPLICATION USAGE

The returned RDMA address must be communicated to a network peer in order to be used in RDMA operations. The Consumer is responsible for performing this communication.

Because the RDMA address is in network byte order, a Consumer wishing to perform address arithmetic must first convert it to host byte order, which may be done using the *it_ntoh64* function. Derived addresses must be converted back to network byte order using *it_hton64* before being used in RDMA operations.

## SEE ALSO

*it_post_rdma_write*(), *it_ntoh64*, *it_hton64*, *it_make_rdma_addr_absolute*, *it_addr_mode_t*

# it_mem_map()

**NAME**

it_mem_map – map virtually addressed buffer to I/O address space

**SYNOPSIS**

```
#include <it_api.h>

it_status_t it_mem_map(
    IN   it_ia_handle_t  ia_handle,
    IN   void           *vaddr,
    IN   uint64_t        aspace_id,
    IN   size_t          len,
    OUT  it_io_addr_t   *io_addr,
    OUT  size_t         *mapped_len
);
```

**APPLICABILITY**

*it_mem_map* is usable only by Privileged Mode Consumers.

**DESCRIPTION**

| | |
|---|---|
| *ia_handle* | The Interface Adapter on which *io_addr* will be used. |
| *vaddr* | Local address. |
| *aspace_id* | Local address space identifier. An O/S-dependent and architecture-specific value that combines with *vaddr* to yield a virtual address valid in any O/S context. |
| *len* | Length of buffer addressed by *vaddr*. |
| *io_addr* | I/O address of buffer corresponding to the first *mapped_len* bytes of *vaddr*. The buffer returned in *io_addr* is contiguous for *mapped_len* bytes in the I/O address space of the IA. |
| *mapped_len* | Number of bytes of *vaddr* mapped into *io_addr*. |

The *it_mem_map* routine maps a local virtually addressed buffer *vaddr* to a buffer that is contiguous in the I/O address space usable by the Interface Adapter identified by *ia_handle*. If *it_mem_map* is unable to map all of *vaddr* in one contiguous I/O buffer it returns the number of bytes actually mapped in *mapped_len*.

**RETURN VALUE**

A successful call returns IT_SUCCESS. Otherwise, an immediate error is returned.

| | |
|---|---|
| IT_ERR_NO_PERMISSION | The Consumer did not have the proper permissions to perform the requested operation. |
| IT_ERR_FAULT | Part or all of the supplied address range was invalid. |
| IT_ERR_INVALID_IA | The Interface Adapter Handle (*ia_handle*) was invalid. |

| | |
|---|---|
| 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. |

**APPLICATION USAGE**

The Consumer may use *it_mem_map* to construct an IOBL for input to *it_lmr_link* and other API routines.

When *it_mem_map* returns *mapped_len* as less than *len*, the Consumer should store *io_addr* and *mapped_len* into an IOBL entry, increment *vaddr* by *mapped_len* bytes, subtract *mapped_len* bytes from *len*, and call *it_mem_map* again with the revised arguments and store the result into a subsequent IOBL entry. The Consumer should follow this practice iteratively until the entire buffer is mapped.

**SEE ALSO**

*it_mem_unmap*(), *it_lmr_link*(), *it_lmr_create*(), *it_io_addr_t*, *it_iobl_t*

# it_mem_unmap()

## NAME

it_mem_unmap – undo a mapping established by *it_mem_map*

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_mem_unmap(
    IN  it_ia_handle_t  ia_handle,
    IN  it_io_addr_t    io_addr,
    IN  size_t          mapped_len
);
```

## APPLICABILITY

*it_mem_unmap* is usable only by Privileged Mode Consumers.

## DESCRIPTION

| | |
|---|---|
| *ia_handle* | The Inteface Adapter on which *io_addr* will be used. |
| *io_addr* | I/O address of buffer returned earlier by *it_mem_map*. |
| *mapped_len* | Size of buffer starting at *io_addr* in bytes. |

The *it_mem_unmap* routine tears down a mapping established by *it_mem_map* on the Interface Adapter implicitly identified by *pz_handle*.

Calling *it_mem_unmap* on an address range still in use by the Consumer can result in undefined behavior up to and including unreported data corruption.

Passing *io_addr* or *mapped_len* values that do not correspond to values returned by *it_mem_map* will yield an immediate error.

## RETURN VALUE

A successful call returns IT_SUCCESS. Otherwise, an immediate error is returned.

| | |
|---|---|
| IT_ERR_NO_PERMISSION | The Consumer did not have the proper permissions to perform the requested operation. |
| IT_ERR_FAULT | Part or all of the supplied address range was invalid. |
| IT_ERR_INVALID_IA | The Interface Adapter Handle (*ia_handle*) was invalid. |
| 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. |

## SEE ALSO

*it_mem_map*(), *it_lmr_link*(), *it_lmr_create*(), *it_io_addr_t*, *it_iobl_t*

# it_post_atomic()

## NAME

it_post_atomic – post an Atomic DTO to an Endpoint

## SYNOPSIS

```
#include <it_api.h>

typedef enum {
    IT_COMPARE_AND_SWAP,
    IT_FETCH_AND_ADD
} it_atomic_op_t;

it_status_t it_post_atomic (
    IN        it_ep_handle_t     ep_handle,
    IN        it_atomic_op_t     op,
    IN        uint64_t           swap_or_add,
    IN        uint64_t           compare,
    IN const  it_lmr_triplet_t  *result,
    IN        it_dto_cookie_t    cookie,
    IN        it_dto_flags_t     dto_flags,
    IN        it_rmr_context_t   rmr_context
);
```

## APPLICABILITY

*it_post_atomic* is applicable only to Endpoints created for the RC service type and is supported only if the Interface Adapter attribute *atomic_support* (see *it_ia_info_t*) is IT_TRUE.

## DESCRIPTION

| | |
|---|---|
| *ep_handle* | Handle for the Endpoint – the local side of the Connection. |
| *op* | Atomic operation to be performed. Must be one of IT_COMPARE_AND_SWAP or IT_FETCH_AND_ADD. |
| *swap_or_add* | Swap operand when *op* is IT_COMPARE_AND_SWAP. Add operand when *op* is IT_FETCH_AND_ADD. |
| *compare* | Compare operand when *op* is IT_COMPARE_AND_SWAP. Not used when *op* is IT_FETCH_AND_ADD. |
| *result* | One *it_lmr_triplet_t* data structure that specifies the local buffer where the result of the operation will be deposited. The local buffer must be at least eight bytes in size. |
| *cookie* | Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the Atomic. |
| *dto_flags* | Flags for posted Atomic. |
| *rdma_addr* | The starting address of the remote buffer (target of Atomic operation). Remote buffer starting address must correspond to an I/O Address that is aligned to a modulo eight byte boundary at the remote and the remote |

| | | |
|---|---|---|
| | | buffer must be contiguous in the I/O Address space for eight bytes. Ensuring that the virtual address of the buffer is aligned to a modulo eight-byte boundary is sufficient on supported architectures. |
| | *rmr_context* | The RMR Context for the remote buffer (target of Atomic operation). |

*it_post_atomic* posts a request to the *ep_handle* Endpoint to initiate an Atomic operation on the remote buffer, *rdma_addr*, via the reliable Connection of the *ep_handle* Endpoint.

Two operations specified in *op* are supported, IT_FETCH_AND_ADD and IT_COMPARE_AND_SWAP.

When *op* is specified as IT_FETCH_AND_ADD, the contents of *rdma_addr* are fetched from the remote and stored into *result*. The value in *swap_or_add* is then added to the contents of *rdma_addr*. The fetch and add operations are performed atomically at the remote. Conversion to the endianness of the remote system is performed by the Implementation and addition is performed in the native endianness of the remote system using 2's complement arithmetic without a carry (i.e., addition may overflow/underflow – wrapping around within the operand). The value fetched into *result* is converted into the native endianness of the local system. See [IB-R1.2], Section 9.4.5. It is the Consumer's responsibility to interpret whether or not the operands and results are signed arithmetically.

When *op* is specified as IT_COMPARE_AND_SWAP, the contents of *rdma_addr* are compared with that in *compare*. If the values match, the swap is performed as follows: The contents of *rdma_addr* are fetched from the remote and stored into *result* and the value in *swap_or_add* is written into the contents of *rdma_addr*. The comparison and swap operations are performed atomically at the remote. The endiannesses of the local and remote systems are honored: the comparison is done in the native endianness of the remote. The value fetched into *result* is converted into the native endianness of the local system. See [IB-R1.2] Section 9.4.5.

All Atomic operation requests made to the same remote IA, referencing the same target buffer (*rdma_addr*) shall appear serialized with respect to each other.

Atomic operations made to an IA are not guaranteed to be serialized with respect to RDMA operation requests made to it or other IAs in the system, or with respect to operations performed by other system components such as processors. Because of this behavior, if Atomic operations on a particular area of memory are used to implement locks, all accesses to that memory must be done using Atomic operations. In particular, it is not safe to use an RDMA read or Send/Receive to see if a lock is held, and it is not safe to use an RDMA write or Send/Receive to clear a lock.

For a remote buffer with Absolute Addressing, the starting address *rdma_addr* must be an absolute address within the remote linear address space, and for a remote buffer with Relative Addressing, *rdma_addr* must be a byte offset relative to the first byte of the remote buffer. In either case, *rdma_addr* must represent a buffer aligned as described above. See *it_addr_mode_t* for details on addressing modes.

The remote buffer represented by *rmr_context* must have memory access privileges including both remote read and remote write access.

An LMR used in *result* (data sink) must have local write access.

The cookie (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work Request. This identifier is completely under Consumer control and opaque to the

Implementation. The cookie is returned to the Consumer in the Completion Event for the posted Work Request.

The *dto_flags* value is used as specified in *it_dto_flags_t*.

A successful call returns IT_SUCCESS, which means that the Atomic operation was successfully posted to the transport layer.

The completion of the posted Atomic is reported asynchronously to the Consumer according to the rules defined in *it_dto_flags_t*. An Atomic Completion Event is of type *it_dto_cmpl_event_t*. Any generated Atomic Completion Event manifests on the EVD associated with the Endpoint Send Queue. See *it_ep_rc_create*, *it_dto_status_t*, and *it_dto_events*. Once a successful Completion Event has been generated for the Atomic, the contents of the local buffer corresponds to the result of the Atomic operation. Prior to the Completion Event being generated, the content of the local buffer is Implementation-dependent.

A Consumer shall not modify the local buffer specified by *result* until the DTO is completed. When a Consumer does not adhere to this rule, the behavior of the Implementation and the underlying transport is not defined. A Consumer does get back the ownership of the *it_lmr_triplet_t* represented by *result* (but not the local buffer identified by LMR triplet) when *it_post_atomic* returns and is free to use LMR triplet for other calls, to modify it, or to destroy it.

If the reported status of the completion DTO Event corresponding to the posted Atomic is not IT_DTO_SUCCESS, the content of the local buffer is not defined.

The Implementation ensures that the Atomic in no way corresponds to any Send or Recv Data Transfer Operations over the same Connection.

The Implementation ensures that subsequent Atomic DTOs posted to the same Endpoint start and complete in post order.

In general, Work Requests following an RDMA Read (or Atomic operation) may start execution while the RDMA Read (or Atomic operation) is in progress (but may not complete before the RDMA Read (or Atomic operation) completes). To ensure deterministically that an Atomic operation following an RDMA Read DTO (or following an Atomic operation) starts after the RDMA Read (or Atomic operation) completes, specify the IT_BARRIER_FENCE_FLAG on the following Atomic operation. This technique can be used to ensure that the Atomic DTOs place their data payloads into their local sink buffer in post order. It can also be used to prevent exceeding the IRD of the remote Endpoint.

Any Work Request posted to an Endpoint's Send Queue (thus also an Atomic) after a call to *it_rmr_link* or *it_rmr_unlink* will not begin execution until the Link or Unlink operation has completed.

The Implementation ensures that all data for a given Atomic operation is transferred from the remote buffer into the local buffer before an Atomic completion is generated with the status of IT_DTO_SUCCESS.

**RETURN VALUE**

A successful call returns IT_SUCCESS.

Posting to an Endpoint that is not in the IT_EP_STATE_CONNECTED or IT_EP_STATE_ NONOPERATIONAL state will return the IT_ERR_INVALID_EP_STATE immediate error.

The possible immediate errors for *it_post_atomic* are listed below:

| | |
|---|---|
| IT_ERR_INVALID_DTO_FLAGS | The Data Transfer Operation flags (*dto_flags*) value was invalid. |
| IT_ERR_INVALID_ATOMIC_OP | The Atomic operation (*op*) value was invalid. |
| IT_ERR_INVALID_EP | The Endpoint Handle (*ep_handle*) was invalid. |
| IT_ERR_INVALID_EP_STATE | The Endpoint was not in the proper state for the attempted operation. |
| IT_ERR_INVALID_EP_TYPE | The attempted operation was invalid for the Service Type of the Endpoint. |
| IT_ERR_TOO_MANY_POSTS | The operation failed due to an overflow of a Work Queue. |
| 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 |

**ASYNCHRONOUS ERRORS**

For Work Requests posted to an Endpoint in the IT_EP_STATE_CONNECTED state, a completion status (see *it_dto_status_t*) other than IT_DTO_SUCCESS will break the Connection by moving the Endpoint to the IT_EP_STATE_NONOPERATIONAL state and deliver an IT_CM_MSG_CONN_BROKEN_EVENT Event to the Connect EVD of *ep_handle*. Once the Connection is broken, all outstanding and in-progress operations on the Connection will complete with an error status.

Any posting to an Endpoint that is in the IT_EP_STATE_NONOPERATIONAL state will be flushed with completion status set to IT_DTO_ERR_FLUSHED.

For locally or remotely detected errors that can be reported as Completion Errors, see also *it_dto_status_t*.

An Atomic operation may result in a remotely detected access violation at the remote buffer. Such a violation may be due to misalignment or other reasons.

The remote Implementation verifies the accessibility of the remote buffer represented by *rmr_context* based on the buffer's access rights, the buffer's association with the remote Endpoint that processes the operation, and the incoming RDMA operations allowed on the remote Endpoint. It declares an access violation if either the remote buffer has insufficient access rights, if *rmr_context* represents an LMR or Wide RMR whose Protection Zone does not match the Protection Zone of the remote Endpoint, if *rmr_context* represents a Narrow RMR that is associated with a different remote Endpoint, or if the targeted remote Endpoint does not have incoming Atomic operations enabled. An access violation at the data source is also declared if the Atomic operation exceeds the bounds of the remote buffer.

An access violation at the data source will surface remotely as an IT_ASYNC_AFF_EP_L_ACCESS_VIOLATION Affiliated Asynchronous Error on the IB transport. Atomics are not supported on the iWARP Transport.

An access violation at the remote buffer will surface locally as follows:

For the IB transport, an IT_DTO_ERR_REMOTE_ACCESS Completion Error will occur. Atomics are not supported on the iWARP Transport.

An Atomic operation may result in a locally detected access violation at the local data sink.

The local Implementation verifies the accessibility of the local buffer represented by result based on the segment access rights and association with the local Endpoint, and the incoming operations allowed on the local Endpoint. It declares an access violation if an LMR in a local segment has insufficient access rights or a Protection Zone that does not match the Protection Zone of the local Endpoint. An access violation is also declared if the Atomic operation exceeds the bounds of an LMR in a local segment.

An access violation at the local data sink will surface locally as follows:

For the IB transport, an IT_DTO_ERR_LOCAL_PROTECTION Completion Error will occur. Atomics are not supported on the iWARP Transport.

Despite using the RC service, an Atomic DTO may fail to successfully deliver the contents of the section of the remote buffer into the local buffer. This may result in a broken Connection or lead to data corruption in the local buffer, which is detected locally with high probability. In case of local data corruption, an IT_DTO_ERR_TRANSPORT Completion Error will occur.

**APPLICATION USAGE**

This function is used after a Connection has been established to perform atomic Fetch and Add or Compare and Swap operations on a Consumer-specified remote buffer and yield a result into a Consumer-specified local buffer.

The Consumer should use unique identifiers for *cookie* if they desire to identify each DTO. If the Consumer does not require a unique DTO identifier, the value of zero or NULL (IT_NO_ADDR) can be used.

It is the Consumer's responsibility to ensure *rdma_addr* corresponds to a remote buffer that is 64-bit physically aligned and is physically contiguous for 64 bits. For Absolute Addressing, the Consumer must ensure that *rdma_addr* is 8-byte aligned. For Relative Addressing, the Consumer must ensure that "address + *rdma_addr*" is 8-byte aligned where "address" is the absolute address of the first byte of the remote memory region.

For Relative Addressing, the alignment of the remote buffer cannot be determined at the Atomic-initiating side by examination of the lower bits of *rdma_addr* alone (since *rdma_addr* is just a byte offset relative to the starting address of the remote buffer, which may have arbitrary alignment). It is the Consumer's responsibility to ensure that the Atomic-initiating side has sufficient information to generate an *rdma_addr* byte offset that results in an 8-byte aligned absolute address in the remote buffer's address space.

If the remote buffer corresponds to memory registered with the IT_LMR_FLAG_NONCOHERENT (i.e., the buffer is in non-coherent memory), then it is the Consumer's responsibility to ensure the synchronization of the view of the memory using the *it_lmr_flush_to_mem* and *it_lmr_refresh_from_mem* APIs appropriately.

For best Atomic operation performance, the Consumer should align the buffer segment of *result* to the *dto_alignment_hint* in the IA attributes obtained via *it_ia_query*.

When an EVD holding the DTO Event Stream associated with an Endpoint overflows, on some transports the behavior of the Endpoint and/or the EVD is undefined, making recovery from such an overflow either impossible or at the very least IA-dependent. Consumers should therefore take care to avoid this situation. Each Work Request that a Consumer posts to a Work Queue can result in a Completion Event being enqueued in an EVD, due to the possibility of an error occurring during Work Request processing. To avoid any possibility of overflow, the Consumer must therefore ensure that the total number of Outstanding Operations on all Work Queues that can enqueue Completion Events into a given EVD never exceeds the minimum queue size (as reported by *it_evd_query*) of that EVD.

**SEE ALSO**

*it_post_send*(), *it_post_send_and_unlink*(), *it_post_recv*(), *it_post_rdma_read*(), *it_post_rdma_read_to_rmr*(), *it_post_rdma_write*(), *it_ep_rc_create*(), *it_ia_query*(), *it_addr_mode_t*, *it_dto_cookie_t*, *it_dto_events*, *it_dto_flags_t*, *it_dto_status_t*, *it_ia_info_t*, *it_lmr_triplet_t*

# it_post_rdma_read()

## NAME

it_post_rdma_read – post an RDMA Read DTO to an Endpoint

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_post_rdma_read (
    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         it_rdma_addr_t     rdma_addr,
    IN         it_rmr_context_t   rmr_context
);
```

## APPLICABILITY

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

## DESCRIPTION

| | |
|---|---|
| *ep_handle* | Handle for the Endpoint – the local side of the Connection. |
| *local_segments* | Vector of *it_lmr_triplet_t* data structures that specifies the local buffer where data should be deposited. Can be NULL (IT_NO_ADDR) for a zero-sized message. |
| *num_segments* | Number of *it_lmr_triplet_t* data structures in *local_segments*. Can be zero for a zero-sized message. |
| *cookie* | Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the RDMA Read. |
| *dto_flags* | Flags for posted RDMA Read. |
| *rdma_addr* | The starting address of the section of the remote buffer from which to read. |
| *rmr_context* | The RMR Context corresponding to the remote buffer from which reads will occur. |

*it_post_rdma_read* posts a request to the *ep_handle* Endpoint to transfer data from the section of the remote buffer (data source) specified by *rmr_context* and starting address *rdma_addr* into the local buffer (data sink) specified by *local_segments* and *num_segments*, via the reliable Connection of the *ep_handle* Endpoint. If *num_segments* is non-zero, then the size of the data transferred is given by the sum of the segment lengths specified by *local_segments.* A zero-sized message may be transferred. Like all other RDMA Read operations, the maximum number of *local_segments* is limited by the Endpoint attribute *max_rdma_read_segments*.

For a remote buffer with Absolute Addressing, the starting address *rdma_addr* must be an absolute address within the remote linear address space, and for a remote buffer with Relative

Addressing, *rdma_addr* must be a byte offset relative to the first byte of the remote buffer. See *it_addr_mode_t* for details on addressing modes.

The Implementation ensures that an LMR Triplet supports byte alignment for Data Transfer Operations. If the buffer segments described by *local_segments* overlap, the resulting content of the local buffer is undefined.

The remote buffer represented by *rmr_context* (data source) must have memory access privileges including remote read access.

An LMR used in *local_segments* (data sink) must have memory access privileges including remote write access if the Interface Adapter attribute *rdma_read_requires_remote_write* equals IT_TRUE, or including local write access otherwise. See also Extended Description and Application Usage. The Direct LMR Handle may not be used.

The *cookie* (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work Request. This identifier is completely under Consumer control and opaque to the Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted Work Request.

The *dto_flags* value is used as specified in *it_dto_flags_t*.

A successful call returns IT_SUCCESS, which means that the RDMA Read operation was successfully posted to the transport layer.

The completion of the posted RDMA Read is reported asynchronously to the Consumer according to the rules defined in *it_dto_flags_t*. An RDMA Read Completion Event is of type *it_dto_cmpl_event_t*. Any generated RDMA Read Completion Event manifests on the EVD associated with the Endpoint Send Queue. See *it_ep_rc_create*, *it_dto_status_t*, and *it_dto_events*. Once a successful Completion Event has been generated for the RDMA Read, the order of the bytes in the local buffer specified by *num_segments* and *local_segments* corresponds to the order defined by the section of the remote buffer unless there is local overlap. If there is local overlap, the content of the local buffer is undefined. Prior to the Completion Event being generated, the content of the local buffer is Implementation-dependent.

A Consumer shall not modify the local buffer specified by *num_segments* and *local_segments* until the DTO is completed. When a Consumer does not adhere to this rule, the behavior of the Implementation and the underlying transport is not defined. A Consumer does get back the ownership of the array specified by the *local_segments* and *num_segments* arguments (but not the local buffer identified by this array) when *it_post_rdma_read* returns and is free to use this array for other calls, to modify it, or to destroy it.

If the reported status of the completion DTO Event corresponding to the posted RDMA Read is not IT_DTO_SUCCESS, the content of the local buffer is not defined.

The Implementation ensures that the RDMA Read in no way corresponds to any Send or Recv Data Transfer Operations over the same Connection.

The Implementation ensures that subsequent RDMA Read DTOs posted to the same Endpoint start and complete in post order. However, the Implementation does not ensure that the RDMA Read DTOs place their data payloads into their local sink buffers in post order; if the local sink buffers overlap, their contents will be indeterminate.

In general, Work Requests following an RDMA Read may start execution while the RDMA Read is in progress (but may not complete before the RDMA Read completes). To ensure

deterministically that an RDMA Read DTO following an RDMA Read DTO starts after the first RDMA Read completes, specify the IT_BARRIER_FENCE_FLAG on the RDMA Read following the first RDMA Read. This technique can be used to ensure that the RDMA Read DTOs place their data payloads into their local sink buffers in post order. It can also be used to prevent exceeding the IRD of the remote Endpoint.

Any Work Request posted to an Endpoint's Send Queue (thus also an RDMA Read) after a call to *it_rmr_link* or *it_rmr_unlink* will not begin execution until the Link or Unlink operation has completed.

The Implementation ensures that all data for a given RDMA Read operation is transferred from the section of the remote buffer into the local buffer before an RDMA Read completion is generated with the status of IT_DTO_SUCCESS.

**EXTENDED DESCRIPTION**

For InfiniBand, the IA writing to a local buffer (data sink) in an RDMA Read DTO is considered a local write access, as indicated by the *it_ia_info_t* attribute *rdma_read_requires_remote_write* being IT_FALSE; consequently, the local buffer need only have the local write access privilege set and the Endpoint *ep_handle* need not have the *rdma_write_enable* attribute set to IT_TRUE.

For iWARP, however, the IA writing to a local buffer in an RDMA Read DTO is considered a remote write access (by an incoming RDMA Read Response message), as indicated by the *it_ia_info_t* attribute *rdma_read_requires_remote_write* being IT_TRUE; hence, the local buffer must have the remote write access privilege set and the Endpoint *ep_handle* must have the *rdma_write_enable* attribute set to IT_TRUE.

**RETURN VALUE**

A successful call returns IT_SUCCESS.

Posting to an Endpoint that is not in the IT_EP_STATE_CONNECTED or IT_EP_STATE_NONOPERATIONAL state will return the IT_ERR_INVALID_EP_STATE immediate error.

The possible immediate errors for *it_post_rdma_read* are listed below:

| | |
|---|---|
| IT_ERR_INVALID_DTO_FLAGS | The Data Transfer Operation flags (*dto_flags*) value was invalid. |
| IT_ERR_INVALID_EP | The Endpoint Handle (*ep_handle*) was invalid. |
| IT_ERR_INVALID_EP_STATE | The Endpoint was not in the proper state for the attempted operation. |
| IT_ERR_INVALID_EP_TYPE | The attempted operation was invalid for the Service Type of the Endpoint. |
| IT_ERR_INVALID_NUM_SEGMENTS | The requested number of segments (*num_segments*) was larger than the Endpoint supports. |
| IT_ERR_TOO_MANY_POSTS | The operation failed due to an overflow of a Work Queue. |
| 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

**ASYNCHRONOUS ERRORS**

For Work Requests posted to an Endpoint in the IT_EP_STATE_CONNECTED state, a completion status (see *it_dto_status_t*) other than IT_DTO_SUCCESS will break the Connection by moving the Endpoint to the IT_EP_STATE_NONOPERATIONAL state and deliver an IT_CM_MSG_CONN_BROKEN_EVENT Event to the Connect EVD of *ep_handle*.

An Affiliated Asynchronous Error that is associated with an Endpoint will move that Endpoint to the IT_EP_STATE_NONOPERATIONAL state; if this state transition was from the IT_EP_STATE_CONNECTED state, an IT_CM_MSG_CONN_BROKEN_EVENT Event will be delivered to the Connect EVD of *ep_handle*.

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

Any posting to an Endpoint that is in the IT_EP_STATE_NONOPERATIONAL state will be flushed with completion status set to IT_DTO_ERR_FLUSHED.

For locally or remotely detected errors that can be reported as Completion Errors, see also *it_dto_status_t*.

The handling of remotely detected errors and some locally detected errors is transport and Implementation-dependent. For the iWARP Transport, an IA may provide advanced RDMA Read error handling support; i.e., RDMAP/DDP support for mapping an error due to an inbound RDMA Read Response to a Completion Error, support for on-the-fly conversion of an RDMAP Terminate message into a Completion Error and, optionally, early detection of a local access violation due to an RDMA Read Request.

An RDMA Read operation may result in a remotely detected access violation (IRRQ access violation for iWARP) at the data source.

The remote Implementation verifies the accessibility of the remote buffer represented by *rmr_context* based on the buffer's access rights, the buffer's association with the remote Endpoint that processes the operation, and the incoming RDMA operations allowed on the remote Endpoint. In case of a non-zero sized data transfer, it declares an access violation if *rmr_context* does not represent a valid and linked remote buffer, if the remote buffer has insufficient access rights, if *rmr_context* represents an LMR or Wide RMR whose Protection Zone does not match the Protection Zone of the remote Endpoint, if *rmr_context* represents a Narrow RMR that is associated with a different remote Endpoint, if the targeted remote Endpoint does not have incoming RDMA Read operations enabled, or if the RDMA Read operation exceeds the bounds of the remote buffer. In case of a zero-sized data transfer, no access violation will result regardless of the values of *rmr_context* and *rdma_addr*.

An access violation at the data source will surface remotely as an IT_ASYNC_AFF_EP_L_ACCESS_VIOLATION Affiliated Asynchronous Error on the IB transport, and as an IT_ASYNC_AFF_EP_L_IRRQ_ACCESS_VIOLATION Affiliated Asynchronous Error on the iWARP Transport.

An access violation at the data source will surface locally as follows:

For the IB transport, an IT_DTO_ERR_REMOTE_ACCESS Completion Error will occur.

For the iWARP Transport and an IA with advanced RDMA Read error handling support, an IT_DTO_ERR_REMOTE_ACCESS Completion Error will occur. For the iWARP Transport and an IA without such support, an IT_ASYNC_AFF_EP_R_ACCESS_VIOLATION Affiliated Asynchronous Error will surface.

An RDMA Read operation may result in a locally detected access violation at the data sink.

The local Implementation verifies the accessibility of the local buffer represented by *local_segments* and *num_segments* based on each segment's access rights and association with the local Endpoint, and the incoming RDMA operations allowed on the local Endpoint. In case of a non-zero sized data transfer, it declares an access violation if a local segment does not refer to a valid and linked LMR, if an LMR in a local segment has insufficient access rights or a Protection Zone that does not match the Protection Zone of the local Endpoint, if the *rdma_read_requires_remote_write* IA attribute equals IT_TRUE and the local Endpoint has incoming RDMA Write operations disabled, or if the RDMA Read operation exceeds the bounds of an LMR in a local segment. In case of a zero-sized data transfer, no access violation will result regardless of *num_segments* and any (zero-sized) segments in the *local_segments* vector.

An access violation at the data sink will surface locally as follows:

For the IB transport, an IT_DTO_ERR_LOCAL_PROTECTION Completion Error will occur.

For the iWARP Transport and an IA with advanced RDMA Read error handling support, an IT_DTO_ERR_LOCAL_PROTECTION Completion Error will occur. For the iWARP Transport and an IA without such support, an IT_ASYNC_AFF_EP_L_ACCESS_VIOLATION Affiliated Asynchronous Error will surface.

Despite using the RC service, an RDMA Read DTO may fail to successfully deliver the contents of the section of the remote buffer into the local buffer. A failure such as an unfinished data delivery or a CRC error that cannot be corrected by the transport will manifest locally as follows:

For the IB transport, an IT_DTO_ERR_TRANSPORT Completion Error will occur.

For the iWARP Transport and an IA with advanced RDMA Read error handling support, an IT_DTO_ERR_TRANSPORT Completion Error will occur. For the iWARP Transport and an IA without such support, an IT_ASYNC_AFF_EP_L_LLP_ERROR Affiliated Asynchronous Error will surface.

If a type of remotely detected error can manifest locally in different ways – i.e., as an Affiliated Asynchronous Error or as a Completion Error – transport-independent programming requires Consumers to be prepared to deal with either.

If the Direct LMR Handle is used to directly specify a local buffer in an RDMA Read Work Request, then a Completion Error with DTO status IT_DTO_ERR_LOCAL_PROTECTION shall be returned.

**APPLICATION USAGE**

This function is used after a Connection has been established to transfer data from a Consumer-specified section of a remote buffer to a Consumer-specified local buffer.

For writing transport-independent code, the Consumer should specify the remote and local write access privileges of an RDMA Read DTO's data sink and the *rdma_write_enable* attribute of the local Endpoint according to the IA attribute *rdma_read_requires_remote_write* obtained via

*it_ia_query*. The Consumer may also enable both remote write and local write access for the data sink, regardless of the *rdma_read_requires_remote_write* attribute.

The Consumer should use unique identifiers for *cookie* if they desire to identify each DTO. If the Consumer does not require a unique DTO identifier, the value of zero or NULL (IT_NO_ADDR) can be used.

For best RDMA Read operation performance, the Consumer should align each buffer segment of *local_segments* to the *dto_alignment_hint* in the IA attributes obtained via *it_ia_query*.

When an EVD holding the DTO Event Stream associated with an Endpoint overflows, on some transports the behavior of the Endpoint and/or the EVD is undefined, making recovery from such an overflow either impossible or at the very least IA-dependent. Consumers should therefore take care to avoid this situation. Each Work Request that a Consumer posts to a Work Queue can result in a Completion Event being enqueued in an EVD, due to the possibility of an error occurring during Work Request processing. To avoid any possibility of overflow, the Consumer must therefore ensure that the total number of Outstanding Operations on all Work Queues that can enqueue Completion Events into a given EVD never exceeds the minimum queue size (as reported by *it_evd_query*) of that EVD.

**SEE ALSO**

*it_post_atomic*(), *it_post_send*(), *it_post_send_and_unlink*(), *it_post_recv*(), *it_post_rdma_read_to_rmr*(), *it_post_rdma_write*(), *it_ep_rc_create*(), *it_addr_mode_t*, *it_ia_query*(), *it_dto_cookie_t*, *it_dto_events*, *it_dto_flags_t*, *it_dto_status_t*, *it_ia_info_t*, *it_lmr_triplet_t*

# it_post_rdma_read_to_rmr()

## NAME

it_post_rdma_read_to_rmr – post an RDMA Read DTO to an Endpoint

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_post_rdma_read_to_rmr (
    IN         it_ep_handle_t     ep_handle,
    IN  const  it_rmr_triplet_t  *local_segments,
    IN         size_t             num_segments,
    IN         it_dto_cookie_t    cookie,
    IN         it_dto_flags_t     dto_flags,
    IN         it_rdma_addr_t     rdma_addr,
    IN         it_rmr_context_t   rmr_context
);
```

## APPLICABILITY

*it_post_rdma_read_to_rmr* is applicable only to Endpoints created for the RC service type and is supported only if the Interface Adapter attribute *rdma_read_local_extensions* (see *it_ia_info_t*) is IT_TRUE.

## DESCRIPTION

| | |
|---|---|
| *ep_handle* | Handle for the Endpoint – the local side of the Connection. |
| *local_segments* | Vector of *it_rmr_triplet_t* data structures that specifies the local buffer where data should be deposited. Can be NULL (IT_NO_ADDR) for a zero-sized message. |
| *num_segments* | Number of *it_rmr_triplet_t* data structures in *local_segments*. Can be zero for a zero-sized message. |
| *cookie* | Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the RDMA Read. |
| *dto_flags* | Flags for posted RDMA Read. |
| *rdma_addr* | The starting address of the section of the remote buffer from which to read. |
| *rmr_context* | The RMR Context corresponding to the remote buffer from which reads will occur. |

*it_post_rdma_read_to_rmr* posts a request to the *ep_handle* Endpoint to transfer data from the section of the remote buffer (data source) specified by *rmr_context* and *rdma_addr* into the local buffer (data sink) specified by *local_segments* and *num_segments*, via the reliable Connection of the *ep_handle* Endpoint. If *num_segments* is non-zero, then the size of the data transferred is given by the sum of the segment lengths specified by *local_segments.* A zero-sized message may be transferred. Like all other RDMA Read operations, the maximum number of *local_segments* is limited by the Endpoint attribute *max_rdma_read_segments*.

For a remote buffer with Absolute Addressing, the starting address *rdma_addr* must be an absolute address within the remote linear address space, and for a remote buffer with Relative Addressing, *rdma_addr* must be a byte offset relative to the first byte of the remote buffer. See *it_addr_mode_t* for details on addressing modes.

The Implementation ensures that an RMR Triplet supports byte alignment for Data Transfer Operations. If the buffer segments described by *local_segments* overlap, the resulting content of the local buffer is undefined.

The remote buffer represented by *rmr_context* (data source) must have memory access privileges including remote read access.

An RMR used in *local_segments* (data sink) must have memory access privileges including remote write access.

The *cookie* (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work Request. This identifier is completely under Consumer control and opaque to the Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted Work Request.

The *dto_flags* value is used as specified in *it_dto_flags_t*.

A successful call returns IT_SUCCESS, which means that the RDMA Read operation was successfully posted to the transport layer.

The completion of the posted RDMA Read is reported asynchronously to the Consumer according to the rules defined in *it_dto_flags_t*. An RDMA Read Completion Event is of type *it_dto_cmpl_event_t*. Any generated RDMA Read Completion Event manifests on the EVD associated with the Endpoint Send Queue. See *it_ep_rc_create*, *it_dto_status_t*, and *it_dto_events*. Once a successful Completion Event has been generated for the RDMA Read, the order of the bytes in the local buffer specified by *num_segments* and *local_segments* corresponds to the order defined by the section of the remote buffer unless there is local overlap. If there is local overlap, the content of the local buffer is undefined. Prior to the Completion Event being generated, the content of the local buffer is Implementation-dependent.

A Consumer shall not modify the local buffer specified by *num_segments* and *local_segments* until the DTO is completed. When a Consumer does not adhere to this rule, the behavior of the Implementation and the underlying transport is not defined. A Consumer does get back the ownership of the array specified by the *local_segments* and *num_segments* arguments (but not the local buffer identified by this array) when *it_post_rdma_read_to_rmr* returns and is free to use this array for other calls, to modify it, or to destroy it.

If the reported status of the completion DTO Event corresponding to the posted RDMA Read is not IT_DTO_SUCCESS, the content of the local buffer is not defined.

The Implementation ensures that the RDMA Read in no way corresponds to any Send or Recv Data Transfer Operations over the same Connection.

The Implementation ensures that subsequent RDMA Read DTOs posted to the same Endpoint start and complete in post order. However, the Implementation does not ensure that the RDMA Read DTOs place their data payloads into their local sink buffers in post order; if the local sink buffers overlap, their contents will be indeterminate.

In general, Work Requests following an RDMA Read may start execution while the RDMA Read is in progress (but may not complete before the RDMA Read completes). To ensure

deterministically that an RDMA Read DTO following an RDMA Read DTO starts after the first RDMA Read completes, specify the IT_BARRIER_FENCE_FLAG on the RDMA Read following the first RDMA Read. This technique can be used to ensure that the RDMA Read DTOs place their data payloads into their local sink buffers in post order. It can also be used to prevent exceeding the IRD of the remote Endpoint.

Any Work Request posted to an Endpoint's Send Queue (thus also an RDMA Read) after a call to *it_rmr_link* or *it_rmr_unlink* will not begin execution until the Link or Unlink operation has completed.

The Implementation ensures that all data for a given RDMA Read operation is transferred from the section of the remote buffer into the local buffer before an RDMA Read completion is generated with the status of IT_DTO_SUCCESS.

**EXTENDED DESCRIPTION**

This call is supported only by iWARP. It allows using local RMRs as data sinks.

For the iWARP Transport, the IA writing to a local buffer (data sink) in an RDMA Read DTO is considered a remote write access (by an incoming RDMA Read Response message), as indicated by the *it_ia_info_t* attribute *rdma_read_requires_remote_write* being IT_TRUE; hence, the local buffer must have the remote write access privilege set and the Endpoint *ep_handle* must have the *rdma_write_enable* attribute set to IT_TRUE.

**RETURN VALUE**

A successful call returns IT_SUCCESS.

Posting to an Endpoint that is not in the IT_EP_STATE_CONNECTED or IT_EP_STATE_NONOPERATIONAL state will return the IT_ERR_INVALID_EP_STATE immediate error.

The possible immediate errors for *it_post_rdma_read_to_rmr* are listed below:

| | |
|---|---|
| IT_ERR_INVALID_DTO_FLAGS | The Data Transfer Operation flags (*dto_flags*) value was invalid. |
| IT_ERR_INVALID_EP | The Endpoint Handle (*ep_handle*) was invalid. |
| IT_ERR_INVALID_EP_STATE | The Endpoint was not in the proper state for the attempted operation. |
| IT_ERR_INVALID_EP_TYPE | The attempted operation was invalid for the Service Type of the Endpoint. |
| IT_ERR_INVALID_NUM_SEGMENTS | The requested number of segments (*num_segments*) was larger than the Endpoint supports. |
| IT_ERR_OP_NOT_SUPPORTED | The operation failed as it is not supported on the IA. |
| IT_ERR_TOO_MANY_POSTS | The operation failed due to an overflow of a work queue. |
| 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 |

**ASYNCHRONOUS ERRORS**

For Work Requests posted to an Endpoint in the IT_EP_STATE_CONNECTED state, a completion status (see *it_dto_status_t*) other than IT_DTO_SUCCESS will break the Connection by moving the Endpoint to the IT_EP_STATE_NONOPERATIONAL state and deliver an IT_CM_MSG_CONN_BROKEN_EVENT Event to the Connect EVD of *ep_handle*.

An Affiliated Asynchronous Error that is associated with an Endpoint will move that Endpoint to the IT_EP_STATE_NONOPERATIONAL state; if this state transition was from the IT_EP_STATE_CONNECTED state, an IT_CM_MSG_CONN_BROKEN_EVENT Event will be delivered to the Connect EVD of *ep_handle*.

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

Any posting to an Endpoint that is in the IT_EP_STATE_NONOPERATIONAL state will be flushed with completion status set to IT_DTO_ERR_FLUSHED.

For locally or remotely detected errors that can be reported as Completion Errors, see also *it_dto_status_t*.

The handling of remotely detected errors and some locally detected errors is transport and Implementation-dependent. For the iWARP Transport, an IA may provide advanced RDMA Read error handling support; i.e., RDMAP/DDP support for mapping an error due to an inbound RDMA Read Response to a Completion Error, support for on-the-fly conversion of an RDMAP Terminate message into a Completion Error and, optionally, early detection of a local access violation due to an RDMA Read Request.

An RDMA Read operation may result in a remotely detected access violation (IRRQ access violation for iWARP) at the data source.

The remote Implementation verifies the accessibility of the remote buffer represented by *rmr_context* based on the buffer's access rights, the buffer's association with the remote Endpoint that processes the operation, and the incoming RDMA operations allowed on the remote Endpoint. In case of a non-zero sized data transfer, it declares an access violation if *rmr_context* does not represent a valid and linked remote buffer, if the remote buffer has insufficient access rights, if *rmr_context* represents an LMR or Wide RMR whose Protection Zone does not match the Protection Zone of the remote Endpoint, if *rmr_context* represents a Narrow RMR that is associated with a different remote Endpoint, if the targeted remote Endpoint does not have incoming RDMA Read operations enabled, or if the RDMA Read operation exceeds the bounds of the remote buffer. In case of a zero-sized data transfer, no access violation will result regardless of the values of *rmr_context* and *rdma_addr*.

An access violation at the data source will surface remotely as an IT_ASYNC_AFF_EP_L_ ACCESS_VIOLATION Affiliated Asynchronous Error on the IB transport, and as an IT_ASYNC_AFF_EP_L_IRRQ_ACCESS_VIOLATION Affiliated Asynchronous Error on the iWARP Transport.

An access violation at the data source will surface locally as follows:

For the IB transport, an IT_DTO_ERR_REMOTE_ACCESS Completion Error will occur.

For the iWARP Transport and an IA with advanced RDMA Read error handling support, an IT_DTO_ERR_REMOTE_ACCESS Completion Error will occur. For the iWARP Transport

and an IA without such support, an IT_ASYNC_AFF_EP_R_ACCESS_VIOLATION Affiliated Asynchronous Error will surface.

An RDMA Read operation may result in a locally detected access violation at the data sink.

The local Implementation verifies the accessibility of the local buffer represented by *local_segments* and *num_segments* based on each segment's access rights and association with the local Endpoint and the incoming RDMA operations allowed on the local Endpoint. In case of a non-zero sized data transfer, it declares an access violation if a local segment does not refer to a valid and linked RMR, if an RMR in a local segment has insufficient access rights or a Protection Zone that does not match the Protection Zone of the local Endpoint, if an RMR in a local segment is a Narrow RMR that is associated with a different local Endpoint, if the *rdma_read_requires_remote_write* IA attribute equals IT_TRUE and the local Endpoint has incoming RDMA Write operations disabled, or if the RDMA Read operation exceeds the bounds of an RMR in a local segment. In case of a zero-sized data transfer, no access violation will result regardless of *num_segments* and any (zero-sized) segments in the *local_segments* vector.

An access violation at the data sink will surface locally as follows:

For the IB transport, an IT_DTO_ERR_LOCAL_PROTECTION Completion Error will occur.

For the iWARP Transport and an IA with advanced RDMA Read error handling support, an IT_DTO_ERR_LOCAL_PROTECTION Completion Error will occur. For the iWARP Transport and an IA without such support, an IT_ASYNC_AFF_EP_L_ACCESS_VIOLATION Affiliated Asynchronous Error will surface.

Despite using the RC service, an RDMA Read DTO may fail to successfully deliver the contents of the section of the remote buffer into the local buffer. A failure such as an unfinished data delivery or a CRC error that cannot be corrected by the transport will manifest locally as follows:

For the IB transport, an IT_DTO_ERR_TRANSPORT Completion Error will occur.

For the iWARP Transport and an IA with advanced RDMA Read error handling support, an IT_DTO_ERR_TRANSPORT Completion Error will occur. For the iWARP Transport and an IA without such support, an IT_ASYNC_AFF_EP_L_LLP_ERROR Affiliated Asynchronous Error will surface.

If a type of remotely detected errors can manifest locally in different ways – i.e., as an Affiliated Asynchronous Error or as a Completion Error – transport-independent programming requires Consumers to be prepared to deal with either.

**APPLICATION USAGE**

This function is used after a Connection has been established to transfer data from a Consumer-specified section of a remote buffer to a Consumer-specified local buffer.

For writing transport-independent code, the Consumer should specify the remote write access privilege of an RDMA Read DTO's data sink and the *rdma_write_enable* attribute of the local Endpoint according to *rdma_read_requires_remote_write* in the IA attributes obtained via *it_ia_query*. The Consumer may also enable both remote write and local write access for the data sink, regardless of the *rdma_read_requires_remote_write* attribute.

The Consumer should use unique identifiers for *cookie* if they desire to identify each DTO. If the Consumer does not require a unique DTO identifier, the value of zero or NULL (IT_NO_ADDR) can be used.

For best RDMA Read operation performance, the Consumer should align each buffer segment of *local_segments* to the *dto_alignment_hint* in the IA attributes obtained via *it_ia_query*.

When an EVD holding the DTO Event Stream associated with an Endpoint overflows, on some transports the behavior of the Endpoint and/or the EVD is undefined, making recovery from such an overflow either impossible or at the very least IA-dependent. Consumers should therefore take care to avoid this situation. Each Work Request that a Consumer posts to a Work Queue can result in a Completion Event being enqueued in an EVD, due to the possibility of an error occurring during Work Request processing. To avoid any possibility of overflow, the Consumer must therefore ensure that the total number of Outstanding Operations on all Work Queues that can enqueue Completion Events into a given EVD never exceeds the minimum queue size (as reported by *it_evd_query*) of that EVD.

**SEE ALSO**

*it_post_atomic*(), *it_post_send*(), *it_post_send_and_unlink*(), *it_post_sendto*(), *it_post_recv*(), *it_post_recvfrom*(), *it_post_rdma_read*(), *it_post_rdma_write*(), *it_rmr_link*(), *it_rmr_unlink*(), *it_ep_rc_create*(), *it_ia_query*(), *it_addr_mode_t*, *it_dto_cookie_t*, *it_dto_events*, *it_dto_flags_t*, *it_dto_status_t*, *it_ia_info_t*, *it_rmr_triplet_t*

# it_post_rdma_write()

## NAME

it_post_rdma_write – post an RDMA Write DTO to an Endpoint

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_post_rdma_write (
    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         it_rdma_addr_t     rdma_addr,
    IN         it_rmr_context_t   rmr_context
);
```

## APPLICABILITY

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

## DESCRIPTION

| | |
|---|---|
| *ep_handle* | Handle for the Endpoint – the local side of the Connection. |
| *local_segments* | Vector of *it_lmr_triplet_t* data structures that specifies the local buffer that contains data to be transferred. Can be NULL (IT_NO_ADDR) for a zero-sized message. |
| *num_segments* | Number of *it_lmr_triplet_t* data structures in *local_segments*. Can be zero for a zero-sized message. |
| *cookie* | Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the RDMA Write. |
| *dto_flags* | Flags for posted RDMA Write. |
| *rdma_addr* | The starting address of the section of the remote buffer to which to write. |
| *rmr_context* | The RMR Context corresponding to the remote buffer to which writes will occur. |

*it_post_rdma_write* posts a request to the *ep_handle* Endpoint to transfer all the data from the local buffer (data source) specified by *local_segments* and *num_segments* into the section of the remote buffer (data sink) specified by *rmr_context* and *rdma_addr*, via the reliable Connection of the *ep_handle* Endpoint. If *num_segments* is non-zero, then the size of the data transferred is given by the sum of the segment lengths specified by *local_segments.* A zero-sized message may be transferred.

For a remote buffer with Absolute Addressing, the starting address *rdma_addr* must be an absolute address within the remote linear address space, and for a remote buffer with Relative Addressing, *rdma_addr* must be a byte offset relative to the first byte of the remote buffer. See *it_addr_mode_t* for details on addressing modes.

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.

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).

The remote buffer represented by *rmr_context* (data sink) must have memory access privileges including remote write access.

The *cookie* (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work Request. This identifier is completely under Consumer control and opaque to the Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted Work Request.

The *dto_flags* value is used as specified in *it_dto_flags_t*.

A successful call returns IT_SUCCESS, which means that the RDMA Write operation was successfully posted to the transport layer.

The completion of the posted RDMA Write is reported asynchronously to the Consumer according to the rules defined in *it_dto_flags_t*. An RDMA Write Completion Event is of type *it_dto_cmpl_event_t*. Any generated RDMA Write Completion Event manifests on the EVD associated with the Endpoint Send Queue. See *it_ep_rc_create*, *it_dto_status_t*, and *it_dto_events*. The Completion Event for the *it_post_rdma_write* call indicates to the Consumer that the local buffer is under Consumer control again and that the contents of the local buffer will be transported reliably, but does not guarantee that these contents have been successfully delivered into the memory of the remote Consumer; wherever necessary, this restriction is made explicit by designating a Completion as local or by stating that operations complete locally. See Section 5.4 of [DDP-IETF] for background information. However, once the contents of the local buffer reach the remote Consumer memory, the order of the bytes in the remote memory corresponds to the order defined by the *local_segments*.

A Consumer should not modify the local buffer specified by *num_segments* and *local_segments* until the DTO is locally completed. When a Consumer does not adhere to this rule, the behavior of the Implementation and the underlying transport is not defined. A Consumer does get back the ownership of the array specified by the *local_segments* and *num_segments* arguments (but not the local buffer identified by this array) when *it_post_rdma_write* returns and is free to use this array for other calls, to modify it, or to destroy it.

The Implementation ensures that the RDMA Write in no way corresponds to any Receive Data Transfer Operations over the same Connection.

The Implementation ensures that subsequent RDMA Write DTOs posted to the same Endpoint start and complete locally in post order. However, the Implementation does not ensure that the RDMA Write DTOs place their data payloads into their remote buffers in post order; if the targeted sections of the remote buffers overlap, their contents will be indeterminate.

The Implementation ensures that each RDMA Write DTO posted to an Endpoint prior to a Send DTO posted to the same Endpoint has its complete data payload delivered to the remote memory prior to the completion of the Receive DTO at the remote side matching that Send. However, the Implementation does not ensure that the RDMA Write DTO places its entire data payload into its remote buffer before the Receive DTO places the data payload of the incoming Send into its

receive buffer; if the targeted sections of the remote buffers overlap, their contents will be indeterminate.

In general, Work Requests following an RDMA Read may start execution while the RDMA Read is in progress (but may not complete before the RDMA Read completes). To ensure deterministically that an RDMA Write DTO following an RDMA Read DTO starts after the RDMA Read completes, specify the IT_BARRIER_FENCE_FLAG on the RDMA Write DTO.

Any Work Request posted to an Endpoint's Send Queue (thus also an RDMA Write) after a call to *it_rmr_link* or *it_rmr_unlink* will not begin execution until the Link or Unlink operation has completed.

**EXTENDED DESCRIPTION**

See *it_lmr_refresh_from_mem* for a discussion of ramifications of RDMA Write use on non-coherent systems.

On the InfiniBand Transport, the Implementation ensures that subsequent RDMA Write DTOs or an RDMA Write DTO followed by a Send DTO posted to the same Endpoint cause data payloads to be placed in remote memory in post order; relying on such placement ordering represents a transport-dependent programming practice.

**RETURN VALUE**

A successful call returns IT_SUCCESS.

Posting to an Endpoint that is not in the IT_EP_STATE_CONNECTED or IT_EP_STATE_NONOPERATIONAL state will return the IT_ERR_INVALID_EP_STATE immediate error.

The possible immediate errors for *it_post_rdma_write* are listed below:

| | |
|---|---|
| IT_ERR_INVALID_DTO_FLAGS | The Data Transfer Operation flags (*dto_flags*) value was invalid. |
| IT_ERR_INVALID_EP | The Endpoint Handle (*ep_handle*) was invalid. |
| IT_ERR_INVALID_EP_STATE | The Endpoint was not in the proper state for the attempted operation. |
| IT_ERR_INVALID_EP_TYPE | The attempted operation was invalid for the Service Type of the Endpoint. |
| IT_ERR_INVALID_NUM_SEGMENTS | The requested number of segments (*num_segments*) was larger than the Endpoint supports. |
| IT_ERR_TOO_MANY_POSTS | The operation failed due to an overflow of a Work Queue. |
| 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. |

**ASYNCHRONOUS ERRORS**

For Work Requests posted to an Endpoint in the IT_EP_STATE_CONNECTED state, a completion status (see *it_dto_status_t*) other than IT_DTO_SUCCESS will break the Connection by moving the Endpoint to the IT_EP_STATE_NONOPERATIONAL state and deliver an IT_CM_MSG_CONN_BROKEN_EVENT Event to the Connect EVD of *ep_handle*. Once the

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

An Affiliated Asynchronous Error that is associated with an Endpoint will move that Endpoint to the IT_EP_STATE_NONOPERATIONAL state; if this state transition was from the IT_EP_STATE_CONNECTED state, an IT_CM_MSG_CONN_BROKEN_EVENT Event will be delivered to the Connect EVD of *ep_handle*.

Any posting to an Endpoint that is in the IT_EP_STATE_NONOPERATIONAL state will be flushed with completion status set to IT_DTO_ERR_FLUSHED.

For locally or remotely detected errors that can be reported as Completion Errors, see also *it_dto_status_t*.

The handling of remotely detected errors is transport and Implementation-dependent; end-to-end completions are not supported for certain transports or DTOs.

An RDMA Write operation may result in a remotely detected access violation, also referred to as a protection error by some transports.

The remote Implementation verifies the accessibility of the remote buffer represented by *rmr_context* based on the buffer's access rights, the buffer's association with the remote Endpoint that processes the operation, and the RDMA operations allowed on the remote Endpoint. In case of a non-zero sized data transfer, it declares an access violation if *rmr_context* does not represent a valid and linked remote buffer, if the remote buffer has insufficient access rights, if *rmr_context* represents an LMR or Wide RMR whose Protection Zone does not match the Protection Zone of the remote Endpoint, if *rmr_context* represents a Narrow RMR that is associated with a different remote Endpoint, if the targeted remote Endpoint has incoming RDMA Write operations disabled, or if the RDMA Write operation exceeds the bounds of the remote buffer. In case of a zero-sized data transfer, no access violation will result regardless of the values of *rmr_context* and *rdma_addr*.

An access violation due to an RDMA Write operation will surface remotely as an IT_ASYNC_AFF_EP_L_ACCESS_VIOLATION Affiliated Asynchronous Error on both the IB and iWARP Transports.

An access violation due to an RDMA Write operation will surface locally as follows:

For the IB transport, an IT_DTO_ERR_REMOTE_ACCESS Completion Error will occur.

For the iWARP Transport, either an IT_ASYNC_AFF_EP_R_ACCESS_VIOLATION Affiliated Asynchronous Error will surface after the DTO has already completed with IT_DTO_SUCCESS or, if the Implementation is able to convert a Terminate message to a Completion Error, an IT_DTO_ERR_REMOTE_ACCESS Completion Error will occur.

Despite using the RC service, an RDMA Write operation may fail to successfully deliver the contents of the local buffer into the remote buffer. A failure such as an unfinished data delivery or a CRC error that cannot be corrected by the transport is detected as a transport error and manifests as follows:

For the IB transport, a transport error causes the RDMA Write DTO to complete with an IT_DTO_ERR_TRANSPORT Completion Error.

For the iWARP Transport, a transport error will surface remotely as an IT_ASYNC_AFF_EP_L_LLP_ERROR Affiliated Asynchronous Error, and locally as an

IT_ASYNC_AFF_EP_R_ERROR Affiliated Asynchronous Error after the RDMA Write DTO has already completed with IT_DTO_SUCCESS.

If a type of remotely detected error can manifest locally in different ways – i.e., as an Affiliated Asynchronous Error or as a Completion Error – transport-independent programming requires Consumers to be prepared to deal with either.

If the Direct LMR Handle is used to specify a local buffer in a Work Request and the Endpoint to which the Work Request is posted is not Privileged, then a Completion Error with the DTO status IT_DTO_ERR_LOCAL_PROTECTION shall be returned.

**APPLICATION USAGE**

This function is used after a Connection has been established to transfer data from a Consumer-specified local buffer to a section of a remote buffer on the other side of the Connection.

The Consumer should use unique identifiers for *cookie* if they desire to identify each DTO. If the Consumer does not require a unique DTO identifier, the value of zero or NULL (IT_NO_ADDR) can be used.

For best RDMA Write operation performance, the Consumer should align each buffer segment of *local_segments* to the *dto_alignment_hint* in the IA attributes obtained via *it_ia_query*.

There are a variety of ways to guarantee the delivery of a local buffer via RDMA Write into the memory of the remote Consumer. One way would be for the Consumer to send a message over the Connection after the RDMA Write had completed, and then wait for the remote Consumer to reply to that message. When the remote Consumer reaps the Receive Completion for the Send from the local Consumer, the payload of the RDMA Write must have been delivered into the remote memory.

When an EVD holding the DTO Event Stream associated with an Endpoint overflows, on some transports the behavior of the Endpoint and/or the EVD is undefined, making recovery from such an overflow either impossible or at the very least IA-dependent. Consumers should therefore take care to avoid this situation. Each Work Request that a Consumer posts to a Work Queue can result in a Completion Event being enqueued in an EVD, due to the possibility of an error occurring during Work Request processing. To avoid any possibility of overflow, the Consumer must therefore ensure that the total number of Outstanding Operations on all Work Queues that can enqueue Completion Events into a given EVD never exceeds the minimum queue size (as reported by *it_evd_query*) of that EVD.

**SEE ALSO**

*it_post_atomic*(), *it_post_send*(), *it_post_send_and_unlink*(), *it_post_sendto*(), *it_post_recv*(), *it_post_recvfrom*(), *it_post_rdma_read*(), *it_post_rdma_read_to_rmr*(), *it_rmr_link*(), *it_rmr_unlink*(), *it_ep_rc_create*(), *it_ia_query*(), *it_addr_mode_t*, *it_dto_cookie_t*, *it_dto_events*, *it_dto_status_t*, *it_dto_flags_t*, *it_ia_info_t*, *it_lmr_triplet_t*

# it_post_recv()

## NAME

it_post_recv – post a Receive DTO to an Endpoint or Shared Receive Queue

## SYNOPSIS

```
#include <it_api.h>

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
);
```

## APPLICABILITY

*it_post_recv* is applicable only to Endpoints created for the RC service type or to Shared Receive Queues.

## DESCRIPTION

| | |
|---|---|
| *handle* | Handle for the Endpoint or S-RQ. |
| *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. |
| *num_segments* | Number of *it_lmr_triplet_t* data structures in *local_segments*. Can be zero for a zero-sized message. |
| *cookie* | Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the Receive. |
| *dto_flags* | Flags for posted Receive. |

*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.

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).

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.

The *cookie* (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work Request. This identifier is completely under Consumer control and opaque to the Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted Work Request.

The *dto_flags* value is used as specified in *it_dto_flags_t*.

A successful call returns IT_SUCCESS, which means that the Receive operation was successfully posted to the transport layer.

If an Endpoint has an associated S-RQ and an incoming Send arrives for that Endpoint, then the Implementation behaves as if a Receive Work Request is dequeued from the S-RQ and enqueued onto the Endpoint's local RQ. Receive Work Requests are not necessarily dequeued from the S-RQ in the order in which Send messages arrive. Depending on Implementation, an out-of-order arrival of a Send message may cause more than one Receive WR to be dequeued from the S-RQ. An Interface Adapter may support the Endpoint Soft High Watermark and/or Endpoint Hard High Watermark mechanisms (see *it_ep_attributes_t* and *it_ia_info_t*), allowing Consumers to bound the effects of out-of-order arrivals.

The completion of the posted Receive is reported asynchronously to the Consumer according to the rules defined in *it_dto_flags_t*. A Receive Completion Event is of type *it_dto_cmpl_event_t*. Exactly one Receive Completion Event is always generated and manifests on the EVD associated with the Endpoint to which the corresponding incoming Send operation was directed (see *it_ep_rc_create*).

If the reported *dto_status* of the Receive Completion Event (see *it_dto_cmpl_event_t*) is IT_DTO_SUCCESS (see *it_dto_status_t*), then all data from an incoming Send message has been transferred into the Receive buffer and the size of the received data can be retrieved through the *transferred_length* member of the Event. Reception of a zero-sized message is supported and will consume a posted Receive. Once a successful Completion Event has been generated for the Receive, the order of the bytes in the local buffer specified by *local_segments* and *num_segments* corresponds to the order defined by the *local_segments* of the corresponding Send operation unless there is overlap among the segments of the local Receive buffer. If there is such an overlap, the content of the local Receive buffer after the Completion Event has been generated is undefined. Prior to the Completion Event being generated, the content of the local buffer is Implementation-dependent.

A Consumer should not modify the local buffer specified by *num_segments* and *local_segments* until the DTO is completed. When a Consumer does not adhere to this rule, the behavior of the Implementation and the underlying transport is not defined. A Consumer does get back the ownership of the array specified by the *local_segments* and *num_segments* arguments (but not the local buffer identified by this array) when *it_post_recv* returns and is free to use this array for other calls, to modify it, or to destroy it.

The Implementation ensures that each Receive corresponds to one and only one remote Send and in no way corresponds to any RDMA Read or RDMA Write Data Transfer Operations over the same Connection.

The Implementation ensures that an RDMA Write DTO from a remote Endpoint preceding a Send DTO from the same remote Endpoint has fully delivered its payload prior to the completion of the Receive DTO corresponding to the Send DTO. However, the Implementation does not ensure that the RDMA Write DTO places its entire data payload into its Destination

buffer before the Receive DTO places the data payload of the incoming Send into its Receive buffer; if these two buffers overlap, their contents will be indeterminate.

The Implementation ensures that Receive DTOs matching subsequent Send DTOs from the same remote Endpoint complete in the post order of these Send DTOs. However, the Implementation does not ensure that the matching Receive DTOs place the data payloads into their Receive buffers in the post order of the Send DTOs; if the Receive buffers overlap, their contents will be indeterminate. If a collection of Endpoints is using an S-RQ, the Receive Completion Events will be generated for a particular Endpoint in the same order that the corresponding Sends were done on the associated remote Endpoint.

For Receives that are posted to an Endpoint, the Implementation ensures that all Receives start and complete in the order posted. The same is not true for Receives that are posted to an S-RQ; such Receive Work Requests do not necessarily complete in the order posted, nor in the order in which Send messages arrive, nor in the order in which the Receive Work Requests are dequeued from the S-RQ.

There is no order relationship between completions of Receive DTOs and any other Work Requests posted to an Endpoint.

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_INVALID_DTO_FLAGS | The Data Transfer Operation flags (*dto_flags*) value was invalid. |
| IT_ERR_INVALID_EP | The input handle (*handle*) was invalid. |
| IT_ERR_INVALID_EP_TYPE | The attempted operation was invalid for the Service Type of the Endpoint. |
| IT_ERR_INVALID_NUM_SEGMENTS | The requested number of segments (*num_segments*) was larger than the Endpoint or S-RQ supports. |
| IT_ERR_INVALID_SRQ | The S-RQ handle was invalid. |
| IT_ERR_TOO_MANY_POSTS | The operation failed due to an overflow of a Work Queue. |
| 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. |

**ASYNCHRONOUS ERRORS**

A completion status (see *it_dto_status_t*) other than IT_DTO_SUCCESS will break the Connection associated with the Endpoint that received the corresponding Send message by moving that Endpoint to the IT_EP_STATE_NONOPERATIONAL state and delivering an IT_CM_MSG_CONN_BROKEN_EVENT Event to the Connect EVD of that Endpoint. Once that Connection is broken, all outstanding and in-progress operations on that Connection will complete with an error status. If the reported completion status is not IT_DTO_SUCCESS, the content of the local buffer is not defined.

An Affiliated Asynchronous Error that is associated with an Endpoint will move that Endpoint to the IT_EP_STATE_NONOPERATIONAL state; if this state transition was from the IT_EP_STATE_CONNECTED state, an IT_CM_MSG_CONN_BROKEN_EVENT Event will be delivered to the Connect EVD of *ep_handle*.

Any posting to an Endpoint that is in the IT_EP_STATE_NONOPERATIONAL state will be flushed with completion status set to IT_DTO_ERR_FLUSHED.

For errors that can be reported as Completion Errors, see also *it_dto_status_t*.

If no Receive buffers are posted in time, then an incoming Send will fail to deliver its data. In order to avoid this error, the Consumer should post Receive resources prior to the remote Consumer posting a Send. The lack of a Receive buffer is handled as follows:

For IB and iWARP, an IT_ASYNC_AFF_EP_L_RECV_ACCESS_VIOLATION Affiliated Asynchronous Error is generated.

For the IB transport, the Send DTO that was posted remotely completes with an IT_DTO_ERR_REMOTE_RESPONDER error status.

For the iWARP Transport, an IT_ASYNC_AFF_EP_R_RECV_ACCESS_VIOLATION Affiliated Asynchronous Error will surface remotely after the Send DTO posted remotely has already completed with IT_DTO_SUCCESS.

If the Receive buffer is not sufficient in size for the data payload of the incoming Send message, a length error will occur. In order to avoid length errors, the Consumer should post a buffer large enough for the incoming Send data. Length errors are handled as follows:

For both the IB and iWARP Transports, the Receive DTO completes with an IT_DTO_ERR_LOCAL_LENGTH error status.

For the IB transport, the Send DTO that was posted remotely completes with an IT_DTO_ERR_REMOTE_RESPONDER error status.

For the iWARP Transport, an IT_ASYNC_AFF_EP_R_RECV_LENGTH_ERROR Affiliated Asynchronous Error will surface remotely after the Send DTO posted remotely has already completed with IT_DTO_SUCCESS.

Despite using the RC service, an incoming Send message may fail to successfully deliver its data. A failure such as an unfinished data delivery or a CRC error that cannot be corrected by the transport is detected as a transport error and manifests as follows:

For the IB transport, a transport error causes the Send DTO posted remotely and the Receive DTO matching the Send DTO to complete with an IT_DTO_ERR_TRANSPORT Completion Error.

For the iWARP Transport, a transport error causes the Receive DTO to complete with an IT_DTO_ERR_TRANSPORT Completion Error and an IT_ASYNC_AFF_EP_R_ERROR Affiliated Asynchronous Error to surface remotely after the Send DTO posted remotely has already completed with IT_DTO_SUCCESS.

If the Direct LMR Handle is used to specify a local buffer in a Work Request and the Endpoint to which the Work Request is posted is not Privileged, then a Completion Error with the DTO status IT_DTO_ERR_LOCAL_PROTECTION shall be returned.

If the Direct LMR Handle is used to specify a local buffer in a Receive Work Request posted to an S-RQ and the Endpoint that finally processes the Receive Work Request is not Privileged, then a Completion Error with the DTO status IT_DTO_ERR_LOCAL_PROTECTION shall be returned.

**APPLICATION USAGE**

This function is used to post a Receive DTO to an Endpoint or Shared Receive Queue, requesting the transfer of data into a Consumer-specified local buffer from a buffer specified by the corresponding Send operation on the other side of the Connection.

A Receive DTO can be posted to an Endpoint in any RC Endpoint state (see *it_ep_state_t*) other than IT_EP_STATE_NONOPERATIONAL – i.e., even before Connection establishment – in order to prepare the Endpoint for reception.

The Consumer should use unique identifiers for *cookie* if they desire to identify each DTO. If the Consumer does not require a unique DTO identifier, the value of zero or NULL (IT_NO_ADDR) can be used.

For best Receive operation performance, the Consumer should align each buffer segment of *local_segments* to the *dto_alignment_hint* in the IA attributes obtained via *it_ia_query*.

When an EVD holding the DTO Event Stream associated with an Endpoint overflows, on some transports the behavior of the Endpoint and/or the EVD is undefined, making recovery from such an overflow either impossible or at the very least IA-dependent. Consumers should therefore take care to avoid this situation. Each Work Request that a Consumer posts to a Work Queue can result in a Completion Event being enqueued in an EVD, due to the possibility of an error occurring during Work Request processing. To avoid any possibility of overflow, the Consumer must therefore ensure that the total number of Outstanding Operations on all Work Queues that can enqueue Completion Events into a given EVD never exceeds the minimum queue size (as reported by *it_evd_query*) of that EVD.

**SEE ALSO**

*it_post_atomic*(), *it_post_send*(), *it_post_send_and_unlink*(), *it_post_sendto*(), *it_post_recvfrom*(), *it_post_rdma_read*(), *it_post_rdma_read_to_rmr*(), *it_post_rdma_write*(), *it_dto_status_t*, *it_dto_events*, *it_dto_flags_t*, *it_ep_rc_create*(), *it_lmr_triplet_t*, *it_ia_query*(), *it_dto_cookie_t*, *it_ia_info_t*, *it_ep_attributes_t*, *it_ep_state_t*

# it_post_recvfrom()

## NAME

it_post_recvfrom – post a Receive DTO to a datagram Endpoint

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_post_recvfrom(
    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
);
```

## APPLICABILITY

*it_post_recvfrom* is applicable only to Endpoints created for the UD service type.

## DESCRIPTION

| | |
|---|---|
| *ep_handle* | Handle for the local datagram Endpoint. |
| *local_segments* | Vector of *it_lmr_triplet_t* data structures that specifies the local buffer that will contain the received data. Local buffer must be at least 40 bytes. |
| *num_segments* | Number of *it_lmr_triplet_t* data structures in *local_segments*. Must be at least one. |
| *cookie* | Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the Receive. |
| *dto_flags* | Flags for posted Receive. |

*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*.

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.

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).

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.

The *cookie* (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work Request. This identifier is completely under Consumer control and opaque to the Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted Work Request.

The *dto_flags* value is used as specified in *it_dto_flags_t*.

A successful call returns IT_SUCCESS, which means that the Receive operation was successfully posted to the transport layer.

The completion of the posted Receive is reported asynchronously to the Consumer according to the rules defined in *it_dto_flags_t*. Exactly one Receive Completion Event of type *it_all_dto_cmpl_event_t* is always generated and manifests on the EVD associated with the Endpoint Receive Queue (see *it_ep_ud_create*).

If the reported *dto_status* of the Receive Completion Event (see *it_all_dto_cmpl_event_t*) is IT_DTO_SUCCESS (see *it_dto_status_t*), then the size of the received data can be retrieved through the *transferred_length* member of the Event. Once a successful Completion Event has been generated for the Receive, the order of the bytes in the local buffer specified by *local_segments* and *num_segments* corresponds to the order defined by the *local_segments* of the corresponding Send operation unless there is overlap among the segents of the local Receive buffer. If there is such an overlap, the content of the local buffer after the Completion Event has been generated is undefined. Prior to the Completion Event being generated, the content of the local buffer is Implementation-dependent. A successful Completion Event indicates that the data has been delivered uncorrupted into the local buffer.

If the reported completion status is not IT_DTO_SUCCESS, the content of the local buffer is not defined.

A Consumer should not modify the local buffer specified by *num_segments* and *local_segments* until the DTO is completed. When a Consumer does not adhere to this rule, the behavior of the Implementation and the underlying transport is not defined. A Consumer does get back the ownership of the array specified by the *local_segments* and *num_segments* arguments (but not the local buffer identified by this array) when *it_post_recvfrom* returns and is free to use this array for other calls, to modify it, or to destroy it.

The Implementation ensures that each Receive corresponds to one and only one remote Send.

There is no relationship guaranteed on the order of Receive completions and the order of the posting of the corresponding Sends at remote datagram Endpoints.

The Implementation ensures that all Receives start and complete in the order posted.

The Implementation ensures that all data from a given Send operation is transferred from the remote buffer and into the local buffer before a Receive completion is generated with IT_DTO_SUCCESS.

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_INVALID_DTO_FLAGS | The Data Transfer Operation flags (*dto_flags*) value was invalid. |
| IT_ERR_INVALID_EP | The Endpoint Handle (*ep_handle*) was invalid. |

| | |
|---|---|
| IT_ERR_INVALID_EP_TYPE | The attempted operation was invalid for the Service Type of the Endpoint. |
| IT_ERR_INVALID_NUM_SEGMENTS | The requested number of segments (*num_segments*) was larger than the Endpoint supports. |
| IT_ERR_TOO_MANY_POSTS | The operation failed due to an overflow of a Work Queue. |
| 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. |

**ASYNCHRONOUS ERRORS**

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

An Affiliated Asynchronous Error that is associated with an Endpoint will move that Endpoint to the IT_EP_STATE_UD_NONOPERATIONAL state.

For errors that can be reported as Completion Errors, see also *it_dto_status_t*.

If no Receive buffers are posted in time, then an incoming Send will fail to deliver its data. In order to avoid this error, the Consumer should post Receive resources prior to the remote Consumer posting a Send. The lack of a Receive buffer is handled as follows:

For the IB transport, an IT_ASYNC_AFF_EP_L_RECV_ACCESS_VIOLATION Affiliated Asynchronous Error is generated.

If the size of an incoming message is larger than the size of the local buffer or larger than the MTU of the local Spigot, a length error will occur. In order to avoid length errors, the Consumer should post a buffer large enough for the incoming Send data. Length errors are handled as follows:

For the IB transport, the Receive DTO completes with an IT_DTO_ERR_LOCAL_LENGTH error status.

If the Direct LMR Handle is used to specify a local buffer in a Work Request and the Endpoint to which the Work Request is posted is not Privileged, then a Completion Error with the DTO status IT_DTO_ERR_LOCAL_PROTECTION shall be returned.

**APPLICATION USAGE**

This function is used to transfer data into a Consumer-specified local buffer from a buffer specified by the corresponding Send operation at the remote Endpoint.

The Consumer should use unique identifiers for *cookie* if they desire to identify each DTO. If the Consumer does not require a unique DTO identifier, the value of zero or NULL (IT_NO_ADDR) can be used.

For best Receive operation performance, the Consumer should align each buffer segment of *local_segments* to the *dto_alignment_hint* in the IA attributes obtained via *it_ia_query*.

When an EVD holding the DTO Event Stream associated with an Endpoint overflows, on some transports the behavior of the Endpoint and/or the EVD is undefined, making recovery from

such an overflow either impossible or at the very least IA-dependent. Consumers should therefore take care to avoid this situation. Each Work Request that a Consumer posts to a Work Queue can result in a Completion Event being enqueued in an EVD, due to the possibility of an error occurring during Work Request processing. To avoid any possibility of overflow, the Consumer must therefore ensure that the total number of Outstanding Operations on all Work Queues that can enqueue Completion Events into a given EVD never exceeds the minimum queue size (as reported by *it_evd_query*) of that EVD.

**SEE ALSO**

*it_post_atomic*(), *it_post_send*(), *it_post_send_and_unlink*(), *it_post_sendto*(), *it_post_recv*(), *it_post_rdma_read*(), *it_post_rdma_read_to_rmr*(), *it_post_rdma_write*(), *it_dto_status_t*, *it_dto_events*, *it_dto_flags_t*, *it_ep_ud_create*(), *it_lmr_triplet_t*, *it_ia_query*(), *it_dto_cookie_t*, *it_ia_info_t*

# it_post_send()

## NAME

it_post_send – post a Send DTO to a connected Endpoint

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_post_send(
    IN         it_ep_handle_t     ep_handle,
    IN  const  it_lmr_triplet_t  *local_segments,
    IN         size_t             num_segments,
    IN         it_dto_cookie_t    cookie,
    IN         it_dto_flags_t     dto_flags
);
```

## APPLICABILITY

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

## DESCRIPTION

| | |
|---|---|
| *ep_handle* | Handle for the Endpoint – the local side of the Connection. |
| *local_segments* | Vector of *it_lmr_triplet_t* data structures that specifies the local buffer that contains data to be transferred. Can be NULL (IT_NO_ADDR) for a zero-sized message. |
| *num_segments* | Number of *it_lmr_triplet_t* data structures in *local_segments*. Can be zero for a zero-sized message. |
| *cookie* | Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the Send. |
| *dto_flags* | Flags for posted Send. |

*it_post_send* posts a request to the *ep_handle* Endpoint to transfer over the reliable Connection of that Endpoint all the data from the local buffer (data source) specified by *local_segments* and *num_segments* into a remote buffer specified by a single corresponding Receive on the other side of the Connection. If *num_segments* is non-zero, then the size of the data transferred is given by the sum of the segment lengths specified by *local_segments.* A zero-sized message may be transferred over the Connection and will consume a buffer specified by the corresponding Receive.

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.

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).

The *cookie* (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work Request. This identifier is completely under Consumer control and opaque to the

Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted Work Request.

The *dto_flags* value is used as specified in *it_dto_flags_t*.

A successful call returns IT_SUCCESS, which means that the Send operation was successfully posted to the transport layer.

The completion of the posted Send is reported asynchronously to the Consumer according to the rules defined in *it_dto_flags_t*. A Send Completion Event is of type *it_dto_cmpl_event_t*. Any generated Send Completion Event manifests on the EVD associated with the Endpoint Send Queue. See *it_ep_rc_create*, *it_dto_status_t*, and *it_dto_events*. The Completion Event for the *it_post_send* call indicates to the Consumer that the local buffer is under Consumer control again and that the contents of the local buffer will be transported reliably, but does not guarantee that these contents have been successfully received by the remote Consumer; wherever necessary, this restriction is made explicit by designating a Completion as local or by stating that operations complete locally. See Section 5.4 of [DDP-IETF] for background information. The contents of the local buffer are only guaranteed to have reached the remote Consumer's memory when the remote Consumer reaps a successful completion for the Receive operation that matches the Send initiated by the *it_post_send* call. However, once the contents of the local buffer reach the remote Consumer memory, the order of the bytes in the remote buffer specified by the Receive operation at the remote Endpoint corresponds to the order defined by the Send side *local_segments*, subject to overlap constraints. See *it_post_recv* for details on overlap constraints.

A Consumer should not modify the local buffer specified by *num_segments* and *local_segments* until the DTO is locally completed. When a Consumer does not adhere to this rule, the content of the local buffer at the receiving side is undefined after the matching Receive operation completes. A Consumer does get back the ownership of the *num_segments* and *local_segments* arguments (but not the local buffer identified by them) when *it_post_send* returns and is free to use the *num_segments* and *local_segments* arguments for other calls, to modify them, or to destroy them. A Consumer does get back the ownership of the array specified by the *local_segments* and *num_segments* arguments (but not the local buffer identified by this array) when *it_post_send* returns and is free to use this array for other calls, to modify it, or to destroy it.

The Implementation ensures that each Send corresponds to one and only one remote Receive and in no way corresponds to any locally or remotely posted RDMA Read or RDMA Write Data Transfer Operations over the same Connection.

The Implementation ensures that each RDMA Write DTO posted to an Endpoint prior to a Send DTO posted to the same Endpoint has its complete data payload delivered to the remote memory prior to the completion of the Receive DTO at the remote side matching the Send DTO. However, the Implementation does not ensure that the RDMA Write DTO places its entire data payload into its remote buffer before the Receive DTO places the data payload of the incoming Send into its Receive buffer; if the remote buffers overlap, their contents will be indeterminate.

The Implementation ensures that subsequent Send DTOs posted to the same Endpoint start and complete locally in post order and that the Receive DTOs at the remote side matching the Send DTOs complete in the same order. However, the Implementation does not ensure that the matching Receive DTOs place the data payloads into their Receive buffers in the post order of the Send DTOs; if the Receive buffers overlap, their contents will be indeterminate.

In general, Work Requests following an RDMA Read may start execution while the RDMA Read is in progress (but may not complete before the RDMA Read completes). To ensure deterministically that a Send DTO following an RDMA Read DTO starts after the RDMA Read completes, specify the IT_BARRIER_FENCE_FLAG on the Send DTO.

Any Work Request posted to the Send Queue of an Endpoint (thus also a Send) after a call to *it_rmr_link* or *it_rmr_unlink* will not begin execution until the Link or Unlink operation has completed.

**EXTENDED DESCRIPTION**

On the InfiniBand Transport, the Implementation ensures that an RDMA Write DTO followed by a Send DTO posted to the same Endpoint cause data payloads to be placed in remote memory in post order; relying on such placement ordering represents a transport-dependent programming practice.

**RETURN VALUE**

A successful call returns IT_SUCCESS.

Posting to an Endpoint that is not in the IT_EP_STATE_CONNECTED or IT_EP_STATE_NONOPERATIONAL state will return the IT_ERR_INVALID_EP_STATE immediate error.

The possible immediate errors for *it_post_send* are listed below:

| | |
|---|---|
| IT_ERR_INVALID_DTO_FLAGS | The Data Transfer Operation flags (*dto_flags*) value was invalid. |
| IT_ERR_INVALID_EP | The Endpoint Handle (*ep_handle*) was invalid. |
| IT_ERR_INVALID_EP_STATE | The Endpoint was not in the proper state for the attempted operation. |
| IT_ERR_INVALID_EP_TYPE | The attempted operation was invalid for the Service Type of the Endpoint. |
| IT_ERR_INVALID_NUM_SEGMENTS | The requested number of segments (*num_segments*) was larger than the Endpoint supports. |
| IT_ERR_TOO_MANY_POSTS | The operation failed due to an overflow of a Work Queue. |
| IT_ERR_INVALID_LMR | The *local_segments* contains the Direct LMR Handle on an IA that does not support Direct LMRs. |
| 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. |

**ASYNCHRONOUS ERRORS**

For Work Requests posted to an Endpoint in the IT_EP_STATE_CONNECTED state, a completion status (see *it_dto_status_t*) other than IT_DTO_SUCCESS will break the Connection by moving the Endpoint to the IT_EP_STATE_NONOPERATIONAL state and deliver an IT_CM_MSG_CONN_BROKEN_EVENT Event to the Connect EVD of *ep_handle*. Once the Connection is broken, all outstanding and in-progress operations on the Connection will complete with an error status.

An Affiliated Asynchronous Error that is associated with an Endpoint will move that Endpoint to the IT_EP_STATE_NONOPERATIONAL state; if this state transition was from the IT_EP_STATE_CONNECTED state, an IT_CM_MSG_CONN_BROKEN_EVENT Event will be delivered to the Connect EVD of *ep_handle*.

Any posting to an Endpoint that is in the IT_EP_STATE_NONOPERATIONAL state will be flushed with completion status set to IT_DTO_ERR_FLUSHED.

For locally or remotely detected errors that can be reported as Completion Errors, see also *it_dto_status_t*.

The handling of remotely detected errors is transport and Implementation-dependent; end-to-end completions are not supported for certain transports or DTOs.

If no Receive buffers are posted in time at the remote end, then a Send will fail to successfully deliver the contents of the local buffer. In order to avoid this error, the remote Consumer should post Receive resources prior to the local Consumer posting the Send. The lack of a remote Receive buffer is handled as follows:

For IB and iWARP, an IT_ASYNC_AFF_EP_L_RECV_ACCESS_VIOLATION Affiliated Asynchronous Error will surface remotely.

For the IB transport, the Send DTO completes with an IT_DTO_ERR_REMOTE_RESPONDER error status.

For the iWARP Transport, an IT_ASYNC_AFF_EP_R_RECV_ACCESS_VIOLATION Affiliated Asynchronous Error will surface after the Send DTO has already completed with IT_DTO_SUCCESS.

If the Receive buffer of the Receive DTO at the remote side matching the Send DTO is not sufficient in size for the data payload of the Send DTO, a length error will occur. In order to avoid length errors, the remote Consumer should post a buffer large enough for the incoming Send data. Remotely detected length errors are handled as follows:

For both the IB and iWARP Transports, the Receive DTO at the remote side completes with an IT_DTO_ERR_LOCAL_LENGTH error status.

For the IB transport, the Send DTO completes with an IT_DTO_ERR_REMOTE_RESPONDER error status.

For the iWARP Transport, an IT_ASYNC_AFF_EP_R_RECV_LENGTH_ERROR Affiliated Asynchronous Error will surface after the Send DTO has already completed with IT_DTO_SUCCESS.

Despite using the RC service, a Send operation may fail to successfully deliver the contents of the local buffer. A failure such as an unfinished data delivery or a CRC error that cannot be corrected by the transport is detected as a transport error and manifests as follows:

For the IB transport, a transport error causes the Send DTO and possibly the Receive DTO matching the Send DTO to complete with an IT_DTO_ERR_TRANSPORT Completion Error.

For the iWARP Transport, a transport error causes the Receive DTO matching the Send DTO to complete with an IT_DTO_ERR_TRANSPORT Completion Error and an IT_ASYNC_AFF_EP_R_ERROR Affiliated Asynchronous Error to surface locally after the Send DTO has already completed with IT_DTO_SUCCESS.

If a type of remotely detected error can manifest locally in different ways – i.e., as an Affiliated Asynchronous Error or as a Completion Error – transport-independent programming requires Consumers to be prepared to deal with either.

If the Direct LMR Handle is used to specify a local buffer in a Work Request and the Endpoint to which the Work Request is posted is not Privileged, then a Completion Error with the DTO status IT_DTO_ERR_LOCAL_PROTECTION shall be returned.

**APPLICATION USAGE**

This function is used after a Connection has been established to transfer data from a Consumer-specified local buffer to a buffer specified by the corresponding Receive operation on the other side of the Connection.

The Consumer should use unique identifiers for *cookie* if they desire to identify each DTO. If the Consumer does not require a unique DTO identifier, the value of zero or NULL (IT_NO_ADDR) can be used.

For best Send operation performance, the Consumer should align each buffer segment of *local_segments* to the *dto_alignment_hint* in the IA attributes obtained via *it_ia_query*.

When an EVD holding the DTO Event Stream associated with an Endpoint overflows, on some transports the behavior of the Endpoint and/or the EVD is undefined, making recovery from such an overflow either impossible or at the very least IA-dependent. Consumers should therefore take care to avoid this situation. Each Work Request that a Consumer posts to a Work Queue can result in a Completion Event being enqueued in an EVD, due to the possibility of an error occurring during Work Request processing. To avoid any possibility of overflow, the Consumer must therefore ensure that the total number of Outstanding Operations on all Work Queues that can enqueue Completion Events into a given EVD never exceeds the minimum queue size (as reported by *it_evd_query*) of that EVD.

**SEE ALSO**

*it_post_atomic*(), *it_post_send_and_unlink*(), *it_post_sendto*(), *it_post_recv*(), *it_post_recvfrom*(), *it_post_rdma_read*(), *it_post_rdma_read_to_rmr*(), *it_post_rdma_write*(), *it_dto_status_t*, *it_dto_events*, *it_dto_flags_t*, *it_ep_rc_create*(), *it_lmr_triplet_t*, *it_ia_query*(), *it_dto_cookie_t*, *it_ia_info_t*

# it_post_send_and_unlink()

## NAME

it_post_send_and_unlink – post a Send and Unlink DTO to a connected Endpoint

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_post_send_and_unlink(
    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         it_rmr_context_t   rmr_context
);
```

## APPLICABILITY

*it_post_send_and_unlink* is applicable only to Endpoints created for the RC service type and is supported only if the Interface Adapter attribute *post_send_unlink_remote_support* (see *it_ia_info_t*) is IT_TRUE.

## DESCRIPTION

| | |
|---|---|
| *ep_handle* | Handle for the Endpoint – the local side of the Connection. |
| *local_segments* | Vector of *it_lmr_triplet_t* data structures that specifies the local buffer that contains data to be transferred. Can be NULL (IT_NO_ADDR) for a zero-sized message. |
| *num_segments* | Number of *it_lmr_triplet_t* data structures in *local_segments*. Can be zero for a zero-sized message. |
| *cookie* | Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the Send and Unlink operation. |
| *dto_flags* | Flags for posted Send and Unlink operation. |
| *rmr_context* | RMR Context to be unlinked at Remote. |

*it_post_send_and_unlink* posts a request to the *ep_handle* Endpoint to transfer over the reliable Connection of that Endpoint all the data from the local buffer (data source) specified by *local_segments* and *num_segments* into a remote buffer specified by a single corresponding Receive on the other side of the Connection. In addition, on successful transmission to the remote, the remote buffer registered remotely using the RMR Context *rmr_context* is unlinked.

If *num_segments* is non-zero, then the size of the data transferred is given by the sum of the segment lengths specified by *local_segments.* A zero-sized message may be transferred over the Connection and will consume a buffer specified by the corresponding Receive.

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.

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).

The *cookie* (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work Request. This identifier is completely under Consumer control and opaque to the Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted Work Request.

The *dto_flags* value is used as specified in *it_dto_flags_t*.

The *rmr_context* is used to specify an LMR or RMR at the remote that, if the Send portion of this operation completes successfully, will be unlinked after the Send data has been placed at the remote.

A successful call returns IT_SUCCESS, which means that the Send operation was successfully posted to the transport layer.

The completion of the posted Send is reported asynchronously to the Consumer according to the rules defined in *it_dto_flags_t*. A Send Completion Event is of type *it_dto_cmpl_event_t*. Any generated Send Completion Event manifests on the EVD associated with the Endpoint Send Queue. See *it_ep_rc_create*, *it_dto_status_t*, and *it_dto_events*. The Completion Event for the *it_post_send_and_unlink* call indicates to the Consumer that the local buffer is under Consumer control again and that the contents of the local buffer will be transported reliably, but does not guarantee that these contents have been successfully received by the remote Consumer; wherever necessary, this restriction is made explicit by designating a Completion as local or by stating that operations complete locally. See Section 5.4 of [DDP-IETF] for background information. The contents of the local buffer are only guaranteed to have reached the remote Consumer's memory when the remote Consumer reaps a successful completion for the Receive operation that matches the Send initiated by the *it_post_send_and_unlink* call. However, once the contents of the local buffer reach the remote Consumer memory, the order of the bytes in the remote buffer specified by the Receive operation at the remote Endpoint corresponds to the order defined by the Send side *local_segments*, subject to overlap constraints. See *it_post_recv* for details on overlap constraints.

A Consumer should not modify the local buffer specified by *num_segments* and *local_segments* until the DTO is locally completed. When a Consumer does not adhere to this rule, the content of the local buffer at the receiving side is undefined after the matching Receive operation completes. A Consumer does get back the ownership of the *num_segments* and *local_segments* arguments (but not the local buffer identified by them) when *it_post_send_and_unlink* returns and is free to use the *num_segments* and *local_segments* arguments for other calls, to modify them, or to destroy them. A Consumer does get back the ownership of the array specified by the *local_segments* and *num_segments* arguments (but not the local buffer identified by this array) when *it_post_send_and_unlink* returns and is free to use this array for other calls, to modify it, or to destroy it.

The Implementation ensures that each Send corresponds to one and only one remote Receive and in no way corresponds to any locally or remotely posted RDMA Read or RDMA Write Data Transfer Operations over the same Connection.

The Implementation ensures that each RDMA Write DTO posted to an Endpoint prior to a Send DTO posted to the same Endpoint has its complete data payload delivered to the remote memory prior to the completion of the Receive DTO at the remote side matching the Send DTO.

However, the Implementation does not ensure that the RDMA Write DTO places its entire data payload into its remote buffer before the Receive DTO places the data payload of the incoming Send into its Receive buffer; if the remote buffers overlap, their contents will be indeterminate.

The Implementation ensures that subsequent Send DTOs posted to the same Endpoint start and complete locally in post order and that the Receive DTOs at the remote side matching the Send DTOs complete in the same order. However, the Implementation does not ensure that the matching Receive DTOs place the data payloads into their Receive buffers in the post order of the Send DTOs; if the Receive buffers overlap, their contents will be indeterminate.

In general, Work Requests following an RDMA Read may start execution while the RDMA Read is in progress (but may not complete before the RDMA Read completes). To ensure deterministically that a Send DTO following an RDMA Read DTO starts after the RDMA Read completes, specify the IT_BARRIER_FENCE_FLAG on the Send DTO.

Any Work Request posted to the Send Queue of an Endpoint (thus also a Send) after a call to *it_rmr_link* or *it_rmr_unlink* will not begin execution until the Link or Unlink operation has completed.

**EXTENDED DESCRIPTION**

On the InfiniBand Transport, the Implementation ensures that an RDMA Write DTO followed by a Send DTO posted to the same Endpoint cause data payloads to be placed in remote memory in post order; relying on such placement ordering represents a transport-dependent programming practice.

**RETURN VALUE**

A successful call returns IT_SUCCESS.

Posting to an Endpoint that is not in the IT_EP_STATE_CONNECTED or IT_EP_STATE_NONOPERATIONAL state will return the IT_ERR_INVALID_EP_STATE immediate error.

The possible immediate errors for *it_post_send_and_unlink* are listed below:

| | |
|---|---|
| IT_ERR_INVALID_DTO_FLAGS | The Data Transfer Operation flags (*dto_flags*) value was invalid. |
| IT_ERR_INVALID_EP | The Endpoint Handle (*ep_handle*) was invalid. |
| IT_ERR_INVALID_EP_STATE | The Endpoint was not in the proper state for the attempted operation. |
| IT_ERR_INVALID_EP_TYPE | The attempted operation was invalid for the Service Type of the Endpoint. |
| IT_ERR_INVALID_NUM_SEGMENTS | The requested number of segments (*num_segments*) was larger than the Endpoint supports. |
| IT_ERR_TOO_MANY_POSTS | The operation failed due to an overflow of a Work Queue. |
| IT_ERR_INVALID_LMR | The *local_segments* contains the Direct LMR Handle on an IA that does not support Direct LMRs. |
| 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.

**ASYNCHRONOUS ERRORS**

For Work Requests posted to an Endpoint in the IT_EP_STATE_CONNECTED state, a completion status (see *it_dto_status_t*) other than IT_DTO_SUCCESS will break the Connection by moving the Endpoint to the IT_EP_STATE_NONOPERATIONAL state and deliver an IT_CM_MSG_CONN_BROKEN_EVENT Event to the Connect EVD of *ep_handle*. Once the Connection is broken, all outstanding and in-progress operations on the Connection will complete with an error status.

An Affiliated Asynchronous Error that is associated with an Endpoint will move that Endpoint to the IT_EP_STATE_NONOPERATIONAL state; if this state transition was from the IT_EP_STATE_CONNECTED state, an IT_CM_MSG_CONN_BROKEN_EVENT Event will be delivered to the Connect EVD of *ep_handle*.

Any posting to an Endpoint that is in the IT_EP_STATE_NONOPERATIONAL state will be flushed with completion status set to IT_DTO_ERR_FLUSHED.

For locally or remotely detected errors that can be reported as Completion Errors, see also *it_dto_status_t*.

The handling of remotely detected errors is transport and Implementation-dependent; end-to-end completions are not supported for certain transports or DTOs.

If no Receive buffers are posted in time at the remote end, then a Send will fail to successfully deliver the contents of the local buffer. In order to avoid this error, the remote Consumer should post Receive resources prior to the local Consumer posting the Send. The lack of a remote Receive buffer is handled as follows:

For IB and iWARP, an IT_ASYNC_AFF_EP_L_RECV_ACCESS_VIOLATION Affiliated Asynchronous Error will surface remotely.

For the IB transport, the Send DTO completes with an IT_DTO_ERR_REMOTE_RESPONDER error status.

For the iWARP Transport, an IT_ASYNC_AFF_EP_R_RECV_ACCESS_VIOLATION Affiliated Asynchronous Error will surface after the Send DTO has already completed with IT_DTO_SUCCESS.

If the Receive buffer of the Receive DTO at the remote side matching the Send DTO is not sufficient in size for the data payload of the Send DTO, a length error will occur. In order to avoid length errors, the remote Consumer should post a buffer large enough for the incoming Send data. Remotely detected length errors are handled as follows:

For both the IB and iWARP Transports, the Receive DTO at the remote side completes with an IT_DTO_ERR_LOCAL_LENGTH error status.

For the IB transport, the Send DTO completes with an IT_DTO_ERR_REMOTE_RESPONDER error status.

For the iWARP Transport, an IT_ASYNC_AFF_EP_R_RECV_LENGTH_ERROR Affiliated Asynchronous Error will surface after the Send DTO has already completed with IT_DTO_SUCCESS.

Despite using the RC service, a Send operation may fail to successfully deliver the contents of the local buffer. A failure such as an unfinished data delivery or a CRC error that cannot be corrected by the transport is detected as a transport error and manifests as follows:

For the IB transport, a transport error causes the Send DTO and possibly the Receive DTO matching the Send DTO to complete with an IT_DTO_ERR_TRANSPORT Completion Error.

For the iWARP Transport, a transport error causes the Receive DTO matching the Send DTO to complete with an IT_DTO_ERR_TRANSPORT Completion Error and an IT_ASYNC_AFF_EP_R_ERROR Affiliated Asynchronous Error to surface locally after the Send DTO has already completed with IT_DTO_SUCCESS.

If a type of remotely detected error can manifest locally in different ways – i.e., as an Affiliated Asynchronous Error or as a Completion Error – transport-independent programming requires Consumers to be prepared to deal with either.

If the Direct LMR Handle is used to specify a local buffer in a Work Request and the Endpoint to which the Work Request is posted is not Privileged, then a Completion Error with the DTO status IT_DTO_ERR_LOCAL_PROTECTION shall be returned.

Unlinking a remote RMR Context (RMR/LMR) through *it_post_send_and_unlink* may result in a remotely detected error if the remote RMR Context is invalid or corresponds to the remote Direct LMR Handle, or if the remote PZ of the remote RMR Context does not match the remote PZ of the remote EP, or if the remote RMR is a Wide RMR, or if the remote RMR is a linked Narrow RMR associated with an Endpoint other than the remote Endpoint of the connection, or if the remote RMR is a narrow RMR in another PZ than that of the remote Endpoint of the connection, or if the RMR Context corresponds to a remote LMR that has bound RMRs or the LMR is shared or the LMR does not have remote access or the LMR cannot be unlinked due to the way it was linked (IB-only) or if the LMR was already in the unlinked state. Such a remotedly detected error manifests remotely by returning a Completion Error for the matching Receive and by terminating the connection. The remote RMR/LMR remains linked. Such remotely detected erors manifest locally in a transport-dependent manner – for InfiniBand, subsequent Work Requests to the same Endpoint will fail with an IT_DTO_ERR_REMOTE_RESPONDER Completion Error; for iWARP, the connection will be torn down abortively and no Completion Error will be generated.

**APPLICATION USAGE**

This function is used after a Connection has been established to transfer data from a Consumer-specified local buffer to a buffer specified by the corresponding Receive operation on the other side of the Connection. In addition, this function supplies an RMR Context that is to be unlinked at the remote – the remote Consumer should examine the RMR Context found in the corresponding Receive Completion Event to determine if one of their local Memory Regions (LMR or RMR) has been unlinked.

The Consumer should use unique identifiers for *cookie* if they desire to identify each DTO. If the Consumer does not require a unique DTO identifier, the value of zero or NULL (IT_NO_ADDR) can be used.

For best Send operation performance, the Consumer should align each buffer segment of *local_segments* to the *dto_alignment_hint* in the IA attributes obtained via *it_ia_query*.

When an EVD holding the DTO Event Stream associated with an Endpoint overflows, on some transports the behavior of the Endpoint and/or the EVD is undefined, making recovery from

such an overflow either impossible or at the very least IA-dependent. Consumers should therefore take care to avoid this situation. Each Work Request that a Consumer posts to a Work Queue can result in a Completion Event being enqueued in an EVD, due to the possibility of an error occurring during Work Request processing. To avoid any possibility of overflow, the Consumer must therefore ensure that the total number of Outstanding Operations on all Work Queues that can enqueue Completion Events into a given EVD never exceeds the minimum queue size (as reported by *it_evd_query*) of that EVD.

**SEE ALSO**

*it_post_atomic*(), *it_post_send*(), *it_post_sendto*(), *it_post_recv*(), *it_post_recvfrom*(), *it_post_rdma_read*(), *it_post_rdma_read_to_rmr*(), *it_post_rdma_write*(), *it_dto_status_t*, *it_dto_events*, *it_dto_flags_t*, *it_ep_rc_create*(), *it_lmr_triplet_t*, *it_ia_query*(), *it_dto_cookie_t*, *it_ia_info_t*

# it_post_sendto()

## NAME

it_post_sendto – post a Send DTO to a datagram Endpoint

## SYNOPSIS

```
#include <it_api.h>

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
);
```

## APPLICABILITY

*it_post_sendto* is applicable only to Endpoints created for the UD service type.

## DESCRIPTION

| | |
|---|---|
| *ep_handle* | Handle for the local datagram Endpoint. |
| *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. |
| *num_segments* | Number of *it_lmr_triplet_t* data structures in *local_segments*. Can be zero for a zero-sized message. |
| *cookie* | Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the Send. |
| *dto_flags* | Flags for posted Send. |
| *remote_ep_addr* | Remote datagram Endpoint address. |

*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.

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.

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).

The *cookie* (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work Request. This identifier is completely under Consumer control and opaque to the

Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted Work Request.

The *dto_flags* value is used as specified in *it_dto_flags_t*.

*remote_ep_addr* specifies the Destination for the *it_post_sendto* operation. See *it_dg_remote_ep_addr_t* for details on the format of this data structure.

A successful call returns IT_SUCCESS, which means that the Send operation was successfully posted to the transport layer.

The completion of the posted Send is reported asynchronously to the Consumer according to the rules defined in *it_dto_flags_t*. A Send Completion Event is of type *it_all_dto_cmpl_event_t*. Any generated Send Completion Event manifests on the EVD associated with the Endpoint Send Queue. See *it_ep_ud_create*, *it_dto_status_t* and *it_dto_events*.

Once a successful Completion Event has been generated at the receiver, the order of the bytes in the remote buffer specified by the Receive operation at the remote Endpoint corresponds to the order defined by the Send side *local_segments* subject to overlap constraints. See *it_post_recvfrom* for details on overlap constraints.

A Consumer should not modify the local buffer specified by *num_segments* and *local_segments* until the DTO is completed. When a Consumer does not adhere to this rule, the behavior of the Implementation and the underlying transport is not defined. A Consumer does get back the ownership of the array specified by the *local_segments* and *num_segments* arguments (but not the local buffer identified by this array) when *it_post_sendto* returns and is free to use this array for other calls, to modify it, or to destroy it.

The Implementation ensures that all Sends start and complete in the order posted.

The Implementation makes no delivery order guarantees for Unreliable Datagrams.

There is no delivery order or completion order between Receive Data Transfer Operations on different Destinations that correspond to the Sends posted in order to the same Unreliable Datagram Endpoint.

When *it_post_sendto* completes with IT_DTO_SUCCESS or IT_DTO_ERR_LOCAL_EP there is no guarantee that the DTO has reached the remote Endpoint.

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_INVALID_AH | The Address Handle within *remote_ep_addr* was invalid or the does not match the *spigot_id* of the Endpoint. |
| IT_ERR_INVALID_DTO_FLAGS | The Data Transfer Operation flags (*dto_flags*) value was invalid. |
| IT_ERR_INVALID_EP | The Endpoint Handle (*ep_handle*) was invalid. |
| IT_ERR_INVALID_EP_TYPE | The attempted operation was invalid for the Service Type of the Endpoint. |
| IT_ERR_INVALID_NUM_SEGMENTS | The requested number of segments (*num_segments*) was larger than the Endpoint supports. |

| | |
|---|---|
| IT_ERR_TOO_MANY_POSTS | The operation failed due to an overflow of a Work Queue. |
| IT_ERR_INVALID_LMR | The *local_segments* contains the Direct LMR Handle on an IA that does not support Direct LMRs. |
| 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. |

**ASYNCHRONOUS ERRORS**

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

An Affiliated Asynchronous Error that is associated with an Endpoint will move that Endpoint to the IT_EP_STATE_UD_NONOPERATIONAL state.

For errors that can be reported as Completion Errors, see also *it_dto_status_t*.

If no Receive buffers are posted in time at the remote end, then a Send will fail to successfully deliver the contents of the local buffer. In order to avoid this error, the remote Consumer should post Receive resources prior to the local Consumer posting the Send. The lack of a remote Receive buffer is handled as follows:

For the IB transport, an IT_ASYNC_AFF_EP_L_RECV_ACCESS_VIOLATION Affiliated Asynchronous Error will surface remotely.

If the Receive buffer of the Receive DTO at the remote side matching the Send DTO is not sufficient in size for the data payload of the Send DTO, a length error will occur. In order to avoid length errors, the remote Consumer should post a buffer large enough for the incoming Send data. Remotely detected length errors are handled as follows:

For the IB transport, the Receive DTO at the remote side completes with an IT_DTO_ERR_LOCAL_LENGTH error status.

A completion status IT_DTO_ERR_LOCAL_EP may be returned in case of an Address Handle inconsistency (see Application Usage).

If the Direct LMR Handle is used to specify a local buffer in a Work Request and the Endpoint to which the Work Request is posted is not Privileged, then a Completion Error with the DTO status IT_DTO_ERR_LOCAL_PROTECTION shall be returned.

**APPLICATION USAGE**

This function is used to transfer data from a Consumer-specified local buffer to a buffer specified by the corresponding Receive operation at a remote datagram Endpoint.

The Consumer should use unique identifiers for *cookie* if they desire to identify each DTO. If the Consumer does not require a unique DTO identifier, the value of zero or NULL (IT_NO_ADDR) can be used.

For best Send operation performance, the Consumer should align each buffer segment of *local_segments* to the *dto_alignment_hint* in the IA attributes obtained via *it_ia_query*.

An Address Handle corresponds to a specific Spigot on an IA. Attempting to *it_post_sendto* on an Endpoint using an Address Handle that does not correspond to the Spigot associated with the

Endpoint must be avoided by the Consumer. If the Consumer persists in this practice, they must write error handling code to deal with three possible error cases: One – the *it_post_sendto* call will return the IT_ERR_INVALID_AH error immediately. Or two – the DTO will complete in error with the *it_dto*_status set to IT_DTO_ERR_LOCAL_EP. Or three – there will be no indication of error. The three possible cases represent the allowable implementations of the underlying technology.

When an EVD holding the DTO Event Stream associated with an Endpoint overflows, on some transports the behavior of the Endpoint and/or the EVD is undefined, making recovery from such an overflow either impossible or at the very least IA-dependent. Consumers should therefore take care to avoid this situation. Each Work Request that a Consumer posts to a Work Queue can result in a Completion Event being enqueued in an EVD, due to the possibility of an error occurring during Work Request processing. To avoid any possibility of overflow, the Consumer must therefore ensure that the total number of Outstanding Operations on all Work Queues that can enqueue Completion Events into a given EVD never exceeds the minimum queue size (as reported by *it_evd_query*) of that EVD.

**SEE ALSO**

*it_post_atomic*(), *it_post_send*(), *it_post_send_and_unlink*(), *it_post_recv*(), *it_post_recvfrom*(), *it_post_rdma_read*(), *it_post_rdma_read_to_rmr*(), *it_post_rdma_write*(), *it_dto_status_t*, *it_dto_events*, *it_dg_remote_ep_addr_t*, *it_dto_flags_t*, *it_ep_ud_create*(), *it_lmr_triplet_t*, *it_ia_query*(), *it_dto_cookie_t*, *it_ia_info_t*

# it_pz_create()

## NAME

it_pz_create – create a new Protection Zone

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_pz_create(
    IN   it_ia_handle_t   ia_handle,
    OUT  it_pz_handle_t  *pz_handle
);
```

## DESCRIPTION

| | |
|---|---|
| *ia_handle* | Interface Adapter on which the Protection Zone will be created. |
| *pz_handle* | Handle of new Protection Zone |

The *it_pz_create* routine creates a new Protection Zone that may be used to create Local Memory Regions, Remote Memory Regions, transport Endpoints, or Address Handles on the Interface Adapter identified by *ia_handle*. The Protection Zone is returned in *pz_handle*.

## RETURN VALUE

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

| | |
|---|---|
| IT_ERR_RESOURCES | The requested operation failed due to insufficient resources. |
| IT_ERR_INVALID_IA | The Interface Adapter Handle (*ia_handle*) was invalid. |
| 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. |

## APPLICTION USAGE

An LMR, RMR, Endpoint, or Address Handle cannot be created without supplying a Protection Zone. An LMR, RMR, or Address Handle may only be used in concert with an Endpoint having the same Protection Zone. In DTO, Link, and Unlink operations, the Protection Zone of the local Endpoint and the LMR must match, or the operation will fail. In RDMA operations, the Protection Zone of the RMR associated with the RMR Context must match that of the remote Endpoint. In datagram DTO operations, the Protection Zone of the local Address Handle identifying the Destination must match that of the local Endpoint. In Link and Unlink operations, the Protection Zone of the LMR and RMR must match.

## SEE ALSO

*it_pz_free*(), *it_pz_query*()

# it_pz_free()

## NAME

it_pz_free – destroy a Protection Zone

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_pz_free(
    IN  it_pz_handle_t  pz_handle
);
```

## DESCRIPTION

| | |
|---|---|
| *pz_handle* | Handle of Protection Zone to be destroyed. |

The *it_pz_free* routine destroys the Protection Zone *pz_handle*. On successful return, the *pz_handle* may no longer be used. An attempt to free a Protection Zone that is still referenced by undestroyed Endpoints, Local Memory Regions, Remote Memory Regions, or Address Handles will fail with IT_ERR_PZ_BUSY, and the Protection Zone will be unaffected.

## RETURN VALUE

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

| | |
|---|---|
| IT_ERR_INVALID_PZ | The Protection Zone Handle (*pz_handle*) was invalid. |
| IT_ERR_PZ_BUSY | The Protection Zone was still in use. |
| 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. |

## SEE ALSO

*it_pz_create*(), *it_pz_query*()

# it_pz_query()

## NAME

it_pz_query – get attributes of a Protection Zone

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_pz_query(
    IN   it_pz_handle_t      pz_handle,
    IN   it_pz_param_mask_t  mask,
    OUT  it_pz_param_t       *params
);

typedef enum {
    IT_PZ_PARAM_ALL  = 0x01,
    IT_PZ_PARAM_IA   = 0x02
} it_pz_param_mask_t;

typedef struct {
    it_ia_handle_t ia;  /* IT_PZ_PARAM_IA */
} it_pz_param_t;
```

## DESCRIPTION

| | |
|---|---|
| *pz_handle* | Protection Zone. |
| *mask* | Bitwise OR of flags for desired parameters. |
| *params* | Structure whose members are written with the desired parameters. |

The *it_pz_query* routine returns the desired parameters of the Protection Zone *pz_handle* in the structure pointed to by params. On return, each field of *params* is only valid if the corresponding flag as shown in the Synopsis is set in the mask argument. The mask value IT_PZ_PARAM_ALL causes all fields to be returned.

The definition of each field of *params* follows:

| | |
|---|---|
| *ia* | The Interface Adapter Handle specified to create the Protection Zone. |

## RETURN VALUE

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

| | |
|---|---|
| IT_ERR_INVALID_PZ | The Protection Zone Handle (*pz_handle*) was invalid. |
| IT_ERR_INVALID_MASK | The *mask* contained invalid flag values. |
| 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. |

## SEE ALSO

*it_pz_create*(), *it_pz_free*()

# it_reject()

## NAME

it_reject – reject an incoming Connection Request or Connection Reply

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_reject(
    IN         it_cn_est_identifier_t  cn_est_id,
    IN  const  unsigned char          *private_data,
    IN         size_t                  private_data_length
);

typedef uint64_t it_cn_est_identifier_t;
```

## APPLICABILITY

*it_reject* is applicable only to the RC service type.

## DESCRIPTION

| | |
|---|---|
| *cn_est_id* | Connection Establishment Identifier associated with the Connection Request to be rejected. Calling *it_reject* destroys the identifier. See *it_ep_accept* for a definition of this data type. |
| *private_data* | Opaque Private Data to be sent in the IT_CM_MSG_CONN_PEER_REJECT_EVENT Event delivered to the Remote Consumer. If the IA does not support Private Data, *private_data_length* must be zero. |
| *private_data_length* | Length of *private_data.* This field must be 0 if the IA does not support Private Data. |

*it_reject* rejects an incoming Connection Request or Connection Reply. The Remote Endpoint will receive an IT_CM_MSG_CONN_PEER_REJECT_EVENT Event on its IT_CM_MSG_EVENT_STREAM Simple Event Dispatcher, and that Endpoint will transition into the IT_EP_STATE_NONOPERATIONAL state.

For two-way Connection establishment, *it_reject* can only be called on the Passive side in response to the IT_CM_REQ_CONN_REQUEST_EVENT Event.

For three-way Connection establishment, *it_reject* can be called on the Passive side in response to the IT_CM_REQ_CONN_REQUEST_EVENT, or on the Active side in response to the IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT. If *it_reject* is called on the active side, the local Endpoint associated with the Connection establishment transitions to the IT_EP_STATE_NONOPERATIONAL state. See the *it_ep_state_t* reference page for a description of this Endpoint state.

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.

The Connection Establishment Identifier, *cn_est_id*, is freed by *it_reject.*

## RETURN VALUE

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

| | |
|---|---|
| IT_ERR_INVALID_CN_EST_ID | The Connection Establishment Identifier (*cn_est_id*) was invalid. |
| 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 IAs capabilities to support Private Data. |
| IT_ERR_INVALID_PDATA_LENGTH | The Interface Adapter supports Private Data, but the length specified exceeded the Interface Adapter's capabilities. |
| 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. |

## APPLICATION USAGE

1. 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 in an invalid Connection Establishment Identifier is indeterminate.

2. Calling *it_reject* does not necessarily mean that 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 the remote does get a peer reject Event, the Private Data contained within that Event will be exactly what was furnished to *it_reject*.

## SEE ALSO

*it_ep_accept*(), *it_ep_connect*(), *it_cm_req_events*, *it_cm_msg_events*, *it_ep_state_t*, *it_handoff*(), *it_ia_query*()

# it_rmr_create()

## NAME

it_rmr_create – create a Remote Memory Region (RMR)

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_rmr_create(
    IN   it_pz_handle_t    pz_handle,
    IN   it_rmr_type_t     rmr_type,
    OUT  it_rmr_handle_t  *rmr_handle
);
```

## APPLICABILITY

Remote Memory Regions can be used only for the RC service type.

## DESCRIPTION

| | |
|---|---|
| *pz_handle* | Protection Zone in which the Remote Memory Region will be created. |
| *rmr_type* | Desired type for the Remote Memory Region. RMR types are defined in *it_rmr_type_t*. |
| *rmr_handle* | Handle of new Remote Memory Region. |

The *it_rmr_create* routine creates a Remote Memory Region within a Protection Zone identified by the *pz_handle* argument and associated with the Interface Adapter implicitly identified by *pz_handle*. RMRs can be accessed only remotely (for setting access privileges, see *it_rmr_link*). The restriction to remote access does not preclude a special case where a “local RMR” is used locally in a DTO – see the Extended Description for details.

The desired RMR type is specified through the *rmr_type* argument. The allowable RMR types are Implementation-dependent (see *it_rmr_type_t* for details).

If the Consumer specifies an *rmr_type* of IT_RMR_TYPE_DEFAULT, the Implementation will choose a type of RMR that is supported by the underlying IA, which may be either a Narrow or a Wide RMR. Requesting an RMR type that is not supported by the Implementation will cause *it_rmr_create* to fail.

An RMR cannot be used as a target for DTOs until it has been linked with *it_rmr_link*.

An unlinked Wide or Narrow RMR may be used as a target for Link operations via all Endpoints in the PZ of the RMR.

A linked Wide RMR may be used as a target for DTOs and Link/Unlink operations via all Endpoints in the PZ of the RMR. A linked Narrow RMR may be used as a target for DTOs and Unlink operations only via the Endpoint through which it was linked.

## EXTENDED DESCRIPTION

InfiniBand Implementations at least support Wide RMRs and, if Verb Extensions (BMM) are provided, also Narrow RMRs; iWARP Implementations at least support Narrow RMRs.

For iWARP only, RMRs can also be used locally, namely as data sinks in an *it_post_rdma_read_to_rmr* call; see the Extended Description of *it_post_rdma_read_to_rmr* for why iWARP nevertheless considers the writing to a local data sink of an RDMA Read DTO a remote write access.

**BACKWARDS COMPATIBILITY**

*it_rmr_create* (v2.0) called with *rmr_type* set to IT_RMR_TYPE_WIDE behaves identically as *it_rmr_create* (v1.0), which creates Wide RMRs only.

*it_rmr_create* (v2.0) called with *rmr_type* set to IT_RMR_TYPE_DEFAULT selects the appropriate RMR type for the given Implementation.

See also Appendix C.

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_INVALID_PZ | The Protection Zone Handle (*pz_handle*) was invalid. |
| IT_ERR_INVALID_RMR_TYPE | The RMR type (*rmr_type*) was invalid or not supported by the Implementation. |
| IT_ERR_RESOURCES | The requested operation failed due to insufficient resources. |
| 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. |

**APPLICATION USAGE**

The returned RMR must be linked to a Local Memory Region by means of *it_rmr_link* before it can be used in Data Transfer Operations. Remote use of the linked RMR also requires exposing the RMR to a remote Consumer through an associated RMR Context.

Creating an RMR is a relatively expensive operation. Once created, however, an RMR may be linked repeatedly to different LMR address ranges using the more efficient *it_rmr_link* call, as long as a linked RMR is first unlinked through *it_rmr_unlink*. Linking an RMR is much more efficient than changing remote access privileges using *it_lmr_modify*.

**SEE ALSO**

*it_rmr_link*(), *it_rmr_free*(), *it_rmr_query*()

# it_rmr_free()

## NAME

it_rmr_free – destroy a Remote Memory Region

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_rmr_free(
    IN  it_rmr_handle_t  rmr_handle
);
```

## APPLICABILITY

Remote Memory Regions can be used only for the RC service type. See *it_rmr_create*.

## DESCRIPTION

| | |
|---|---|
| *rmr_handle* | Handle of Remote Memory Region to be destroyed. |

The *it_rmr_free* routine destroys the Remote Memory Region *rmr_handle*. If the RMR is currently linked to an LMR, then the RMR linking is also destroyed. On return, the *rmr_handle* may no longer be used, and the associated RMR Context may no longer be used. RMRs with memory ranges that overlap the range of *rmr_handle* are not affected by its destruction.

Outstanding remote DTOs that use the RMR Context of this RMR may either complete successfully or fail with an access violation error.

## RETURN VALUE

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

| | |
|---|---|
| IT_ERR_INVALID_RMR | The Remote Memory Region Handle (*rmr_handle*) was invalid. |
| 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. |

## APPLICATION USAGE

Since the number of possible RMR Context values is finite, the Implementation will eventually re-use previously freed values in a new linking. If a DTO using an RMR Context is posted after that Context is freed, it is theoretically possible for the Context to be re-used before the DTO completes, and for the DTO to complete under the new linking for the Context, resulting in data corruption. To avoid this, the Consumer should not free an RMR which may be the target of outstanding DTOs. This may require coordination between local and remote Consumers, and such coordination is the Consumer's responsibility.

## SEE ALSO

*it_rmr_create*(), *it_rmr_query*()

# it_rmr_link()

## NAME

it_rmr_link – post operation to Link a Remote Memory Region to a memory range

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_rmr_link(
    IN   it_rmr_handle_t   rmr_handle,
    IN   it_lmr_handle_t   lmr_handle,
    IN   void              *addr,
    IN   it_length_t       length,
    IN   it_addr_mode_t    addr_mode,
    IN   it_mem_priv_t     privs,
    IN   it_ep_handle_t    ep_handle,
    IN   it_dto_cookie_t   cookie,
    IN   it_dto_flags_t    dto_flags,
    OUT  it_rmr_context_t  *rmr_context
);
```

## APPLICABILITY

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

## DESCRIPTION

| | |
|---|---|
| *rmr_handle* | Handle of RMR that will be linked. |
| *lmr_handle* | LMR to which RMR will be linked. |
| *addr* | Starting address of RMR to be linked. |
| *length* | Length in bytes of RMR to be linked. Must not be 0. |
| *addr_mode* | Addressing mode of RMR to be linked. |
| *privs* | Bitwise OR of remote access privilege flags for linked RMR, taken from *it_mem_priv_t*. |
| *ep_handle* | Endpoint on which to post the Link operation. |
| *cookie* | Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the RMR Link operation. |
| *dto_flags* | Bitwise OR of options for operation handling. |
| *rmr_context* | Returned Context allowing remote access to the linked RMR. |

The *it_rmr_link* routine posts to Endpoint *ep_handle* an operation to Link the Remote Memory Region *rmr_handle* to the segment of an LMR specified by the *lmr_handle*, *addr*, and *length* arguments. It returns a new *rmr_context* value in network byte order that can be transferred by the local Consumer to a remote Consumer to be used for an RDMA operation.

The arguments *addr* and *length* provide the starting address, also known as the Base Address, and the length in bytes, respectively, of the region to be linked. The starting address *addr* must be in the same linear address space as used by the underlying LMR given by *lmr_handle*. The specified address range must fall within the address range of the underlying LMR.

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

An RMR can be linked only if the Endpoint *ep_handle* is in IT_EP_STATE_CONNECTED state.

While any RMR is associated with a Protection Zone, successfully linking a Narrow RMR causes the RMR to be also associated with the Endpoint *ep_handle* to which the RMR Link operation is posted (see *it_rmr_type_t*).

A Narrow RMR can be linked if it is not currently in linked state, while a Wide RMR can be (re)linked regardless of whether it is currently linked.

The RMR handle must be valid. The LMR handle must be valid. For any RMR type, the Protection Zones of the RMR, underlying LMR, and Endpoint must match. For any RMR type and regardless of the *addr_mode* argument, the underlying LMR must use Absolute Addressing. An RMR of type IT_RMR_TYPE_NARROW must be in unlinked state to be linkable.

Remote access to an RMR is enforced with byte-level granularity.

RMRs can be accessed only remotely, and any local access privileges specified in the *privs* argument are ignored. The type of remote access to be allowed is specified by the *privs* argument as a bitwise-inclusive OR of one or more of the bit values IT_PRIV_REMOTE_READ and IT_PRIV_REMOTE_WRITE. Either or both of IT_PRIV_REMOTE_READ and IT_PRIV_REMOTE_WRITE must be included; see *it_mem_priv_t* for bit definitions and predefined bit combinations.

It is legal to request remote access rights that exceed the remote access rights of the underlying LMR.

The *cookie* (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work Request. This identifier is completely under Consumer control and opaque to the Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted Work Request.

Request handling is specified by the *dto_flags* argument and is the bitwise OR of zero or more of the following flags:

IT_COMPLETION_FLAG
IT_NOTIFY_FLAG
IT_BARRIER_FENCE_FLAG

For the definition of these flags, see *it_dto_flags_t*. In addition, *it_rmr_link* automatically fences all DTO, Link, and Unlink operations subsequently submitted on the Endpoint *ep_handle* such that none of these operations starts until the currently posted Link operation has completed.

The completion of the posted Link operation is reported asynchronously to the Consumer according to the rules defined in *it_dto_flags_t*. An RMR Link Completion Event is of type *it_dto_cmpl_event_t*. Any RMR Link Completion Event generated manifests on the EVD associated with the Endpoint Send Queue. The Event type is IT_RMR_LINK_CMPL_EVENT.

The value of *rmr_context* is immediately available when *it_rmr_link* returns, but it may not be used by a remote host for an RDMA operation until the Link Completion Event occurs. Violation of this rule may result in an error and a broken Connection for the reliable Connection Endpoint on which the RDMA operation is posted. See Application Usage for more details.

After a successful Link Completion Event, any previous linking for the RMR is invalidated. Successfully linking an RMR results in a new RMR Context being associated with the RMR and revokes remote access using any previous RMR Contexts, as long as the new RMR Context is not a re-use of a previously generated RMR Context. The revocation of remote access may affect operations that are outstanding when *it_rmr_link* is called. Consumers should make sure that such operations complete before calling *it_rmr_link*.

The new linking remains valid until the next Link or Unlink operation completes successfully, or until the RMR is destroyed. A Link operation will never be partially successful over a subset of the requested memory range; it either succeeds completely or fails without invalidating any portion of the previous linking.

If *it_rmr_link* returns successfully, but the Link Completion Event status indicates failure, then any previous linking and RMR Context remains valid.

Any Work Request posted to an Endpoint's Send Queue after a call to *it_rmr_link* will not begin execution until the Link operation has completed.

**EXTENDED DESCRIPTION**

For the InfiniBand Transport, specifying the IT_PRIV_REMOTE_WRITE access privilege for the RMR requires that the underlying LMR be enabled for local write access.

For the InfiniBand Transport, linking a Wide RMR with Relative Addressing is typically not supported (even if the IA attribute *addr_mode_relative_support* indicates support for Relative Addressing); in this case, an IT_ERR_INVALID_ADDR_MODE immediate error is returned.

**BACKWARDS COMPATIBILITY**

*it_rmr_link* called for a Wide RMR and with *addr_mode* set to IT_ADDR_MODE_ABSOLUTE behaves identically as *it_rmr_bind* (v1.0).

See also Appendix C.

**RETURN VALUE**

A successful call returns IT_SUCCESS. Otherwise, an immediate error is returned and, if the RMR was previously linked, the previous linking and RMR Context remains valid. It is possible for *it_rmr_link* to return success but for the Completion Event to indicate failure.

Posting to an Endpoint that is not in the IT_EP_STATE_CONNECTED or IT_EP_STATE_NONOPERATIONAL state will return the IT_ERR_INVALID_EP_STATE immediate error.

The possible immediate errors for *it_rmr_link* are listed below.

| | |
|---|---|
| IT_ERR_ADDRESS | The address (*addr*) fell outside the boundaries specified by the Local Memory Region. |
| IT_ERR_INVALID_ADDR_MODE | The addressing mode (*addr_mode*) was invalid, unsupported, or unsupported for the RMR type of the given RMR. |
| IT_ERR_INVALID_DTO_FLAGS | The Data Transfer Operation flags (*dto_flags*) value was invalid. |
| IT_ERR_INVALID_EP | The Endpoint Handle (*ep_handle*) was invalid. |
| IT_ERR_INVALID_EP_STATE | The Endpoint was not in the proper state for the attempted operation. |
| IT_ERR_INVALID_EP_TYPE | The attempted operation was invalid for the Service Type of the Endpoint. |
| IT_ERR_INVALID_LENGTH | The value of *length* fell outside the boundaries of the Local Memory Region or the value of *length* was 0. |
| IT_ERR_INVALID_LMR | The Local Memory Region Handle (*lmr_handle*) was invalid. |
| IT_ERR_INVALID_PRIVS | The requested memory privileges (*privs*) contained an invalid flag. |
| IT_ERR_INVALID_RMR | The Remote Memory Region Handle (*rmr_handle*) was invalid. |
| IT_ERR_TOO_MANY_POSTS | The operation failed due to an overflow of a Work Queue. |
| 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. |

**ASYNCHRONOUS ERRORS**

For Work Requests posted to an Endpoint in the IT_EP_STATE_CONNECTED state, a completion status (see *it_dto_status_t*) other than IT_DTO_SUCCESS will break the Connection by moving the Endpoint to the IT_EP_STATE_NONOPERATIONAL state and deliver an IT_CM_MSG_CONN_BROKEN_EVENT Event to the Connect EVD of *ep_handle*. Once the Connection is broken, all outstanding and in-progress operations on the Connection will complete with an error status.

Any posting to an Endpoint that is in the IT_EP_STATE_NONOPERATIONAL state will be flushed with completion status set to IT_DTO_ERR_FLUSHED.

If the RMR address range specified with *addr* and *length* exceeds the address range of the underlying LMR, a Completion Error is generated with completion status set to IT_RMR_OPERATION_FAILED.

An IT_RMR_OPERATION_FAILED completion status also results in case of an invalid RMR handle or invalid LMR handle, unless the Implementation handles these errors as immediate errors.

An IT_RMR_OPERATION_FAILED completion status also results if the Protection Zones of the RMR, underlying LMR, and Endpoint do not match, if the underlying LMR does not use Absolute Addressing, or if the RMR is a Narrow RMR and not in unlinked state.

For InfiniBand, it is invalid to request remote write access if the memory access privileges of the underlying LMR do not include local write access. Such an attempt also causes an IT_RMR_OPERATION_FAILED completion status.

Binding a Narrow or Wide RMR with Remote Read permission when the LMR permissions do not include Local Read results in a Completion Error with the IT_RMR_OPERATION_FAILED completion status.

Linking an RMR to an LMR that uses Relative Addressing is invalid and results in a Completion Error with the IT_DTO_ERR_LOCAL_MM_OPERATION completion status.

For the InfiniBand Transport, linking a Narrow RMR with a length of zero is invalid and results in a Completion Error with the IT_DTO_ERR_LOCAL_MM_OPERATION completion status.

**APPLICATION USAGE**

The *it_rmr_link* operation is lightweight compared to creating an RMR or an LMR. An application concerned with efficiency would typically create one or more RMRs at initialization time that could be linked multiple times to enable remote access to different peers as needed.

The Consumer should use unique identifiers for *cookie* if they desire to identify each DTO. If the Consumer does not require a unique DTO identifier, the value of zero or NULL (IT_NO_ADDR) can be used.

The local Consumer has several options for ensuring that the remote Consumer does not use *rmr_ context* before the Link Completion Event occurs. One is to wait for the Completion Event on the Send EVD of the specified Endpoint *ep_handle* before sending the *rmr_context* to a peer. Another option is to send the *rmr_context* to a peer by posting a DTO to the same Endpoint *ep_handle* that was used to Link the RMR. The barrier-fencing behavior of *it_rmr_link* ensures that the DTO does not start until the Link Completion Event has occurred. If the Link fails with a Completion Error, the Connection will be broken and the DTO flushed, so the *rmr_context* will not be sent.

For reasons already described, the completion of an RMR Link operation represents an important change that Consumers may need to monitor. One way to do this is to set IT_COMPLETION_FLAG in *dto_flags*, which will generate a Completion Event to indicate when the RMR has been linked. Consumers who do not set IT_COMPLETION_FLAG must rely on ordering semantics to infer when the RMR has been successfully linked. For example, if a subsequent DTO posted to the Send Queue of the same EP completes successfully, then the Link operation has completed, because DTOs posted to the Send Queue of an EP must wait for a Link operation to complete before processing. If the Link operation fails, a Link Completion Event is generated regardless of the use of IT_COMPLETION_FLAG.

On the InfiniBand Transport, it is possible to prevent further accesses to a (linked) RMR by linking it with *privs* set to 0, but it is recommended and preferable to use *it_rmr_unlink* instead.

When an EVD holding the DTO Event Stream associated with an Endpoint overflows, on some transports the behavior of the Endpoint and/or the EVD is undefined, making recovery from such an overflow either impossible or at the very least IA-dependent. Consumers should therefore take care to avoid this situation. Each Work Request that a Consumer posts to a Work Queue can result in a Completion Event being enqueued in an EVD, due to the possibility of an

error occurring during Work Request processing. To avoid any possibility of overflow, the Consumer must therefore ensure that the total number of Outstanding Operations on all Work Queues that can enqueue Completion Events into a given EVD never exceeds the minimum queue size (as reported by *it_evd_query*) of that EVD.

**FUTURE DIRECTIONS**

Calling *it_rmr_link* on a Wide RMR that is already in the linked state is currently allowed, while the same operation is not allowed for a linked Narrow RMR. A future version of the IT-API may require the Consumer to unlink any linked RMR using *it_rmr_unlink* before linking it, regardless of the transport.

**SEE ALSO**

*it_lmr_create*(), *it_rmr_create*(), *it_rmr_unlink*(), *it_dto_flags_t*, *it_dto_events*, *it_addr_mode_t*, *it_mem_priv_t*

# it_rmr_query()

## NAME

it_rmr_query – get attributes of a Remote Memory Region

## SYNOPSIS

```
#include <it_api.h>

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
);

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;

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;
```

## APPLICABILITY

Remote Memory Regions can be used only for the RC service type. See *it_rmr_create*.

## DESCRIPTION

| | |
|---|---|
| *rmr_handle* | Remote Memory Region. |
| *mask* | Bitwise OR of flags for desired parameters. |
| *params* | Structure whose members are written with the desired parameters. |

The *it_rmr_query* routine returns the desired attributes of the Remote Memory Region *rmr_handle* in the structure pointed to by *params*. The *mask* argument specifies which fields of

*params* are returned, and the values returned in other fields are undefined. See the Synopsis for the correspondence between *mask* values and fields. The *mask* value IT_RMR_PARAM_ALL causes all fields to be returned.

The definition of each field of *params* follows, some of which depend on whether the RMR is currently linked to an LMR as a result of using *it_rmr_link*:

| | |
|---|---|
| *ia* | The Interface Adapter Handle specified to create the RMR. |
| *pz* | The Protection Zone Handle specified to create the RMR. |
| *linked* | IT_TRUE if the RMR is currently linked; otherwise, IT_FALSE. |
| *lmr* | The Local Memory Region to which the RMR is currently linked. Undefined if RMR is not linked. |
| *addr* | The currently linked starting address or, equivalently, Base Address of the RMR. To be interpreted as an absolute address, regardless of *addr_mode*. Undefined if RMR is not linked. |
| *length* | The currently linked length in bytes of the RMR. Undefined if RMR is not linked. |
| *privs* | The currently linked memory access privileges of the RMR. Undefined if RMR is not linked. For the definition of bits, see *it_mem_priv_t*. |
| *rmr_context* | The currently linked RMR Context associated with the RMR. Undefined if RMR is not linked. Returned in network byte order. |
| *type* | The type of the RMR, either IT_RMR_TYPE_NARROW or IT_RMR_TYPE_WIDE. |
| *addr_mode* | The addressing mode of the RMR. Undefined if RMR is not linked. |

If the Consumer calls *it_rmr_query* after posting an RMR Link or RMR Unlink operation, and before dequeueing the Completion Event of such an operation, then the returned *linked*, *lmr*, *addr*, *length*, *privs*, *rmr_context*, and *addr_mode* fields may represent the RMR state as it was prior to posting, or a new RMR state. The Consumer should not rely on the value of these fields during this time.

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_INVALID_MASK | The *mask* contained invalid flag values. |
| IT_ERR_INVALID_RMR | The Remote Memory Region Handle (*rmr_handle*) was invalid. |
| 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. |

**SEE ALSO**

*it_rmr_create*(), *it_rmr_free*(), *it_rmr_link*(), *it_rmr_context_t*, *it_addr_mode_t*, *it_mem_priv_t*, *it_rmr_type_t*

# it_rmr_unlink()

## NAME

it_rmr_unlink – post operation to Unlink a Remote Memory Region from its memory range

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_rmr_unlink(
    IN  it_rmr_handle_t  rmr_handle,
    IN  it_ep_handle_t   ep_handle,
    IN  it_dto_cookie_t  cookie,
    IN  it_dto_flags_t   dto_flags
);
```

## APPLICABILITY

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

## DESCRIPTION

| | |
|---|---|
| *rmr_handle* | Handle of RMR that will be unlinked. |
| *ep_handle* | Endpoint on which to post the operation. |
| *cookie* | Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the operation. |
| *dto_flags* | Bitwise OR of options for operation handling. |

The *it_rmr_unlink* routine posts to Endpoint *ep_handle* an Unlink operation to Unlink the Remote Memory Region specified by *rmr_handle*.

An RMR can be unlinked only if the Endpoint *ep_handle* is in IT_EP_STATE_CONNECTED state.

For any RMR type, the Protection Zones of the RMR and Endpoint must match. For a linked Narrow RMR, the Endpoint *ep_handle* to which the Unlink operation is posted must match the Endpoint through which the RMR was linked (see *it_rmr_type_t*).

An RMR can be unlinked regardless of whether it is currently linked or unlinked.

The *cookie* (*it_dto_cookie_t*) allows the Consumer to associate an identifier with each Work Request. This identifier is completely under Consumer control and opaque to the Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted Work Request.

Request handling is specified by the *dto_flags* argument as the bitwise OR of zero or more of the following flags:

IT_COMPLETION_FLAG
IT_NOTIFY_FLAG
IT_BARRIER_FENCE_FLAG
IT_UNLINK_FENCE_FLAG

For the definition of these flags, see *it_dto_flags_t*.

The completion of the posted Unlink operation is reported asynchronously to the Consumer according to the rules defined in *it_dto_flags_t*. An RMR Unlink Completion Event is of type *it_dto_cmpl_event_t*. Any generated RMR Unlink Completion Event manifests on the EVD associated with the Endpoint Send Queue. The Event type is IT_RMR_LINK_CMPL_EVENT.

After a successful Unlink Completion Event, any previous linking for the RMR is invalidated, and the RMR Context for the RMR is no longer defined. Any RDMA operation that uses the previous RMR Context will fail with a protection violation; beware that this may include operations that are outstanding when *it_rmr_unlink* is called. The Consumer must ensure that such operations have completed prior to calling *it_rmr_unlink* if successful completions are desired. An Unlink operation will never be partially successful over a subset of the requested memory range; it either succeeds completely or fails without invalidating any portion of the previous linking.

If *it_rmr_unlink* returns successfully but the Completion Event status indicates failure, then any previous linking and RMR Context remains valid.

Any Work Request posted to an Endpoint's Send Queue after a call to *it_rmr_unlink* will not begin execution until the Unlink operation has completed.

**RETURN VALUE**

A successful call returns IT_SUCCESS. Otherwise, an immediate error is returned and, if the RMR was previously linked the previous linking of the RMR remains valid. It is possible for *it_rmr_unlink* to return success but for the Completion Event to indicate failure.

Posting to an Endpoint that is not in the IT_EP_STATE_CONNECTED or IT_EP_STATE_NONOPERATIONAL state will return the IT_ERR_INVALID_EP_STATE immediate error.

The possible immediate errors for *it_rmr_unlink* are listed below:

| | |
|---|---|
| IT_ERR_INVALID_DTO_FLAGS | The Data Transfer Operation flags (*dto_flags*) value was invalid. |
| IT_ERR_INVALID_EP | The Endpoint Handle (*ep_handle*) was invalid. |
| IT_ERR_INVALID_EP_STATE | The Endpoint was not in the proper state for the attempted operation. |
| IT_ERR_INVALID_EP_TYPE | The attempted operation was invalid for the Service Type of the Endpoint. |
| IT_ERR_INVALID_RMR | The Remote Memory Region Handle (*rmr_handle*) was invalid. |
| IT_ERR_TOO_MANY_POSTS | The operation failed due to an overflow of a Work Queue. |
| 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. |

## ASYNCHRONOUS ERRORS

For Work Requests posted to an Endpoint in the IT_EP_STATE_CONNECTED state, a completion status (see *it_dto_status_t*) other than IT_DTO_SUCCESS will break the Connection by moving the Endpoint to the IT_EP_STATE_NONOPERATIONAL state and deliver an IT_CM_MSG_CONN_BROKEN_EVENT Event to the Connect EVD of *ep_handle*. Once the Connection is broken, all outstanding and in-progress operations on the Connection will complete with an error status.

Any posting to an Endpoint that is in the IT_EP_STATE_NONOPERATIONAL state will be flushed with completion status set to IT_DTO_ERR_FLUSHED.

An IT_RMR_OPERATION_FAILED completion status results in case of an invalid RMR handle, unless the Implementation handles this error as an immediate error.

An IT_RMR_OPERATION_FAILED completion status also results if the Protection Zones of the RMR and the Endpoint do not match or, for a linked Narrow RMR, if the Endpoint to which the Unlink operation is posted does not match the Endpoint through which the RMR was linked.

## APPLICATION USAGE

*it_rmr_unlink* may be used to revoke remote Consumer access to an RMR that was previously granted. In addition, the Consumer must Unlink all RMRs that refer to an LMR in order to destroy or modify the LMR. Note that the RMR is not considered unlinked until a successful Completion Event is generated; thus, the Consumer should dequeue the Completion Event before calling *it_lmr_free*.

A difficulty can arise if the Endpoint that the Consumer was using to Link the RMR has become disconnected, because an Unlink operation can only be posted to a connected Endpoint. The preferred solution is to destroy the RMR by calling *it_rmr_free*. In case of a Wide RMR and for the InfiniBand Transport only, an alternative is for the Consumer to create a special pair of Endpoints within the same Protection Zone as the Wide RMR and to connect these Endpoints in loopback mode to each other; the Wide RMR can then be unlinked through one of the special Endpoints.

For reasons already described, the completion of an RMR Unlink operation represents an important change that Consumers may need to monitor. One way to do this is to set IT_COMPLETION_FLAG in *dto_flags*, which will generate a Completion Event to indicate when the RMR has been unlinked. Consumers who do not set IT_COMPLETION_FLAG must rely on ordering semantics to infer when the RMR has been successfully unlinked. For example, if a subsequent DTO posted to the Send Queue of the same EP completes successfully, then the Link operation has completed, because DTOs posted to the Send Queue of an EP must wait for a Link operation to complete before processing. If the Unlink operation fails, a Completion Event is generated regardless of the use of IT_COMPLETION_FLAG.

When an EVD holding the DTO Event Stream associated with an Endpoint overflows, on some transports the behavior of the Endpoint and/or the EVD is undefined, making recovery from such an overflow either impossible or at the very least IA-dependent. Consumers should therefore take care to avoid this situation. Each Work Request that a Consumer posts to a Work Queue can result in a Completion Event being enqueued in an EVD, due to the possibility of an error occurring during Work Request processing. To avoid any possibility of overflow, the Consumer must therefore ensure that the total number of Outstanding Operations on all Work Queues that can enqueue Completion Events into a given EVD never exceeds the minimum queue size (as reported by *it_evd_query*) of that EVD.

**SEE ALSO**

*it_rmr_create*(), *it_rmr_link*(), *it_dto_flags_t*

# it_set_consumer_context()

## NAME

it_set_consumer_context – associate a Consumer Context with an IT Object Handle

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_set_consumer_context(
    IN  it_handle_t   handle,
    IN  it_context_t  context
);
```

## DESCRIPTION

*handle* — Handle for the IT-API object to be associated with the Consumer Context.

*context* — The Consumer Context to be associated with the object Handle.

*it_set_consumer_context* associates a Consumer Context with the specified *handle*. See *it_handle_t* for a description of the valid Handle types.

Only a single Consumer Context is provided for any IT Object Handle. If there is a previous Consumer Context associated with the specified Handle, the new Context replaces the old one. The value of Context is opaque to the Implementation. The Consumer can disassociate the existing Context by providing a NULL (IT_NO_ADDR) value for the Context. The Implementation makes no attempt to synchronize access to the Context.

## RETURN VALUE

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

IT_ERR_INVALID_HANDLE — The *handle* was invalid.

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.

## EXAMPLES

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

```
it_lmr_handle_t lmr;
it_context_t cxt = 1234;
it_set_consumer_context( (it_handle_t) lmr, cxt);
```

## SEE ALSO

*it_get_consumer_context*(), *it_context_t*, *it_handle_t*

# it_socket_convert()

## NAME

it_socket_convert – convert a connected socket into a connected IT-API Endpoint

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_socket_convert(
    IN         int             sd,
    IN         it_ep_handle_t  ep_handle,
    IN         it_sc_flags_t   flags,
    IN  const  unsigned char*  msg,
    IN         size_t          len
);

typedef enum {
    IT_SC_DEFAULT     = 0x0000,
    IT_SC_NO_REQ_REP  = 0x0001,
} it_sc_flags_t;
```

## APPLICABILITY

it_socket_convert is supported only if the Interface Adapter attribute *socket_conversion_support* (see *it_ia_info_t*) is IT_TRUE.

## DESCRIPTION

*sd* Socket descriptor.

*ep_handle* Handle for an instance of the local Endpoint.

*flags* Flags for the Conversion attempt.

| 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 |

*msg* Last ULP streaming mode message for delivery to the remote responding Consumer. Must be NULL (IT_NO_ADDR) when the Consumer is in the Conversion Responder role.

*len* Length of *msg*. This field must be zero when the Consumer is in the Conversion Responder role.

The *it_socket_convert* routine converts a connected TCP/IP socket (*sd*) into a connected IT-API Endpoint (*ep_handle*).

Calling *it_socket_convert* with an *sd* parameter not corresponding to a TCP connection or with an *ep_handle* corresponding to an Endpoint created on an IA where *socket_conversion_support* is IT_FALSE shall yield the return code IT_ERR_OP_NOT_SUPPORTED.

This routine is for use by the Consumer both when initiating and when responding to a Conversion attempt.

The Consumer must provide a connected socket, *sd*, to the *it_socket_convert* routine. Prior to calling the *it_socket_convert* routine, the Consumer must quiesce all traffic in both directions on the connection. How the Consumer quiesces the connection is outside the scope of this specification. Failure to quiesce the connection prior to calling this routine may result in breaking the TCP/IP connection.

The connection parameter, *sd*, is a socket descriptor as returned by the *socket*() API routine.

After this call returns with IT_SUCCESS and before the Conversion process completes, the *sd* must not be used by the Consumer. Use of the *sd* prior to the Conversion process completing may abort the Conversion process and cause the generation of a Completion Error. After the Conversion process has completed, the Consumer can only legally use the *sd* in two other system calls: they can pass it to the *setsockopt*(2) system call to modify only the SO_KEEPALIVE option, or they pass it to *close*(2) to close it. Support for doing anything else with the *sd* is Implementation-specific. Support for modifying the SO_KEEPALIVE option on a converted socket is optional; if the Implementation doesn't support this, *setsockopt*(2) will return an error when the Consumer attempts to do so, but no undefined behavior will result. If the Consumer attempts to pass *sd* to any other system call besides the two previous specified, or attempts to use *setsockopt*(2) to modify any option other than SO_KEEPALIVE, undefined behavior up to and including unreported data corruption may result. Calling *close*(2) on the *sd* has no effect upon the Endpoint.

The Consumer must provide an Endpoint, *ep_handle*, of the Reliable Connected service type to the *it_socket_convert* routine. The Endpoint must be in the IT_EP_STATE_UNCONNECTED state when input to the routine.

The *flags* parameter for this routine allows the Consumer to control use of underlying features of the iWARP Transport. Unless stated otherwise, the bitwise OR of any combination of *flags* values is allowed.

The *flags* value IT_SC_DEFAULT causes the underlying implementation to enable MPA startup.

Setting the bit IT_SC_NO_REQ_REP in *flags* allows the Consumer to specify that the underlying Implementation not use the IETF MPA startup. See [MPA-RDMAC] for more details.

To initiate the Conversion process, the Consumer must provide a Last ULP Streaming-Mode Message buffer *msg* of length *len* greater than zero to the *it_socket_convert* routine. The Last ULP Streaming-Mode Message will be conveyed to the remote Consumer via the underlying connection. The Consumer is required to supply a Last ULP Streaming-Mode Message to initiate the Conversion process regardless of whether or not the IETF MPA Startup is used.

Prior to initiating the Conversion process by calling *it_socket_convert* with a non-NULL (i.e., not equal to IT_NO_ADDR) *msg* buffer, the Consumer should post at least one Receive DTO to the Endpoint *ep_handle* passed to the *it_socket_convert* routine. Failure to do so may result in the Endpoint failing to reach the IT_EP_STATE_CONNECTED state.

To respond to an incoming Conversion attempt by a remote Consumer, the Consumer must specify the *msg* parameter as NULL (IT_NO_ADDR) and specify the length, *len*, parameter as zero. The Consumer should not call the *it_socket_convert* routine in responding mode prior to consuming the Last ULP Streaming-Mode Message sent by the remote initiating Consumer (e.g., the local Consumer should *read*() from the *sd* to consume the remote Consumer's Last ULP Streaming-Mode Message). The Consumer should receive the peer's Last ULP Streaming-Mode Message completely so that the *sd* is quiesced prior to calling *it_socket_convert*. How the Consumer determines the size of the peer's Last ULP Streaming-Mode Message on the responding side is outside of the scope of this specification. Failure to consume the peer's Last ULP Streaming-Mode Message completely and thereby to quiesce *sd* before calling *it_socket_convert* may result in the TCP/IP connection breaking. The Consumer also must not attempt to send any further streaming mode messages on the *sd* prior to calling *it_socket_convert* as the Conversion Responder.

How a ULP chooses which side of the connection initiates Conversion and which side responds to Conversion is outside of the scope of this specification.

In order to safely post RDMA Read operations to an Endpoint, the Consumer must ensure that the Endpoint at the other end of the converted Connection has compatible *rdma_read_ord* (ORD) and *rdma_read_ird* (IRD) attributes. See *it_ep_attributes_t* for a description of these attributes. See Chapter 5 for more details on use of IRD and ORD.

The *it_socket_convert* routine returns control to the Consumer prior to completion of the Conversion operation. The completion of the Conversion operation is reported asynchronously via the *connect_sevd_handle* parameter furnished to the *it_ep_rc_create* routine on Endpoint creation. Successful completion of the Conversion process is indicated by an IT_CM_MSG_CONN_ESTABLISHED_EVENT queued on the IT_CM_MSG_EVENT_ STREAM SEVD represented by *connect_sevd_handle*. The Consumer may distinguish the Event corresponding to their Conversion attempt by comparing the *it_ep_handle_t* returned in the event with that of the Endpoint, *ep_handle*, passed into the *it_socket_convert* routine.

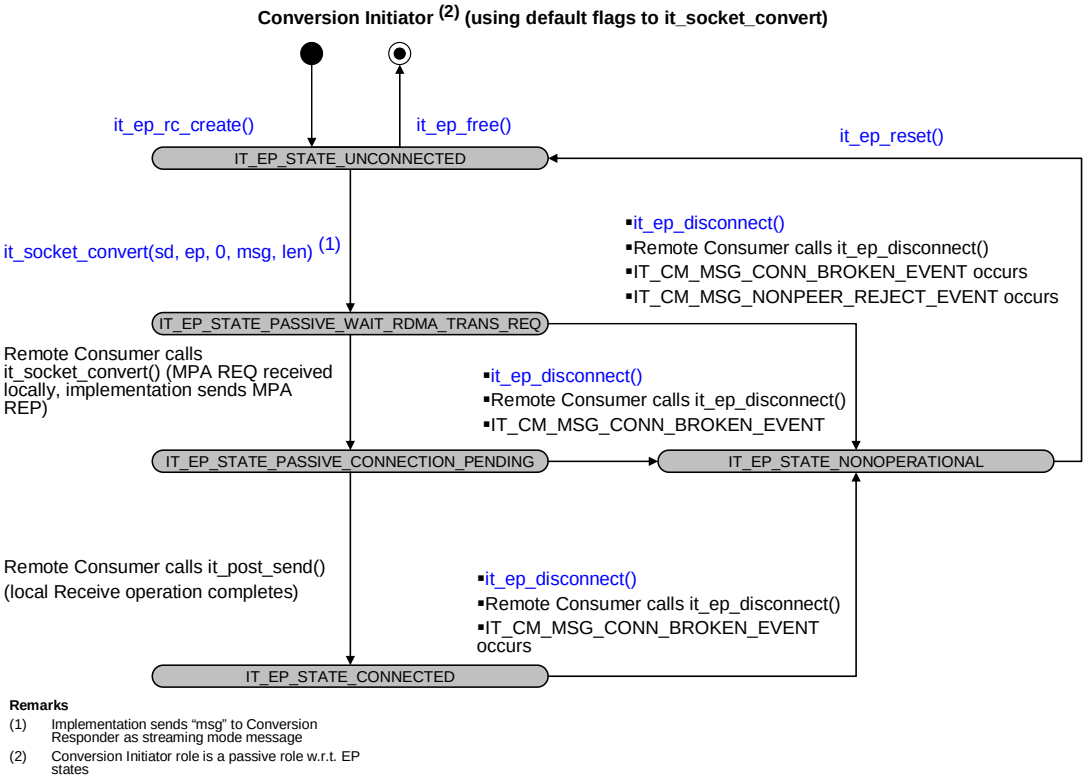
For a complete definition of Endpoint state and a more complete description of the state transitions, see the Extended Description below. If for any reason an Endpoint Connection fails to be established the Endpoint will transition into the IT_EP_STATE_NONOPERATIONAL state and any Receive DTO operations that were successfully posted to the Endpoint will be completed with an IT_DTO_ERR_FLUSHED status.

The Conversion process may not be completed for the Conversion Initiating Consumer unless the Conversion Responding Consumer posts a Send message via *it_post_send* to the remote Endpoint that is connected to, *ep_handle*. The Consumer may abort a Conversion attempt in progress by making use of the *it_ep_disconnect* routine or the *it_ep_free* routine.

**EXTENDED DESCRIPTION**

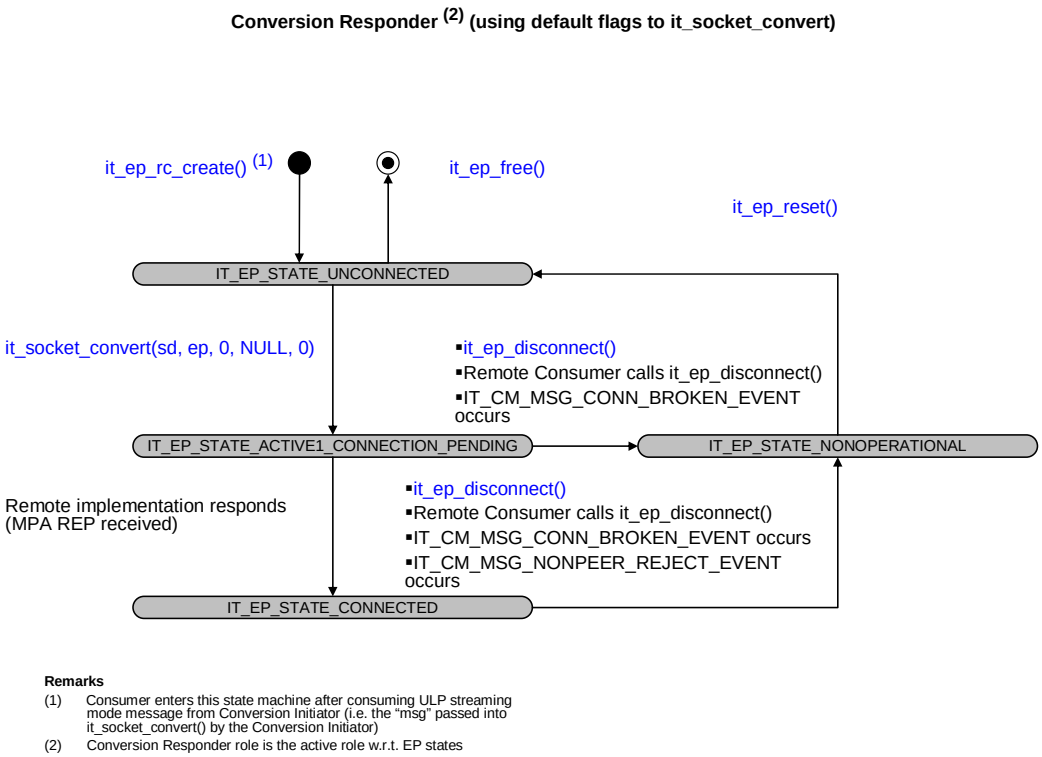
The Endpoint state transitions due to the Conversion process depend on the use of the *flags* parameter.

Setting *flags* to the value IT_SC_DEFAULT yields the following state diagrams for the initiating and responding sides:




8130

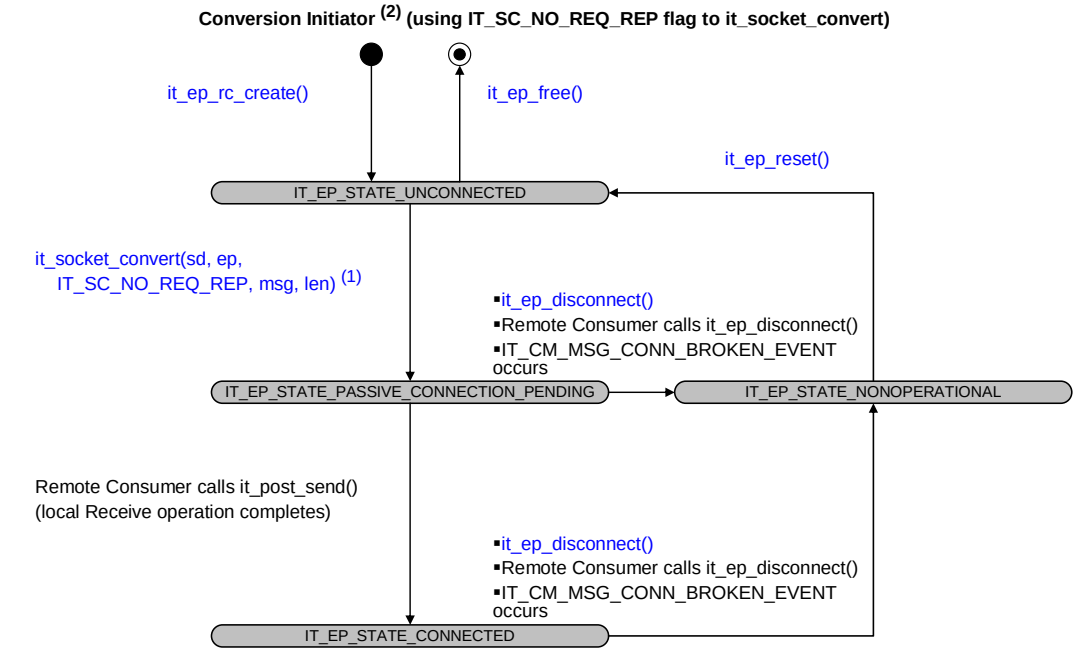
8131 **Figure 4: Conversion Initiator with *flags* set to IT_SC_DEFAULT**




8132

8133 **Figure 5: Conversion Responder with *flags* set to IT_SC_DEFAULT**

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

**Figure 6: Conversion Initiator with the IT_SC_NO_REQ_REP bit set in *flags***

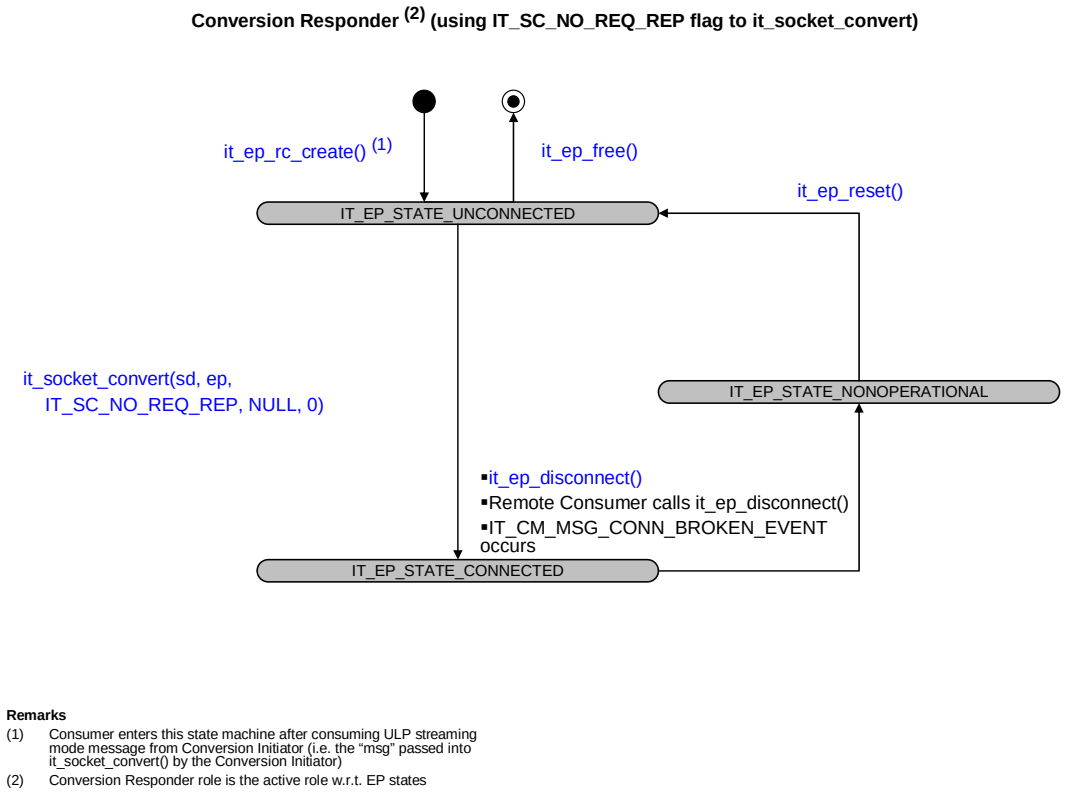



**Figure 7: Conversion Responder with the IT_SC_NO_REQ_REP bit set in *flags***

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.

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.

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*().

## RETURN VALUE

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

| | |
|---|---|
| IT_ERR_OP_NOT_SUPPORTED | The operation failed as it is not supported on the IA. |
| IT_ERR_RESOURCES | The operation failed due to resource limitations. |
| IT_ERR_INVALID_SD | The socket descriptor *sd* was invalid. |

| | |
|---|---|
| IT_ERR_INVALID_SD_STATE | The socket descriptor *sd* is not in the proper state to be converted. |
| IT_ERR_INVALID_EP | The *ep_handle* was invalid. |
| IT_ERR_INVALID_EP_STATE | The Endpoint is not in the proper state to be connected. |
| IT_ERR_INVALID_EP_TYPE | The Endpoint Service Type does not support this operation. |
| IT_ERR_INVALID_ FLAGS | The *flags* values was invalid. |
| 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. |

**ASYNCHRONOUS ERRORS**

If the Conversion Initiator uses a Non-permissive AV-RNIC/IETF and the Conversion Responder uses an RDMAC AV-RNIC, then an IT_CM_MSG_NONPEER_REJECT_EVENT with the IT_CN_REJ_BAD_CONN_PARMS reject reason code indicating incompatible protocol versions will be generated at both Conversion Initiator and Conversion Responder. See [INTEROP-IETF].

Conversely, if the Conversion Initiator uses an RDMAC AV-RNIC and the Conversion Responder uses a Non-permissive AV-RNIC/IETF, then an IT_CM_MSG_CONN_BROKEN_ EVENT will be generated at the Conversion Initiator and an IT_CM_MSG_NONPEER_ REJECT_EVENT with an IT_CN_REJ_BAD_CONN_PARMS reject reason code indicating incompatible protocol versions will be generated at the Conversion Responder. See [INTEROP-IETF].

**APPLICATION USAGE**

The *it_socket_convert* call may be used for implementing iSER and SDP ULPs.

It is the Consumer's responsibility to monitor forward progress of the Conversion process. The *it_socket_convert* call will never time out waiting for a reply. The *it_socket_convert* call may abort if the LLP closes due to an error. Consumers are advised not to perform any operations on *sd* while the Conversion of *sd* is in progress. Consumers may *close*(2) the *sd* after Conversion, which indicates only that resources associated with *sd* are no longer needed and has no effect on the Endpoint provided to *it_socket_convert*.

The iWARP transport currently does not provide a standardized, Endpoint-based interface for manipulating LLP options after Socket Conversion, such as an iWARP equivalent to the *setsockopt*(2) system call. Some iWARP Interface Adapters may allow setting a limited set of LLP options (such as the TCP SO_KEEPALIVE socket option) after a completed Conversion by *accepting setsockopt*(2) on the *sd* that was provided to *it_socket_convert*. Consumers having a need to set LLP options are therefore advised to either use *setsockopt*(2) prior to Conversion or to be prepared for handling failures of *setsockopt*(2) calls if invoked after a completed Conversion.

**SEE ALSO**

*it_ep_rc_create*(), *it_ep_modify*(), *it_ep_free*(), *it_ep_disconnect*(), *it_ia_info_t*, *it_post_send*(), *it_post_recv*(), *it_post_rdma_write*(), *it_post_rdma_read*()

# it_srq_create()

## NAME

it_srq_create – create a Shared Receive Queue (S-RQ) on an Interface Adapter

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_srq_create(
    IN   it_pz_handle_t    pz_handle,
    IN   size_t            max_recv_segments,
    IN   size_t            max_recv_dtos,
    OUT  it_srq_handle_t  *srq_handle
);
```

## APPLICABILITY

*it_srq_create* is applicable only to Endpoints created for the RC service type and is supported only if the Interface Adapter attribute *srq_support* (see *it_ia_info_t*) is IT_TRUE. Endpoints of the UD service type cannot use an S-RQ.

## DESCRIPTION

| | |
|---|---|
| *pz_handle* | Protection Zone in which to create the Shared Receive Queue. |
| *max_recv_segments* | Maximum number of data segments for a local buffer that the Consumer may specify for a posted Receive DTO. |
| *max_recv_dtos* | Maximum number of outstanding Receive DTOs that the Consumer can have outstanding on the S-RQ. |
| *srq_handle* | Returned Handle for the Shared Receive Queue. |

*it_srq_create* creates a Shared Receive Queue on the Interface Adapter implicitly identified by the input *pz_handle*. The *srq_handle* output is only valid if the call to *it_srq_create* returns IT_SUCCESS. An S-RQ is an IT Object to which Receive DTOs can be posted using the *it_post_recv* routine. By default when a Reliable Connection Endpoint is created (via the *it_ep_rc_create* routine) it has its own private Receive Queue, but the Consumer can override the default and instead specify that the Endpoint should use a Shared Receive Queue that the Consumer created using the *it_srq_create* routine.

The Protection Zone *pz_handle* allows the Consumer to control what local memory an incoming Send operation can access. An incoming Send DTO that is paired with a Receive DTO posted to an S-RQ can only access memory that is in the same Protection Zone as the S-RQ. In contrast, an incoming RDMA Read or RDMA Write DTO targeted at an Endpoint that is using an S-RQ can only access memory that is in the same Protection Zone as the Endpoint. The Protection Zone of the Endpoint and the Protection Zone of the S-RQ that the Endpoint is using are allowed to be different. See the Application Usage section below for an example of how this could be used.

When an Endpoint is using an S-RQ and an incoming Send operation targets that Endpoint, the Implementation shall attempt to dequeue a Receive DTO from the S-RQ and provide it to the targeted Endpoint for further processing. The semantics associated with the processing of the

Receive operation are described in the *it_post_recv* routine. If an incoming Send must be processed and no Receive DTO is available to be dequeued in the S-RQ, or if an error occurs while processing the Receive DTO (e.g., the Send DTO it gets paired with contains more data than the Receive DTO can handle) the Connection of the targeted Endpoint shall be broken and the Endpoint moved to the IT_EP_STATE_NONOPERATIONAL state, but other Endpoints that are using the same S-RQ shall not be affected. Endpoints that use a common S-RQ maintain the same isolation from errors on other Endpoints that they would have if they weren't using an S-RQ.

When an S-RQ is first created, the S-RQ Low Watermark mechanism is disabled. It can only be enabled by calling *it_srq_modify* to explicitly enable it.

If the call to *it_srq_create* returns IT_SUCCESS the total number of segments allowed in a Receive operation posted to the S-RQ will be greater than or equal to the *max_recv_segments* that were requested. To find out exactly how many segments will be allowed, use *it_srq_query*. Similarly, if the call to *it_srq_create* returns IT_SUCCESS the total number of Receive DTOs that can be outstanding for the S-RQ will be greater than or equal to the *max_recv_dtos* that were requested. To find out exactly how many Receive operations can safely be outstanding, use *it_srq_query*. (If the Consumer attempts to have more Receive DTOs outstanding on an S-RQ than is given by *it_srq_query*, the Implementation may refuse to allow more Receive DTOs to be posted; see *it_post_recv* for details.)

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_INVALID_PZ | The Protection Zone Handle (*pz_handle*) was invalid. |
| IT_ERR_SRQ_NOT_SUPPORTED | The IA associated with *pz_handle* does not support Shared Receive Queues. |
| IT_ERR_RESOURCES | The requested operation failed due to insufficient resources. |
| IT_ERR_RESOURCE_RECV_DTO | The underlying transport could not allocated the requested *max_recv_dtos* resources at this time. |
| IT_ERR_RESOURCE_RSEG | The underlying transport could not allocated the requested *max_recv_segments* resources at this time. |
| 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. |

**APPLICATION USAGE**

If a Consumer wishes to use an S-RQ they must first create the S-RQ by calling *it_srq_create*. They can then furnish the handle for the S-RQ as one of the attributes of the Endpoint when creating the Endpoint using the *it_ep_rc_create* routine.

Before creating a connection with an Endpoint that is using an S-RQ the Consumer will typically ensure that the Endpoint that it is connecting will be able to process at least one incoming Send operation by using the *it_post_recv* routine to post a Receive DTO to the S-RQ.

One example of a situation where it could be useful to allow the Protection Zone associated with an S-RQ and the Protection Zone associated with an Endpoint that is using the S-RQ to be different is when multiple clients are performing RDMA Write operations to the same server and the server wants to prevent an RDMA Write operation from one client from accessing the RDMA target buffer associated with a different client. Assume that we have two clients C1 and C2, and that the server processes incoming DTOs from C1 through Endpoint E1, and similarly processes incoming DTOs from C2 through Endpoint E2. The server can then prevent C1 from accessing C2's RDMA target buffer (and *vice versa*) by placing the RDMA target buffer for C1 in Protection Zone P1 and assigning P1 to E1, and by placing the RDMA target buffer for C2 in Protection Zone P2 and assigning P2 to E2. The server also needs to receive incoming Send operations from both C1 and C2, and may therefore decide to have both E1 and E2 share a single S-RQ. Since an S-RQ and all Receive buffers used on the S-RQ must have the same Protection Zone, the PZ associated with the S-RQ will be one of P1 or P2 and thus be different from the PZ of either E2 or E1, respectively. The PZ associated with the S-RQ could also be a third PZ that is different from P1 and P2.

**SEE ALSO**

*it_srq_free*(), *it_srq_query*(), *it_srq_modify*(), *it_post_recv*(), *it_ep_rc_create*()

# it_srq_free()

## NAME

it_srq_free – destroy a Shared Receive Queue

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_srq_free(
    IN  it_srq_handle_t  srq_handle
);
```

## APPLICABILITY

*it_srq_free* is applicable only to Endpoints created for the RC service type and is supported only if the Interface Adapter attribute *srq_support* (see *it_ia_info_t*) is IT_TRUE. Endpoints of the UD service type cannot use an S-RQ.

## DESCRIPTION

| | |
|---|---|
| *srq_handle* | Handle of the Shared Receive Queue to be destroyed. |

The *it_srq_free* routine destroys the Shared Receive Queue *srq_handle*. On return, the Handle *srq_handle* may no longer be used. A Shared Receive Queue may not be destroyed if it is still being used by an Endpoint; an attempt to do so will fail and the S-RQ will not be affected.

If this routine returns success, for each Receive DTO that was outstanding on the S-RQ the Consumer is guaranteed that either a Completion Event has been enqueued or that no Completion Event will be enqueued. In addition, the Implementation shall release ownership of the local buffers associated with all Receive DTOs that were outstanding on the S-RQ back to the Consumer.

## RETURN VALUE

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

| | |
|---|---|
| IT_ERR_INVALID_SRQ | The Shard Receive Queue Handle srq_*handle*) was invalid. |
| IT_ERR_SRQ_BUSY | The Shared Receive Queue was still referenced by an Endpoint. |
| 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. |

## SEE ALSO

*it_srq_create*, *it_srq_query*(), *it_srq_modify*()

# it_srq_modify()

## NAME

it_srq_modify – modify selected attributes of a Shared Receive Queue

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_srq_modify(
    IN         it_srq_handle_t      srq_handle,
    IN         it_srq_param_mask_t  mask,
    IN  const  it_srq_param_t      *params
);
```

## APPLICABILITY

*it_srq_modify* is applicable only to Endpoints created for the RC service type and is supported only if the Interface Adapter attribute *srq_support* (see *it_ia_info_t*) is IT_TRUE. Endpoints of the UD service type cannot use an S-RQ.

## DESCRIPTION

| | |
|---|---|
| *srq_handle* | The Shared Receive Queue whose attributes are to be modified. |
| *mask* | Bitwise OR of flags for specified parameters. |
| *params* | Structure whose members contain the new parameter values. |

The *it_srq_modify* routine changes selected attributes of the Shared Receive Queue *srq_handle*. Attributes to be modified are specified by flags in *mask*. New values for the attributes are specified by the corresponding fields in the structure pointed to by *params*. Fields and their corresponding flag values are shown in *it_srq_param_t*. Note that attributes represented by fields of *it_srq_param_t* that are not shown below cannot be modified. The definition of each field follows:

| | |
|---|---|
| *max_recv_dtos* | The maximum number of Receive DTOs that the Consumer can safely have outstanding on the S-RQ. This can only be modified if the IA associated with the S-RQ supports dynamically resizing the S-RQ. See *it_ia_info_t* for details of how to determine this. If the Consumer attempts to modify this to a value that is less than the total number of Receive DTOs that are currently enqueued within the S-RQ, an error shall be returned. |
| *low_watermark* | The S-RQ Low Watermark threshold. If the Consumer attempts to modify this to a value of zero, an error shall be returned. If the Consumer attempts to modify this to a value that is greater than *max_recv_dtos*, an error shall be returned. If the Consumer modifies this to any non-zero value (including the existing non-zero value if the S-RQ Low Watermark mechanism is already armed) and *it_srq_modify* returns success, the S-RQ Low Watermark mechanism shall be armed/rearmed. Once the mechanism is armed, it shall remain armed until either the S-RQ is destroyed by a call to *it_srq_free*, or the total number of Receive Work Requests outstanding on the S-RQ falls below the threshold. If the mechanism is disarmed because the total number of Receive Work Requests outstanding on the S-RQ falls below this threshold, an Affiliated Asynchronous Event shall be enqueued |

on the Affiliated Asynchronous Event Stream for the IA that the S-RQ is associated with; see *it_affiliated_event_t* for details. Once the S-RQ Low Watermark mechanism is disarmed, it can only be rearmed by calling *it_srq_modify* to establish a new *low_watermark* value.

Modifying the maximum number of Receive DTOs that the S-RQ can safely have outstanding while there are Endpoints actively using the S-RQ can have an adverse impact upon performance, and can potentially cause the Connections on Endpoints that are using the S-RQ to break. The Consumer is therefore advised to only modify this attribute of the S-RQ when data transfer activity on the Endpoints associated with the S-RQ has been quiesced to minimize the risk of Connections breaking.

The Consumer is allowed to call *it_srq_modify* to modify the S-RQ Low Watermark for an S-RQ whose S-RQ Low Watermark mechanism is already armed. The result of doing so is that the mechanism shall remain armed, but the new S-RQ Low Watermark value shall replace the old one, subject to the normal bounds checking of the supplied S-RQ Low Watermark value that is done by the *it_srq_modify* routine.

*it_srq_modify* shall succeed in modifying all the requested attributes atomically; if the attempt to modify any of the requested attributes generates an error, none of the other attributes supplied to the call will be applied.

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_INVALID_SRQ | The Shared Receive Queue Handle *srq_handle* was invalid. |
| IT_ERR_INVALID_MASK | The *mask* contained invalid flag values. |
| IT_ERR_RESOURCES | The requested operation failed due to insufficient resources. |
| IT_ERR_INVALID_SRQ_SIZE | An attempt was made to set the total number of entries in the S-RQ (*max_recv_dtos*) to a value that is less than the total number of Receive DTOs currently enqueued in the S-RQ. |
| IT_ERR_SRQ_LOW_WATERMARK | An attempt was made to either set the S-RQ Low Watermark (*low_watermark*) to a value greater than *max_recv_dtos*, or to set the S-RQ Low Watermark to a value of zero. |
| 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. |

**SEE ALSO**

*it_srq_create*(), *it_srq_free*(), *it_srq_query*()

# it_srq_query()

## NAME

it_srq_query – get attributes of a Shared Receive Queue

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_srq_query(
    IN   it_srq_handle_t      srq_handle,
    IN   it_srq_param_mask_t  mask,
    OUT  it_srq_param_t      *params
);

typedef enum {
    IT_SRQ_PARAM_ALL            = 0x000001,
    IT_SRQ_PARAM_IA             = 0x000002,
    IT_SRQ_PARAM_PZ             = 0x000004,
    IT_SRQ_PARAM_MAX_RECV_DTO   = 0x000008,
    IT_SRQ_PARAM_MAX_RECV_SEG   = 0x000010,
    IT_SRQ_PARAM_LOW_WATERMARK  = 0x000020
} it_srq_param_mask_t;

typedef struct {
    it_ia_handle_t  ia;             /* IT_SRQ_PARAM_IA */
    it_pz_handle_t  pz;             /* IT_SRQ_PARAM_PZ */
    size_t          max_recv_dtos;  /* IT_SRQ_PARAM_MAX_RECV_DTO */
    size_t          max_recv_segs;  /* IT_SRQ_PARAM_MAX_RECV_SEG */
    size_t         low_watermark;   /* IT_SRQ_PARAM_LOW_WATERMARK */
} it_srq_param_t;
```

## APPLICABILITY

*it_srq_query* is applicable only to Endpoints created for the RC service type and is supported only if the Interface Adapter attribute *srq_support* (see *it_ia_info_t*) is IT_TRUE. Endpoints of the UD service type cannot use an S-RQ.

## DESCRIPTION

| | |
|---|---|
| *srq_handle* | The Shared Receive Queue whose attributes are being queried. |
| *mask* | Bitwise OR of flags for desired parameters. |
| *params* | Structure whose members are written with the desired parameters. |

The *it_srq_query* routine returns the desired attributes of the Shared Receive Queue *srq_handle* in the structure pointed to by *params*. On return, each field of *params* is only valid if the corresponding flag as shown in the Synopsis is set in the *mask* argument. The *mask* value IT_SRQ_PARAM_ALL causes all fields to be returned.

The definition of each field of *params* follows:

| | |
|---|---|
| *ia* | The Interface Adapter Handle associated with the S-RQ. |
| *pz* | The Protection Zone Handle specified to create the S-RQ. |

| | | |
|---|---|---|
| *max_recv_dtos* | The maximum number of Receive DTOs that the Consumer can safely have outstanding on the S-RQ. |
| *max_recv_segs* | The maximum number of data segments for a local buffer that the Consumer may specify for a Receive DTO posted to the S-RQ. |
| *low_watermark* | 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. |

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_INVALID_SRQ | The Shared Receive Queue Handle (*srq_handle*) was invalid. |
| IT_ERR_INVALID_MASK | The *mask* contained invalid flag values. |
| 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. |

**SEE ALSO**

*it_srq_create*(), *it_srq_free*(), *it_srq_modify*()

# it_ud_service_reply()

## NAME

it_ud_service_reply – return the information necessary to communicate via Unreliable Datagram (UD) messages with the entity specified by the Connection Qualifier in the UD Service Request Event

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_ud_service_reply (
    IN        it_ud_svc_req_identifier_t  ud_svc_req_id,
    IN        it_ud_svc_req_status_t      status,
    IN        it_remote_ep_info_t         ep_info,
    IN  const unsigned char              *private_data,
    IN        size_t                      private_data_length
);

typedef uint64_t it_ud_svc_req_identifier_t;
```

## APPLICABILITY

*it_ud_service_reply* is applicable only to the UD service type.

## DESCRIPTION

| | |
|---|---|
| *ud_svc_req_id* | 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 *it_ud_service_reply*. |
| *status* | Status to return in the IT_CM_MSG_UD_SERVICE_REPLY_EVENT data indicating the outcome of the UD Service Request. |
| *ep_info* | End-point information to be used by the UD Service Requester to communicate with this UD Service. |
| *private_data* | Opaque Private Data provided by the Consumer which will be sent as part of the *it_ud_service_reply*. If the IA does not support Private Data, *private_data_length* must be 0. |
| *private_data_length* | Length of the *private_data* provided by the Consumer. If the IA does not support Private Data, this field must be 0. |

The *it_ud_service_reply* routine will be called by the Consumer to respond to an IT_CM_REQ_UD_SERVICE_REQUEST_EVENT Event. The IT_CM_REQ_UD_SERVICE_REQUEST_EVENT Event data (*it_ud_svc_request_event_t*) contains a unique Service Request Handle, the Connection Qualifier of interest, Source address information, and optional Private Data. The recipient of the IT_CM_REQ_UD_SERVICE_REQUEST_EVENT Event needs to respond to the request by calling *it_ud_service_reply*.

The *ud_svc_req_id* is a unique identifier allowing this response to be correlated to the request being responded to. The *ud_svc_req_id* should be copied from the IT_CM_REQ_UD_SERVICE_REQUEST_EVENT Event data, *ud_svc_req_id* field. Once *it_ud_service_reply* has

been successfully invoked, the supplied *ud_svc_req_id* is no longer valid. The resources associated with the *ud_svc_req_id* are released and the *ud_svc_req_id* cannot be re-used.

Valid status codes for the *status* field are defined in *it_ud_svc_req_status_t*. A valid status code must be provided. IT_UD_REQ_REDIRECTED cannot be supplied as input for this parameter, even though it may appear in the Event given to the requester. The Implementation is responsible for redirection, not the Consumer.

The *ep_info* is only used by this routine if the *status* field is set to IT_UD_SVC_EP_INFO_VALID. See *it_ud_svc_req_status_t* for details.

The IA can be queried via *it_ia_query* to determine whether it supports the transfer of Private Data. This is indicated by the *private_data_support* field of the *it_ia_info_t* structure. If Private Data is not supported, *private_data_length* must be 0. The maximum length of *private_data* can be determined by examining the *ud_rep_private_data_len* member of the *it_ia_info_t* structure.

**EXTENDED DESCRIPTION**

*it_ud_service_reply* is called by the Consumer in response to receiving an IT_CM_REQ_UD_SERVICE_REQUEST_EVENT Event. The Consumer chooses how to respond to the Service Request and makes that choice known via the value of *status* passed into the *it_ud_service_reply* call. The value of *status* determines whether the Implementation uses the *ep_info* input parameter. The table below describes the meaning of each *status* value, and whether the Implementation uses the *ep_info* input parameter when that *status* value is present.

| ***status* Value** | **Implication of the *status* Value** |
|---|---|
| IT_UD_SVC_EP_INFO_VALID | The supplied *ep_info* (*it_remote_ep_info_t*) is valid and can be used by the recipient of the *it_ud_service_reply* to communicate with this service via UD messages. The Consumer must supply an *it_remote_ep_info_t* structure containing a valid *ud_ep_id* and *ud_ep_key*. |
| IT_UD_SVC_ID_NOT_SUPPORTED | The service described by the *conn_qual* (*it_conn_qual_t*) in the *it_ud_svc_request_event_t* is not supported by this service. The Implementation does not use the *ep_info* parameter. |
| IT_UD_SVC_REQ_REJECTED | Rejects the request for UD Service information. The Implementation does not use the *ep_info* parameter. |
| IT_UD_NO_EP_AVAILABLE | The Consumer responding via *it_ud_service_reply* does not have any Endpoints available for UD communication. The Implementation does not use the *ep_info* parameter. |
| IT_UD_REQ_REDIRECTED | The Consumer cannot set this status. This status can only be set by the Implementation. |

In order for the Implementation to be able to correctly correlate this *it_ud_service_reply* call with the request Event being responded to, the Consumer must supply the *ud_svc_req_id* from the *it_ud_svc_request_event_t* as the *ud_svc_req_id* passed into the *it_ud_service_reply* call.

It is possible to receive duplicate UD Service Requests as a result of the active side retrying an *it_ud_service_request* operation. It is the Consumer's responsibility to detect and handle duplicate requests. Requests are uniquely identified by a combination of the *ud_svc_req_id* and

the *source_addr* from the *it_ud_svc_request_event_t* data. This combination can be used to detect duplicate UD Service Requests.

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_PDATA_NOT_SUPPORTED | Private Data was supplied by the Consumer, but this Interface Adapter does not support Private Data. |
| IT_ERR_INVALID_PDATA_LENGTH | The Interface Adapter supports Private Data, but the length specified exceeded the Interface Adapter's capabilities. |
| IT_ERR_INVALID_UD_SVC_REQ_ID | The Unreliable Datagram Service Request ID (*ud_svc_req_id*) was invalid. |
| IT_ERR_INVALID_UD_STATUS | The Unreliable Datagram Service Request *status* was invalid. |
| 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. |

**SEE ALSO**

*it_ia_query*(), *it_ud_service_request*(), *it_ep_attributes_t*, *it_cm_msg_events*, *it_cm_req_events*

# it_ud_service_request()

**NAME**

it_ud_service_request – request that the recipient of this message return the information necessary to communicate via Unreliable Datagram (UD) messages to the entity specified by the UD Service Handle

**SYNOPSIS**

```
#include <it_api.h>

it_status_t it_ud_service_request (
    IN  it_ud_svc_req_handle_t  ud_svc_handle
);
```

**APPLICABILITY**

*it_ud_service_request* is applicable only to the UD service type.

**DESCRIPTION**

| | |
|---|---|
| *ud_svc_handle* | UD Service Request Handle created by a call to *it_ud_service_request_handle_create*. This Handle uniquely identifies this UD Service Request operation. The UD Service Request Handle is associated with a specific UD Service described during the creation of the UD Service Request Handle. |

The *it_ud_service_request* routine is called by a Consumer to request a remote entity specified by the UD Service Handle to return information necessary to communicate via Unreliable Datagram messages.

The *ud_svc_handle* provides the Consumer with a means of correlating this *it_ud_service_request* with the IT_CM_MSG_UD_SERVICE_REPLY_EVENT Event that the Consumer will receive when the remote Endpoint responds to this UD Service Request. See *it_cm_msg_events*.

Due to the nature of Unreliable Datagrams, even though an invocation of *it_ud_service_request* returns success, the target of the UD Service Request may not receive it. Therefore, the Consumer may have to call *it_ud_service_request* multiple times with the same *ud_svc_handle* before the recipient actually receives the request and is able to reply to it. In addition, if the Consumer issues multiple requests with the same *ud_svc_handle*, the Consumer may receive multiple replies. It is up to the Consumer to detect and handle duplicate replies.

The *ud_svc_req_id* (*it_ud_svc_req_identifier_t*) associated with a given *ud_svc_handle* does not change. Therefore, all retries using a given *ud_svc_handle* will result in the same *ud_svc_req_id* being presented to the recipient of the IT_CM_REQ_UD_SERVICE_REQUEST_EVENT Event in the Event data (*it_ud_svc_request_event_t*).

Upon a successful invocation and transmission of the *it_ud_service_request*, once the recipient of the request replies via *it_ud_service_reply*, the Consumer will receive an IT_CM_MSG_UD_SERVICE_REPLY_EVENT Event. The IT_CM_MSG_UD_SERVICE_REPLY_EVENT Event data (*it_ud_svc_reply_event_t*) contains the results of the Service Request query. The *status* field of the *it_ud_svc_reply_event_t* structure in the IT_CM_MSG_UD_SERVICE_REPLY_EVENT indicates the state of the information in the *it_ud_svc_reply_event_t* structure. See *it_cm_msg_events*.

**EXTENDED DESCRIPTION**

The *ud_service_request* call requests information from the remote UD Service. Once that remote UD Service responds, an IT_CM_MSG_UD_SERVICE_REPLY Event will be generated. The data associated with the Event, *it_ud_svc_reply_event_t*, contains information the Consumer needs in order to perform Data Transfer Operations with the remote UD Service. The *status* (*it_ud_svc_req_status_t*) field of the *it_ud_svc_reply_event_t* indicates the validity of other fields in the structure. The *status* field should be checked by the Consumer prior to making any assumptions about the data in the rest of the structure. The table below summarizes the *status* values and the implications on the data in the *it_ud_svc_reply_event_t* structure:

| ***status* Value** | **Implication for *it_ud_svc_reply_event_t* Data** |
|---|---|
| IT_UD_SVC_EP_INFO_VALID | The *ep_info* (*it_remote_ep_info_t*) is valid. The *ud_ep_id* and *ud_ep_key* from the *it_remote_ep_info_t* structure, combined with the *it_path_t* from the *ud_svc_handle* provides the Consumer with the necessary information to perform Data Transfer Operations with the remote UD Service. All fields except *destination_path* contain valid data. |
| IT_UD_SVC_ID_NOT_SUPPORTED | The Service described by the *connection_qualifier* (*it_conn_qual_t*) in the *ud_svc_handle* is not supported on the Spigot to which the *it_ud_service_request* was sent. All fields except *ep_info* and *destination_path* contain valid data. |
| IT_UD_SVC_REQ_REJECTED | The recipient of the *it_ud_service_request* rejected the UD Service Request operation. All fields except *ep_info* and *destination_path* contain valid data. |
| IT_UD_NO_EP_AVAILABLE | The recipient of the *it_ud_service_request* 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 *ep_info* and *destination_path* contain valid data. |
| IT_UD_REQ_REDIRECTED | The Implementation on the receiving side of the *it_ud_service_request* has requested that the Consumer redirect the Service Request operation to a new Destination. The *destination_path* (*it_path_t*) contains valid data. The *destination_path* should be used to create a new *ud_svc_handle* to be used in another call to *it_ud_service_request*. All fields except *ep_info*, *private_data*, and *private_data_length* contain valid data. |

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_INVALID_UD_SVC | The Unreliable Datagram Service Handle (*ud_svc_handle*) was invalid. |
| 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. |

**SEE ALSO**

*it_ud_service_request_handle_create*(), *it_ud_service_reply*(), *it_cm_msg_events*, *it_path_t*, *it_ep_attributes_t*

# it_ud_service_request_handle_create()

**NAME**

it_ud_service_request_handle_create – create an Unreliable Datagram (UD) Service Request Handle

**SYNOPSIS**

```
#include <it_api.h>

it_status_t it_ud_service_request_handle_create (
    IN  const  it_conn_qual_t          *conn_qual,
    IN         it_evd_handle_t          reply_evd,
    IN  const  it_path_t                *destination_path,
    IN  const  unsigned char            *private_data,
    IN         size_t                    private_data_length,
    OUT        it_ud_svc_req_handle_t   *ud_svc_handle
);
```

**APPLICABILITY**

*it_ud_service_request_handle_create* is applicable only to the UD service type.

**DESCRIPTION**

| | |
|---|---|
| *conn_qual* | The Connection Qualifier describing the UD Service for which the Consumer is requesting information. |
| *reply_evd* | The Simple EVD on which the IT_CM_MSG_UD_SERVICE_REPLY_EVENT Event will be received. *reply_evd* must be of the IT_CM_MSG_EVENT_STREAM Event Stream Type. See *it_cm_msg_events*. |
| *destination_path* | *destination_path* specifies a Path to the Destination of the *it_ud_service_request* operation. |
| *private_data* | Opaque Private Data provided by the Consumer which will be sent as part of the *it_ud_service_request*. If the IA does not support Private Data, *private_data_length* must be 0. |
| *private_data_length*: | Length of the *private_data* provided by the Consumer. If the IA does not support Private Data, this field must be 0. |
| *ud_svc_handle* | UD Service Request Handle created by this call. This Handle will be used in a call to *it_ud_service_request*. |

The *it_ud_service_request_handle_create* routine is called by the Consumer to create an Unreliable Datagram Service Request Handle to be used in a call to *it_ud_service_request*.

The *destination_path* can be obtained by calling *it_get_pathinfo*. The *spigot_id* in the *it_path_t* will be the Spigot Identifier used for this UD Service Request.

The IA can be queried via *it_ia_query* to determine whether it supports the transfer of Private Data. This is indicated by the *private_data_support* field of the *it_ia_info_t* structure. If Private Data is not supported, *private_data_length* must be 0. The maximum length of *private_data* can be determined by examining the *ud_req_private_data_len* member of the *it_ia_info_t* structure.

The returned *ud_svc_handle* is used to identify the UD Service Request. It provides the Consumer with a means of correlating this *it_ud_service_request* with the IT_CM_MSG_UD_SERVICE_REPLY_EVENT Event that the Consumer will receive when the remote Endpoint responds to this UD Service Request.

The *ud_svc_req_id* (*it_ud_svc_req_identifier_t*) associated with a given *ud_svc_handle* does not change. Therefore, all retries using a given *ud_svc_handle* will result in the same *ud_svc_req_id* being presented to the recipient of the IT_CM_REQ_UD_SERVICE_REQUEST_EVENT Event in the Event data (*it_ud_svc_request_event_t*).

**EXTENDED DESCRIPTION**

The members of the *it_path_t* structure that are pertinent for creating a UD Service Request Handle are listed in the table below:

| ***it_path_t* Member** | **Description** |
|---|---|
| *spigot_id* | Spigot Identifier |
| *ib.partition_key* | Partition Key |
| *ib.local_port_lid* | Source LID |
| *ib.remote_port_lid* | Destination LID |
| *ib.sl* | Service Level |

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_INVALID_CONN_EVD | The Connection Simple Event Dispatcher Handle was invalid. |
| IT_ERR_INVALID_EVD_TYPE | The Event Stream Type for the Event Dispatcher was invalid. |
| IT_ERR_PDATA_NOT_SUPPORTED | Private Data was supplied by the Consumer, but this Interface Adapter does not support Private Data. |
| IT_ERR_INVALID_PDATA_LENGTH | The Interface Adapter supports Private Data, but the length specified exceeded the Interface Adapter's capabilities. |
| IT_ERR_INVALID_CONN_QUAL | The Connection Qualifier (*conn_qual*) was invalid. |
| IT_ERR_INVALID_SOURCE_PATH | One of the components of the Source portion of the supplied Path was invalid. |
| IT_ERR_INVALID_SPIGOT | An invalid Spigot ID was specified (*spigot_id* member of the *destination_path*). |
| IT_ERR_RESOURCES | The requested operation failed due to insufficient resources. |
| 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.

**APPLICATION USAGE**

The resulting *ud_svc_handle* (*it_ud_svc_req_handle_t*) produced by this call will be used in calls to *it_ud_service_request* to obtain information describing how to communicate with the remote UD Service described by *conn_qual* (*it_conn_qual_t*).

The *it_ud_service_request* call requests information from the remote UD Service. Once that remote UD Service responds, an IT_CM_MSG_UD_SERVICE_REPLY Event will be generated. The data associated with the Event, *it_ud_svc_reply_event_t*, contains an *it_remote_ep_info_t* structure and other information. The *ud_ep_id* and *ud_ep_key* from the *it_remote_ep_info_t*, combined with the information from the *destination_path* (*it_path_t*), provides the Consumer the necessary information to perform Data Transfer Operations with the remote UD Service.

Note that the *spigot_id* of the Endpoint that will be used for Data Transfer Operations with the UD Service being requested must match the *spigot_id* in the *destination_path*.

See *it_ud_service_request* and *it_ud_service_reply* for more information.

**SEE ALSO**

*it_ud_service_request_handle_free*(), *it_ud_request_handle_query*(), *it_ia_query*(), *it_ud_service_request*(), *it_get_pathinfo*(), *it_path_t*, *it_cm_msg_events*, *it_ep_attributes_t*

# it_ud_service_request_handle_free()

## NAME

it_ud_service_request_handle_free – free a previously created *it_ud_svc_req_handle_t*

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_ud_service_request_handle_free (
    IN  it_ud_svc_req_handle_t  ud_svc_handle
);
```

## APPLICABILITY

*it_ud_service_request_handle_free* is applicable only to the UD service type.

## DESCRIPTION

| | |
|---|---|
| ud_svc_handle | Unreliable Datagram (UD) Service Request Handle previously created by a call to *it_ud_service_request_handle_create*. |

*it_ud_service_request_handle_free* removes an existing UD Service Request Handle and frees all associated underlying resources. Once *it_ud_service_request_handle_free* returns, *ud_svc_handle* can no longer be used in UD Service Request operations. In addition, once *it_ud_service_request_handle_free* returns, any replies to outstanding UD Service Request operations associated with this *ud_svc_handle* will be silently dropped.

Any IT_CM_MSG_UD_SERVICE_REPLY_EVENT Events associated with this request that have been enqueued on the Event Dispatcher (EVD) will not be removed. It is the Consumer's responsibility to dequeue and dispose of them.

## RETURN VALUE

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

| | |
|---|---|
| IT_ERR_INVALID_UD_SVC | The Unreliable Datagram Service Handle (*ud_svc_handle*) was invalid. |
| 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. |

## SEE ALSO

*it_ud_service_request_handle_create*(), *it_ud_service_request_handle_query*(), *it_cm_msg_events*

# it_ud_service_request_handle_query()

## NAME

it_ud_service_request_handle_query – return information about a specified *it_ud_svc_req_handle_t*

## SYNOPSIS

```
#include <it_api.h>

it_status_t it_ud_service_request_handle_query (
    IN   it_ud_svc_req_handle_t      ud_svc_handle,
    IN   it_ud_svc_req_param_mask_t  mask,
    OUT  it_ud_svc_req_param_t      *ud_svc_handle_info
);

typedef enum {
    IT_UD_PARAM_ALL               = 0x00000001,
    IT_UD_PARAM_IA_HANDLE         = 0x00000002,
    IT_UD_PARAM_REQ_ID            = 0x00000004,
    IT_UD_PARAM_REPLY_EVD         = 0x00000008,
    IT_UD_PARAM_CONN_QUAL         = 0x00000010,
    IT_UD_PARAM_DEST_PATH         = 0x00000020,
    IT_UD_PARAM_PRIV_DATA         = 0x00000040,
    IT_UD_PARAM_PRIV_DATA_LENGTH  = 0x00000080
} it_ud_svc_req_param_mask_t;

/*
 * The it_ud_svc_req_param_mask_t value in the comment above
 * each attribute in the it_ud_svc_req_param_t structure below
 * is the mask value used to select that attribute in a call
 * to it_ud_service_request_handle_query.
 */
typedef struct {
    it_ia_handle_t   ia;                /* IT_UD_PARAM_IA_HANDLE */
    uint32_t         request_id;        /* IT_UD_PARAM_REQ_ID */
    it_evd_handle_t  reply_evd;         /* IT_UD_PARAM_REPLY_EVD */
    it_conn_qual_t   conn_qual;         /* IT_UD_PARAM_CONN_QUAL */
    it_path_t        destination_path;  /* IT_UD_PARAM_DEST_PATH */
    unsigned char    private_data[IT_MAX_PRIV_DATA];
                                        /* IT_UD_PARAM_PRIV_DATA */
    size_t           private_data_length;
                                   /* IT_UD_PARAM_PRIV_DATA_LENGTH */
} it_ud_svc_req_param_t;
```

## APPLICABILITY

*it_ud_service_request_handle_query* is applicable only to the UD service type.

## DESCRIPTION

| | |
|---|---|
| ud_svc_handle | Unreliable Datagram (UD) Service Request Handle previously created by a call to *it_ud_service_request_handle_create*. |

| | |
|---|---|
| mask | Bitwise OR of flags for the requested UD Service Request Handle parameters. |
| ud_svc_handle_info | Data structure containing information about the UD Service Request Handle, *ud_svc_handle*. |

*it_ud_service_request_handle_query* collects the desired information about the *ud_svc_handle* passed in and returns that information in the *it_ud_svc_req_param_t* structure provided in *ud_svc_handle_info*. On return, each field of *ud_svc_handle_info* is only valid if the corresponding flag is set in the *mask* argument. The flag values for the *mask* appear in the comments above each of the fields in the *it_ud_svc_req_param_t* structure. The *mask* value IT_UD_PARAM_ALL causes all fields to be returned.

The definition of each field in the *it_ud_svc_req_param_t* structure is as follows:

| | |
|---|---|
| *ia* | Handle for the Interface Adapter associated with this UD Service Request. |
| *request_id* | Unique identifier associated with the *it_ud_svc_req_handle_t*. |
| *reply_evd* | The Simple EVD for reply Events associated with the *it_ud_svc_req_handle_t*. |
| *conn_qual* | Connection Qualifier describing the UD Service associated with the *it_ud_svc_req_handle_t*. |
| *destination_path* | Path to the Destination of the *it_ud_service_request* operation associated with the *it_ud_svc_req_handle_t*. |
| *private_data* | Opaque Private Data provided by the Consumer if the IA supports Private Data. |
| *private_data_length* | Length of the Private Data supplied by the Consumer. |

**RETURN VALUE**

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

| | |
|---|---|
| IT_ERR_INVALID_UD_SVC | The Unreliable Datagram Service Handle (*ud_svc_handle*) was invalid. |
| IT_ERR_INVALID_MASK | The *mask* contained invalid flag values. |
| 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. |

**SEE ALSO**

*it_ud_service_request_handle_create*(), *it_ud_service_request_handle_free*(), *it_ud_service_request*()

8827 # 7 Data Type Reference Pages

| | |
|---|---|
| *it_addr_mode_t* | Addressing mode of a Local Memory Region or Remote Memory Region |
| *it_aevd_notification_event_t* | Aggregate Event Dispatcher Notification Event type |
| *it_affiliated_event_t* | Affiliated Asynchronous Event type |
| *it_boolean_t* | The Boolean type used by the IT-API |
| *it_cm_msg_events* | Communication Management Message Events |
| *it_cm_req_events* | Communication Management Request Events |
| *it_conn_qual_t* | Encapsulates all supported Connection Qualifier types |
| *it_context_t* | Structure describing a Consumer Context |
| *it_dg_remote_ep_addr_t* | DatagramTransport Endpoint address |
| *it_dto_cookie_t* | DTO Cookie type |
| *it_dto_events* | Completion Event types |
| *it_dto_flags_t* | Flags for Send, Receive, RDMA Read & Write, RMR Link & Unlink |
| *it_dto_status_t* | Definition of DTO and RMR completion asynchronous status |
| *it_ep_attributes_t* | Endpoint attributes |
| *it_ep_state_t* | RC and UD Endpoint state type definition |
| *it_event_t* | Definition of Event data structures |
| *it_handle_t* | Enumeration and type definitions for IT Handles |
| *it_ia_info_t* | Encapsulates all Interface Adapter attributes and Spigot information |
| *it_io_addr_t* | O/S-dependent structure describing a bus or physical address. |
| *it_iobl_t* | I/O buffer list. |
| *it_lmr_triplet_t* | Structure describing a DTO buffer in a Local Memory Region |
| *it_mem_priv_t* | Memory access privileges for Local and Remote Memory Regions |
| *it_net_addr_t* | Encapsulates all supported Network Address types |
| *it_path_t* | Describes the Path between a pair of Spigots |
| *it_rmr_triplet_t* | Structure describing a DTO buffer in a Remote Memory Region |
| *it_rmr_type_t* | Definition of RMR type |
| *it_software_event_t* | Software Event type |
| *it_status_t* | Definition of IT-API call return status |
| *it_unaffiliated_event_t* | Unaffiliated Asynchronous Event type |

# it_addr_mode_t

## NAME

it_addr_mode_t – definition of addressing mode of a Local Memory Region or Remote Memory Region

## SYNOPSIS

```
#include <it_api.h>

typedef enum {
    IT_ADDR_MODE_ABSOLUTE  = 0,
    IT_ADDR_MODE_RELATIVE  = 1
} it_addr_mode_t;
```

## DESCRIPTION

Addressing mode of an LMR or RMR, with two possible values as follows:

| | |
|---|---|
| IT_ADDR_MODE_ABSOLUTE | Indicates Absolute Addressing. 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.When an RMR is linked to an LMR with Absolute Addressing, the *addr* parameter of the RMR Link operation and the *addr* 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 *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. |
| IT_ADDR_MODE_RELATIVE | Indicates Relative Addressing. The *addr* 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 *addr.rel* member of an LMR Triplet or the *rdma_addr* 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 *addr* parameter of the RMR Link operation and the *addr* 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 *addr* 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 *addr.rel* member of an RMR Triplet or the *rdma_addr* parameter passed to an RDMA DTO is interpreted as a byte offset |

relative to the first byte of the RMR. The total offset relative to the first byte of the underlying LMR is obtained by adding to this byte offset the difference between the *addr* attributes of the RMR and the LMR established at RMR Link time.

**EXTENDED DESCRIPTION**

On InfiniBand, Relative Addressing is supported only if the Verb Extensions are supported including the “ZBVA” option. Moreover, Relative Addressing may be available only for Narrow RMRs (Type 2 Memory Windows).

**APPLICATION USAGE**

The desired addressing mode of an LMR must be specified when the LMR is created or linked. The desired addressing mode of an RMR must be specified when the RMR is linked.

The *addr_mode_relative_support* field of the *it_ia_info_t* structure, which can be queried through *it_ia_query*, indicates whether the IA supports Relative Addressing.

**SEE ALSO**

*it_lmr_create*(), *it_rmr_link*(), *it_post_atomic*(), *it_post_send*(), *it_post_sendto*(), *it_post_recv*(), *it_post_recvfrom*(), *it_post_rdma_write*(), *it_post_rdma_read*(), *it_lmr_triplet_t*, *it_rmr_triplet_t*

# it_aevd_notification_event_t

## NAME

it_aevd_notification_event_t – Aggregate Event Dispatcher Notification Event type

## SYNOPSIS

```
#include <it_api.h>

typedef struct {
    it_event_type_t  event_number;
    it_evd_handle_t  aevd;
    it_evd_handle_t  sevd;
} it_aevd_notification_event_t;
```

## DESCRIPTION

| | |
|---|---|
| *event_number* | Identifier of the Event type.<br>Valid values: IT_AEVD_NOTIFICATION_EVENT |
| *aevd* | Handle for the Aggregate Event Dispatcher (AEVD) where the Event was queued. |
| *sevd* | Handle to the Simple Event Dispatcher (SEVD) that experienced a Notification Event. |

An IT_AEVD_NOTIFICATION_EVENT_STREAM Event is generated when a Notification has occurred on an SEVD associated with an AEVD with the IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* set (see *it_evd_create*).

The AEVD Notification Event passes the Handle for the associated SEVD on which a Notification Event has occurred.

The AEVD Notification Event only applies to AEVDs. AEVDs do not overflow.

## SEE ALSO

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

# it_affiliated_event_t

## NAME

it_affiliated_event_t – Affiliated Asynchronous Event type

## SYNOPSIS

```
#include <it_api.h>

typedef struct {
    it_event_type_t  event_number;
    it_evd_handle_t  evd;

    union {
        it_evd_handle_t  sevd;
        it_ep_handle_t   ep;
        it_srq_handle_t  srq;
    } cause;
} it_affiliated_event_t;
```

## DESCRIPTION

| | |
|---|---|
| *event_number* | Identifier of the Event type.<br>Valid values: IT_ASYNC_AFF_SEVD_ENQUEUE_FAILURE,<br>IT_ASYNC_AFF_EP_FAILURE,<br>IT_ASYNC_AFF_EP_BAD_TRANSPORT_OPCODE,<br>IT_ASYNC_AFF_EP_REQ_DROPPED,<br>IT_ASYNC_AFF_EP_RDMAW_ACCESS_VIOLATION,<br>IT_ASYNC_AFF_EP_RDMAW_CORRUPT_DATA,<br>IT_ASYNC_AFF_EP_RDMAR_ACCESS_VIOLATION,<br>IT_ASYNC_AFF_EP_LOCAL_ACCESS_VIOLATION,<br>IT_ASYNC_AFF_EP_L_ACCESS_VIOLATION,<br>IT_ASYNC_AFF_EP_L_RECV_ACCESS_VIOLATION,<br>IT_ASYNC_AFF_EP_L_IRRQ_ACCESS_VIOLATION,<br>IT_ASYNC_AFF_EP_L_TRANSPORT_ERROR,<br>IT_ASYNC_AFF_EP_L_LLP_ERROR, IT_ASYNC_AFF_EP_R_ERROR,<br>IT_ASYNC_AFF_EP_R_ACCESS_VIOLATION,<br>IT_ASYNC_AFF_EP_R_RECV_ACCESS_VIOLATION,<br>IT_ASYNC_AFF_EP_R_RECV_LENGTH_ERROR,<br>IT_ASYNC_AFF_EP_SOFT_HI_WATERMARK<br>IT_ASYNC_AFF_SRQ_LOW_WATERMARK |
| *evd* | Handle for the Event Dispatcher where the Event was queued. |
| *sevd* | The Handle for the SEVD for which the Implementation failed to enqueue an Event. Valid only for the IT_ASYNC_AFF_SEVD_ENQUEUE_FAILURE Event type. |
| *ep* | The Handle for the Endpoint that experienced the Event. Valid for all asynchronous errors affiliated with Endpoint other than IT_ASYNC_AFF_SEVD_ENQUEUE_FAILURE and IT_ASYNC_AFF_SRQ_LOW_WATERMARK. |
| *srq* | The Handle for the S-RQ whose Low Watermark threshold has just been crossed. |

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*.

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.

The following table maps the Affiliated Asynchronous Error values in the *it_event_type_t* enumeration to a transport-independent description.

| ***it_event_type_t* 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 *it_evd_overflowed* 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. |

| ***it_event_type_t* Value** | **Generic Event Description** |
|---|---|
| IT_ASYNC_AFF_EP_L_RECV_ACCESS_VIOLATION | The Receive Queue or Shared Receive Queue of the local Endpoint detected an access violation while processing an incoming Send. Note that not all incoming Sends that cause an access violation will cause an Affiliated Asynchronous Event to be generated. |
| IT_ASYNC_AFF_EP_L_IRRQ_ACCESS_VIOLATION | The Inbound RDMA Read Queue of the local Endpoint detected an access violation while processing an incoming Read Request. |
| IT_ASYNC_AFF_EP_L_TRANSPORT_ERROR | The local Endpoint detected a transport error while processing an incoming request. Note that not all incoming requests that cause a transport error will cause an Affiliated Asynchronous Event to be generated. |
| IT_ASYNC_AFF_EP_L_LLP_ERROR | The local Endpoint detected a LLP error while processing an incoming request. |
| IT_ASYNC_AFF_EP_R_ERROR | The local Endpoint received an indication that the connected remote Endpoint detected an error while processing an incoming request. |
| IT_ASYNC_AFF_EP_R_ACCESS_VIOLATION | The local Endpoint received an indication that the connected remote Endpoint detected an access violation while processing an incoming RDMA request. |
| IT_ASYNC_AFF_EP_R_RECV_ACCESS_VIOLATION | The local Endpoint received an indication that the connected remote Endpoint could not place an incoming Send due to an MSN inconsistency or the lack of a Receive buffer. |
| IT_ASYNC_AFF_EP_R_RECV_LENGTH_ERROR | The local Endpoint received an indication that the connected remote Endpoint could not place an incoming Send due to insufficient length of the Receive buffer. |
| IT_ASYNC_AFF_EP_SOFT_HI_WATERMARK | The total number of Receive DTOs being processed by the Endpoint has exceeded the Endpoint Soft High Watermark threshold that was established by either the *it_ep_rc_create*() or *it_ep_modify*() routine. |

| *it_event_type_t* 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 *it_srq_modify*(). |

All Events on an IT_ASYNC_AFF_EVENT_STREAM SEVD cause Notification. See *it_evd_create* for details of Notification.

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.

**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].

| *it_event_type_t* 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_LLP_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.) |

| *it_event_type_t* 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].

| it_event_type_t Value | iWARP "Asynchronous Event Identifier" Name(s) |
|---|---|
| IT_ASYNC_AFF_SEVD_ENQUEUE_FAILURE | • CQ/SQ Error<br>• 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 | • Invalid STag<br>• Base and bounds violation<br>• Access Rights violation<br>• Invalid PD ID<br>• Wrap error<br>(Locally detected access violation due to incoming RDMA Write or RDMA Read Response message.) |
| IT_ASYNC_AFF_EP_L_RECV_ACCESS_VIOLATION | • Invalid MSN – no buffer available<br>• Invalid MSN – MSN range not valid/gap in MSN<br>(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 | • Invalid STag<br>• Base and bounds violation<br>• Access Rights violation<br>• Invalid PD ID<br>• Wrap error<br>• Invalid MSN – MSN range not valid/gap in MSN<br>(Locally detected access violation due to incoming RDMA Read Request.) |
| IT_ASYNC_AFF_EP_L_TRANSPORT_ERROR | • Invalid DDP Queue Number<br>• Invalid RDMA Read Request<br>• No 'L' bit when expected<br>(Locally detected operation error or protocol error due to incoming requests.) |
| IT_ASYNC_AFF_EP_L_LLP_ERROR | • LLP Connection Lost<br>• LLP Connection Reset<br>• LLP Integrity Error: Segment size invalid<br>• LLP Integrity Error: Invalid CRC<br>• Bad FPDU<br>(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:<br>• Invalid MSN – no buffer available<br>• Invalid MSN – MSN range not valid/gap in MSN<br>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:<br>• DDP Message too long for available buffer<br>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 |

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_LLP_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

**NAME**

it_boolean_t – the Boolean type used by the API

**SYNOPSIS**

```
#include <it_api.h>

typedef enum {
    IT_FALSE  = 0,
    IT_TRUE   = 1
} it_boolean_t;
```

**DESCRIPTION**

The *it_boolean_t* type is used in several data structures in the API to describe a value that can exist in one of two different states: true (IT_TRUE), or false (IT_FALSE).

**SEE ALSO**

*it_cm_msg_events*, *it_cm_req_events*, *it_ep_attributes_t*, *it_evd_create*(), *it_evd_modify*(), *it_evd_query*(), *it_ia_info_t*, *it_rmr_query*()

# it_cm_msg_events

**NAME**

Communication Management Message Events – definitions for communication management Events other than Connection Requests and definition of Unreliable Datagram service resolution reply Event

**SYNOPSIS**

```
#include <it_api.h>

#define IT_MAX_PRIV_DATA 256

typedef enum {
    IT_CN_REJ_OTHER           = 0,
    IT_CN_REJ_TIMEOUT         = 1,
    IT_CN_REJ_BAD_PATH        = 2,
    IT_CN_REJ_STALE_CONN      = 3,
    IT_CN_REJ_BAD_ORD         = 4,
    IT_CN_REJ_RESOURCES       = 5,
    IT_CN_REJ_BAD_CONN_PARMS  = 6
} it_conn_reject_code_t;

typedef struct {
    it_event_type_t         event_number;
    it_evd_handle_t         evd;
    it_cn_est_identifier_t  cn_est_id;
    it_ep_handle_t          ep;
    uint32_t                rdma_read_ird;
    uint32_t                rdma_read_ord;
    it_path_t               dst_path;
    it_conn_reject_code_t   reject_reason_code;
    unsigned char           private_data[IT_MAX_PRIV_DATA];
    it_boolean_t            private_data_present;
} it_connection_event_t;

typedef enum {
    IT_UD_SVC_EP_INFO_VALID     = 0,
    IT_UD_SVC_ID_NOT_SUPPORTED  = 1,
    IT_UD_SVC_REQ_REJECTED      = 2,
    IT_UD_NO_EP_AVAILABLE       = 3,
    IT_UD_REQ_REDIRECTED        = 4
} it_ud_svc_req_status_t;

typedef struct {
    it_event_type_t         event_number;
    it_evd_handle_t         evd;
    it_ud_svc_req_handle_t  ud_svc;
    it_ud_svc_req_status_t  status;
    it_remote_ep_info_t     ep_info;
    it_path_t               dst_path;
    unsigned char           private_data[IT_MAX_PRIV_DATA];
    it_boolean_t            private_data_present;
} it_ud_svc_reply_event_t;
```

**DESCRIPTION**

The Communication Management Message Event Stream, IT_CM_MSG_EVENT_STREAM, generates Events for all of the possible state transitions following a Connection Request as well as for Unreliable Datagram Service Resolution replies. These Events are all the Communication Management Events except those invoked by incoming requests (see *it_cm_req_events* for those).

Only one Event will be generated when a Connection is destroyed for any reason, either the IT_CM_MSG_CONN_DISCONNECT_EVENT or the IT_CM_MSG_CONN_BROKEN_ EVENT, but not both. The Consumer should be ready to handle either of these Events being generated even when the local or remote Consumer called *it_ep_disconnect.*

The Connection Events are represented by the *it_connection_event_t* structure and the Unreliable Datagram Service Resolution replies are represented by the *it_ud_svc_reply_event_ t* structure.

The *it_connection_event_ t* structure has the following members:

| | |
|---|---|
| *event_number* | Identifier of the Event type. Valid values:<br>IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT,<br>IT_CM_MSG_CONN_ESTABLISHED_EVENT,<br>IT_CM_MSG_CONN_PEER_REJECT_EVENT,<br>IT_CM_MSG_CONN_NONPEER_REJECT_EVENT,<br>IT_CM_MSG_CONN_DISCONNECT_EVENT,<br>IT_CM_MSG_CONN_BROKEN_EVENT |
| *evd* | Handle for the Event Dispatcher where the Event was queued. |
| *cn_est_id* | Identifier for the Connection establishment Event. |
| *ep* | Endpoint Handle associated with Connection in progress. |
| *rdma_read_ird* | Maximum number of incoming simultaneous RDMA Read operations supported by the remote EP. Only valid if the *it_ia_info.ird_ord_ia_support* value is IT_TRUE and as described under Application Usage below. Not valid if IT_LISTEN_SUPPRESS_IRD_ORD is set for *it_listen_create* or if IT_CONNECT_SUPPRESS_IRD_ORD is set for *it_ep_connect*. |
| *rdma_read_ord* | Maximum number of outgoing simultaneous RDMA Read operations supported by the remote EP. Only valid if the *it_ia_info.ird_ord_ia_support* value is IT_TRUE and as described under Application Usage below. Not valid if IT_LISTEN_SUPRRESS_IRD_ORD is set for *it_listen_create* or if IT_CONNECT_SUPPRESS_IRD_ORD is set for *it_ep_connect*. |
| *dst_path* | Path to Destination node supporting Service. Valid only if remote has rejected the proposed Path (*reject_reason_code* is IT_CN_REJ_BAD_PATH). Consumer should use *dst_path* if they wish to retry Connection attempt. |
| *reject_reason_code* | Reason for rejection of Connection attempt. |

| | |
|---|---|
| *private_data* | Private Data buffer. |
| *private_data_present* | When it has the value IT_TRUE then Private Data is present in the *private_data* buffer above. |

The *it_ud_svc_reply_event_ t* structure has the following members:

| | |
|---|---|
| *event_number* | Identifier of the Event type. Valid values: IT_CM_MSG_UD_SERVICE_REPLY_EVENT |
| *evd* | Handle for the Event Dispatcher where the Event was queued. |
| *ud_svc* | Handle for the corresponding Service Request. |
| *status* | Completion status for Service Request. |
| *ep_info* | Resolution of Connection Qualifier for the UD service to a specific remote Endpoint. Only valid if *status* is IT_UD_SVC_EP_INFO_VALID. See *it_ep_attributes_t* for the definition of the *it_remote_ep_info_t* structure. |
| *dst_path* | Path to Destination node supporting Service. Valid only if remote has redirected (*status* is IT_UD_REQ_REDIRECTED). Path returned is complete. |
| *private_data* | Private Data buffer. |
| *private_data_present* | When it has the value IT_TRUE then Private Data is present in the *private_data* buffer above. |

**EXTENDED DESCRIPTION**

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 *it_ep_accept* 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 *it_ep_disconnect*. 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.<br>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 *it_reject* for the specified Connection establishment request. | Applies to Connection management through the TII (two-way and three-way).<br>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 *it_reject*. 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.<br>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.<br>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
9126 Message Events. For any Event, *event_number* and *evd* are always valid.

| Event Type | Valid Fields |
|---|---|
| IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT | *cn_est_id*, *ep*, *rdma_read_ird*, *rdma_read_ord*, *private_data*, *private_data_present*.<br>The *cn_est_id* may not be valid if the Consumer called *it_ep_disconnect* on the associated Endpoint at any time before the *cn_est_id* is used. |
| IT_CM_MSG_CONN_ESTABLISHED_EVENT | *ep*, *private_data*, *private_data_present*, *rdma_read_ird* (IRD), *rdma_read_ord* (ORD).<br>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 | *ep*, *private_data*, *private_data_present* |
| IT_CM_MSG_CONN_PEER_REJECT_EVENT | *ep*, *private_data*, *private_data_present* |
| IT_CM_MSG_CONN_NONPEER_REJECT_EVENT | *ep*, *reject_reason_code*<br>If the *reject_reason_code* is IT_CN_REJ_ BAD_PATH, then *dst_path* is also valid. |
| IT_CM_MSG_CONN_BROKEN_EVENT | *ep* |

9127 **Table 4: Event Management Event Fields**

9128 The following table describes the meaning of the various *reject_reason_code* values that can be
9129 present in an IT_CM_MSG_CONN_NONPEER_REJECT_EVENT.

| reject_reason_code Value | Description |
|---|---|
| IT_CN_REJ_OTHER | The Connection establishment attempt was rejected for some reason other than those listed below. |

| reject_reason_code Value | Description |
|---|---|
| IT_CN_REJ_TIMEOUT | The Connection could not be established within the timeout period defined by the timeout member of the *it_path_t* that was input to *it_ep_connect*. This *reject_reason_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 *dst_path* 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 *rdma_read_ird* 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. |

9130 **Table 5: reject_reason_code Descriptions**

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. |

**Table 6: UD Service Resolution Reply Event Definitions**

UD service resolution replies return *status* in the Event data structure as described in the following table:

| Status | Description |
|---|---|
| IT_UD_SVC_EP_INFO_VALID | Reply is valid. *ep_info* 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. |

**Table 7: Service Resolution Reply Status**

All Events on an IT_CM_MSG_EVENT_STREAM SEVD cause Notification. See *it_evd_create* for details of Notification.

Default overflow behavior of an IT_CM_MSG_EVENT_STREAM SEVD is 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.

**EXTENDED DESCRIPTION**

For the Infiniband transport, the following table maps the values in the *reject_reason_code* field to their corresponding “Rejection Reason Code” for the REJ message as specified in Volume 1, Chapter 12 of the Infiniband specification. Rejection Reason Codes that are not listed in this 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<br>VipConnectRequest – VIP_NO_MATCH |
| IT_CN_REJ_TIMEOUT | VipConnectAccept – VIP_NOT_REACHABLE<br>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 tranport. |

**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 *connect*(2) 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 *timeout* member of the *it_path_t* structure passed to *it_ep_connect*. |

| 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 *Reject_Codes* as described in the IRD/ORD Header section in Appendix B. |
| IT_CN_REJ_RESOURCES | Due to the following LLP resource failure indications: ENOMEN and ENOBUFS. |
| IT_CN_REJ_BAD_CONN_PARMS | During Connection establishment, the local side did not recognize the version or format of the Connection parameter header received from the remote side. See Chapter 5. |

**Table 8: iWARP reject_reason_code Mapping**

For the InfiniBand Transport, the following table maps the *ep_info* field elements in the UD Service Resolution Event data type to InfiniBand concepts as specified in Volume 1 of the Infiniband specification.

| ep_info Element | IB Concept |
|---|---|
| *it_ud_ep_id_t* | Queue Pair Number (QPN) |
| *it_ud_ep_key_t* | Queue Key |

**Table 9: ep_info Element Mapping**

## APPLICATION USAGE

The Consumer should use the *it_event_t* structure if it is desired to wait for both communication management Events (*it_connection_event_t*) and Unreliable Datagram service resolution reply Events (*it_ud_svc_reply_event_t*) via the same EVD. The *it_event_t* structure is of sufficient size to hold either Event type.

When using three-way Connection establishment, the Consumer may receive an IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT containing *rdma_read_ird* and *rdma_read_ord* values that differ from those of the Endpoint in use. The Consumer should use *it_ep_modify* to adjust the values associated with the Endpoint to agree with those from this Event before issuing the *it_ep_accept* call to complete the Connection establishment.

When using two-way Connection establishment, the active side Consumer may receive an IT_CM_MSG_CONN_ESTABLISHED_EVENT containing *rdma_read_ird* and *rdma_read_ord* values that differ from those of the Endpoint in use. The Consumer may use *it_ep_modify* to adjust *rdma_read_ord* associated with the Endpoint if it is desired to conserve outbound RDMA resources.

When using two-way Connection establishment, the passive side Consumer may receive an IT_CM_MSG_CONN_ESTABLISHED_EVENT Event. The *rdma_read_ird* and *rdma_read_ord* values found in the Event should be ignored by the Consumer (rationale – the Implementation could, at best, expose the original IRD/ORD values sent by the active side, but the active side may choose to change ORD on receipt of the passive side thus invalidating the original values).

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

Neither the Active nor the Passive side Consumer should rely upon the IT_CM_MSG_CONN_ESTABLISHED_EVENT Event containing any Private Data, even if Private Data is input to the final *it_ep_accept* call that causes the Connection to be established. The side that makes the final call to *it_ep_accept* will never see any Private Data in the IT_CM_MSG_CONN_ESTABLISHED_EVENT Event, and because of races and unreliability inherent to the Connection establishment process of many of the transports that the IT-API supports, Private Data can sometimes be dropped on the other side as well.

See the *it_ep_disconnect* reference page for details on the guarantee of delivery of Private Data when disconnecting Connections.

The Consumer is advised to structure their ULP so that the Active side sends the first message after a Connection has been established. This is good practice because under some circumstances the completion of the first Receive operation is what causes the IT_CM_MSG_CONN_ESTABLISHED_EVENT Event to be generated on the Passive side. Depending upon the Passive side to send the first message after a Connection has been established can potentially result in the Connection establishment process timing out rather than completing successfully.

Consult the Application Usage section of *it_cm_req_events* for the discussion of use of Private Data.

When the Consumer receives an IT_CM_MSG_UD_SERVICE_REPLY_EVENT where the *status* is IT_UD_REQ_REDIRECTED, the Consumer can retry the attempt to retrieve UD service information. In order to do so the Consumer should free up its old UD Service Request Handle (by calling *it_ud_service_request_handle_free*), and create a new UD Service Request Handle by passing the *dst_path* returned in the IT_CM_MSG_UD_SERVICE_REPLY_EVENT to *it_ud_service_request_handle_create* to create a new Handle.

No ordering guarantee exists between generation of IT_CM_MSG_EVENT_STREAM Events and generation of IT_DTO_EVENT_STREAM Events. For example, when an Endpoint is disconnected, the IT_CM_MSG_CONN_DISCONNECT_EVENT can be generated on the IT_CM_MSG_EVENT_STREAM Event Stream either before or after the flushed Events (i.e., Events with an IT_DTO_ERR_FLUSHED status) for pending DTOs are generated on the IT_DTO_EVENT_STREAM Event Stream.

**SEE ALSO**

*it_event_t*, *it_evd_create*(), *it_evd_wait*(), *it_ep_modify*(), *it_listen_create*(), *it_ep_accept*(), *it_ep_disconnect*(), *it_reject*(), *it_address_handle_create*(), *it_ud_service_request_handle_free*(), *it_path_t*, *it_ia_info_t*, *it_ep_attributes_t*

# it_cm_req_events

**NAME**

Communication Management Request Events – definitions for Connection Request and Unreliable Datagram service resolution request communication management Events

**SYNOPSIS**

```
#include <it_api.h>

typedef struct {
    it_event_type_t         event_number;
    it_evd_handle_t         evd;
    it_cn_est_identifier_t  cn_est_id;
    it_conn_qual_t          conn_qual;
    it_net_addr_t           source_addr;
    size_t                  spigot_id;
    uint32_t                max_message_size;
    uint32_t                rdma_read_ird;
    uint32_t                rdma_read_ord;
    unsigned char           private_data[IT_MAX_PRIV_DATA];
    it_boolean_t            private_data_present;
} it_conn_request_event_t;

typedef struct {
    it_event_type_t             event_number;
    it_evd_handle_t             evd;
    it_ud_svc_req_identifier_t  ud_svc_req_id;
    it_conn_qual_t              conn_qual;
    it_net_addr_t               source_addr;
    size_t                      spigot_id;
    unsigned char               private_data[IT_MAX_PRIV_DATA];
    it_boolean_t                private_data_present;
} it_ud_svc_request_event_t;
```

**DESCRIPTION**

The Communication Management Request Event Stream, IT_CM_REQ_EVENT_STREAM, generates Events when an incoming Connection Request Event or incoming Unreliable Datagram service resolution request occurs.

Incoming Connection Request Events are represented by the *it_conn_request_event_t* structure and incoming Unreliable Datagram Service Resolution requests are represented by the *it_ud_svc_request_event_t* structure.

The *it_conn_request_event_t* structure has the following members:

| | |
|---|---|
| *event_number* | Identifier of the Event type. Valid values: IT_CM_REQ_CONN_REQUEST_EVENT |
| *evd* | Handle for the Event Dispatcher where the Event was queued. |
| *cn_est_id* | Identifier for the Connection establishment Event. |
| *conn_qual* | Connection Qualifier on which request was received. |

*source_addr* — Source address of requestor.

*spigot_id* — Local Spigot on which the request was received.

*max_message_size* — Largest message supported on Connection by the requesting remote EP. Only valid if *it_ia_info.max_message_size_support* is IT_TRUE.

*rdma_read_ird* — Maximum number of incoming simultaneous RDMA Read operations supported by the requesting remote EP. Only valid if *it_ia_info.ird_ord_ia_support* is IT_TRUE. Not valid if IT_LISTEN_SUPPRESS_IRD_ORD is set for *it_listen_create*.

*rdma_read_ord* — Maximum number of outgoing simultaneous RDMA Read operations supported by the requesting remote EP. Only valid if *it_ia_info.ird_ord_ia_support* is IT_TRUE. Not valid if IT_LISTEN_SUPPRESS_IRD_ORD is set for *it_listen_create*.

*private_data* — Private Data buffer.

*private_data_present* — When it has the value IT_TRUE, then Private Data is present in the *private_data* buffer above.

The *it_ud_svc_request_event_t* structure has the following members:

*event_number* — Identifier of the Event type. Valid values: IT_CM_REQ_UD_SERVICE_REQUEST_EVENT

*evd* — Handle for the Event Dispatcher where the Event was queued.

*ud_svc_req_id* — Identifier for the Service Request. Must be passed into the *it_ud_service_reply* call used to respond.

*conn_qual:* — Connection Qualifier on which request was received.

*source_addr* — Source address of requestor.

*spigot_id* — Local Spigot on which request was received.

*private_data* — Private Data buffer.

*private_data_present*: — When it has the value IT_TRUE then Private Data is present in the *private_data* buffer above.

| **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 (*cn_est_id*) in the Event. |
| IT_CM_REQ_UD_SERVICE_REQUEST_EVENT | An incoming request for Unreliable Datagram service resolution. This request is identified by the *ud_svc_req_id* in the Event. |

**Table 10: Communication Management Request Event Definitions**

All Events on an IT_CM_REQ_EVENT_STREAM SEVD cause Notification. See *it_evd_create* for details of Notification.

Default overflow behavior of an IT_CM_REQ_EVENT_STREAM SEVD is overflow Notification disabled. This default behavior of the SEVD is equivalent to IT_EVD_OVERFLOW_DEFAULT cleared and IT_EVD_OVERFLOW_NOTIFY cleared. See *it_evd_create* for details of overflow detection.

**APPLICATION USAGE**

The Consumer should use the *it_event_t* structure if it is desired to wait for both Connection Request Events (*it_conn_request_event_t*) and Unreliable Datagram service resolution request Events (*it_ud_svc_request_event_t*) via the same EVD. The *it_event_t* structure is of sufficient size to hold either Event type.

The Consumer must *it_evd_create* an IT_CM_REQ_EVENT_STREAM Simple EVD and pass the new EVD and a Connection Qualifier to *it_listen_create* in order to receive IT_CM_REQ_EVENT_STREAM Events via the *it_evd_wait* or *it_evd_dequeue* calls.

The *private_data_present* field indicates whether Private Data is present in the *private_data* buffer. It is the Consumer's responsibility to convey the size of the data contained in the Private Data buffer using their own ULP. Each communication management Event type may have a different maximum Private Data buffer size. The Consumer can determine the maximum possible sizes for the Private Data buffers corresponding to each of the Event types from the *it_ia_info_t* structure (for instance, *connect_private_data_len*).

**SEE ALSO**

*it_event_t*, *it_evd_create*(), *it_evd_wait*(), *it_evd_dequeue*(), *it_ep_modify*(), *it_listen_create*(), *it_ep_accept*(), *it_ep_disconnect*(), *it_reject*(), *it_ud_service_reply*(), *it_ia_info_t*

# it_conn_qual_t

**NAME**

it_conn_qual_t – encapsulates all supported Connection Qualifier types

**SYNOPSIS**

```
#include <it_api.h>

/* Enumerates all the possible Connection Qualifier types supported by
   the API. */
typedef enum {

    /* IANA (TCP/UDP) Port Number */
    IT_IANA_PORT = 0x01,

    /* InfiniBand Service ID, as described in Section 12.7.3 of
       Volume 1 of the InfiniBand specification. */
    IT_IB_SERVICEID = 0x02,

    /* VIA Connection Discriminator */
    IT_VIA_DISCRIMINATOR = 0x04,

    /* iWARP local and remote IP (IANA) port object */
    IT_IANA_LR_PORT = 0x08

} it_conn_qual_type_t;

/* Defines the Connection Qualifier format for a VIA “connection
   discriminator”. The API imposes a fixed upper bound on the
   discriminator size. */

#define IT_MAX_VIA_DISC_LEN  64

typedef struct {

    /* The total number of bytes in the array below */
    /* that are significant. */
    uint16_t  len;

    /* VIA connection discriminator, which is an array of bytes. */
    unsigned char  discriminator[IT_MAX_VIA_DISC_LEN];

} it_via_discriminator_t;

/* This defines the Connection Qualifier for InfiniBand, which is the
   64-bit Service ID. */
typedef uint64_t  it_ib_serviceid_t;

/* Defines an additional Connection Qualifier for iWARP that
   allows specification of both local and remote IANA ports when
   passing this structure to it_ep_connect. */
typedef struct {
    uint16_t  local;
```

```
            uint16_t  remote;
        } it_iana_lr_port_t;

        /* This describes a Connection Qualifier suitable for input to
           several routines in the API. */
        typedef struct {

            /* The discriminator for the union below. */
            it_conn_qual_type_t  type;

            union {

                /* IANA Port Number, in network byte order. */
                uint16_t  port;

                /* InfiniBand Service ID, in network byte order. */
                it_ib_serviceid_t  serviceid;

                /* VIA connection discriminator. */
                it_via_discriminator_t  discriminator;

                /* IANA local/remote Port nunbers, in network byte order. */
                it_iana_lr_port_t  lr_port;

            } conn_qual;

        } it_conn_qual_t;
```

**DESCRIPTION**

The *it_conn_qual_t* type is used by several routines in the API to encapsulate a Connection Qualifier. A Connection Qualifier is used by a Consumer on the Active side of the Connection establishment process in the *it_ep_connect* routine to target the remote Consumer that should be responding to the Connection establishment attempt. It is used on the Passive side of the Connection establishment process in the *it_listen_create* routine to steer incoming Connection Requests to an appropriate EVD for further processing.

Each Spigot on an IA can support one or more types of Connection Qualifier. All Spigots will support the IANA Port Number type of Connection Qualifier, regardless of which transport the IA that houses the Spigot is using. Which types of Connection Qualifier a Spigot supports can be determined using the *it_ia_query* routine.

In order to aid Consumers in writing portable applications that span platforms with different native byte orders, all Connection Qualifiers that are supported by the API with the exception of the VIA “connection discriminator” are required to be input to the API in network byte order, and will be output from the API in network byte order. (The VIA “connection discriminator” is defined to be an array of bytes, and hence is not affected by which native byte order a platform uses.)

**SEE ALSO**

*it_ep_connect*(), *it_listen_create*(), *it_ia_query*()

# it_context_t

**NAME**

it_context_t – structure describing a Consumer Context

**SYNOPSIS**

```
#include <it_api.h>

typedef union {
    void *    ptr;
    uint64_t  index;
} it_context_t;
```

**DESCRIPTION**

The *it_context_t* union describes storage definitions for the Consumer Context associated with an IT Object Handle.

*ptr* Storage space for an address pointer.

*index* Storage space for an unsigned 64-bit integer.

**SEE ALSO**

*it_get_consumer_context*(), *it_set_consumer_context*()

# it_dg_remote_ep_addr_t

## NAME

it_dg_remote_ep_addr_t – datagram transport Endpoint address

## SYNOPSIS

```
#include <it_api.h>

typedef struct
{
    it_addr_handle_t     addr;
    it_remote_ep_info_t  ep_info;
} it_ib_ud_addr_t;

typedef enum
{
    IT_DG_TYPE_IB_UD
} it_dg_type_t;

typedef struct
{
it_dg_type_t  type; /* IT_DG_TYPE_IB_UD */
    union {
        it_ib_ud_addr_t  ud;
    } addr;
} it_dg_remote_ep_addr_t;
```

## APPLICABILITY

The *it_dg_remote_ep_addr_t* data type is applicable only to Endpoints created for the UD Service Type.

## DESCRIPTION

For datagram transports, the Endpoint address is specified in DTO operations using the *it_dg_remote_ep_addr_t* data structure. The structure is intended to allow support of more than one datagram transport type.

The datagram transport type is specified as *type* in the *it_dg_remote_ep_addr_t* structure. In this revision of the API, only the InfiniBand Unreliable Datagram transport, IT_DG_TYPE_IB_UD, is supported.

For InfiniBand Unreliable Datagram, the transport-specific Endpoint address is contained in the *it_ib_ud_addr_t* sub-structure of the *it_dg_remote_ep_addr_t*. The components of the InfiniBand Unreliable Datagram Endpoint address are:

| | |
|---|---|
| addr | An Address Handle created by the Consumer using *it_address_handle_create*. |
| ep_info | An Endpoint Info structure (see *it_ep_attributes_t*) containing the Endpoint ID, *ud_ep_id*, and the Endpoint Key, *ud_ep_key*. The Consumer may make use of *it_ud_service_request* to obtain *ud_ep_id* and *ud_ep_key* or may obtain them by their own means. |

**EXTENDED DESCRIPTION**

For InfiniBand Unreliable Datagram, the Endpoint ID is equivalent to an InfiniBand QP number and the Endpoint Key is equivalent to an InfiniBand *Q_key*.

**FUTURE DIRECTIONS**

Support for Reliable Datagram Service Type may be provided in a future revision of this API.

**SEE ALSO**

*it_post_sendto*(), *it_post_recvfrom*(), *it_address_handle_create*(), *it_ud_service_request*(), *it_ep_attributes_t*

# it_dto_cookie_t

## NAME

it_dto_cookie_t – definition of implementation-opaque Consumer cookie

## SYNOPSIS

```
#include <it_api.h>

typedef uint64_t it_dto_cookie_t;
```

## DESCRIPTION

*it_dto_cookie_t* is an object that can be provided by the Consumer on every DTO or RMR operation and is returned to the Consumer in the corresponding DTO Completion Event (see *it_dto_events*) if a DTO Completion Event is generated (see *it_dto_flags_t*). The *it_dto_cookie_t* object is opaque to the Implementation and is returned unchanged to the Consumer in the DTO Completion Event corresponding to the posted DTO or RMR.

## SEE ALSO

*it_dto_events*, *it_dto_flags_t*, *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_events

**NAME**

DTO and RMR Link/Unlink Completion Event types

**SYNOPSIS**

```
#include <it_api.h>

typedef enum {
    IT_UD_IB_GRH_PRESENT  = 0x01
} it_dto_ud_flags_t;

typedef struct {
    it_event_type_t  event_number;
    it_evd_handle_t  evd;
    it_ep_handle_t   ep;
    it_dto_cookie_t  cookie;
    it_dto_status_t  dto_status;
    uint32_t         transferred_length;
    it_handle_t      unlinked_mr_handle;
} it_dto_cmpl_event_t;

typedef struct {
    it_event_type_t    event_number;
    it_evd_handle_t    evd;
    it_ep_handle_t     ep;
    it_dto_cookie_t    cookie;
    it_dto_status_t    dto_status;
    uint32_t           transferred_length;
    it_handle_t        unlinked_mr_handle;
    it_dto_ud_flags_t  flags;
    it_ud_ep_id_t      ud_ep_id;
    it_path_t          src_path;
} it_all_dto_cmpl_event_t;
```

**APPLICABILITY**

The *it_dto_cmpl_event_t* data type is applicable only to Endpoints created for the RC Service Type. The *it_all_dto_cmpl_event_t* data type is applicable to Endpoints of any Service Type.

**DESCRIPTION**

The DTO Completion Event Stream, IT_DTO_EVENT_STREAM, generates Events for all Data Transfer Operation completions as well as RMR Link and Unlink completions.

Unreliable Datagram Receive Completion Events provide additional data beyond that of the other DTO Completion Events. The additional data is large enough to warrant defining a much smaller Event structure for all other DTO operations usable by Consumers interested in conserving the memory footprint of their application.

With the exception of UD Receive completions, all DTO completions, including RMR Links and Unlinks, can be represented by the *it_dto_cmpl_event_t* structure.

UD Receive completions require use of the *it_all_dto_cmpl_event_t* structure. Consumers wishing to receive UD Receive and Send Completion Events on one Simple EVD or wishing to

handle all possible DTO completions with one Simple EVD must use the *it_all_dto_cmpl_event_t* structure or the encompassing *it_event_t* structure (see *it_event_t*). Failure to use the *it_all_dto_cmpl_event_t* structure or *it_event_t* structure for UD Receive Completion Events can result in program termination.

The *it_dto_cmpl_event_t* structure has the following members:

| | |
|---|---|
| *event_number* | Identifier of the Event type. Valid values:<br>IT_DTO_SEND_CMPL_EVENT,<br>IT_DTO_RC_RECV_CMPL_EVENT,<br>IT_DTO_RDMA_WRITE_CMPL_EVENT,<br>IT_DTO_RDMA_READ_CMPL_EVENT,<br>IT_RMR_BIND_CMPL_EVENT,<br>IT_RMR_LINK_CMPL_EVENT,<br>IT_DTO_FETCH_ADD_CMPL_EVENT,<br>IT_DTO_CMP_SWAP_CMPL_EVENT,<br>IT_LMR_LINK_CMPL_EVENT |
| *evd* | Handle for the Event Dispatcher where the Event was queued. |
| *ep* | For all Event types except IT_DTO_RC_RECV_CMPL_EVENT, this is the Handle for the Endpoint on which the DTO was posted. For the IT_DTO_RC_RECV_CMPL_EVENT, this is the Handle for the Endpoint targeted by an incoming Send operation that resulted in the completion of a Receive DTO. |
| *cookie* | Cookie that the Consumer associated with the DTO when it was posted. See *it_dto_cookie_t* for details. |
| *dto_status* | Status of completed DTO. |
| *transferred_length* | Length of transferred message. |
| *unlinked_mr_handle* | Handle of the LMR or RMR that was unlinked if not equal IT_NULL_HANDLE. Always IT_NULL_HANDLE for UD receives. |

See *it_dto_status_t* for values and definition of *dto_status*.

The *transferred_length* field indicates the amount of data transferred in a Receive operation. The content of this field is undefined for Send, RDMA Read, RDMA Write, RMR Link, and RMR Unlink operations. This field is also only valid if *dto_status* is IT_DTO_SUCCESS; otherwise, the contents are undefined.

The *it_all_dto_cmpl_event_t* structure has the following additional members:

| | |
|---|---|
| *flags*: | Flags indicating additional service specific information. |
| *ud_ep_id* | Remote Endpoint ID from incoming datagram. |
| *src_path* | Partial Source Path information from incoming datagram. |

The IT_DTO_UD_RECV_CMPL_EVENT *event_number* is an additional valid value only for the *it_all_dto_cmpl_event_t* structure or *it_event_t* structure.

The *flags* parameter indicates whether or not the InfiniBand Global Routing Header (GRH) is present in the first 40 bytes of the message payload. If the GRH is present, the IT_UD_IB_

GRH_PRESENT bit will be set in *flags*. If the GRH is not present (IT_UD_IB_GRH_PRESENT bit cleared in *flags)*, the first 40 bytes of the payload are undefined.

For an IT_DTO_UD_RECV_CMPL_EVENT, the *transferred_length* field includes the length of the transferred message plus 40 bytes regardless of the IT_UD_IB_GRH_PRESENT bit value.

The remote Endpoint ID, *ud_ep_id*, is derived from the incoming datagram. See *it_ep_attributes_t* for more details.

Partial Source address information is returned in a datagram Completion Event in the *src_path* structure element. See Application Usage below.

The *src_path* can hold more information than is returned in the Completion Event. The members of the *it_path_t* structure that are pertinent to a datagram Completion Event are listed in the table below. For each member, the corresponding Infiniband datagram addressing information that the member corresponds to is also identified. For a detailed explanation of the semantics associated with the Datagram addressing information, see Chapter 11.4.2.1 Poll For Completion in [IB-R1.2].

| **it_path_t Member** | **Unreliable Datagram Completion Addressing Information** |
|---|---|
| *ib.sl* | Service level |
| *ib.remote_port_lid* | Source LID |

IT_DTO_EVENT_STREAM Events may or may not cause Notification depending on the use of DTO flags (see *it_dto_flags_t*). See *it_evd_create* for details of Notification.

Default overflow behavior of an IT_DTO_EVENT_STREAM SEVD is overflow Notification enabled (for an IA that supports DTO EVD overflow detection, see below). The behavior of the SEVD is equivalent to IT_EVD_OVERFLOW_DEFAULT cleared and IT_EVD_OVERFLOW_NOTIFY set. Once an IT_DTO_EVENT_STREAM SEVD overflows, it cannot be rearmed. See *it_evd_create* for details of overflow detection.

Overflow of an IT_DTO_EVENT_STREAM SEVD is catastrophic for the associated Endpoint or Endpoints. Each Endpoint is transitioned to the IT_EP_STATE_NONOPERATIONAL state as defined in *it_ep_state_t*. The Endpoints are unrecoverable; *it_ep_free* must be called for all Endpoints sharing the same SEVD on which the overflow occurred.

Overflow detection of DTO EVDs is only supported when the *dto_evd_overflow_detection* flag found in *it_ia_info_t* is IT_TRUE.

**APPLICATION USAGE**

Within an IT_DTO_UD_RECV_CMPL_EVENT Event, the *src_path* member returned contains insufficient information to identify the remote Endpoint. To resolve the remote Endpoint Path, the user should pass *src_path* returned in this Event to *it_address_handle_create* with the IT_AH_PATH_COMPLETE bit cleared. *it_address_handle_create* will complete the resolution of the Path.

Overflow of an EVD containing the DTO Event Stream can result in indeterminate behavior when the IA does not support overflow detection (*dto_evd_overflow_detection* is IT_FALSE). On such an IA, the Consumer should structure their ULP to avoid any possibility of this occurring. Indeterminate behavior is constrained to the EVD concerned; overflow on a DTO

Event Stream associated with a particular EVD must not affect any other EVD or Endpoint associated with other EVDs.

No ordering guarantee exists between generation of IT_CM_MSG_EVENT_STREAM Events and generation of IT_DTO_EVENT_STREAM Events. For example, when an Endpoint is disconnected, the IT_CM_MSG_CONN_DISCONNECT_EVENT can be generated on the IT_CM_MSG_EVENT_STREAM Event Stream either before or after the flushed Events (i.e., Events with an IT_DTO_ERR_FLUSHED status) for pending DTOs are generated on the IT_DTO_EVENT_STREAM Event Stream.

Receipt of a successful incoming *it_post_send_and_unlink* DTO will unlink a local LMR or RMR and generate an *unlinked_mr_handle* not equal to IT_NULL_HANDLE in the Event. It is the Consumer's responsibility to correlate the *unlinked_mr_handle* with a local LMR or RMR.

## FUTURE DIRECTIONS

An additional detailed status code may be added to the DTO Completion Event data structures.

## 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_dto_status_t*, *it_dto_flags_t*, *it_event_t*, *it_evd_create*(), *it_evd_wait*(), *it_address_handle_create*(), *it_ep_state_t*, *it_ep_free*(), *it_ep_reset*(), *it_path_t*, *it_dto_cookie_t*, *it_ia_info_t*

# it_dto_flags_t

## NAME

it_dto_flags_t – flags for Send, Receive, RDMA Read, RDMA Write, RMR Link, RMR Unlink, LMR Link, LMR Unlink, and Atomic Work Requests

## SYNOPSIS

```
#include <it_api.h>

typedef enum
{
    /* If flag set, completion generates a local event. */
    IT_COMPLETION_FLAG  = 0x01,

    /* If flag set, completion causes local Notification. */
    IT_NOTIFY_FLAG  = 0x02,

    /* If flag set, receipt of DTO at remote will cause Notification
       at remote. */
    IT_SOLICITED_WAIT_FLAG  = 0x04,

    /* If flag set, a WR posted to an EP's SQ will not start if
       RDMA Reads posted previously to the EP are not complete. */
    IT_BARRIER_FENCE_FLAG  = 0x08,

    /* If flag set, an LMR Unlink or RMR Unlink WR posted to an EP
       will not start if WRs posted previously to the EP's SQ are
       not complete. */
    IT_UNLINK_FENCE_FLAG  = 0x10,

    /* If flag set, the local data sink is unlinked upon WR
       completion */
    IT_UNLINK_LOCAL_SINK = 0x20,

    /* If flag set, attempt to enqueue this WR for later processing */
    IT_COALESCE_WR_FLAG = 0x40
} it_dto_flags_t;
```

## DESCRIPTION

*it_dto_flags* — Flags for posted WRs: Send, Receive, RDMA Read, RDMA Write, RMR Link, RMR Unlink, LMR Link, LMR Unlink, Atomic.

Values for *it_dto_flags* are constructed by a bitwise-inclusive OR of flags from the following discussion.

Any combination of the following may be used subject to restrictions as noted:

### IT_COMPLETION_FLAG

If set, generate a Completion Event for this WR, else do not generate a Completion Event unless there is an error.

If not set, then the completion of a subsequent WR on the same Work Queue of the same Endpoint with this flag set or with error completion will indicate the successful completion of prior WR(s) with this flag cleared.

#### Restrictions

IT_COMPLETION_FLAG may be set or cleared only for Send, RDMA Write, RDMA Read, RMR Link, and RMR Unlink operations on a Reliable Connection Service Type, and may be set or cleared only for Send operation on an Unreliable Datagram Service Type.

IT_COMPLETION_FLAG must be set for all Receive WRs on all Service Types (Reliable Connection and Unreliable Datagram). Posting a Receive WR with IT_COMPLETION_FLAG cleared is an error.

### IT_NOTIFY_FLAG

If set, generate Notification of completion of the WR. If there is an error, Notification of completion will be generated regardless of the IT_NOTIFY_FLAG value.

#### Restrictions

IT_NOTIFY_FLAG may be set or cleared on all WRs on a Reliable Connected Service Type, and may be set or cleared on Send and Receive operations on an Unreliable Datagram Service Type. It is an error to set IT_NOTIFY_FLAG if IT_COMPLETION_FLAG is clear.

A completion will be generated with the Notification for a Receive WR if the matching received Send message had been posted at the remote with the IT_SOLICITED_WAIT_FLAG set regardless of the IT_NOTIFY_FLAG of the posted Receive WR.

### IT_SOLICITED_WAIT_FLAG

If set, the Send WR will request completion Notification for the matching Receive on the other side of the Connection or, for Unreliable Datagram, for the matching Receive at the remote datagram Endpoint.

#### Restrictions

IT_SOLICITED_WAIT_FLAG is supported only for Send operations for all Service Types. It is an error to specify IT_SOLICITED_WAIT_FLAG on other operations.

If set, requests Notification of completion of the matching remote Receive WR regardless of the value of the IT_NOTIFY_FLAG on the Receive WR.

### IT_BARRIER_FENCE_FLAG

If set, a Work Request posted to an Endpoint’s SQ will not execute until all RDMA Read requests previously posted to the Endpoint have completed.

#### Restrictions

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.

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.

### IT_UNLINK_FENCE_FLAG

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.

#### Restrictions

If the *it_ia_info_t* attribute *unlink_fence_support* is IT_FALSE, then it is an error to set this flag.

The IT_UNLINK_FENCE_FLAG may be set only for LMR Unlink (*it_lmr_unlink*) or RMR Unlink (*it_rmr_unlink*) operations. For InfiniBand only, IT_UNLINK_FENCE_FLAG may not be used with Wide RMRs.

### IT_UNLINK_LOCAL_SINK

If set, the local data sink is unlinked after the WR has completed.

#### Restrictions

If the *it_ia_info_t* attribute *rdma_read_local_extensions* is IT_FALSE, then it is an error to set this flag.

The IT_UNLINK_LOCAL_SINK may be set only for RDMA Read (*it_post_rdma_read*) or RDMA Read to RMR (*it_post_rdma_read_to_rmr*) operations and may only be used if *num_segments* is equal to one.

### IT_COALESCE_WR_FLAG

If this bit is set in the *dto_flags* 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.

If this bit is clear in the *dto_flags* 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.

#### Restrictions

While an operation is enqueued for later processing it consumes space in the Endpoint Work Queue.

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_recv_dtos* members, the routine which posts the operation to the Work Queue may return an IT_ERR_TOO_MANY_POSTS error.

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.

**EXTENDED DESCRIPTION**

The following table lists all Work Requests and details the legal *it_dto_flags* values for each.

| **Work Request** | **Legal it_dto_flags Combinations** |
|---|---|
| *it_post_send* | IT_UNLINK_FENCE_FLAG may not be used.<br>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. |
| *it_post_send_and_unlink* | IT_UNLINK_FENCE_FLAG may not be used.<br>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. |
| *it_post_sendto* | IT_BARRIER_FENCE_FLAG may not be used.<br>IT_UNLINK_FENCE_FLAG may not be used.<br>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. |
| *it_post_recv* | IT_COMPLETION_FLAG must be specified.<br>IT_BARRIER_FENCE_FLAG may not be used.<br>IT_UNLINK_FENCE_FLAG may not be used.<br>IT_SOLICITED_WAIT_FLAG may not be used.<br>IT_UNLINK_LOCAL_SINK may not be used. All other possible combinations of the remaining flags are legal. |
| *it_post_recvfrom* | IT_COMPLETION_FLAG must be specified.<br>IT_BARRIER_FENCE_FLAG may not be used.<br>IT_UNLINK_FENCE_FLAG may not be used.<br>IT_SOLICITED_WAIT_FLAG may not be used.<br>IT_UNLINK_LOCAL_SINK may not be used. All other possible combinations of the remaining flags are legal. |
| *it_post_rdma_read* | IT_SOLICITED_WAIT_FLAG may not be used.<br>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. |

| | |
|---|---|
| *it_post_rdma_read_to_rmr* | IT_SOLICITED_WAIT_FLAG may not be used.<br>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. |
| *it_post_rdma_write* | IT_SOLICITED_WAIT_FLAG may not be used.<br>IT_UNLINK_FENCE_FLAG may not be used.<br>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. |
| *it_rmr_link* | IT_SOLICITED_WAIT_FLAG may not be used.<br>IT_UNLINK_FENCE_FLAG may not be used.<br>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. |
| *it_rmr_unlink* | IT_SOLICITED_WAIT_FLAG may not be used.<br>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. |
| *it_post_atomic* | IT_SOLICITED_WAIT_FLAG may not be used.<br>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. |
| *it_lmr_link* | IT_SOLICITED_WAIT_FLAG may not be used.<br>IT_UNLINK_FENCE_FLAG may not be used.<br>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. |
| *it_lmr_unlink* | IT_SOLICITED_WAIT_FLAG may not be used.<br>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 |

As stated in the Description section, IT_COMPLETION_FLAG must be set on *Recv* and *Recvfrom* WRs.

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.

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/unsignaled 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 |

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 |

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/unsignaled 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 |

**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.

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.

See Application Usage in *it_ep_state_t* for discussion of flushing WR completions when the WRs have IT_COMPLETION_FLAG cleared.

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.

**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

## NAME

it_dto_status_t – definition of Work Request (DTO, LMR, or RMR operation) completion status

## SYNOPSIS

```
#include <it_api.h>

typedef enum {
    IT_DTO_SUCCESS                = 0,
    IT_DTO_ERR_LOCAL_LENGTH       = 1,
    IT_DTO_ERR_LOCAL_EP           = 2,
    IT_DTO_ERR_LOCAL_PROTECTION   = 3,
    IT_DTO_ERR_FLUSHED            = 4,
    IT_RMR_OPERATION_FAILED       = 5,
    IT_DTO_ERR_BAD_RESPONSE       = 6,
    IT_DTO_ERR_REMOTE_ACCESS      = 7,
    IT_DTO_ERR_REMOTE_RESPONDER   = 8,
    IT_DTO_ERR_TRANSPORT          = 9,
    IT_DTO_ERR_RECEIVER_NOT_READY = 10,
    IT_DTO_ERR_PARTIAL_PACKET     = 11,
    IT_DTO_ERR_LOCAL_MM_OPERATION = 12
} it_dto_status_t;
```

## DESCRIPTION

Any successfully initiated Work Request (i.e., Send, Receive, RDMA Read, RDMA Write, Atomic, LMR Link, LMR Unlink, RMR Link, or RMR Unlink) can return its completion status asynchronously via an Event enqueued on an SEVD. For some WRs, the Consumer can control whether an Event is generated via the IT_COMPLETION_FLAG (see *it_dto_flags_t*). If an Event is generated, the completion status is contained in the *it_dto_status_t*.

If the completion status is anything other than IT_DTO_SUCCESS for a Reliable Connected Endpoint, the Connection will be broken.

The table below enumerates all of the allowed values for *it_dto_status_t*. For each value, a 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<br>Send<br>Recv<br>RDMA Read<br>RDMA Write<br>LMR Link<br>LMR Unlink<br>RMR Link<br>RMR Unlink | Sendto<br>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 *max_dto_payload_size* for the Endpoint | Recv | Recvfrom |
| IT_DTO_ERR_ LOCAL_LENGTH | The length of the outgoing DTO was larger than *max_dto_payload_size* for the Endpoint. | Send<br>RDMA Read<br>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<br>Recv<br>RDMA Read<br>RDMA Write<br>RMR Link<br>RMR Unlink<br>Atomic<br>LMR Link<br>LMR Unlink | Sendto<br>Recvfrom |

| it_dto_ status_t Value | Description | Applicable to Work Requests | |
|---|---|---|---|
| | | RC | UD |
| IT_DTO_ERR_ LOCAL_ PROTECTION | 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:<br>• The LMR handle or local RMR handle was invalid.<br>• The LMR or local RMR was not in the linked state.<br>• The address range specified by *addr* and *length* was outside the bounds of the LMR or local RMR.<br>• 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.<br>• For *it_post_rdma_read_to_rmr*, the local Narrow RMR was associated with an Endpoint different from the Endpoint to which the DTO was posted.<br>• An attempt was made to access the LMR or local RMR in a way that conflicted with its access permissions.<br>• For the RC service, the local Endpoint was not enabled for the incoming RDMA operation.<br>• The LMR handle was the Direct LMR Handle and the EP is not a Privileged Mode Endpoint (*priv_ops_enable* was not set on EP creation).<br>• For *it_post_rdma_read* or *it_post_rdma_read_to_rmr* with the IT_UNLINK_LOCAL_SINK DTO flag set, the local sink LMR had an RMR still bound to it.<br>• An incoming *it_post_send_and_unlink* 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<br>Recv<br>RDMA Read<br>RDMA Write<br>Atomic | Sendto<br>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<br>Recv<br>RDMA Read<br>RDMA Write<br>RMR Link<br>RMR Unlink<br>LMR Link<br>LMR Unlink<br>Atomic | Sendto<br>Recvfrom |
| IT_RMR_ OPERATION_ FAILED | An RMR operation failed due to a protection violation. Possible causes for this error are as follows.<br>For *it_rmr_link*:<br>• The RMR handle was invalid.<br>• The Protection Zones associated with the RMR, LMR, and EP to which the operation was posted didn't match.<br>• The RMR was a Narrow RMR that was already linked.<br>• The address range specified by the *addr* and *length* arguments was outside the bounds of the LMR.<br>• Remote write access was requested for the RMR, but the LMR did not allow local write access (InfiniBand only).<br>• An attempt was made to grant access through the RMR that conflicted with the access allowed by either the LMR or the EP.<br>For *it_rmr_unlink*:<br>• The RMR handle was invalid<br>• The Protection Zones associated with the RMR and EP to which the operation was posted didn't match.<br>• 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<br>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<br>RDMA Read<br>RDMA Write<br>Atomic | N/A |

| it_dto_ status_t Value | Description | Applicable to Work Requests | |
|---|---|---|---|
| | | RC | UD |
| IT_DTO_ERR_ REMOTE_ACCESS | 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:<br>• *rmr_context* did not represent a valid and linked remote buffer.<br>• The Protection Zone associated with the remote buffer didn't match the Protection Zone of the remote Endpoint.<br>• The remote buffer was a Narrow RMR that was linked via a remote Endpoint not associated with the Connection.<br>• The address range specified by *rdma_addr* and the length implicitly given by *local_segments* and *num_segments* was outside the bounds of the remote buffer.<br>• An attempt was made to access the remote buffer in a way that conflicted with its access permissions.<br>• An attempt was made to access the remote buffer in a way that conflicted with the access permissions of the remote Endpoint.<br>• The remote Endpoint was not enabled for the incoming RDMA operation. | RDMA Read<br>RDMA Write<br>Atomic | N/A |
| IT_DTO_ERR_ REMOTE_ RESPONDER | 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. | Send<br>RDMA Read<br>RDMA Write<br>Atomic | N/A |
| IT_DTO_ERR_ TRANSPORT | 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. | Send<br>Receive<br>RDMA Read<br>RDMA Write<br>Atomic | N/A |
| IT_DTO_ERR_ RECEIVER_NOT_ READY | The DTO operation could not be processed because the responding side repeatedly indicated that it had no resources to do so. | Send<br>RDMA Read<br>RDMA Write<br>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 | A local memory management operation failed due to one of the following causes.<br>For *it_rmr_link*:<br>• The LMR to link to used Relative Addressing.<br>• A Narrow RMR was provided and the *length* argument was zero (only for InfiniBand).<br>For *it_lmr_link*:<br>• The LMR handle was invalid.<br>• The PZ of LMR and EP did not match or *privs* were invalid.<br>• The LMR was already in linked state.<br>• The LMR Handle was the Direct LMR Handle.<br>• The number of IOBL elements exceeded the *iobl_num_elts* LMR attribute, the IOBL included elements of an unsupported length, the IOBL had an invalid First-Byte Offset (*fbo*), or if the *length* of the LMR exceeded the length of the IOBL or if *length* equaled zero.<br>• 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 (*addr*) modulo the length (*elt_len*) of the first IOBL element did not equal the First-Byte Offset (*fbo*) in the IOBL.<br>• The EP was not in Privileged Mode.<br>For *it_lmr_unlink*:<br>• The LMR handle was invalid.<br>• The LMR was not in linked state.<br>• The LMR Handle was the Direct LMR Handle.<br>• The PZ of the LMR did not match the PZ of the EP.<br>• The LMR had at least one RMR bound to it (via *it_rmr_link*).<br>• The LMR was in the shared state.<br>• The LMR could not be unlinked due to the way it was created (only for InfiniBand). | RMR Link<br>LMR Link<br>LMR Unlink | N/A |

**EXTENDED DESCRIPTION**

For the InfiniBand Transport, the following table maps the values in the *it_dto_status_t* enumeration to their corresponding "Completion Return Status" values as specified in Chapter 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 |

For the iWARP Transport, the following table maps the values in the *it_dto_status_t* enumeration to the Completion Status Codes in Section 9.5.2 of [VERBS-RDMAC].

| **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 | STag to Invalidate had Invalid PD or Access Rights, STag Not In Invalid State, Invalid Region (Bind), Invalid Window (Bind) |

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.) |

**SEE ALSO**

*it_post_atomic*(), *it_post_send*(), *it_post_sendto*(), *it_post_recv*(), *it_post_recvfrom*(), *it_post_rdma_read*(), *it_post_rdma_read_to_rmr*(), *it_post_rdma_write*(), *it_rmr_link*(), *it_rmr_unlink*()

# it_ep_attributes_t

**NAME**

it_ep_attributes – Endpoint attributes

**SYNOPSIS**

```
#include <it_api.h>

typedef  uint32_t  it_ud_ep_id_t;
typedef  uint32_t  it_ud_ep_key_t;

typedef enum {
    IT_EP_PARAM_ALL                 = 0x00000001,
    IT_EP_PARAM_IA                  = 0x00000002,
    IT_EP_PARAM_SPIGOT              = 0x00000004,
    IT_EP_PARAM_STATE               = 0x00000008,
    IT_EP_PARAM_SERV_TYPE           = 0x00000010,
    IT_EP_PARAM_PATH                = 0x00000020,
    IT_EP_PARAM_PZ                  = 0x00000040,
    IT_EP_PARAM_REQ_SEVD            = 0x00000080,
    IT_EP_PARAM_RECV_SEVD           = 0x00000100,
    IT_EP_PARAM_CONN_SEVD           = 0x00000200,
    IT_EP_PARAM_RDMA_RD_ENABLE      = 0x00000400,
    IT_EP_PARAM_RDMA_WR_ENABLE      = 0x00000800,
    IT_EP_PARAM_MAX_RDMA_READ_SEG   = 0x00001000,
    IT_EP_PARAM_MAX_RDMA_WRITE_SEG  = 0x00002000,
    IT_EP_PARAM_MAX_IRD             = 0x00004000,
    IT_EP_PARAM_MAX_ORD             = 0x00008000,
    IT_EP_PARAM_EP_ID               = 0x00010000,
    IT_EP_PARAM_EP_KEY              = 0x00020000,
    IT_EP_PARAM_MAX_PAYLOAD         = 0x00040000,
    IT_EP_PARAM_MAX_REQ_DTO         = 0x00080000,
    IT_EP_PARAM_MAX_RECV_DTO        = 0x00100000,
    IT_EP_PARAM_MAX_SEND_SEG        = 0x00200000,
    IT_EP_PARAM_MAX_RECV_SEG        = 0x00400000,
    IT_EP_PARAM_SRQ                 = 0x00800000,
    IT_EP_PARAM_SOFT_HI_WATERMARK   = 0x01000000,
    IT_EP_PARAM_HARD_HI_WATERMARK   = 0x02000000,
    IT_EP_PARAM_ATOMICS_ENABLE      = 0x04000000,
    IT_EP_PARAM_PRIV_OPS_ENABLE     = 0x08000000
} it_ep_param_mask_t;

/*
 * The it_ep_param_mask_t value in the comment beside or
 * following each attribute is the mask value used to select
 * the attribute in the it_ep_query and it_ep_modify calls
 */
typedef struct {
    it_boolean_t  rdma_read_enable;
        /* IT_EP_PARAM_RDMA_RD_ENABLE */
    it_boolean_t  rdma_write_enable;
        /* IT_EP_PARAM_RDMA_WR_ENABLE */
    size_t  max_rdma_read_segments;
```

```
            /* IT_EP_PARAM_MAX_RDMA_READ_SEG */
        size_t  max_rdma_write_segments;
            /* IT_EP_PARAM_MAX_RDMA_WRITE_SEG */
        uint32_t  rdma_read_ird;
            /* IT_EP_PARAM_MAX_IRD */
        uint32_t  rdma_read_ord;
            /* IT_EP_PARAM_MAX_ORD */
        it_srq_handle_t  srq;
            /* IT_EP_PARAM_SRQ */
        size_t  soft_hi_watermark;
            /* IT_EP_PARAM_SOFT_HI_WATERMARK */
        size_t  hard_hi_watermark;
            /* IT_EP_PARAM_HARD_HI_WATERMARK */
        it_boolean_t  atomics_enable;
            /* IT_EP_PARAM_ATOMICS_ENABLE */

    } it_rc_only_attributes_t;

    #define IT_HARD_HI_WATERMARK_DISABLE  ((size_t) -1)

    typedef struct {
        it_ud_ep_id_t   ud_ep_id;   /* IT_EP_PARAM_EP_ID */
        it_ud_ep_key_t  ud_ep_key;  /* IT_EP_PARAM_EP_KEY */
    } it_remote_ep_info_t;

    typedef struct {
        it_remote_ep_info_t  ep_info;

    } it_ud_only_attributes_t;

    typedef union {
        it_rc_only_attributes_t  rc;
        it_ud_only_attributes_t  ud;
    } it_service_attributes_t;

    typedef struct {
        size_t  max_dto_payload_size;  /* IT_EP_PARAM_MAX_PAYLOAD */
        size_t  max_request_dtos;      /* IT_EP_PARAM_MAX_REQ_DTO */
        size_t  max_recv_dtos;         /* IT_EP_PARAM_MAX_RECV_DTO */
        size_t  max_send_segments;     /* IT_EP_PARAM_MAX_SEND_SEG */
        size_t  max_recv_segments;     /* IT_EP_PARAM_MAX_RECV_SEG */

        it_service_attributes_t  srv;

        it_boolean_t priv_ops_enable;  /* IT_EP_PARAM_PRIV_OPS_ENABLE */
    } it_ep_attributes_t;
```

## DESCRIPTION

| | |
|---|---|
| it_ep_attributes | List of Endpoint attributes. The *it_service_attributes_t* union elements are discriminated by *service_type* found in the *it_ep_param_t* structure in the *it_ep_query* reference page. Mask values for query and modify of Endpoint attributes appear as comments to each attribute. |

| Attribute | Description | Service Type | Modifiable? |
|---|---|---|---|
| *max_dto_payload_size* | 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.<br>For RC only, it also specifies the maximum payload data size for RDMA Reads and Writes posted on the Endpoint. | UD and RC | For RC, only when Endpoint is in the IT_EP_STATE_ NONOPERATIONAL or in the IT_EP_STATE_ UNCONNECTED states. For UD, only on creation. |
| *max_request_dtos* | 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 *it_post_send*, *it_post_rdma_read*, etc., routines. | UD and RC | Subject to the setting of the *ep_work_queues_resizable* field in the *it_ia_info_t* for this IA. |
| *max_recv_dtos* | Maximum number of outstanding *Recv* or *Recvfrom* 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 *it_post_recv* or *it_post_recvfrom* routines.<br>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 *it_ep_query* and any attempt to modify the attribute with *it_ep_modify* shall return an error.<br>For an Endpoint having an associated S-RQ, a corresponding *max_recv_dtos* attribute exists as an S-RQ attribute. | UD and RC | Subject to the setting of the *ep_work_queues_resizable* field in the *it_ia_info_t* for this IA. |
| *max_send_segments* | Maximum number of data segments for a local buffer that the Consumer specifies for a posted Send or Sendto DTO for the Endpoint. | UD and RC | Only on creation. |

| Attribute | Description | Service Type | Modifiable? |
|---|---|---|---|
| *max_recv_segments* | Maximum number of data segments for a local buffer that the Consumer specifies for a posted *Recv* or *Recvfrom* DTO for the Endpoint.<br>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 *it_ep_query* and any attempt to modify the attribute with *it_ep_modify* shall return an error.<br>For an Endpoint having an associated S-RQ, a corresponding *max_recv_segments* attribute exists as an S-RQ attribute. | UD and RC | Only on creation. |
| *ud_ep_id* | Local Endpoint ID for this Endpoint. | UD only | Never – this is a READ-ONLY attribute. |
| *ud_ep_key* | Local Endpoint key for this Endpoint. | UD only | In any state. |
| *rdma_read_enable* | Flag allowing Consumer to enable or disable incoming RDMA Read operations on this Endpoint. | RC only | If the *it_ia_info_t* boolean *ep_rdma_enables_ modifiable* is IT_TRUE, then may be modified in any state. Else, only on creation. |
| *rdma_write_enable* | Flag allowing Consumer to enable or disable incoming RDMA Write operations on this Endpoint.<br>For posting RDMA Read DTOs to this Endpoint, this attribute must be IT_TRUE if the IA attribute *rdma_read_requires_remote_write* equals IT_TRUE. | RC only | If the *it_ia_info_t* boolean *ep_rdma_enables_ modifiable* is IT_TRUE, then may be modified in any state. Else, only on creation. |
| *max_rdma_read_ segments* | 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. |
| *max_rdma_write_ segments* | 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? |
|---|---|---|---|
| *rdma_read_ird* | Maximum number of incoming RDMA Reads from the remote side of the connected Endpoint that can be outstanding simultaneously.<br>If the IA supports Atomics, then this attribute defines the maximum number of incoming Atomics and RDMA Reads that can be outstanding simultaneously. | RC only | When Endpoint is in the IT_EP_STATE_ UNCONNECTED or IT_EP_STATE_ ACTIVE2_ CONNECTION_PENDING state. |
| *rdma_read_ord* | 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.<br>If the IA supports Atomics, then this attribute defines the maximum number of outgoing Atomics and RDMA Reads that can be outstanding simultaneously. | RC only | When Endpoint is in the IT_EP_STATE_ UNCONNECTED or IT_EP_STATE_ ACTIVE2_ CONNECTION_PENDING or IT_EP_STATE_ CONNECTED state. |
| *srq* | The handle for the Shared Receive Queue (if any) associated with the Endpoint. If the Endpoint has no S-RQ, *it_ep_query* will return the value IT_NULL_HANDLE for this attribute. | RC only | Only on creation. |
| *soft_hi_watermark* | The current value of the Endpoint Soft High Watermark threshold.<br>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.<br>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 | RC only | In any state. |

| Attribute | Description | Service Type | Modifiable? |
|---|---|---|---|
| | 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”.)<br><br>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 *it_ep_modify* 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 *it_ep_free*, 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 *it_affiliated_event_t* for details. Once the Endpoint Soft High Watermark mechanism is disarmed, it can only be rearmed by calling *it_ep_modify* to establish a new *soft_hi_watermark* value.<br><br>If the IA does not support the Endpoint Soft High Watermark mechanism, or if the Endpoint does not have an associated S-RQ, *it_ep_query* shall return zero for this attribute.<br><br>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.<br><br>If the Consumer attempts to set this attribute to a value of zero while the Endpoint Soft High Watermark mechanism is armed, the | | |

| Attribute | Description | Service Type | Modifiable? |
|---|---|---|---|
| | Implementation shall ignore the attempt to change the value, and the mechanism shall stay armed. | | |
| *hard_hi_watermark* | The current value of the Endpoint Hard High Watermark.<br>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 IT_HARD_HI_WATERMARK_DISABLE 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 IT_HARD_HI_WATERMARK_DISABLE 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 IT_EP_STATE_NONOPERATIONAL state.<br>If the IA does not support the Endpoint Hard High Watermark mechanism, or if the Endpoint does not have an associated S-RQ, *it_ep_query* shall return zero for this attribute. | RC only | In any state. |
| *ep_state* | The current state of the Endpoint. | UD and RC | Never – this is a READ-ONLY attribute. |

| Attribute | Description | Service Type | Modifiable? |
|---|---|---|---|
| *atomics_enable* | Flag allowing Consumer to enable or disable incoming Atomics operations on this Endpoint. Only usable if *it_ia_info_t* boolean *atomic_support* is IT_TRUE. | RC only | If the *it_ia_info_t* boolean *ep_rdma_enables_ modifiable* is IT_TRUE, then may be modified in any state. Else, only on creation. |
| *priv_ops_enable* | Flag allowing Consumer to request creation of a Privileged Mode Endpoint. | UD and RC | Only on creation. |

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.

Exceeding *max_request_dtos* or *max_recv_dtos* using the post DTO and post RMR operations will result in the post operation returning an error or completing in error.

Posting more RDMA Read operations than specified in *rdma_read_ord* is not an error and will have no adverse effects.

**EXTENDED DESCRIPTION**

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.

**FUTURE DIRECTIONS**

Some new Service Types may be added in the future.

**SEE ALSO**

*it_ep_rc_create*(), *it_ep_ud_create*(), *it_ep_query*(), *it_ep_modify*(), *it_ia_info_t*

# it_ep_state_t

**NAME**

it_ep_state_t – RC and UD Endpoint state type definition

**SYNOPSIS**

```
#include <it_api.h>

typedef enum
{
    IT_EP_STATE_UNCONNECTED                   = 0,
    IT_EP_STATE_ACTIVE1_CONNECTION_PENDING    = 1,
    IT_EP_STATE_ACTIVE2_CONNECTION_PENDING    = 2,
    IT_EP_STATE_PASSIVE_CONNECTION_PENDING    = 3,
    IT_EP_STATE_CONNECTED                     = 4,
    IT_EP_STATE_NONOPERATIONAL                = 5,
    IT_EP_STATE_PASSIVE_WAIT_RDMA_TRANS_REQ   = 6
} it_ep_state_rc_t;

typedef enum
{
    IT_EP_STATE_UD_NONOPERATIONAL  = 0,
    IT_EP_STATE_UD_OPERATIONAL     = 1
} it_ep_state_ud_t;

typedef union
{
    it_ep_state_rc_t  rc;
    it_ep_state_ud_t  ud;
} it_ep_state_t;
```

**DESCRIPTION**

The following table identifies and describes the RC Endpoint states. The Transport-Independent Interface (TII) and the Transport-Dependent Interface (TDI) use these states in a consistent way. For each state, the table lists the API routines that can be legally applied to an RC Endpoint in that state.

Whenever an Endpoint transitions its state, at most one Event is generated for that transition. The Endpoints state transitions before the Communication Management Message Event is enqueued. This guarantees that the Consumer can only dequeue Events after the state transition has occurred. Subsequent state transitions and their related Events will occur regardless of whether the Consumer is dequeueing the Events.

| 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 *it_ep_rc_create* it is in this state. | *it_ep_accept*, *it_ep_connect*, *it_ep_free*, *it_ep_modify*, *it_ep_query*, *it_get_consumer_context*, *it_get_handle_type*, *it_post_recv*, *it_set_consumer_context* *it_socket_convert* |
| IT_EP_STATE_ACTIVE1_ CONNECTION_PENDING | The Active side Endpoint has initiated a Connection establishment. | *it_ep_free*, *it_ep_modify*, *it_ep_query*, *it_get_consumer_context*, *it_get_handle_type*, *it_post_recv*, *it_set_consumer_context*, *it_ep_disconnect* |
| 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. | *it_ep_accept*, *it_ep_free*, *it_ep_modify*, *it_ep_query*, *it_get_consumer_context*, *it_get_handle_type*, *it_post_recv*, *it_reject*, *it_set_consumer_context*, *it_ep_disconnect* |
| IT_EP_STATE_PASSIVE_ CONNECTION_PENDING | The Passive side Consumer has called *it_ep_accept* 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. | *it_ep_free*, *it_ep_modify*, *it_ep_query*, *it_get_consumer_context*, *it_get_handle_type*, *it_post_recv*, *it_set_consumer_context*, *it_ep_disconnect* |

| 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. | *it_ep_disconnect*, *it_ep_free*, *it_ep_modify*, *it_ep_query*, *it_get_consumer_context*, *it_get_handle_type*, *it_post_atomic*, *it_post_rdma_read*, *it_post_rdma_read_to_rmr*, *it_post_rdma_write*, *it_post_recv*, *it_post_send*, *it_rmr_link*, *it_rmr_unlink*, *it_set_consumer_context* |
| 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. | *it_ep_disconnect*, *it_ep_free*, *it_ep_modify*, *it_ep_query*, *it_ep_reset*, *it_get_consumer_context*, *it_get_handle_type*, *it_post_atomic*, *it_post_rdma_read*, *it_post_rdma_read_to_rmr*, *it_post_rdma_write*, *it_post_recv*, *it_post_send*, *it_rmr_link*, *it_rmr_unlink*, *it_set_consumer_context* |
| IT_EP_STATE_PASSIVE_WAIT_ RDMA_TRANS_REQ | The Conversion Initiator has called *it_socket_convert*, 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. | *it_ep_free*, *it_ep_modify*, *it_ep_query*, *it_get_consumer_context*, *it_get_handle_type*, *it_post_recv*, *it_set_consumer_context*, *it_ep_disconnect* |

The following table identifies the RC Endpoint state transitions:

| State | Event | Transition to |
|---|---|---|
| IT_EP_STATE_ UNCONNECTED | *it_ep_connect* called | IT_EP_STATE_ACTIVE1_ CONNECTION_PENDING |

| State | Event | Transition to |
|---|---|---|
| | *it_ep_accept* 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 *it_reject* called on Remote side | IT_EP_STATE_NONOPERATIONAL |
| | *it_ep_disconnect* 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 |
| | *it_ep_disconnect* called | IT_EP_STATE_NONOPERATIONAL |
| IT_EP_STATE_PASSIVE_ CONNECTION_PENDING | Completion of Connection establishment | IT_EP_STATE_CONNECTED |
| | Local or Remote error, or *it_reject* called on Active side | IT_EP_STATE_NONOPERATIONAL |
| | *it_ep_disconnect* 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 |
| | *it_ep_disconnect* called | IT_EP_STATE_NONOPERATIONAL |
| IT_EP_STATE_CONNECTED | Local Error, or *it_ep_disconnect* called, or Remote error or Remote disconnect | IT_EP_STATE_NONOPERATIONAL |
| IT_EP_STATE_ NONOPERATIONAL | *it_ep_reset* called | IT_EP_STATE_UNCONNECTED |

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. | *it_ep_free*, *it_ep_modify*, *it_ep_query*, *it_get_consumer_context*, *it_get_handle_type*, *it_post_recvfrom*, *it_post_sendto*, *it_set_consumer_context*, *it_ep_reset* |
| IT_EP_STATE_UD_OPERATIONAL | Data Transfer Operations can be posted to the Endpoint. When an Endpoint is first created by calling *it_ep_ud_create* it is in this state. | *it_ep_free*, *it_ep_modify*, *it_ep_query*, *it_get_consumer_context*, *it_get_handle_type*, *it_post_recvfrom*, *it_post_sendto*, *it_set_consumer_context*, *it_ep_reset* |

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.

#### IT_EP_STATE_UNCONNECTED

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.

#### IT_EP_STATE_ACTIVE1_CONNECTION_PENDING

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.

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_

PEER_REJECT_EVENT Event and the Endpoint will transition into the IT_EP_STATE_NONOPERATIONAL state.

In the case of three-way Connection establishment, the Active side Endpoint will transition to IT_EP_STATE_ACTIVE2_CONNECTION_PENDING when an IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT Event is enqueued for the Active side Consumer. If the Active Consumer calls *it_reject* after processing the IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT Event, the Endpoint will transition into the IT_EP_STATE_NONOPERATIONAL state. If the Passive side rejects the Connection, the Active side will receive an IT_CM_MSG_CONN_PEER_REJECT_EVENT Event and the Endpoint will transition into the IT_EP_STATE_NONOPERATIONAL state.

#### IT_EP_STATE_ACTIVE2_CONNECTION_PENDING

In the case of three-way Connection establishment, the Endpoint will transition to the IT_EP_STATE_CONNECTED state when the Active side Consumer successfully calls *it_ep_accept*, or the Endpoint will transition to the IT_EP_STATE_NONOPERATIONAL state if the Active side Consumer successfully calls *it_reject*.

In the case of two-way Connection establishment, this state is transient, and the Endpoint will transition to the IT_EP_STATE_CONNECTED state when the Connection is successfully established.

#### IT_EP_STATE_PASSIVE_CONNECTION_PENDING

The Passive side Endpoint transitions into this state when the Consumer calls *it_ep_accept* during Connection establishment. In this state only Receive Data Transfer Operations can be posted. For iWARP only, this is also a transient state during Socket Conversion on the Conversion Initiator side.

From this state the Endpoint will transition into the IT_EP_STATE_CONNECTED state when Connection establishment completes successfully.

#### IT_EP_STATE_CONNECTED

This state is entered when Connection establishment completes. Upon transition to this state, the Implementation delivers an IT_CM_MSG_CONN_ESTABLISHED_EVENT Event. All types of Data Transfer and Link Operations can be posted and will be processed in this state.

If either the Local or Remote Consumer disconnects, the Endpoint will transition into the IT_EP_STATE_NONOPERATIONAL state, and the Implementation will deliver an IT_CM_MSG_CONN_DISCONNECT_EVENT Event.

Local or Remote errors (e.g., protection violations) also cause the Endpoint to transition to the IT_EP_STATE_NONOPERATIONAL state, and the Implementation will deliver an IT_CM_MSG_CONN_BROKEN_EVENT Event.

The transition out of the IT_EP_STATE_CONNECTED state is surfaced by either the IT_CM_CONN_DISCONNECT_EVENT Event or the IT_CM_MSG_CONN_BROKEN_EVENT Event, but never both.

#### IT_EP_STATE_NONOPERATIONAL

In this state no requests will be processed by the Endpoint and any well-formed requests posted will generate Completions with a failing Completion Status. The Endpoint will remain in this state until *it_ep_reset* is used to put the Endpoint back into the IT_EP_STATE_UNCONNECTED state.

#### IT_EP_STATE_PASSIVE_WAIT_RDMA_TRANS_REQ

For iWARP only, the Passive side Endpoint; i.e., the Endpoint of the RDMA Responder or, equivalently, of the Conversion Initiator, transitions into this state when the Conversion Initiator calls *it_socket_convert*. In this state only Receive Data Transfer Operations can be posted. From this state, the Endpoint will transition into the transient IT_EP_STATE_ PASSIVE_CONNECTION_PENDING state when a request for transitioning to RDMA mode is received from the remote side. When Connection establishment completes successfully, the Endpoint will finally transition to the IT_EP_STATE_CONNECTED state.

#### IT_EP_STATE_UD_OPERATIONAL

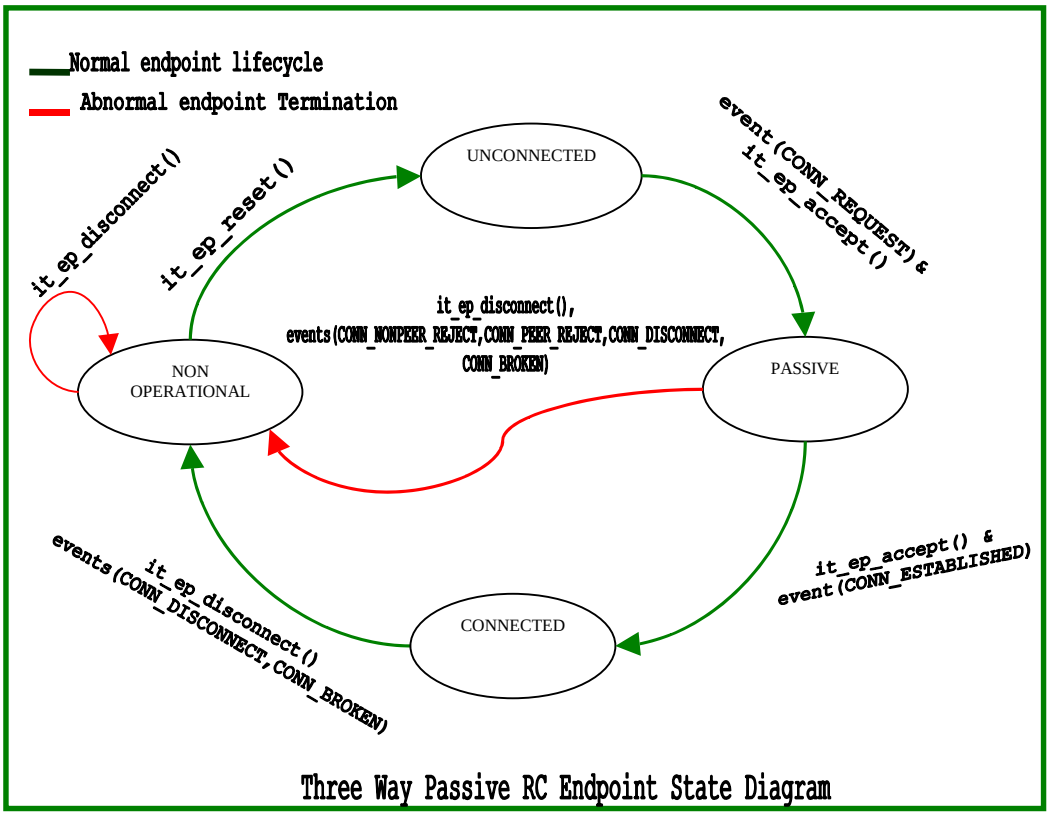
In this state requests will be processed by the Endpoint.

#### IT_EP_STATE_UD_NONOPERATIONAL

In this state no requests will be processed by the Endpoint and any well-formed requests posted will generate Completions with a failing Completion Status. Once an Unreliable Datagram Endpoint enters this state it can only be destroyed with *it_ep_free*.

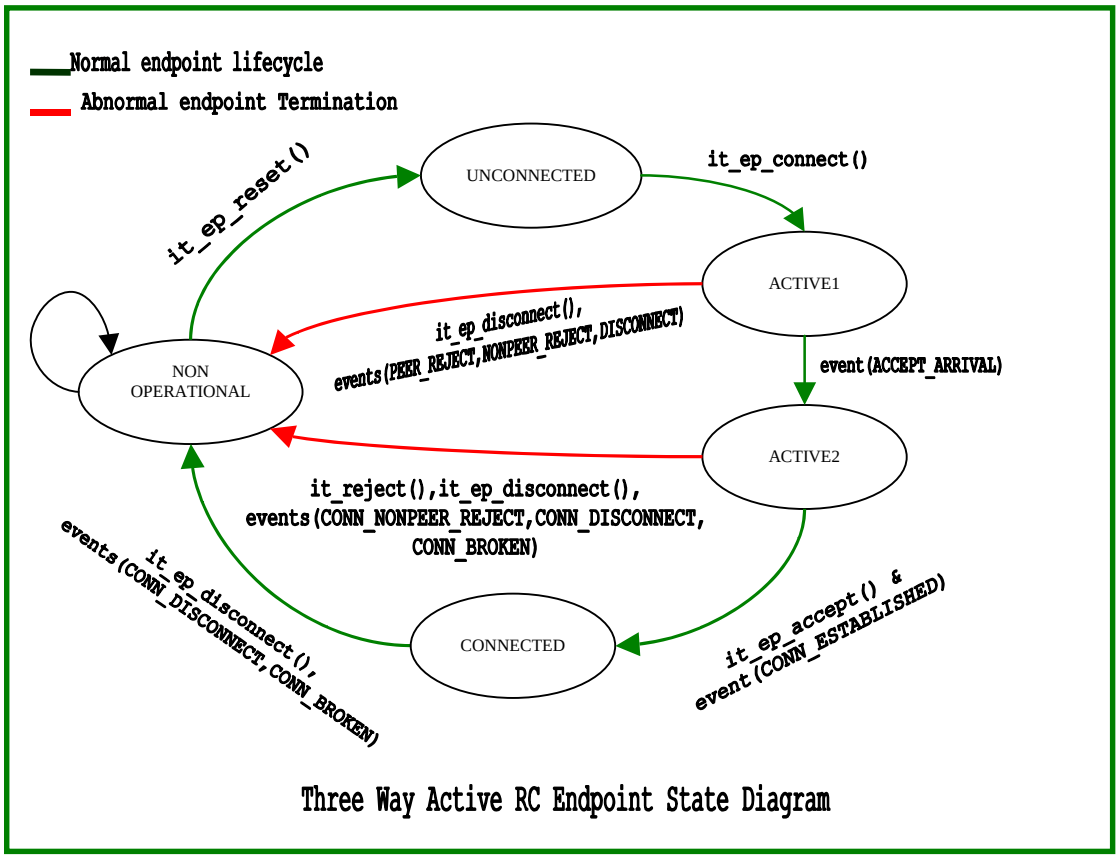
### EXTENDED DESCRIPTION

For the TII, the state diagrams are shown below, separately for three-way and two-way Connection establishment, and for the active and passive sides.



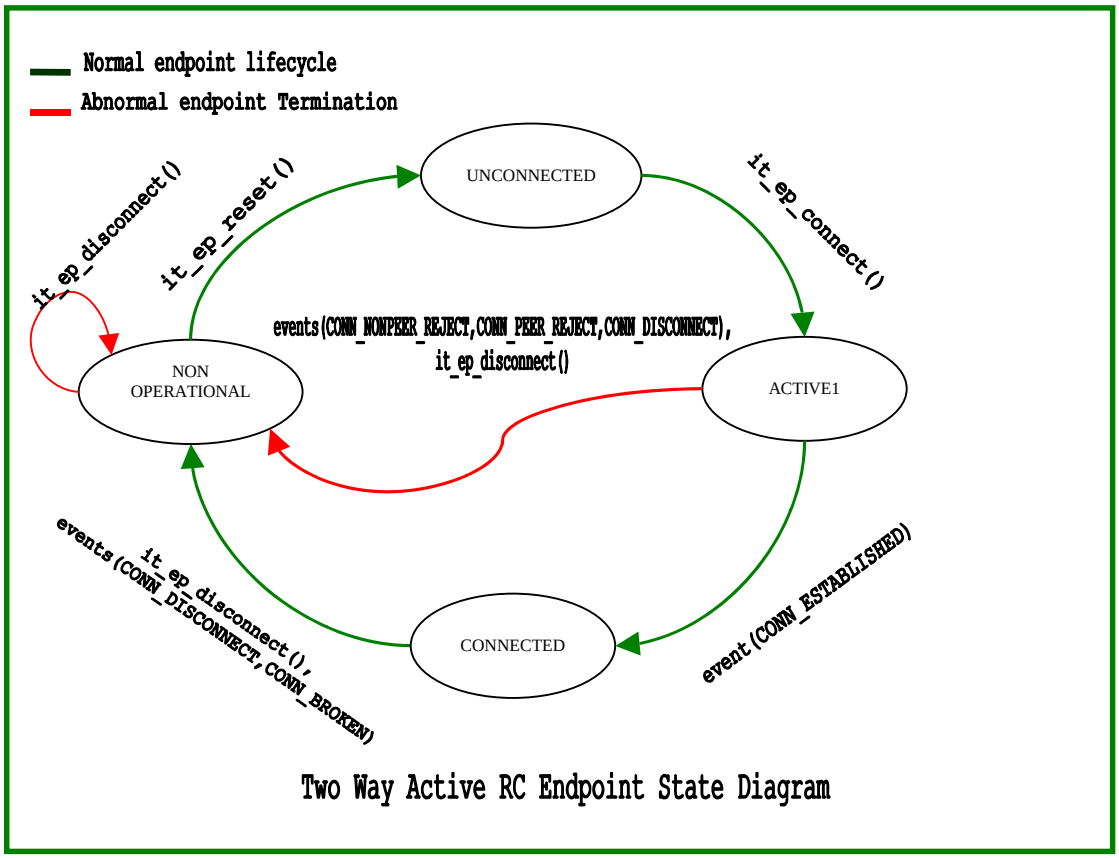

10086

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**

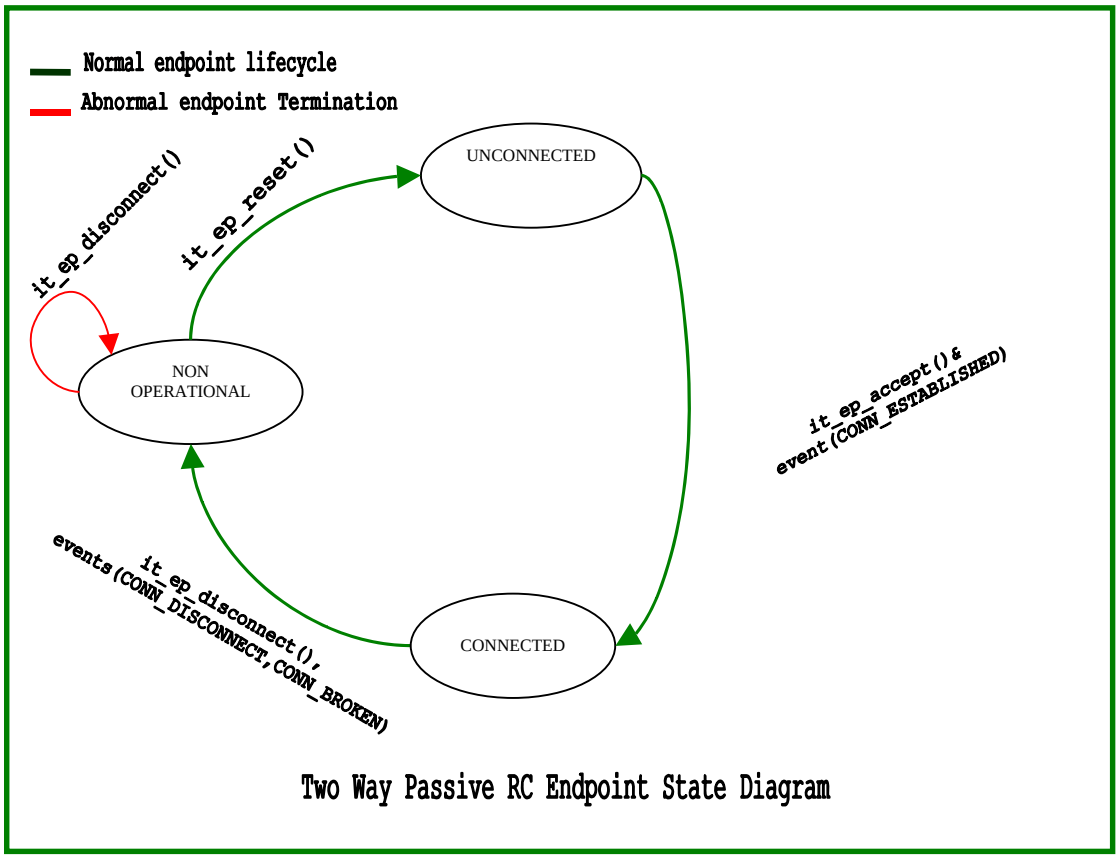



**Figure 11: Two Way Passive RC Endpoint State Diagram**

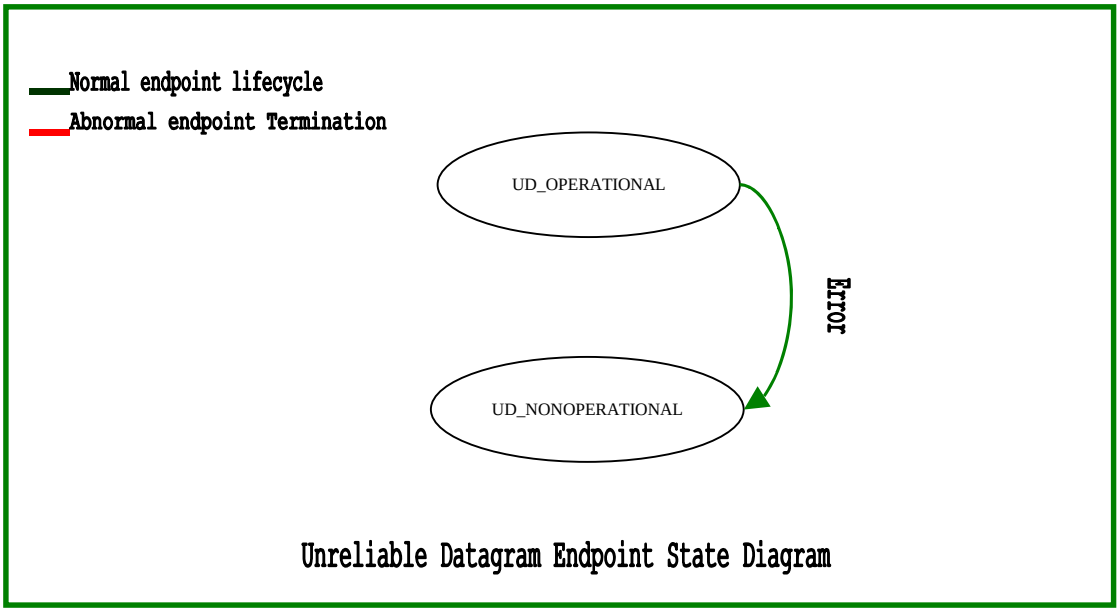



**Figure 12: Unreliable Datagram Endpoint State Diagram**

For the state diagrams applicable to the TDI, see *it_socket_convert*.

**APPLICATION USAGE**

1. If the Consumer cares about the Completion Status of posted Data Transfer or Link Operations after an Endpoint transitions into the IT_EP_STATE_NONOPERATIONAL state, then the Consumer can Post Send and Receive DTOs to the Endpoint to serve as markers in the Endpoint associated EVD. The API guarantees that any posting in IT_EP_STATE_NONOPERATIONAL state will be immediately flushed to the EVDs. The Consumer can reap Completions from the associated EVDs until Completions for the marker DTOs are returned. This way the Consumer can be guaranteed that all Completions associated with the Endpoint have been reaped.

2. The Consumer is responsible for coordinating the use of functions that free a Connection establishment Identifier (*cn_est_i*d) 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.

**SEE ALSO**

*it_ep_accept*(), *it_reject*(), *it_ep_disconnect*(), *it_handoff*(), *it_ep_reset*()

# it_event_t

**NAME**

it_event – definition of Event data structures

**SYNOPSIS**

```
#include <it_api.h>

#define IT_EVENT_STREAM_MASK  0xff000
#define IT_TIMEOUT_INFINITE   ((uint64_t)(-1))

typedef enum
{
    /*
     * Event Stream for WR/DTO completions
     */
    IT_DTO_EVENT_STREAM           = 0x00000,
    IT_DTO_SEND_CMPL_EVENT        = 0x00001,
    IT_DTO_RC_RECV_CMPL_EVENT     = 0x00002,
    IT_DTO_UD_RECV_CMPL_EVENT     = 0x00003,
    IT_DTO_RDMA_WRITE_CMPL_EVENT  = 0x00004,
    IT_DTO_RDMA_READ_CMPL_EVENT   = 0x00005,
    IT_RMR_BIND_CMPL_EVENT        = 0x00006,
    IT_RMR_LINK_CMPL_EVENT        = 0x00006,
    IT_DTO_FETCH_ADD_CMPL_EVENT   = 0x00007,
    IT_DTO_CMP_SWAP_CMPL_EVENT    = 0x00008,
    IT_LMR_LINK_CMPL_EVENT        = 0x00009,

    /*
     * Event Stream for Communication Management Request Events
     */
    IT_CM_REQ_EVENT_STREAM              = 0x01000,
    IT_CM_REQ_CONN_REQUEST_EVENT        = 0x01001,
    IT_CM_REQ_UD_SERVICE_REQUEST_EVENT  = 0x01002,

    /*
     * Event Stream for Communication Management Message Events
     */
    IT_CM_MSG_EVENT_STREAM               = 0x02000,
    IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT  = 0x02001,
    IT_CM_MSG_CONN_ESTABLISHED_EVENT     = 0x02002,
    IT_CM_MSG_CONN_DISCONNECT_EVENT      = 0x02003,
    IT_CM_MSG_CONN_PEER_REJECT_EVENT     = 0x02004,
    IT_CM_MSG_CONN_NONPEER_REJECT_EVENT  = 0x02005,
    IT_CM_MSG_CONN_BROKEN_EVENT          = 0x02006,
    IT_CM_MSG_UD_SERVICE_REPLY_EVENT     = 0x02007,

    /* Event Stream for Affiliated Asynchronous Events */
    IT_ASYNC_AFF_EVENT_STREAM               = 0x04000,
    IT_ASYNC_AFF_SEVD_ENQUEUE_FAILURE       = 0x04001,
    IT_ASYNC_AFF_EP_FAILURE                 = 0x04002,
    IT_ASYNC_AFF_EP_BAD_TRANSPORT_OPCODE    = 0x04003,
    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
```

```
 *     IT_CM_MSG_CONN_NONPEER_REJECT_EVENT
 *     IT_CM_MSG_CONN_DISCONNECT_EVENT
 *     IT_CM_MSG_CONN_BROKEN_EVENT
 *
 * it_ud_svc_reply_event_t supports:
 *     IT_CM_MSG_UD_SERVICE_REPLY_EVENT
 */
    it_connection_event_t    conn;
    it_ud_svc_reply_event_t  ud_svc_reply;

    /*
     * it_affiliated_event_t supports
     * the following Event Stream:
     *     IT_ASYNC_AFF_EVENT_STREAM
     */
    it_affiliated_event_t  aff_async;

    /*
     * it_unaffiliated_event_t supports
     * the following Event Stream:
     *     IT_ASYNC_UNAFF_EVENT_STREAM
     */
    it_unaffiliated_event_t  unaff_async;

    /*
     * it_software_event_t supports
     * the following Event Stream:
     *     IT_SOFTWARE_EVENT_STREAM
     */
    it_software_event_t  sw;

    /*
     * it_aevd_notification_event_t supports
     * the following Event Stream:
     *     IT_AEVD_NOTIFICATION_EVENT_STREAM
     */
    it_aevd_notification_event_t  aevd_notify;
} it_event_t;
```

**DESCRIPTION**

The *it_event_t* defines the format for Events for IT-APIs. Each Event consists of a Handle to the EVD where the Event has been queued, and Event type identifier with the Event type-specific Event data.

Events for a Simple EVD can be fed from only a single Event Stream type.

Multiple Event numbers that can be on the same Event Stream type form an Event group. The Event data formats for all Event types of the same Event Stream type are defined in one or more separate reference page(s) specific to each Event group.

**APPLICATION USAGE**

The Consumer allocates an *it_event_t* object and passes it into the *it_evd_wait* or *it_evd_dequeue* calls in order to retrieve Events. The *it_event_t* object is a union of all possible Event Stream data types, thus it is the size of the largest possible Event type.

If the Consumer wishes to conserve memory and use only the minimally-sized Event data structures found in the *it_event_t* union, they are free to. Use of *it_event_t* structures that are too small for an Event Stream may cause program termination.

The Consumer may use the IT_EVENT_STREAM_MASK to convert from an *it_event_type_t event_number* to Event Stream by masking off the lower bits of the Event number.

**SEE ALSO**

*it_aevd_notification_event_t*, *it_affiliated_event_t*, *it_cm_msg_events*, *it_cm_req_events*, *it_dto_events*, *it_software_event_t*, *it_unaffiliated_event_t*, *it_evd_wait*(), *it_evd_dequeue*(), *it_evd_create*()

# it_handle_t

## NAME

it_handle_t – enumeration and type definitions for IT Handles

## SYNOPSIS

```
#include <it_api.h>

typedef enum {
    IT_HANDLE_TYPE_ADDR,
    IT_HANDLE_TYPE_EP,
    IT_HANDLE_TYPE_EVD,
    IT_HANDLE_TYPE_IA,
    IT_HANDLE_TYPE_LISTEN,
    IT_HANDLE_TYPE_LMR,
    IT_HANDLE_TYPE_PZ,
    IT_HANDLE_TYPE_RMR,
    IT_HANDLE_TYPE_UD_SVC_REQ,
    IT_HANDLE_TYPE_SRQ
} it_handle_type_enum_t;

typedef void  *it_handle_t;
#define IT_NULL_HANDLE  ((it_handle_t) NULL)

typedef struct it_addr_handle_s        *it_addr_handle_t;
typedef struct it_ep_handle_s          *it_ep_handle_t;
typedef struct it_evd_handle_s         *it_evd_handle_t;
typedef struct it_ia_handle_s          *it_ia_handle_t;
typedef struct it_listen_handle_s      *it_listen_handle_t;
typedef struct it_lmr_handle_s         *it_lmr_handle_t;
typedef struct it_pz_handle_s          *it_pz_handle_t;
typedef struct it_rmr_handle_s         *it_rmr_handle_t;
typedef struct it_ud_svc_req_handle_s  *it_ud_svc_req_handle_t;
typedef struct it_srq_handle_s         *it_srq_handle_t;
```

## DESCRIPTION

The *it_handle_type_enum_t* associates an enumerated value with each type of Handle used in the API Implementation. The enumeration is used to describe the type of a Handle returned by *it_get_handle_type*.

The table below defines the relationship of IT-API Handle types and the associated *it_handle_type_enum_t* value.

| **it_handle Type** | **Returned it_handle_type_enum Value** |
|---|---|
| *it_addr_handle_t* | IT_HANDLE_TYPE_ADDR |
| *it_ep_handle_t* | IT_HANDLE_TYPE_EP |
| *it_evd_handle_t* | IT_HANDLE_TYPE_EVD |
| *it_ia_handle_t* | IT_HANDLE_TYPE_IA |

| it_handle Type | Returned it_handle_type_enum Value |
|---|---|
| *it_listen_handle_t* | IT_HANDLE_TYPE_LISTEN |
| *it_lmr_handle_t* | IT_HANDLE_TYPE_LMR |
| *it_pz_handle_t* | IT_HANDLE_TYPE_PZ |
| *it_rmr_handle_t* | IT_HANDLE_TYPE_RMR |
| *it_ud_svc_req_handle_t* | IT_HANDLE_TYPE_UD_SVC_REQ |
| *it_srq_handle_t* | IT_HANDLE_TYPE_SRQ |

10368 **SEE ALSO**

10369 *it_get_handle_type*()

# it_ia_info_t

**NAME**

it_ia_info_t – encapsulates all Interface Adapter attributes and Spigot information

**SYNOPSIS**

```
#include <it_api.h>

/* Enumerates all the transport types supported by the API. */
typedef enum {

    /* InfiniBand Transport */
    IT_IB_TRANSPORT  = 1,

    /* VIA host Interface using IP transport, supporting
       only the Reliable Delivery reliability level */
    IT_VIA_IP_TRANSPORT  = 2,

    /* VIA host Interface, using Fibre Channel transport, supporting
      only the Reliable Delivery reliability level */
    IT_VIA_FC_TRANSPORT  = 3,

    /* iWARP over TCP transport */
    IT_IWARP_TCP_TRANSPORT  = 4,

    /* Vendor-proprietary Transport */
    IT_VENDOR_TRANSPORT  = 1000

} it_transport_type_t;

/* Transport Service Type definitions. */
typedef enum {

    /* Reliable Connected Transport Service Type */
    IT_RC_SERVICE  = 0x1,

    /* Unreliable Datagram Transport Service Type */
    IT_UD_SERVICE  = 0x2,

} it_transport_service_type_t;

/* The following structure describes an Interface Adapter Spigot */
typedef struct {

    /* Spigot identifier */
    size_t  spigot_id;

    /* Maximum sized Send operation for the RC service
      on this Spigot. */
    size_t  max_rc_send_len;

    /* Maximum sized RDMA Read/Write operation for the RC service on
       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. */
```

```
        it_iobl_type_t  iobl_types_supported;
} it_ia_info_t;
```

**DESCRIPTION**

The *it_ia_info_t* structure is returned by the *it_ia_query* routine.

The *it_ia_info_t* structure specifies the capabilities and attributes of an Interface Adapter. It also identifies the Interface Adapter's Spigots and their Network Addresses.

Spigot identifiers are required inputs to the *it_get_pathinfo* and *it_ listen_create* routines. Spigot Network Addresses may be used in the advertisement of local Consumer-provided services to remote Consumers.

The IA can be in one of two states: enabled or disabled. An IA will be in the enabled state when it is created (via *it_ia_create*). Normally an IA will remain in the enabled state, but if it encounters a catastrophic error it will move into the disabled state. The Implementation guarantees that no unreported data corruption has occurred as a result of the IA entering the disabled state. If the Consumer calls any API routine other than *it_ia_free* while the IA is in the disabled state, neither the IA nor any of the Implementation data structures associated with it will be modified in any way. Instead, the routine will return the error code IT_ERR_IA_CATASTROPHE, and none of the output parameters from the routine will be valid. Once an IA has entered the disabled state the only recovery action that the Consumer can perform is to free the IA.

**EXTENDED DESCRIPTION**

The supported I/O Buffer List types are described by *iobl_types_supported*, which is a combination of bits from *it_iobl_type_t*, as follows:

On the InfiniBand 1.1 Transport, no IOBL types are supported.

On the InfiniBand 1.2 and iWARP Transports, *iobl_types_supported* must include one of the combinations IT_IOBL_PAGE_LIST (Page List mode), IT_IOBL_BLOCK_LIST (Block List superset mode), or IT_IOBL_PAGE_LIST | IT_IOBL_BLOCK_LIST (Page List / Block List discriminating mode), and it may further include IT_IOBL_VE_PAGE_LIST.

**FUTURE DIRECTIONS**

Quality of Service control for VIA and other transports may be added in the future.

**SEE ALSO**

*it_ia_query*(), *it_get_pathinfo*(), *it_listen_create*(), *it_iobl_type_t*

# it_io_addr_t

**NAME**

it_io_addr_t – O/S-dependent structure describing a bus or physical address

**SYNOPSIS**

```
typedef struct it_io_addr_s      *it_io_addr_t;
```

**DESCRIPTION**

The *it_io_addr_t* structure describes a bus or physical address in an O/S-dependent manner.

**APPLICATION USAGE**

The *it_io_addr_t* is used to describe bus or physical addresses in LMR Triplets as well as in I/O Buffer Lists used as inputs to the *it_lmr_create* and *it_lmr_link* routines.

**SEE ALSO**

*it_lmr_create*(), *it_post_atomic*(), *it_post_send*(), *it_post_sendto*(), *it_post_recv*(), *it_post_recvfrom*(), *it_post_rdma_write*(), *it_post_rdma_read*(), *it_addr_mode_t*

# it_iobl_t

## NAME

it_iobl_t – I/O Buffer List (IOBL)

## SYNOPSIS

```
#include <it_api.h>

typedef enum {
    IT_IOBL_PAGE_LIST     = 0x0001,
    IT_IOBL_BLOCK_LIST    = 0x0002,
    IT_IOBL_VE_PAGE_LIST  = 0x0004
} it_iobl_type_t;

typedef struct {
    /* I/O Address - bus address or physical address.
       it_io_addr_t is an OS-dependent type. */
    it_io_addr_t io_addr;

    /* IOBL element length for this element */
    it_length_t elt_len;
} it_iobl_ve_elt_t; /* Variable-element-length list element */

typedef struct {
    it_iobl_type_t iobl_type; /* Union discriminator */
    union {
        struct {
            it_io_addr_t *list_elt; /* Head of list */
            it_length_t elt_len; /* Constant IOBL element length */
        } ce_list; /* Constant-element-length list */
        struct {
            it_iobl_ve_elt_t *list_elt; /* Head of list */
        } ve_list; /* Variable-element-length list */
    } list;
    size_t num_elts; /* Number of elements in list */
    it_length_t fbo; /* First-Byte Offset */
} it_iobl_t;
```

## DESCRIPTION

The *it_iobl_t* structure describes a buffer list containing constant-length or variable-length elements. Its members are defined as follows:

*iobl_type* IOBL type, discriminating the union *list*. An IOBL can be a Page List (constant or variable page size) or a Block List, depending on the supported IOBL types (see the *iobl_types_supported* attribute in *it_ia_info_t*). If *iobl_types_supported* includes more than one bit, any one of these bits can be selected as *iobl_type*. If *iobl_types_supported* includes both IT_IOBL_PAGE_LIST and IT_IOBL_BLOCK_LIST (Page List / Block List discriminating mode), then the Consumer can assist the transport in IOBL verification/discrimination by indicating the exact IOBL type rather than declaring each IOBL as a Block List.

*list* Either an array of *it_io_addr_t* elements of constant size or an array of elements each including an *it_io_addr_t* and a length.

*num_elts* Number of elements in the *list*.

*fbo* First-Byte Offset in first element of list. Must be greater than or equal to 0 and less than *elt_len* (where *elt_len* is either the constant element length of a *ce_list* or the element length of the first element of a *ve_list*).

**APPLICATION USAGE**

The I/O Buffer List is an input parameter describing a local buffer segment for the *it_lmr_link* and *it_lmr_create* operations.

The following code demonstrates the various uses of the it_iobl_t type:

```
#include <it_api.h>

#define PLEN 4096
#define BLEN 512

/*
 * for the sake of a simple coding example
 * declare a page-aligned buffer
 */
#pragma ALIGN 4096
char                page_buffer[PLEN];

/*
 * for the sake of a simple coding example
 * declare a block-aligned buffer
 */
#pragma ALIGN 512
char                block_buffer[BLEN];

size_t              mapped_len;
it_io_addr_t        io_addr[2];
it_iobl_ve_elt_t    ve_elt[2];
it_iobl_t           pl_iobl;
it_iobl_t           bl_iobl;
it_iobl_t           ve_pl_iobl;
it_lmr_handle       lmr_handle;

/* map the page-aligned buffer to an I/O address */
it_mem_map(ia, page_buffer, aspace(page_buffer), PLEN, &io_addr[0],
&mapped_len);

/* create a page list it_iobl_t using the ioaddr from above */
pl_iobl.type = IT_IOBL_PAGE_LIST;
pl_iobl.list.ce_list.list_elt = &io_addr[0];
pl_iobl.list.ce_list.elt_len = PLEN;
pl_iobl.num_elts = 1;
pl_iobl.fbo = 0;

/* create an LMR using the page list and absolute addressing */
it_lmr_create(pz, page_buffer, &pl_iobl, PLEN, IT_ADDR_MODE_ABSOLUTE,
IT_PRIV_LOCAL, IT_LMR_FLAG_NONE, 0, &lmr_handle, IT_NO_ADDR);
```

```
        /* map the block-aligned buffer to an I/O address */
        it_mem_map(ia, block_buffer, aspace(block_buffer), BLEN, &io_addr[1],
        &mapped_len);

        /* create a block list it_iobl_t using the ioaddr from above */
        bl_iobl.type = IT_IOBL_BLOCK_LIST;
        bl_iobl.list.ce_list.list_elt = &io_addr[1];
        bl_iobl.list.ce_list.elt_len = BLEN; /* say block size is 512 */
        bl_iobl.num_elts = 1;
        bl_iobl.fbo = 40; /* as an example of fbo use, set First-Byte Offset of
        data to 40 */

        /* create an LMR using the page list and relative addressing */
        it_lmr_create(pz, 0, &bl_iobl, (BLEN-40), IT_ADDR_MODE_RELATIVE,
        IT_PRIV_LOCAL, IT_LMR_FLAG_NONE, 0, &lmr_handle, IT_NO_ADDR);

        /* reuse the two mappings to demonstrate VE lists */
        ve_elt[0].io_addr = &io_addr[0];
        ve_elt[0].elt_len = PLEN;
        ve_elt[1].io_addr = &io_addr[1];
        ve_elt[2].elt_len = BLEN;
        ve_pl_iobl.type = IT_IOBL_VE_PAGE_LIST;
        ve_pl_iobl.list.ve_list.list_elt = ve_elt;
        ve_pl_iobl.num_elts = 2;
        ve_pl_iobl.fbo = 0;

        /* create an LMR using the variable element page list */
        it_lmr_create(pz, 0, &ve_pl_iobl, (PLEN+BLEN), IT_ADDR_MODE_RELATIVE,
        IT_PRIV_LOCAL, IT_LMR_FLAG_NONE, 0, &lmr_handle, IT_NO_ADDR);
```

**SEE ALSO**

*it_lmr_create*(), *it_lmr_link*(), *it_post_atomic*(), *it_post_send*(), *it_post_sendto*(), *it_post_recv*(), *it_post_recvfrom*(), *it_post_rdma_write*(), *it_post_rdma_read*(), *it_addr_mode_t*

# it_lmr_triplet_t

## NAME

it_lmr_triplet_t – structure describing a DTO buffer in a Local Memory Region

## SYNOPSIS

```
#include <it_api.h>

typedef struct {
    it_lmr_handle_t  lmr;
    union {
        void        *abs;
        it_length_t  rel;
        it_io_addr_t io;
    } addr;
    it_length_t      length;
} it_lmr_triplet_t;
```

## DESCRIPTION

The *it_lmr_triplet_t* structure describes a local Source or Destination buffer segment for Data Transfer Operations. Its members are defined as follows:

*lmr* Handle of LMR in which the local buffer resides.

*addr* Starting address of the local buffer segment, interpreted according to the *addr_mode* (addressing mode) attribute of the LMR (see *it_addr_mode_t*) either as an absolute address (*abs*) or as an address offset (*rel*) relative to the Base Address of the LMR. If LMR is the Direct LMR Handle, then *addr* is interpreted as an *it_io_addr_t*.

*length* Length in bytes of the local buffer segment.

## APPLICATION USAGE

The LMR Triplet is an input parameter describing a local buffer segment for DTOs such as *it_post_send*.

## SEE ALSO

*it_post_atomic*(), *it_post_send*(), *it_post_sendto*(), *it_post_recv*(), *it_post_recvfrom*(), *it_post_rdma_write*(), *it_post_rdma_read*(), *it_addr_mode_t*

# it_mem_priv_t

**NAME**

it_mem_priv_t – Memory access privileges for Local Memory Regions and Remote Memory Regions

**SYNOPSIS**

```
#include <it_api.h>

typedef enum {
    IT_PRIV_LOCAL_READ    = 0x0001,
    IT_PRIV_LOCAL_WRITE   = 0x0002,
    IT_PRIV_LOCAL         = 0x0003,
    IT_PRIV_REMOTE_READ   = 0x0004,
    IT_PRIV_REMOTE_WRITE  = 0x0008,
    IT_PRIV_REMOTE        = 0x000c,
    IT_PRIV_ATOMIC        = 0x0010,
    IT_PRIV_ALL           = 0x001f
} it_mem_priv_t;
```

**DESCRIPTION**

*privs* Memory access privileges for an LMR or RMR, formed by a bitwise-inclusive OR of values from *it_mem_priv_t*.

Individual bit values defined in *it_mem_priv_t* are as follows:

| | |
|---|---|
| IT_PRIV_LOCAL_READ | Grants local read access to the IA, enabling the use of an LMR as a Source buffer for locally posted Send or RDMA Write DTOs. |
| IT_PRIV_LOCAL_WRITE | Grants local write access to the IA, enabling the use of the LMR as a Destination buffer for locally posted Receive or RDMA Read DTOs. |
| IT_PRIV_REMOTE_READ | Grants remote read access to the IA, enabling the use of an LMR or RMR as a Source buffer for incoming RDMA Read requests. |
| IT_PRIV_REMOTE_WRITE | Grants remote write access to the IA, enabling the use of an LMR or RMR as a Destination buffer for incoming RDMA Write operations or RDMA Read Responses. |
| IT_PRIV_ATOMIC | Grants remote Atomic access to the IA, enabling the use of an LMR or RMR as a Source/Destination buffer for incoming Atomic operations. |

Predefined bit combinations defined in *it_mem_priv_t* are as follows:

| | |
|---|---|
| IT_PRIV_LOCAL | Local read and local write access, for an LMR. |
| IT_PRIV_REMOTE | Remote read and remote write access, for an LMR or RMR. |
| IT_PRIV_ALL | All access privileges, for an LMR. |

**APPLICATION USAGE**

Memory access privileges of a Local Memory Region are specified when the LMR is created. Memory access privileges of a Remote Memory Region are specified when the RMR is linked to an LMR.

**SEE ALSO**

*it_lmr_create*(), *it_lmr_query*(), *it_rmr_link*(), *it_rmr_query*()

# it_net_addr_t

**NAME**

it_net_addr_t – encapsulates all supported Network Address types

**SYNOPSIS**

```
#include <it_api.h>

/* Enumerates all the possible Network Address types supported
   by the API. */
typedef enum {

    /* IPv4 address */
    IT_IPV4 = 0x1,

    /* IPv6 address */
    IT_IPV6 = 0x2,

    /* InfiniBand GID */
    IT_IB_GID = 0x3,

    /* VIA Network Address */
    IT_VIA_HOSTADDR = 0x4

} it_net_addr_type_t;

/* 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. */

#define IT_MAX_VIA_ADDR_LEN  64

typedef struct {

    /* The number of bytes in the array below that are
       significant. */
    uint16_t  len;

    /* VIA host address, which is an array of bytes. */
    unsigned char  hostaddr[IT_MAX_VIA_ADDR_LEN];

} it_via_net_addr_t;

/* This defines the Network Address format for the InfiniBand
   GID, which is just an IPv6 address. */
typedef struct in6_addr  it_ib_gid_t;

/* This describes a Network Address suitable for input to several
   routines in the API. */
typedef struct {

    /* The discriminator for the union below. */
    it_net_addr_type_t  addr_type;
```

```
        union {

            /* IPv4 address, in network byte order. */
            struct in_addr  ipv4;

            /* IPv6 address, in network byte order. */
            struct in6_addr  ipv6;

            /* InfiniBand GID, in network byte order. */
            it_ib_gid_t  gid;

            /* VIA Network Address. */
            it_via_net_addr_t  via;

        } addr;

    } it_net_addr_t;
```

**DESCRIPTION**

The *it_net_addr_t* type is used by several routines in the API to encapsulate a Network Address associated with a Spigot on an IA. The *it_net_addr_t* is the name for a Spigot when it is being accessed remotely. (When it is being accessed locally, it is named by a Spigot identifier, not by an *it_net_addr_t*.) Each Spigot on an IA has at least one *it_net_addr_t* associated with it. A Spigot can have more than one Network Address associated with it, and these Network Addresses can be of different types. (For example, an InfiniBand HCA might have both an IPv4 address and an InfiniBand GID associated with one of its Spigots.) The set of Network Addresses that can be used to refer to a Spigot on an IA can be determined using the *it_ia_query* routine.

In order to aid Consumers in writing portable applications that span platforms with different native byte orders, all Network Addresses that are supported by the API with the exception of the VIA “host address” are required to be input to the API in network byte order, and will be output from the API in network byte order. (The VIA “host address” is defined to be an array of bytes, and hence is not affected by which native byte order a platform uses.)

**SEE ALSO**

*it_get_pathinfo*(), *it_ia_query*()

# it_path_t

**NAME**

it_path_t – describes the Path between a pair of Spigots

**SYNOPSIS**

```
#include <it_api.h>

/* This is the Path information for the InfiniBand Transport. */
typedef struct {

    /* Partition Key, as defined in the REQ message for the IB
       CM protocol. */
    uint16_t  partition_key;

    /* Path Packet Payload MTU, as defined in the REQ message
       for the IB CM protocol. */
    uint8_t  path_mtu : 4;

    /* 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. */
    uint8_t  packet_lifetime : 6;

    /* 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. */
    uint16_t  local_port_lid;

    /* 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. */
    uint16_t  remote_port_lid;

    /* 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. */
    it_ib_gid_t  local_port_gid;

    /* 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. */
    it_ib_gid_t  remote_port_gid;

    /* 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. */
    uint8_t  packet_rate : 6;
```

```
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
```

```
            /* Remote path element */
            union {
                struct in_addr   ipv4;
                struct in6_addr  ipv6;
            } raddr;
        } it_iwarp_net_endpoint_t;

        /* This is the Path data structure used by several routines in
           the API. */
        typedef struct {

            /* Identifier for the Spigot to be used on the local IA.
               Note that this data structure is always used in a
               Context where the IA associated with the Spigot can be
               deduced. */
            size_t  spigot_id;

            /* The transport-independent timeout parameter for how long
               to wait, in microseconds, before timing out a Connection
               establishment attempt using this Path. The timeout
               period for establishing a Connection
               can only be specified on the Active side; the timeout
               period cannot be changed on the Passive side. */
            uint64_t  timeout;

            /* The remote component of the Path. */
            union {

                /* For use with InfiniBand. */
                it_ib_net_endpoint_t  ib;

                /* For use with VIA. */
                it_via_net_endpoint_t  via;

                /* For use with iWARP. */
                it_iwarp_net_endpoint_t  iwarp;

            } remote;

        } it_path_t;
```

**DESCRIPTION**

The *it_path_t* type is used by several routines in the API to encapsulate a Path between two Spigots. The *it_path_t* contains a local Spigot identifier, a remote Spigot address, and a specification of all information that determines the properties of the Path that messages will take between the two Spigots. The local Spigot to be used is identified in a transport-independent manner, but the remote Spigot and the Path to that Spigot are specified in a transport-dependent manner.

The *it_path_t* structure also contains a *timeout* member. This *timeout* member defines how long the local Consumer is willing to wait for a response to its attempt to establish a Connection when the *it_path_t* is used with the *it_ep_connect* routine. If the Consumer retrieves the Path using the *it_get_pathinfo* routine, the Implementation will provide a *timeout* that should be sufficiently

long to establish a Connection under most circumstances, and so the Consumer should have no need to modify this value. If the Consumer chooses to provide their own value for the *timeout* member, the Consumer should take care to choose a value that is compatible with any underlying transport-specific timeout values governing Connection establishment that may be present in the transport-specific portion of the *it_path_t*. Choosing a value of *timeout* that is incompatible with the transport-specific timeout values governing Connection establishment will result in an error being returned from the *it_ep_connect* routine.

All data contained within the *it_path_t* data structure appears in host byte order unless otherwise noted in the comments associated with the members of the data structure. The contents of the *path_t* returned from the *it_get_pathinfo* routine are only valid on the node on which the routine was invoked.

**SEE ALSO**

*it_get_pathinfo*(), *it_ep_connect*(), *it_address_handle_create*(), *it_ud_service_request_handle_create*()

# it_rmr_triplet_t

## NAME

it_rmr_triplet_t – structure describing a DTO buffer in a Remote Memory Region

## SYNOPSIS

```
#include <it_api.h>

typedef struct {
    it_rmr_handle_t  rmr;
    union {
        void        *abs;
        it_length_t  rel;
    } addr;
    it_length_t      length;
} it_rmr_triplet_t;
```

## APPLICABILITY

Remote Memory Regions can be used only for the RC service type. See *it_rmr_create*.

## DESCRIPTION

The *it_rmr_triplet_t* structure describes a local Destination buffer segment for an RDMA Read DTO initiated by the *it_post_rdma_read_to_rmr*() call. Its members are defined as follows:

| | |
|---|---|
| *rmr* | Handle of RMR in which the local buffer resides. |
| *addr* | Starting address of the local buffer segment, interpreted according to the *addr_mode* (addressing mode) attribute of the RMR (see *it_addr_mode_t*) either as an absolute address (*abs*) or as an address offset (*rel*) relative to the Base Address of the RMR. |
| *length* | Length in bytes of the local buffer segment. |

## APPLICATION USAGE

The RMR Triplet is an input parameter describing a local buffer segment for RDMA Read DTOs initiated by the *it_post_rdma_read_to_rmr*() call only.

## SEE ALSO

*it_post_rdma_read_to_rmr*(), *it_addr_mode_t*

# it_rmr_type_t

## NAME

it_rmr_type_t – definition of RMR type

## SYNOPSIS

```
#include <it_api.h>

typedef enum {
    IT_RMR_TYPE_DEFAULT  = 0,
    IT_RMR_TYPE_NARROW   = 1,
    IT_RMR_TYPE_WIDE     = 2
} it_rmr_type_t;
```

## APPLICABILITY

Remote Memory Regions can be used only for the RC service type. See *it_rmr_create*.

## DESCRIPTION

Type of a Remote Memory Region, with three possible values as follows:

### IT_RMR_TYPE_DEFAULT

This value can be used only for creating an RMR. Creating an RMR of the default type results in an RMR of the Narrow or Wide type, depending on the Implementation.

### IT_RMR_TYPE_NARROW

An unlinked Narrow RMR may be used as a target for Link operations via all Endpoints in the Protection Zone of the RMR. A linked Narrow RMR may be used as a target for DTOs and Unlink operations only via the Endpoint through which it was linked.

### IT_RMR_TYPE_WIDE

An unlinked Wide RMR may be used as a target for Link operations via all Endpoints in the Protection Zone of the RMR.

A linked Wide RMR may be used as a target for DTOs and Link/Unlink operations via all Endpoints in the Protection Zone of the RMR.

## APPLICATION USAGE

The desired type of a Remote Memory Region must be specified at RMR creation time. The allowable RMR types are Implementation-dependent. However, InfiniBand Implementations at least support Wide RMRs and, if Verb Extensions (BMM) are provided, also Narrow RMRs; iWARP Implementations at least support Narrow RMRs. The *rmr_types_supported* field of the *it_ia_info_t* structure, which can be queried through *it_ia_query*, indicates which RMR types are supported by the IA; i.e., Narrow RMRs, Wide RMRs, or both.

## SEE ALSO

*it_rmr_create*(), *it_rmr_link*(), *it_rmr_query*()

# it_software_event_t

## NAME

it_software_event_t – Software Event type

## SYNOPSIS

```
#include <it_api.h>

typedef struct {
    it_event_type_t  event_number;
    it_evd_handle_t  evd;
    void            *data;
} it_software_event_t;
```

## DESCRIPTION

| | |
|---|---|
| *event_number* | Identifier of the Event type. Valid values: IT_SOFTWARE_EVENT |
| *evd* | Handle for the Event Dispatcher where the Event was queued. |
| data | The pointer that the Consumer furnished to the *it_evd_post_se* routine. |

An IT_SOFTWARE_EVENT_STREAM Event is generated when the Consumer calls the *it_evd_post_se* routine to post a Software Event.

The IT_SOFTWARE_EVENT_STREAM Event Stream supports Events with the *event_number* IT_SOFTWARE_EVENT (see *it_event_t*).

The Software Event type passes back the same pointer that the Consumer furnished to the *it_evd_post_se* routine.

All Events on an IT_SOFTWARE_EVENT_STREAM SEVD cause Notification. See *it_evd_create* for details of Notification.

The Software Event type EVD does not overflow. Instead, *it_evd_post_se* will generate an immediate error if the Consumer attempts to queue a software Event on a full Software Event EVD.

## SEE ALSO

*it_evd_post_se*(), *it_evd_create*(), *it_evd_wait*(), *it_event_t*

# it_status_t

**NAME**

it_status_t – definition of IT-API call return status

**SYNOPSIS**

```
#include <it_api.h>

typedef enum {
    IT_SUCCESS = 0,
    IT_ERR_ABORT,
    IT_ERR_ACCESS,
    IT_ERR_ADDRESS,
    IT_ERR_AEVD_NOT_ALLOWED,
    IT_ERR_ASYNC_AFF_EVD_EXISTS,
    IT_ERR_ASYNC_UNAFF_EVD_EXISTS,
    IT_ERR_CALLBACK_EXISTS,
    IT_ERR_CANNOT_RESET,
    IT_ERR_CONN_QUAL_BUSY,
    IT_ERR_EP_BUSY,
    IT_ERR_EP_TIMEWAIT,
    IT_ERR_EVD_BUSY,
    IT_ERR_EVD_WAIT,
    IT_ERR_EVD_QUEUE_FULL,
    IT_ERR_FAULT,
    IT_ERR_HARD_HI_WATERMARK,
    IT_ERR_IA_CATASTROPHE,
    IT_ERR_INTERRUPT,
    IT_ERR_INVALID_ADDR_MODE,
    IT_ERR_INVALID_ADDRESS,
    IT_ERR_INVALID_AEVD,
    IT_ERR_INVALID_AH,
    IT_ERR_INVALID_ATIMEOUT,
    IT_ERR_INVALID_ATOMIC_OP,
    IT_ERR_INVALID_CM_RETRY,
    IT_ERR_INVALID_CN_EST_FLAGS,
    IT_ERR_INVALID_CN_EST_ID,
    IT_ERR_INVALID_CONN_EVD,
    IT_ERR_INVALID_CONN_QUAL,
    IT_ERR_INVALID_CONVERSION,
    IT_ERR_INVALID_DTO_FLAGS,
    IT_ERR_INVALID_EP,
    IT_ERR_INVALID_EP_ATTR,
    IT_ERR_INVALID_EP_KEY,
    IT_ERR_INVALID_EP_STATE,
    IT_ERR_INVALID_EP_TYPE,
    IT_ERR_INVALID_EVD,
    IT_ERR_INVALID_EVD_STATE,
    IT_ERR_INVALID_EVD_TYPE,
    IT_ERR_INVALID_FLAGS,
    IT_ERR_INVALID_HANDLE,
    IT_ERR_INVALID_IA,
    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,
```

```
        IT_ERR_RESOURCE_RWSEG,
        IT_ERR_RESOURCE_SSEG,
        IT_ERR_SOFT_HI_WATERMARK,
        IT_ERR_SRQ_BUSY,
        IT_ERR_SRQ_LOW_WATERMARK,
        IT_ERR_SRQ_NOT_SUPPORTED,
        IT_ERR_TIMEOUT_EXPIRED,
        IT_ERR_TOO_MANY_POSTS,
        IT_ERR_WAITER_LIMIT
    } it_status_t;
```

**DESCRIPTION**

Most IT-API function calls return *it_status_t* on function completion. IT_SUCCESS indicates that an IT-API operation was invoked successfully; otherwise, the return code indicates the reason for failure. See each individual reference page for the meaning of a return code in the Context of the function.

Some API function calls are used to initiate asynchronous operations. For those function calls, a return value of IT_SUCCESS indicates only that the operation was successfully initiated; it does not indicate that it was successfully completed. To determine whether an asynchronous operation was successfully completed, the Completion Event for the asynchronous operation should be examined.

**SEE ALSO**

*it_address_handle_create*(), *it_address_handle_free*(), *it_address_handle_modify*(), *it_address_handle_query*(), *it_convert_net_addr*(), *it_ep_accept*(), *it_ep_connect*(), *it_ep_disconnect*(), *it_ep_free*(), *it_ep_modify*(), *it_ep_query*(), *it_ep_rc_create*(), *it_ep_reset*(), *it_ep_ud_create*(), *it_evd_create*(), *it_evd_dequeue*(), *it_evd_free*(), *it_evd_modify*(), *it_evd_post_se*(), *it_evd_query*(), *it_evd_wait*(), *it_get_consumer_context*(), *it_get_handle_type*(), *it_get_pathinfo*(), *it_handoff*(), *it_ia_create*(), *it_ia_free*(), *it_ia_query*(), *it_listen_create*(), *it_listen_free*(), *it_listen_query*(), *it_lmr_create*(), *it_lmr_free*(), *it_lmr_modify*(), *it_lmr_query*(), *it_lmr_flush_to_mem*(), *it_lmr_refresh_from_mem*(), *it_post_atomic*(), *it_post_rdma_read*(), *it_post_rdma_read_to_rmr*(), *it_post_rdma_write*(), *it_post_recv*(), *it_post_recvfrom*(), *it_post_send*(), *it_post_sendto*(), *it_pz_create*(), *it_pz_free*(), *it_pz_query*(), *it_reject*(), *it_rmr_link*(), *it_rmr_create*(), *it_rmr_free*(), *it_rmr_query*(), *it_rmr_unlink*(), *it_set_consumer_context*(), *it_socket_convert*(), *it_srq_create*(), *it_srq_free*(), *it_srq_modify*(), *it_srq_query*(), *it_ud_service_reply*(), *it_ud_service_request*(), *it_ud_service_request_handle_create*(), *it_ud_service_request_handle_free*(), *it_ud_service_request_handle_query*(), *it_dto_events*

# it_unaffiliated_event_t

**NAME**

it_unaffiliated_event_t – Unaffiliated Asynchronous Event type

**SYNOPSIS**

```
#include <it_api.h>

typedef struct {
    it_event_type_t  event_number;
    it_evd_handle_t  evd;
    it_ia_handle_t   ia;

    size_t  spigot_id;
} it_unaffiliated_event_t;
```

**DESCRIPTION**

| | |
|---|---|
| *event_number* | Identifier of the Event type. Valid values:<br>IT_ASYNC_UNAFF_SPIGOT_ONLINE,<br>IT_ASYNC_UNAFF_SPIGOT_OFFLINE,<br>IT_ASYNC_UNAFF_SEVD_ENQUEUE_FAILURE |
| *evd* | Handle for the Event Dispatcher where the Event was queued. |
| *ia* | The Handle for the IA that experienced the Unaffiliated Event. |
| *spigot_id* | The identifier for the Spigot that changed state on the IA. Valid only for the IT_ASYNC_UNAFF_SPIGOT_ONLINE and IT_ASYNC_UNAFF_SPIGOT_OFFLINE Events. |

IT_ASYNC_UNAFF_EVENT_STREAM Events are generated when an Unaffiliated Asynchronous Event occurs. There are several types of Unaffiliated Asynchronous Events, and each type is identified by *event_number*. The Consumer asks for Unaffiliated Asynchronous Events to be delivered when it creates an EVD for the Unaffiliated Asynchronous Event Stream using the *it_evd_create* call.

The following table maps the values in the Unaffiliated Asynchronous Events *it_event_type_t* 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 *it_ia_info.spigot_online_ support* 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 *it_ia_info.spigot_offline_ support* 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*()

# A Implementer's Guide

The IT-API Standard does not prohibit any implementation from providing functionality beyond that specified in the standard. However, we urge that implementions 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.

### it_address_handle_create

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.

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.

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.

### it_affiliated_event_t

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.

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.

### it_cm_msg_events

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.

In general, all transport-specific Connection Management rejection Events not explicitly defined in this API should be implemented as IT_CM_MSG_CONN_NONPEER_REJECT_EVENT with IT_CN_REJ_OTHER reject reason code Events.

When running over the InfiniBand Transport, Implementations have the option to “chew up” a REJ that is returned with reject reason code 1 (No QP available), 3 (No resources available), or 4 (Timeout) rather than immediately posting an Event with status IT_CN_REJ_RESOURCES (for REJ codes 1 and 3) or IT_CN_REJ_OTHER (for REJ code 4). The Implementation may wait until the timeout specified by the Consumer in the *it_path_t* structure input to *it_ep_connect* expires and then enqueue a non-peer reject Event with an IT_CN_REJ_TIMEOUT status. Alternatively, within the specified timeout period the Implementation may retry the Connection establishment attempt on the Consumer's behalf. If a Connection could not be established within the Consumer-specified timeout period, the Implementation should enqueue a non-peer reject Event with an IT_CN_REJ_TIMEOUT status after the timeout period has expired.

When running on the IB transport, there are two different things that can signal the Implementation that it should generate the IT_CM_MSG_CONN_ESTABLISHED_EVENT on the Passive side: receiving an RTU message, or receiving a “Communication Established” Affiliated Asynchronous Event from the HCA. (Due to inherent races in the IB Connection establishment process, it is also possible that both of these conditions could be present.) If the Implementation receives an RTU message while the Endpoint is in the IT_EP_STATE_PASSIVE_CONNECTION_PENDING state and within the timeout period advertised to the Passive side in the REQ message, it should generate an IT_CM_MSG_CONN_ESTABLISHED_EVENT Event and use the Private Data from the RTU as part of that Event. If the Implementation receives the “Communication Established” Affiliated Asynchronous Event without receiving an RTU, the Implementation should generate the IT_CM_MSG_CONN_ESTABLISHED_EVENT Event with a Private Data size of zero, and when/if the RTU for the Connection subsequently arrives the Implementation should ignore it.

For the InfiniBand Transport, there is a potential race condition in the three-way handshake Connection establishment method between the Implementation generating the IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT Event and the Consumer calling *it_ep_disconnect* (or *it_ep_free*).

The conditions for the race arise when a Consumer has called *it_ep_connect*, but before the Connection is successfully established, the Consumer calls *it_ep_disconnect* or *it_ep_free* on the Endpoint. Within the timeframe of this single Connection attempt, the Implementation must order Events as follows.

It is acceptable for the Implementation to generate and queue the IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT Event to the SEVD prior to the IT_CM_MSG_CONN_DISCONNECT_EVENT Event. But, when the IT_CM_MSG_CONN_DISCONNECT_EVENT Event is generated, the Implementation must invalidate the *cn_est_id* found in the IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT Event. Likewise, if the Consumer calls *it_ep_free*, the Implementation must also invalidate the *cn_est_id*. (Note that it is possible that the *cn_est_id* may have already been invalidated by a Consumer call to *it_ep_accept*, *it_reject* or *it_handoff*.)

On the other hand, if the Implementation has generated an IT_CM_MSG_CONN_DISCONNECT_EVENT Event, and subsequently the Implementation receives indication that

the Connection Request has been accepted by the remote side, the Implementation shall not generate an IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT Event. In this situation, the Implementation is still responsible for generating any necessary transport-specific response to the arrived acceptance message.

**it_conn_qual_t**

When the IANA Port Number Connection Qualifier type is used with the VIA transport, the IANA Port Number is mapped into a 2-byte VIA connection discriminator, with byte 0 of the connection discriminator containing the upper 8 bits of the 16-bit IANA Port Number, and byte 1 containing the lower 8 bits of the 16-bit IANA Port Number.

When the IANA Port Number Connection Qualifier type is used with the InfiniBand Transport, 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 | |

**it_dto_cmpl_event_t**

The handling of remotely detected errors is Implementation-dependent.

On the iWARP Transport, when a locally posted RDMA DTO results in a remote access violation, the remote iWARP Implementation tears down the connection abortively. For implementing the IT_DTO_ERR_REMOTE_ACCESS Completion Error on iWARP, the RDMAC Verbs provide a “remote termination error” completion status that allows on-the-fly conversion of a received Terminate message to a Completion Error. Due to delays in the reception of the Terminate message, it may not always be possible to use this mechanism.

**it_dto_flags_t**

The Implementation should attempt to support the use of IT_NOTIFY_FLAG on Receive DTOs. Where the underlying transport does not support Receive DTO Notification Suppression it may be necessary for the Implementation to generate Receive Notifications regardless of the setting of the IT_NOTIFY_FLAG on the Receive DTOs.

**it_ep_accept**

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 that the Event results in. 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.

### it_ep_attributes_t

The Implementation must allocate resources needed for the support of the Consumer-requested attributes. In general, the Implementation can allocate more resources than requested by the Consumer (a few exceptions are noted in the next paragraph). Some of these resources may actually be allocated at Connection establishment time for RC, but Connection establishment cannot fail because resources requested by the Consumer at Endpoint creation time are not available. Connection establishment may fail when the attributes associated with the local and remote Endpoints are mismatched. For example, for the InfiniBand and iWARP Transports, if the local Endpoint's attributes have an *rdma_read_ird* that is less than the remote Endpoint's *rdma_read_ord*, the Connection establishment attempt will fail.

The attributes that the Implementation must allocate in exactly the quantity that the Consumer specified are:

```
it_rc_only_attributes_t.rdma_read_ord
it_ep_attributes_t.max_dto_payload_size
```

The reason *rdma_read_ord* is listed above is that for InfiniBand you cannot establish a Connection unless the ORD value for one side of the Connection is less than or equal to the IRD value for the other side.

The reason *max_dto_payload_size* is listed above is that for VIA you cannot establish a Connection unless the passive side and the active side Endpoints have matching values for this attribute.

Whether the *max_dto_payload_size* limit is actually enforced for data transfers is IA-specific. (It might not be transport-specific; but it is not clear if all VIA Implementations do this checking.)

An attempt to change *max_request_dtos* or *max_recv_dtos* for an Endpoint of an Interface Adapter whose *it_ia_info ep_work_queues_resizable* is clear must not be successful and IT_ERR_INVALID_EP_STATE is the return value for this case.

When running over the InfiniBand or iWARP Transports, the Implementation must set Signaling type to Selectable in order to support *it_dto_flags*.

### it_ep_connect

When running over the InfiniBand Transport, if the Consumer provides a Path to *it_ep_connect* that contains a *P_Key* that is not in the HCA's *P_Key* table, the Implementation shall return IT_ERR_INVALID_SOURCE_PATH.

### it_ep_disconnect

If the EP is already in the IT_EP_STATE_NONOPERATIONAL state, no messages or Events should be generated. In this case, the transport-level Disconnect Request should have been sent when the EP transitioned into the non-operational state.

When running over the InfiniBand Transport, if *it_ep_disconnect* is called while the Endpoint is in the IT_EP_STATE_CONNECTED state, the Implementation should send CM DREQ (Disconnect Request) message.

### it_ep_free

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.

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*.

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.

### it_ep_rc_create

The Implementation should ignore the IT_EP_REUSEADDR *it_ep_rc_creation_flag_t* parameter on transports where the timewait state is not applicable.

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.

For the iWARP transport, the QP attribute for MW bind operations shall be set to “enabled”.

### it_ep_reset

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.

### it_ep_state_t

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.

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*.

### it_evd_create

An IT-API Implementation should meet the following rules for EVD behavior.

### Definitions

1. An arriving (queued on SEVD) Event is a *notification event* if any one of the following is true:
   a. It is an Event for a DTO with IT_NOTIFY_FLAG set when posted.
   b. It is an Event for a Recv DTO where matching Send DTO was originally posted with the IT_SOLICITED_WAIT_FLAG set (regardless of IT_NOTIFY_FLAG on Recv DTO).
   c. It is the *N*th Event to arrive where threshold is set to *N*.
   d. It is an Event for a DTO completing in error regardless of IT_NOTIFY_FLAG and IT_COMPLETION_FLAG.
   e. It is a non-DTO Completion Event.

   The above are called *arriving notification events*.

   The Events of type a, b, d, and e are also called *plain notification events* and retain their notification status[11] on the SEVD queue.
2. An arriving Event is a *non-notification event* if none of the above 1a. to 1e. criteria are met. These are called both *arriving non-notification events* and *plain non-notification events*.
3. IT_THRESHOLD_DISABLED stands for no threshold or threshold == infinity.
4. The desired semantic for IT-API is: *it_evd_wait* returns only:
   a. For SEVD – when there is a Notification Event of 1a, 1b, 1d, or 1e type (above) or when there are a number of Events on the SEVD equal to or more than the threshold on the SEVD queue.
   b. For AEVD – when any one of the associated SEVDs satisfies 4a.

### Rules

1. *it_evd_wait* **will** block[12] when:
   a. For SEVD – queue is empty.
   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.

2. *it_evd_wait* may block if there are no notification events and number of Events is below the threshold:
    a. For SEVD – on the SEVD queue.
    b. For AEVD – on all associated SEVD queues.
3. *it_evd_wait* **will** return if there is a *notification even*[13] or the number of Events is greater than or equal to the *threshold*:
    a. For SEVD – on the SEVD queue.
    b. For AEVD – on any associated SEVDs.
4. If threshold > 1, *it_evd_wait* **should** block[14] if less than threshold number of Events and no *notification events* are[15]:
    a. For SEVD – on the SEVD queue.
    b. For AEVD – on all SEVD queues of AEVD.
5. Each *arriving notification event* **must** unblock at least one waiter[16], but **should** unblock only one waiter.
6. An *arriving notification event* **can** unblock as many waiters as there are Events available.
7. *it_evd_dequeue* **will** return an Event if one exists regardless of waiters from:
    a. For SEVD – the SEVD queue.
    b. For AEVD with IT_EVD_DEQUEUE_NOTIFICATIONS set – the AEVD queue of IT_AEVD_NOTIFICATION_EVENTS
    c. For AEVD with IT_EVD_DEQUEUE_NOTIFICATIONS cleared – any of its 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.

### Heuristic for Wakeup of Multiple Waiters

Rules 5 and 6 in the EVD Rules (above) require that multiple waiters be awakened on every Notification Event because of the possibility of Notification coalescing.

The following heuristic is recommended: In the Notification Handler, check the queue length[18], and wake up that many waiters and no more. This limits the number of threads that wake up, and reduces the number that wake up and find no Event because some other threads(s) has consumed them.

The handler algorithm should be something like:

```
evd_handler() {
    for (;;) {
        nToUnblock = min(nmore(),nwaiters());
        for (n = 0; n != nToUnblock; ++n)
            unblockWaiter();
        if (nwaiters() == 0) break;
        rearmHandler();
        if (nmore() == 0) break;
    }
}
```

### Additional Issues

An AEVD does not have a queue as such. Potentially, an AEVD can be implemented using a bit array with an entry for each feeding SEVD. An SEVD bit "set" for an AEVD with IT_EVD_DEQUEUE_NOTIFICATIONS set means that the SEVD Handle can be returned in an IT_AEVD_NOTIFICATION_EVENT Event from the AEVD wait or dequeue. The SEVD bit “set” for an AEVD with IT_EVD_DEQUEUE_NOTIFICATIONS cleared means that the actual underlying SEVD Event can be returned from the AEVD wait or dequeue.

Ideally, the SEVD bit it cleared as soon as the SEVD is not in the Notification criteria. But it can be cleared as soon as the SEVD becomes empty or, instead, only when both it becomes empty and there is direct (waiter on SEVD) or indirect waiter (AEVD waiter or FD) on the SEVD.

If the SEVD is enabled, then an arriving SEVD Event that causes the SEVD to reach Notification criteria or maintain Notification criteria of the SEVD will set the SEVD bit for the AEVD.

It is recommended that the Implementation employ a “starvation-free” algorithm in returning Events via an AEVD from underlying SEVDs. That is, the Implementation should ensure forward progress on all SEVDs feeding an AEVD.

For IT_EVD_DEQUEUE_NOTIFICATION set, the Implementation should eventually select every feeding SEVD that has reached Notification status when generating the next AEVD Notification Event.

For IT_EVD_DEQUEUE_NOTIFICATION cleared, the Implementation of *it_evd_wait* should 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.

When the requested stream type is IT_ASYNC_AFF_EVENT_STREAM or IT_ASYNC_UNAFF_EVENT_STREAM, the Implementation creates the requested EVD, and fills in the appropriate *evd* field in the associated *it_ia_info_t* structure for that IA.

The Implementation shall not provide to the Consumer Unaffiliated and Affiliated Events that happened before the Consumer created SEVDs for an Unaffiliated and Affiliated Event Stream. The Implementation can provide Events that happened during SEVD creation time.

**it_evd_dequeue**

The Implementation should abide by the EVD rules defined in the *it_evd_create* section of this Implementers Guide.

All synchronization issues between multiple waiters and/or dequeue-ers from the same Event Dispatcher simultaneously are left to the Implementation.

The Implementation is not required to check that the *Event* structure that the Consumer provides is sufficient to hold a returned Event.

**it_evd_post_se**

All the synchronization issues between multiple Consumer Contexts trying to post software Events to an Event Dispatcher instance simultaneously are left to the Implementation.

**it_evd_wait**

The Implementation should abide by the EVD rules defined in the *it_evd_create* section of this Implementers Guide.

All synchronization issues between multiple waiters and/or dequeue-ers from the same Event Dispatcher simultaneously are left to the Implementation.

The Implementation is not required to check that the *event* structure that the Consumer provides is sufficient to hold a returned Event.

**it_event_t**

Each Event Stream has a designated contiguous range of Event numbers with common most-significant bits (as masked by IT_EVENT_STREAM_MASK) representing the Event Stream.

**it_handle_t**

The definition of all object Handles is Implementation-specific, but all object Handles can be typecast to *it_handle_t* and *vice versa*.

**it_handoff**

Once the IA is open, and until all processes close it, the *ia_name* and Spigot identifier need to 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

rules out Relative Addressing), the IOVA output value may differ from the IOVA input value if byte granularity for Memory Regions is not supported; if there is a difference, the Implementation of *it_lmr_create* shall verify the assertion that the IOVA output value equals the lower bound of the "Actual Local Protection Bounds enforced by the Channel Interface" returned by the Query Memory Region verb and, if this assertion is false, deregister the Memory Region and return the IT_ERR_ABORT immediate error.

Using the Register Physical Memory Region verb has the advantage that control over Memory Region pinning/unpinning remains in generic, OS-provided code, while using the Register Memory Region verb would pass such control to IHV-private code. Using the Register Physical Memory Region verb is thus preferred, since it (i) retains sufficient OS control over pinned memory, (ii) supports multiple registrations of Memory Regions in a multi-IHV environment, and (iii) reduces duplication of pinning code in IHV verbs provider drivers.

On the iWARP Transport, *it_lmr_create* can be implemented with the Register Non-Shared Memory Region verb.

For both InfiniBand and iWARP, the Implementation of *it_lmr_create* must set the Enable Memory Window Binding input modifier to "true" in order to allow subsequent calls to *it_rmr_link* using this LMR. The IT_LMR_FLAG_SHARED option can be implemented by registering either a new or a shared Memory Region. The Implementation should search for a matching LMR. If a match is found, call the Register Shared Memory Region Verb; if not, call the Register (Physical) Memory Region verb for InfiniBand and the Register Non-Shared Memory Region verb for iWARP.

The remote access flag for memory region allocation, registration, or re-registration (iWARP and IB-1.2:BMM) is not exposed by IT-API and should be set to "true" by the Implementation when creating an LMR.

The Implementation should protect against race conditions between multiple Consumers and avoid calling memory registration verbs multiple times for the same memory region.

See the advice under *it_rmr_link* for comments concerning *rmr_context* and byte order.

#### it_lmr_create_unlinked

For InfiniBand and iWARP, *it_lmr_create_unlinked* corresponds to the Allocate L_Key verb and the Allocate Non-Shared Memory Region STag verb, respectively.

The remote access flag for memory region allocation, registration, or re-registration (iWARP and IB-1.2:BMM) is not exposed by IT-API and should be set to "true" by the Implementation when creating an unlinked LMR.

#### it_lmr_free

For InfiniBand and iWARP, *it_lmr_free* corresponds to the Deregister Memory Region verb and the Deallocate STag verb, respectively.

Beware that determining when it is permissible to unlock or unpin physical memory is tricky. The Implementation must handle multiple LMRs with overlapping ranges, LMRs on different

IAs with overlapping ranges, and LMRs created in overlapping regions of shared memory by different Consumers. In addition, any of these LMRs may have been created with the IT_LMR_FLAG_SHARED flag set.

#### it_lmr_link

For InfiniBand and iWARP, *it_lmr_link* corresponds to the Fast Register Physical Memory Region verb and the Fast-Register Non-Shared Memory Region verb, respectively.

For both InfiniBand and iWARP, the Implementation of *it_lmr_link* must include the Enable Memory Window Binding input modifier to allow subsequent calls to *it_rmr_link* using this LMR.

#### it_lmr_modify

On the InfiniBand Transport, *it_lmr_modify* corresponds to the Reregister Physical Memory Region or Reregister Memory Region verb. When an LMR enabled for remote access is modified, the Implementation should modify the 8-bit Key Portion of the R_Key (which is shared with the Key Portion of the L_Key) to reduce the potential for problems with stale R_Keys.

On the iWARP Transport, *it_lmr_modify* can be implemented with the Reregister Non-Shared Memory Region verb for a non-shared LMR, and with the Deallocate STag verb followed by the Register Shared Memory Region verb for a shared LMR. When an LMR enabled for remote access is modified, the Implementation should modify the 8-bit STag Key to reduce the potential for problems with stale STags.

#### it_lmr_query

For both InfiniBand and iWARP, *it_lmr_query* can be implemented with the Query Memory Region verb.

For InfiniBand, the *actual_addr* and *actual_length* fields in *params* should be derived from the Actual Remote Protection Bounds returned by the verb, because the Consumer will be most concerned with the degree of exposure of the region to remote Consumers. For a transport Implementation that does not provide Memory Region bounds checking with byte-level granularity, the remote bounds are obtained by rounding the bounds requested by the Consumer to 4K byte boundaries, and the local bounds are obtained by rounding the bounds requested by the Consumer to page boundaries. The Actual Remote Protection Bounds will be contained within the Actual Local Protection Bounds for any IB transport Implementation. This means the Consumer may safely use the returned *actual_addr* and *actual_length* as both local and remote bounds.

For iWARP and any transport supporting Relative Addressing, LMR creation and bounds checking must be performed with byte-level granularity. For InfiniBand 1.2, LMR creation and bounds checking should be performed with byte-level granularity. In case of byte-level granularity, the actual starting address and actual length must equal the requested starting address and requested length, respectively.

#### it_lmr_flush_to_mem
#### it_lmr_refresh_from_mem

There is no defined error code for the case where some portion of the *local_segments* array lies outside the Consumer's valid address space. It is expected that the Implementation will signal the application with the appropriate platform-dependent signal in this case, as would happen for any dereference of an invalid pointer.

#### it_lmr_unlink

For InfiniBand and iWARP, *it_lmr_unlink* corresponds to the Local Invalidate verb and the Invalidate Local STag verb, respectively.

#### it_post_rdma_read

The Implementation should avoid resource allocation as part of *it_post_rdma_read* to ensure that this operation is non-blocking and thread-safe. This operation cannot fail due to insufficient resources. All resource allocation required must be done at Endpoint creation time to ensure that all necessary resources are available at post time.

The Implementation should support zero-copy data transfers and kernel bypass for the RDMA Read operation.

For iWARP, an RNIC may provide advanced RDMA Read error handling support; i.e., RDMAP/DDP support for mapping an error due to an inbound RDMA Read Response to a Completion Error, support for on-the-fly conversion of an RDMAP Terminate message into a Completion Error and, optionally, early detection of a local access violation due to an RDMA Read Request.

RDMAP/DDP support for mapping an error due to an inbound RDMA Read Response to a Completion Error is optional in DDP, but useful. On-the-fly conversion of an RDMAP Terminate message into a Completion Error is useful in case of a remotely detected access violation due to an RDMA Read Request, which results in a Terminate message containing the RDMA Read Request header. Early detection of a local access violation due to an RDMA Read Request is useful since certain access violations pertaining to the local sink buffer or Endpoint can be predicted before an RDMA Read Request is issued via the underlying transport protocol; issuing an RDMA Read Request can be avoided in this case, but incoming RDMA Writes/Read Responses are checked anyway regarding access rights.

#### it_post_rdma_read_to_rmr

This call is available for iWARP only.

The Implementation should avoid resource allocation as part of *it_post_rdma_read_to_rmr* to ensure that this operation is non-blocking and thread-safe. This operation cannot fail due to insufficient resources. All resource allocation required must be done at Endpoint creation time to ensure that all necessary resources are available at post time.

The Implementation should support zero-copy data transfers and kernel bypass for the RDMA Read operation.

An RNIC may provide advanced RDMA Read error handling support; i.e., RDMAP/DDP support for mapping an error due to an inbound RDMA Read Response to a Completion Error, support for on-the-fly conversion of an RDMAP Terminate message into a Completion Error and, optionally, early detection of a local access violation due to an RDMA Read Request.

RDMAP/DDP support for mapping an error due to an inbound RDMA Read Response to a Completion Error is optional in DDP, but useful. On-the-fly conversion of an RDMAP Terminate message into a Completion Error is useful in case of a remotely detected access violation due to an RDMA Read Request, which results in a Terminate message containing the RDMA Read Request header. Early detection of a local access violation due to an RDMA Read Request is useful since certain access violations pertaining to the local sink buffer or Endpoint can be predicted before an RDMA Read Request is issued via the underlying transport protocol; issuing an RDMA Read Request can be avoided in this case, but incoming RDMA Writes/Read Responses are checked anyway regarding access rights.

#### it_post_rdma_write

The Implementation should avoid resource allocation as part of *it_post_rdma_write* to ensure that this operation is non-blocking and thread-safe. This operation cannot fail due to insufficient resources. All resource allocation required must be done at Endpoint creation time to ensure that all necessary resources are available at post time.

The Implementation should support zero-copy data transfers and kernel bypass for the RDMA Write operation.

#### it_post_recv

The Implementation should avoid resource allocation as part of *it_post_recv* to ensure that this operation is non-blocking and thread-safe. This operation cannot fail due to insufficient resources. All resource allocation required must be done at Endpoint or Shared Receive Queue creation time to ensure that all necessary resources are available at post time.

The Implementation should support zero-copy data transfers and kernel bypass for the Receive operation.

#### it_post_recvfrom

The Implementation should avoid resource allocation as part of *it_post_recvfrom* to ensure that this operation is non-blocking and thread-safe. This operation cannot fail due to insufficient resources. All resource allocation required must be done at Endpoint creation time to ensure that all necessary resources are available at post time.

The Implementation should support zero-copy data transfers and kernel bypass for the ReceiveFrom operation.

#### it_post_send

The Implementation should avoid resource allocation as part of *it_post_send* to ensure that this operation is non-blocking and thread-safe. This operation cannot fail due to insufficient

resources. All resource allocation required must be done at Endpoint creation time to ensure that all necessary resources are available at post time.

The Implementation should support zero-copy data transfers and kernel bypass for the Send operation.

#### it_post_sendto

The Implementation should avoid resource allocation as part of *it_post_sendto* to ensure that this operation is non-blocking and thread-safe. This operation cannot fail due to insufficient resources. All resource allocation required must be done at Endpoint creation time to ensure that all necessary resources are available at post time.

For details on handling IT_ERR_INVALID_AH under InfiniBand see compliance statement o10-2.1.1 in [IB-R1.2].

The Implementation should support zero-copy data transfers and kernel bypass for the *SendTo* operation.

#### it_rmr_create

For both InfiniBand and iWARP, *it_rmr_create* corresponds to the Allocate Memory Window verb.

#### it_rmr_free

For InfiniBand and iWARP, *it_rmr_free* corresponds to the Deallocate Memory Window verb and the Deallocate STag verb, respectively.

#### it_rmr_link

On the InfiniBand Transport, *it_rmr_link* must immediately select the appropriate verb for linking an RMR, namely Bind MW for a Wide RMR and Post Send Bind MW for a Narrow RMR.

On the iWARP Transport, *it_rmr_link* translates to the PostSQ verb's Bind Memory Window operation. The Implementation should modify the 8-bit STag Key when binding a Memory Window to reduce the potential for problems with stale STags.

The *rmr_context* returned by *it_rmr_link* is defined to be returned in network byte order; i.e., in big endian format. The intent is that no further reordering of the bytes of the *rmr_context* will be performed by either the local or remote Consumer, or the local or remote Implementation. When the remote Consumer passes the *rmr_context* to a call such as *it_post_rdma_write*, the Implementation of *it_post_rdma_write* will put the first byte of *rmr_context* on the network wire first, the second byte of *rmr_context* second, and so on. Thus, the Implementation of *it_rmr_link* should return the *rmr_context* in the byte order that the IA expects to see on the wire, so that the IA may correctly interpret the incoming *rmr_context*.

#### it_rmr_query

For InfiniBand and iWARP, *it_rmr_query* can be implemented with the Query Memory Window verb.

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.

#### it_rmr_unlink

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.

For iWARP, *it_rmr_unlink* corresponds to the PostSQ verb’s Invalidate Local STag operation.

#### it_ud_service_request_handle_create

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*.

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.

#### it_unaffiliated_event_t

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.

In the case of Unaffiliated Events, the underlying Implementation should copy every Event into each process-specific queue.

# B Implementer's Guide to Connection Management for iWARP

## B.1 Overview

The IT-API provides RDMA services for several underlying RDMA transports, including InfiniBand and iWARP[19]. The specifics of these RDMA transports as well as different application requirements led to the design of two different Connection management interfaces for Reliable Connected (RC) RDMA transports, as introduced below.

The Transport-Independent Interface (TII) provides a unified RDMA Connection management service abstraction for any RC transport supported by the IT-API. The TII includes the *it_ep_connect*, *it_listen_create*, *it_ep_accept*, *it_reject*, and *it_ep_disconnect* calls. The TII allows Consumers to negotiate the IRD and ORD parameters and to exchange Private Data during Connection establishment. Consumers are expected to preconfigure the IRD and ORD parameters of their Endpoints prior to calling *it_ep_connect* or *it_ep_accept*. From the Consumer's viewpoint, the RDMA service is available immediately after Connection establishment. For iWARP, Connection establishment with the TII involves creating and connecting a socket within the API Implementation, as described in Section 5.2. This socket is not exposed through a Lower Layer Protocol (LLP) handle to the Consumer.

The Transport-Dependent Interface (TDI) is available for iWARP only and allows the Conversion of an unconnected Endpoint of the RC type and a connected socket to a connected Endpoint. The TDI includes the *it_socket_convert* and *it_ep_disconnect* calls. It enables Consumers to exchange an arbitrary (but non-zero) amount of streaming-mode data via a TCP socket (and possibly via an SCTP socket in the future) prior to performing a Conversion, which is a requirement for several TCP/IP-based applications. This prior exchange of streaming-mode data occurs outside the IT-API. In contrast to Connection establishment with the TII, the *it_socket_convert* call neither supports the negotiation of IRD and ORD parameters, nor the exchange of Private Data. The main reason for this simplification is that the peers have an opportunity to negotiate IRD and ORD and to exchange Private Data prior to Conversion; moreover, they have an opportunity to modify ORD after the Conversion[20]; i.e., while already in RDMA mode. The Implementation of the TDI is described in Section 5.3.

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].

## B.2 Miscellaneous

### Private Data

The IT-API must report Private Data length supported for the *it_ep_connect*, *it_ep_accept*, and *it_reject* calls as 256 bytes and report the Private Data length supported for the *it_ep_disconnect* routine as zero.

The IT-API must report Private Data length supported for the Unreliable Datagram service as zero as well even though the iWARP Transport does not support the IT_UD_SERVICE service type.

The Private Data fields are fixed-size and it is the Consumer's responsibility to define which portion of the fields contain their own valid data. Such a requirement is legacy behavior from other RDMA transports (specifically InfiniBand).

### Handshake

The IT-API supports only two-way Connection establishment for iWARP. The *three_way_handshake_support* boolean found in the *it_ia_info_t* structure must be reported as IT_FALSE.

## B.3 Interoperability Considerations

The IETF has released a draft document describing modifications to the MPA startup wire protocol [INTEROP-IETF] intended to aid in achieving interoperability between the RDMAC and IETF versions of iWARP.

Where interoperability is not possible – i.e., between Non-permissive AV-RNIC/IETFs and RDMAC AV-RNICS – Connection attempts via the TII or TDI should result in IT_CM_MSG_CONN_NONPEER_REJECT_EVENT Events with the reject reason code IT_CN_REJ_BAD_CONN_PARMS or IT_CM_MSG_CONN_BROKEN_EVENT Events. See the Interoperability Ladder Diagrams found in both the TII and TDI sections of this document.

Where the MPA startup protocol is not used – i.e., between RDMAC AV-RNICs – interoperability concerns are left to the ULP.

## B.4 MPA Marker Control

The RDMAC version of MPA [MPA-RDMAC] mandated the use of MPA framing (Markers) for the use of recovering placement information for DDP frames. The IETF version of MPA [MPA-IETF] made the use of Markers optional and as specified by the receiver.

The IT-API implementations should implement the recommendations of [INTEROP-IETF].

## B.5 Transport-Independent Interface

### Overview

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

The Endpoint finite-state machine uses the active (passive) Endpoint states for the RDMA Initiator (Responder).

Connection establishment for the TII in case of an RNIC without RTR state (see below) is illustrated in Figure 13.

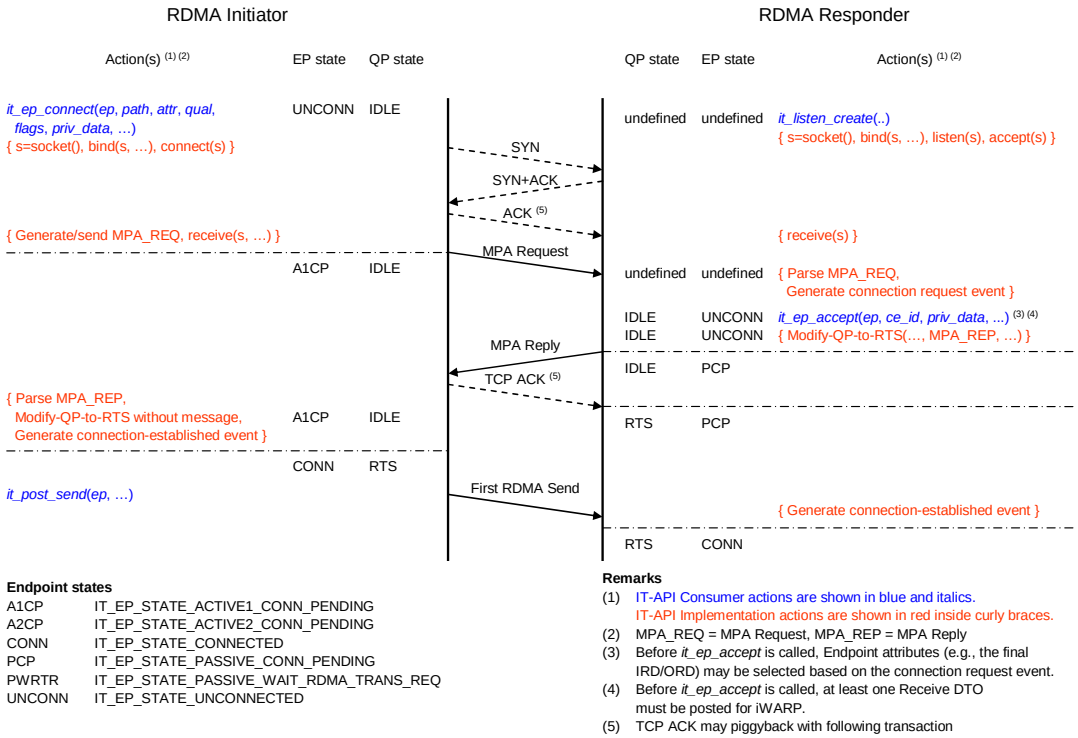



**Figure 13: Connection Establishment with MPA Startup (TII): RNIC without RTR State**

The RDMA Responder uses *it_listen_create* to create a Listen Point, which causes the (API) Implementation to create and bind a socket, to listen for an incoming LLP connection request, and to accept such a request. The RDMA Initiator invokes *it_ep_connect* for connecting an Endpoint of the RC type, which causes the Implementation to create and bind a socket and to attempt a connection. As soon as the LLP connection is established, the Implementation completes TII Connection establishment by performing an Immediate RDMA Transition. 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 visible to the Consumers only through the exchange of Private Data; in particular, the Implementation handles the details of the MPA Startup.

The RDMA Initiator generates and sends an MPA Request message with private data containing the IRD/ORD Header (IOH) and the private data provided by the Consumer with *it_ep_connect*. After receiving a valid MPA Reply message with the MPA Reject bit not set, the RDMA Initiator invokes the Modify-QP-to-RTS verb with a message length of zero and generates a Connection established event for the Consumer providing the remote Endpoint's IRD and ORD parameters as well as the remote Consumer's private data.

The RDMA Responder's Listen Point expects an MPA Request. Upon receiving a valid MPA Request message, it generates a Connection Request event for the Consumer containing the remote Endpoint's IRD and ORD parameters as well as the remote Consumer's Private Data. The Consumer on the RDMA Responder side now invokes either *it_ep_accept* or *it_reject*, optionally with Private Data.

Calling *it_ep_accept* causes the Implementation to associate the Connection establishment ID (and the corresponding LLP connection) with an Endpoint. It then causes the Implementation to generate an MPA Reply message with Private Data containing the IOH and the Private Data provided by the Consumer. Next, the Implementation 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 of *it_ep_accept* generates a Connection established event for the Consumer providing the remote Endpoint's IRD and ORD parameters as well as the remote Consumer's Private Data, and transitions the Endpoint into IT_EP_STATE_CONNECTED state.

For Figure 13, the Modify-QP-to-RTS verb is assumed to transition the QP to RTS state without waiting for the reception of the first iWARP payload (FPDU or RDMA message). While this behavior complies with [VERBS-RDMAC], it will not guarantee that the underlying iWARP Implementation stalls the sending of iWARP payload before the first iWARP payload has been received, as required by [MPA-IETF]. The fact that premature sending of the iWARP payload could result in synchronization problems and that receiving a Completion for an RDMAP Post Receive work request is the only way to detect the arrival of the first iWARP payload on an RNIC complying with [VERBS-RDMAC] has the following, far-reaching consequences:

- For the ULP, after reaping the Connection established event, the RDMA Initiator must post a Send DTO to allow the RDMA Responder to detect the first iWARP payload.
- For the ULP, the RDMA Responder must post at least one Receive DTO prior to calling *it_ep_accept*.
- For the Implementation of *it_ep_accept*, the reception of the first iWARP payload must be detected by noting the Completion of the Consumer's first Receive DTO.

In order to minimize transport-dependent ULP requirements, the IT-API offers an alternative for RNICs that implement an additional *Ready-To-Receive (RTR)* QP state to wait for the first iWARP payload, as well as an additional asynchronous event to indicate the RTR-to-RTS transition when the first iWARP payload is received on the RDMA Responder side. Connection

establishment for the TII in case of an RNIC with an RTR state (extended QP state machine) is depicted in Figure 14.

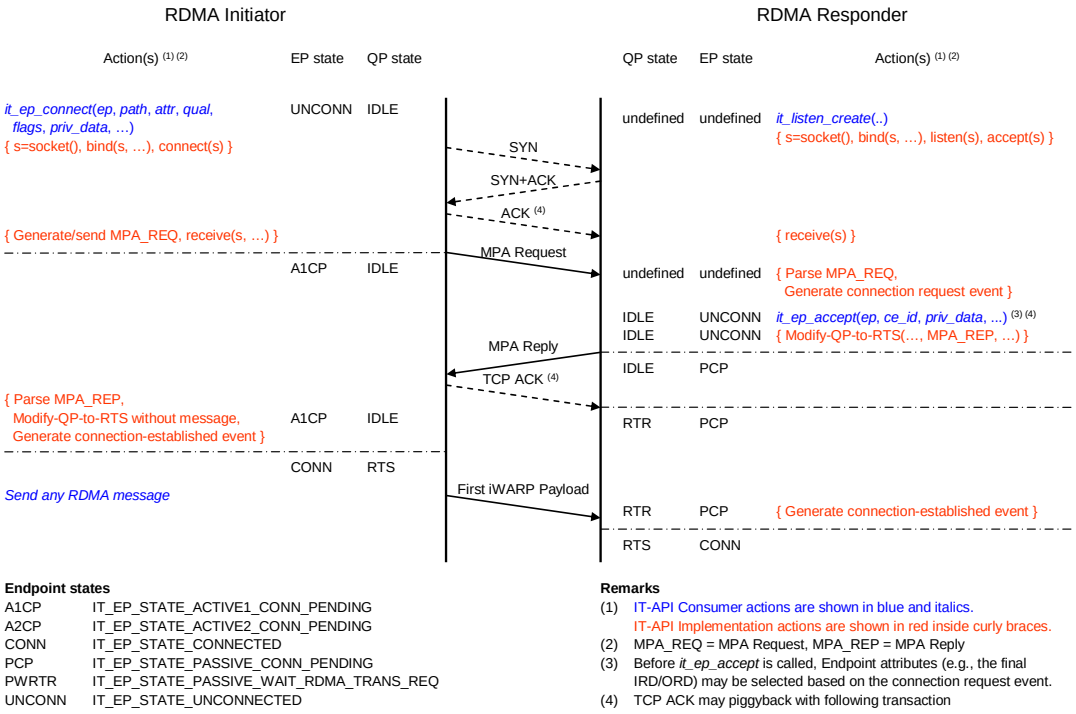



**Figure 14: Connection Establishment with MPA Startup (TII): RNIC with RTR State**

This alternative approach has the following consequences:

- 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.

### IT-API ULP and IRD/ORD Header

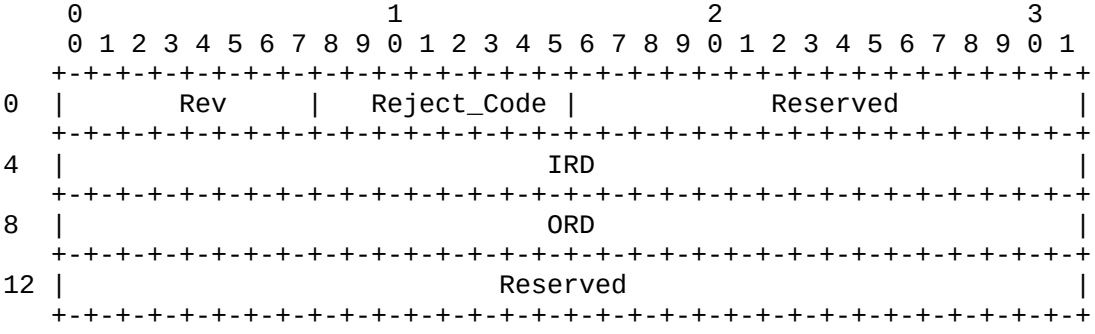
RDMA transitions are not fully specified by the iWARP protocol suite. For instance, [MPA-IETF] leaves it up to its ULP (i) when to enter MPA Startup, (ii) which side initiates the MPA Startup (i.e., sends MPA Request), and (iii) how to use the private data in the MPA Request/Reply messages.

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:

- The MPA Startup Phase shall start immediately after LLP connection setup.
- The peer initiating the LLP connection establishment shall initiate the MPA Startup.
- Due to the lack of standardized IRD/ORD fields in the MPA Request/Reply messages and the requirement to exchange IRD/ORD during Connection establishment, the first 16 bytes of the MPA Private Data buffer shall contain an *IRD/ORD Header (IOH)* specified by the IT-API for IRD/ORD negotiation. Note that there is an effort proposed for the IETF to standardize an equivalent header to the IOH but as of this writing details are not available. Due consideration of incorporation of the IETF proposal shall be taken.

#### IRD/ORD Header

The IT-API specifies the layout of the IRD/ORD Header (IOH) according to Figure 15.




```
Rev: This field contains the Revision of the IRD/ORD header. For this
version of the specification senders MUST set this field to '1'.
Receivers compliant with this version of the specification MUST check
this field for ‘1', and close the connection and report an error
locally if any other value is detected.

Reject_Code: This field is read in combination with the MPA reject bit
and supplies additional information (if available). The field contains
one of the following values: 0x00 means no additional information. 0x01
means iWARP version mismatch (surfaced as IT_CN_REJ_BAD_CONN_PARMS in
IT-API). 0x02 means IRD too small (called IT_CN_REJ_BAD_ORD in IT-API).
All other values are reserved. This field only has validity if the MPA
reject bit is set.

Reserved: This field is reserved for future use. It must be set to zero
when sending, and must not be checked on reception.

IRD: This field contains the sender's Inbound RDMA Read Queue Depth as
a 32-bit word in network byte order.

ORD: This field contains the sender's Outbound RDMA Read Queue Depth as
a 32-bit word in network byte order.

Reserved: This field is reserved for future use. It must be set to zero
when sending, and not checked on reception.
```

**Figure 15: IRD/ORD Header**

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.

## TII Interoperability Ladder Diagrams

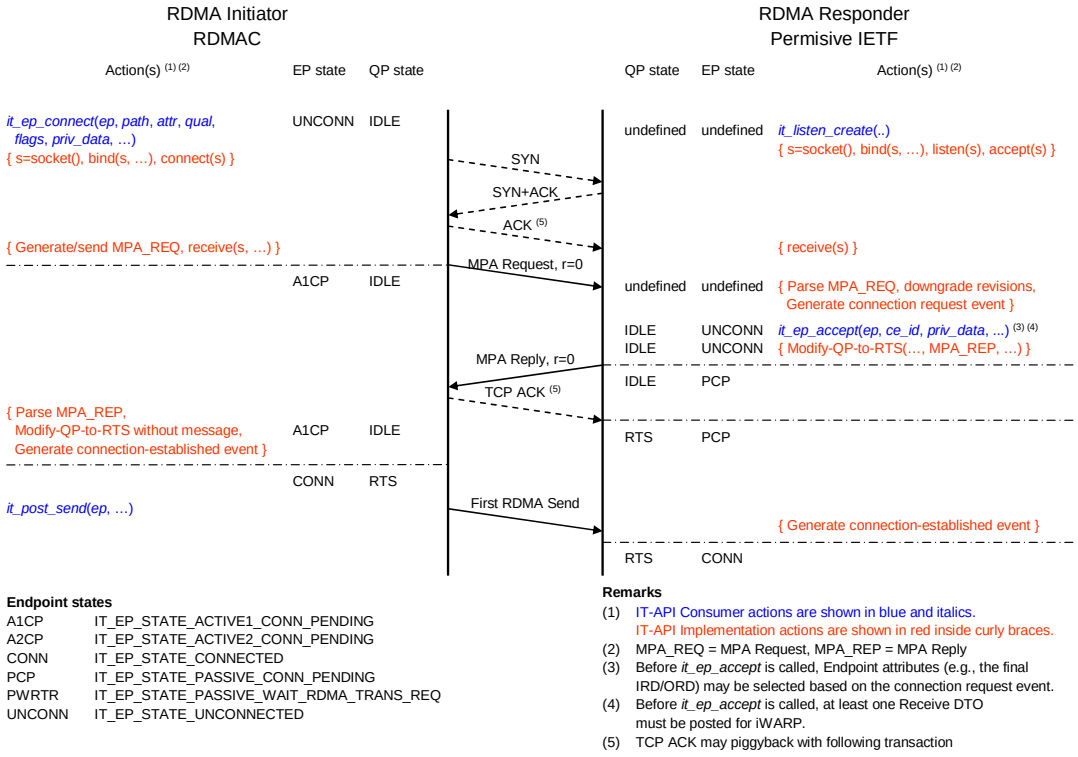
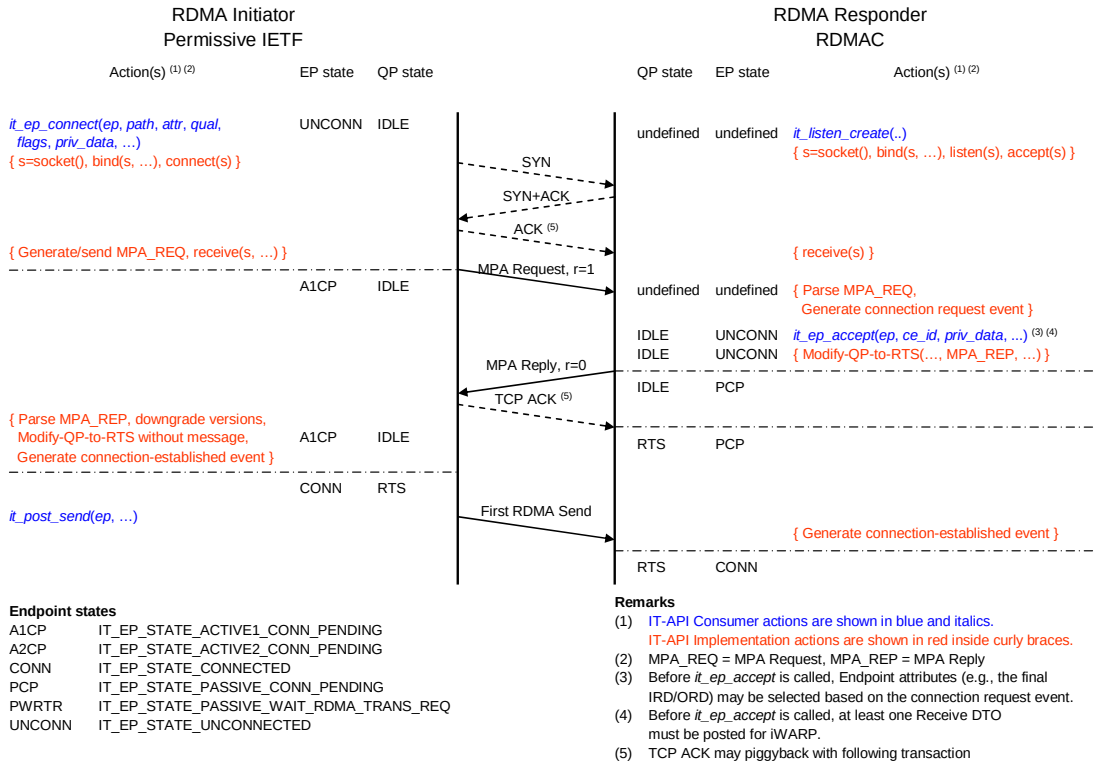
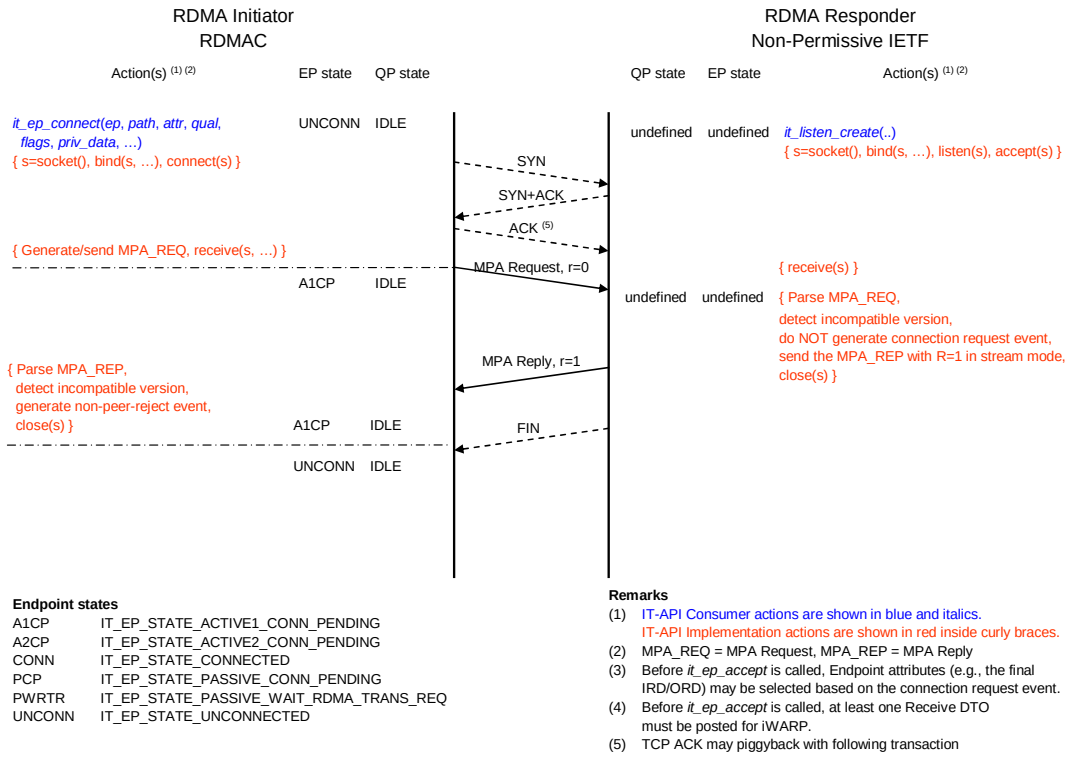



**Figure 16: RDMAC Initiator to Permissive IETF Responder**




12291

12292 **Figure 17: Permissive IETF Initiator to RDMAC Responder**




12293

12294 **Figure 18: RDMAC Initiator to Non-Permissive IETF Responder**

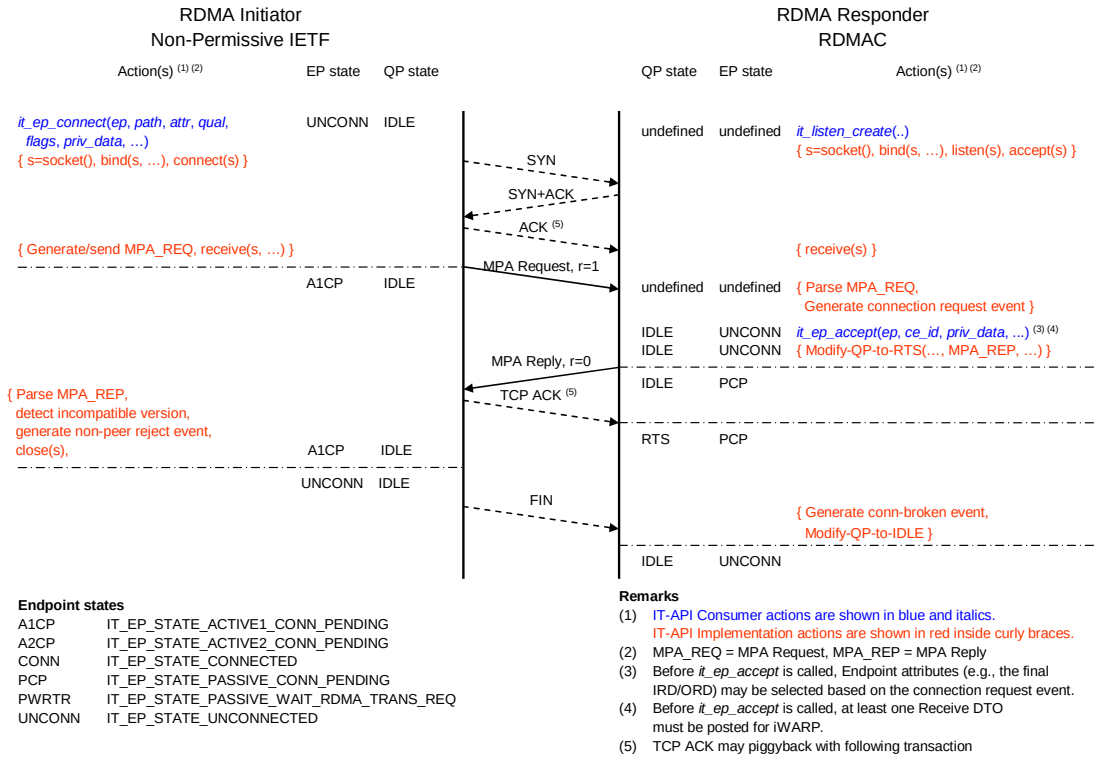



**Figure 19: Non-Permissive IETF Initiator to RDMAC Responder**

## B.6 Transport-Dependent Interface

### Overview

For the *Transport-Dependent Interface (TDI)*, the Consumer calling *it_socket_convert* with a *Last ULP Streaming-Mode (SM) Message* of length greater than zero is referred to as the *Conversion Initiator*, while the Consumer calling *it_socket_convert* with a message length of zero is referred to as the *Conversion Responder*.

As for the TII, the Endpoint finite-state machine uses the active (passive) Endpoint states for the RDMA Initiator (Responder). In order to avoid an inconsistency in the meaning of Endpoint states between TII and TDI, the TDI uses the convention that the Conversion Initiator and the corresponding side of the connection is in the *RDMA Responder* role, while the Conversion Responder and the corresponding side of the connection is in the *RDMA Initiator* role.

The Conversion process in case of an RNIC without RTR state is illustrated in Figure 20.

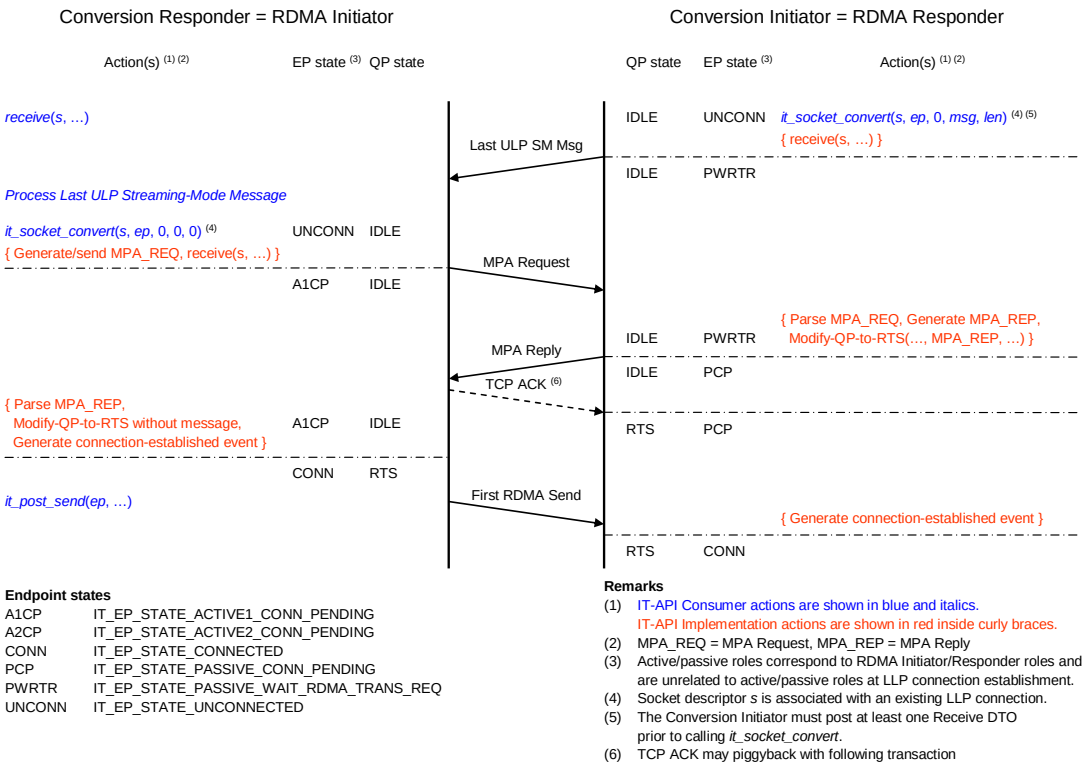



**Figure 20: Conversion Process with MPA Startup (TDI) - RNIC without RTR State**

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.

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.

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.

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

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.

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.

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.

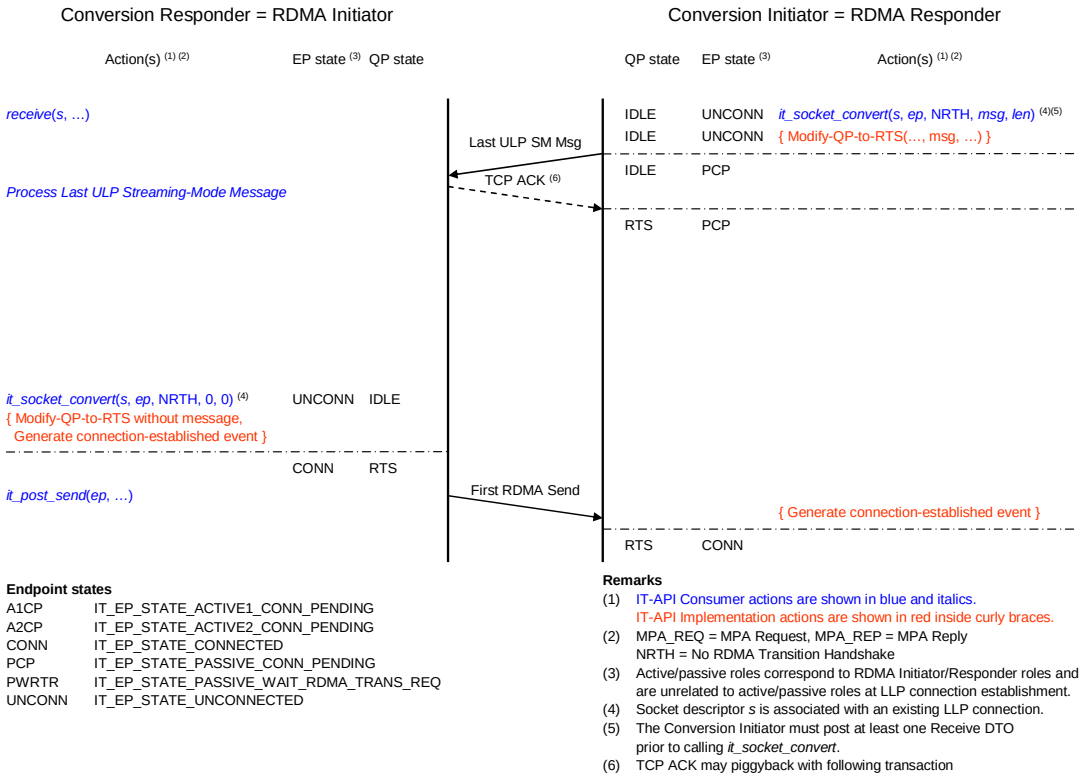
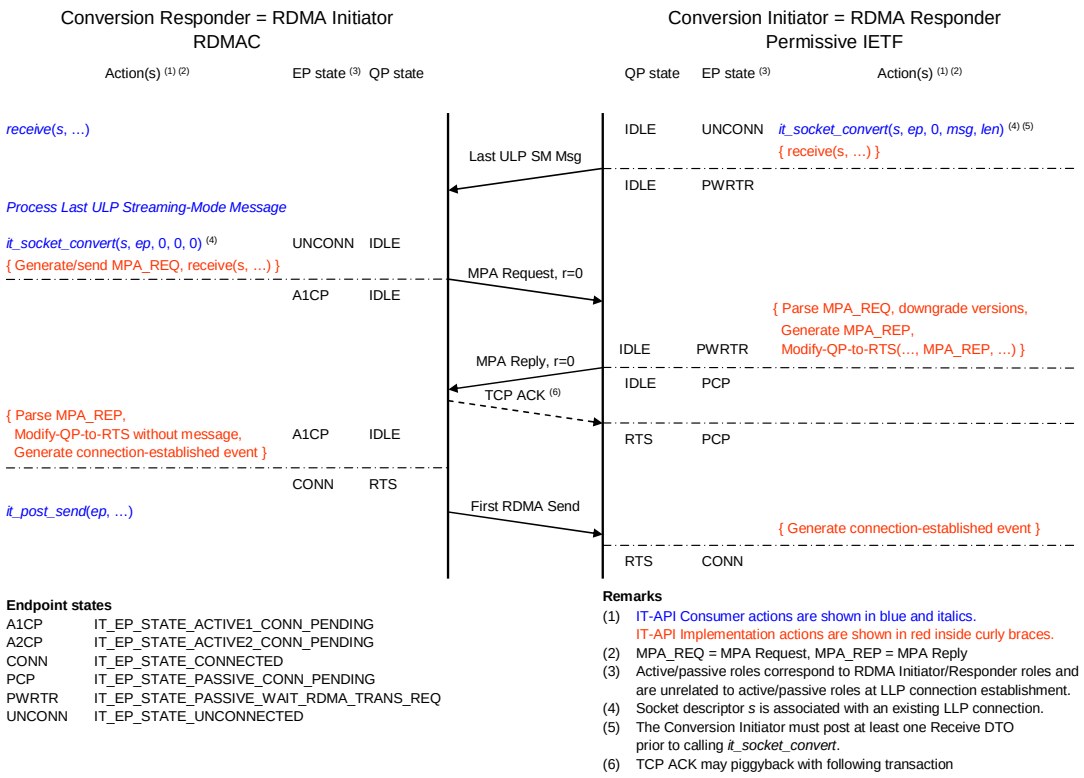


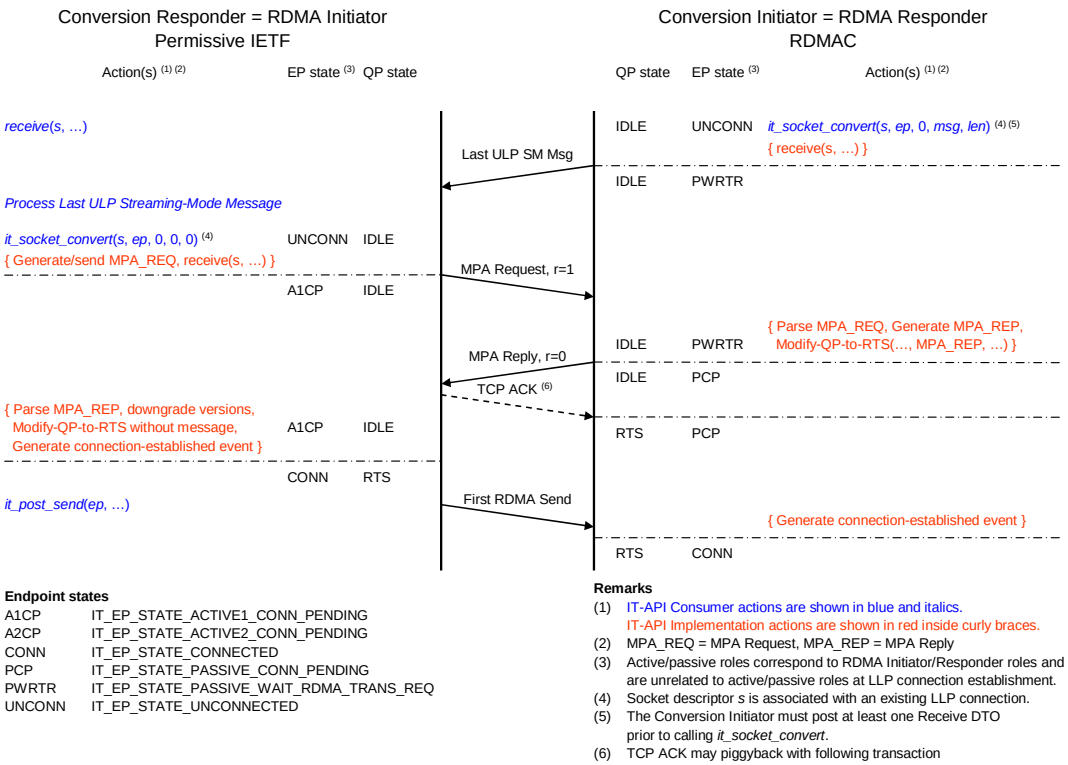

**Figure 21: Conversion Process with MPA Startup Suppression (TDI): RNIC without RTR State**

12349 **TDI Interoperability Ladder Diagrams**



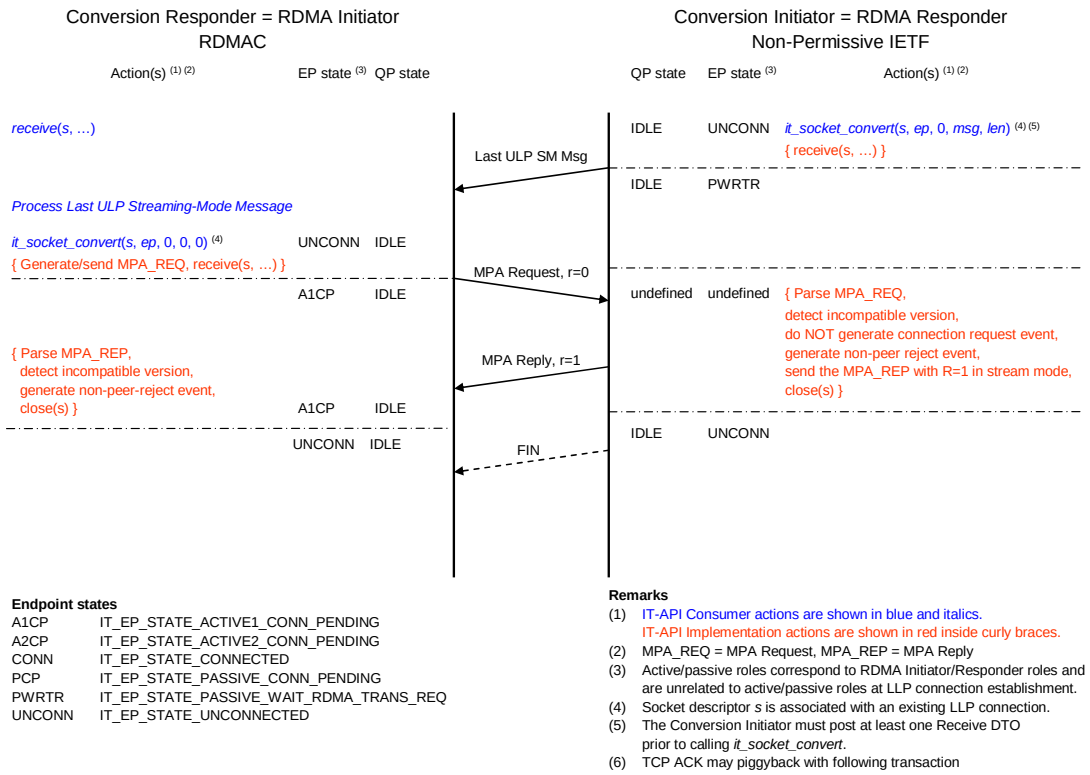

12350

12351 **Figure 22: Permissive IETF Conversion Initiator to RDMAC Conversion Responder**



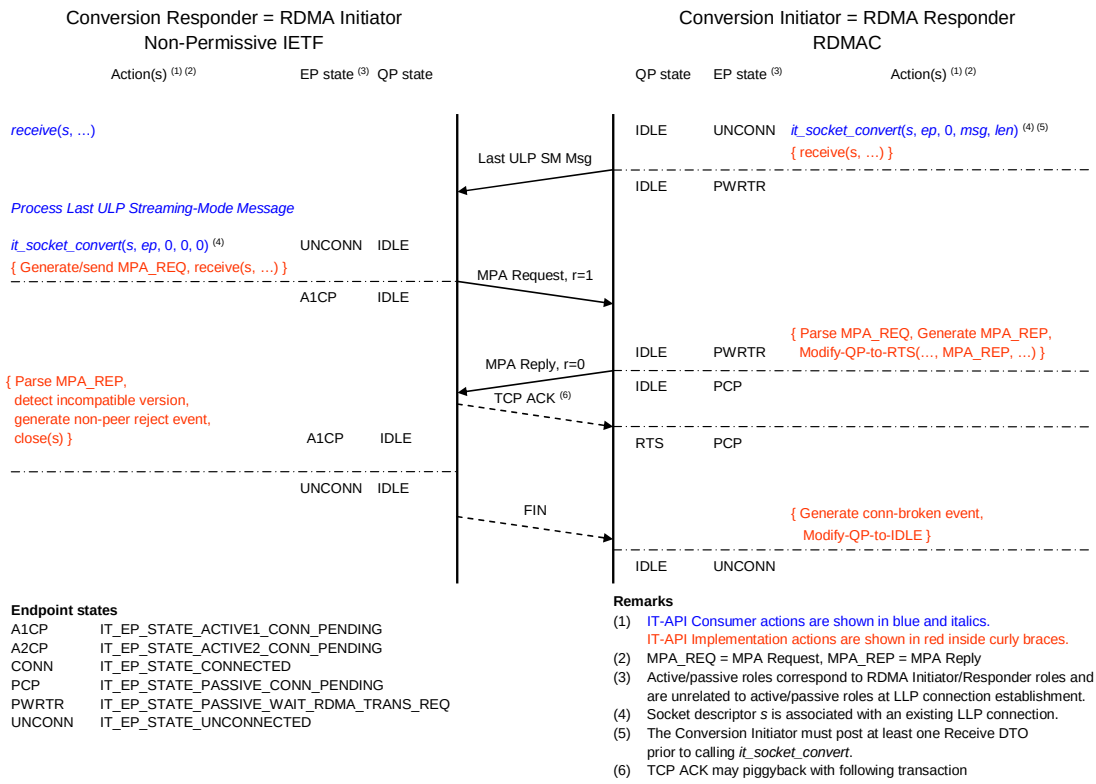

12352

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**

# C Backwards Compatibility with Earlier IT-API Versions

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.

### IT-API Version 2.1 Implementation Requirements

A Version 2.1 Implementation with Version 1.0 backwards-compatibility must provide the following:

- A set of macros in the **<it_api.h>** header file as shown below for mapping the names of deprecated structure fields
- Additional enumeration entries in *it_mem_priv_t* and *it_rmr_param_mask_t* as identified below
- A set of macros in the **<it_api.h>** header file as shown below for mapping the names of modified functions

For each modified function *it_fun*, implementations of the Version 1.0 and the Version 2.1 functions are explicitly called *it_fun*10 and *it_fun*21, 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_bind*21 shall not be provided.

Version 2.0 backwards-compatibility is also provided similarly.

### Application Requirements

The compile-time flag ITAPI_ENABLE_V21_BINDINGS (see **<it_api.h>**) controls the backwards-compatibility mode.

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_fun*10 provided by the Implementation.

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_fun*20 provided by the Implementation.

In any case, it is not recommended to use explicit Version 1.0 function names directly, as the functionality may be incomplete for certain transports and explicit Version 1.0 function names may be removed in a future IT-API version. Similarly, it is not recommended to use explicit Version 2.1 function names directly, as these may be renamed in a future IT-API version by dropping the suffix “21”.

A Consumer wishing to take advantage of Version 2.0 functionality, particularly in the area of memory management, must define the flag ITAPI_ENABLE_V20_BINDINGS before including **<it_api.h>**. The flag ITAPI_ENABLE_V20_BINDINGS should be used mutually exclusively with the flag ITAPI_ENABLE_V21_BINDINGS.

#### Backwards-Compatibility Support in Header File

```
/* default for IT-API is to maintain 1.0 bindings */
#define ITAPI_ENABLE_V10_BINDINGS
#if (defined(ITAPI_ENABLE_V20_BINDINGS) ||
     defined(ITAPI_ENABLE_V21_BINDINGS))
#undef ITAPI_ENABLE_V10_BINDINGS
#endif

/*
 *  Macros restoring some IT-API 1.0 field names:
 */

/* define IT-API 1.0 variable name mappings for
   it_rc_only_attributes_t */
#define rdma_read_inflight_incoming rdma_read_ird
#define rdma_read_inflight_outgoing rdma_read_ord

/* define IT-API 1.0 variable name mappings for it_ia_info_t */
#define ird_support ird_ord_ia_support
#define ord_support ird_ord_ia_support

/* define IT-API 1.0 name mappings for it_make_rdma_addr */
#define it_make_rdma_addr it_make_rdma_addr_absolute

/* define IT-API 1.0 name mappings for it_lmr_sync_* calls */
#define it_lmr_sync_rdma_read it_lmr_flush_to_mem
#define it_lmr_sync_rdma_write it_lmr_refresh_from_mem

/* define IT-API 1.0 name mapping for it_rmr_unbind call */
#define it_rmr_unbind  it_rmr_unlink

/* Need to use “bound” rather than “linked” in IT-API 1.0 */
#ifdef ITAPI_ENABLE_V10_BINDINGS

typedef struct {
    it_ia_handle_t    ia;          /* IT_RMR_PARAM_IA */
    it_pz_handle_t    pz;          /* IT_RMR_PARAM_PZ */
    it_boolean_t      bound;       /* 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 */
```

```
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
```

# D Functional Changes (IT-API Version 2.1 Relative to IT-API Version 2.0)

This Appendix contains a list of all *functional changes* for every reference page in IT-API Version 2.0 and for every new reference page in IT-API Version 2.1, including:

- Changes in call signatures
- New functionality in previously existing calls
- New calls
- Changes in data structures
- New data structures
- Changes and clarifications in the programming model such as:
  - — Additional or modified restrictions and rules
  - — Clarified restrictions and rules that were previously ambiguous
- Errata from Version 2.0 that have been resolved

### Connection Management for iWARP (Section 5.3)

Erroneous text indicating use of MPA Reject for Conversion has been removed resolving Erratum “Bug 180”.

### it_dto_events

A new field, representing a Memory Region handle that has been invalidated, has been added to both the DTO completion event structures.

### it_dto_flags

New flags have been added:

- IT_UNLINK_FENCE_FLAG
- IT_UNLINK_LOCAL_SINK
- 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_recv**

12616 New text describing the hazard of overflowing an EVD has been added to Application Usage,
12617 resolving Erratum “Bug 168”.

12618 **it_post_recv_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”.

# E Functional Changes (IT-API Version 2.0 Relative to IT-API Version 1.0)

This Appendix contains a list of all *functional changes* for every reference page in IT-API Version 1.0 and for every new reference page in IT-API Version 2.0, including:

- Changes in call signatures
- New functionality in previously existing calls
- New calls
- Changes in data structures
- New data structures
- Changes and clarifications in the programming model such as:
  - — Additional or modified restrictions and rules
  - — Clarified restrictions and rules that were previously ambiguous
- Errata from Version 1.0 that have been resolved

### it_addr_mode_t

This is a new data type added for IT-API Version 2.0.

### it_address_handle_create

Flags now identified as bitwise rather than logical, resolving Erratum GN63=IBM1.

### it_address_handle_free

Removed bad advice regarding free handles, resolving Erratum GN50=HP10.

### it_address_handle_modify

Flags now identified as bitwise rather than logically ORed value, resolving Erratum GN63=IBM1.

Replaced use of IT_ADDR_PATH with IT_ADDR_PARAM_PATH, resolving Erratum GN32=OG12.

Replaced use of IT_ERR_INVALID_SLID and IT_ERR_INVALID_SGID with IT_ERR_INVALID_SOURCE_PATH, resolving Erratum GN13=SUN1. Also see *it_status_t* change listing for related change.

**it_address_handle_query**

Flags now identified as bitwise rather than logically ORed value, resolving Erratum GN63=IBM1.

Added additional cross-reference text about relevant fields in *it_path_t*, resolving Erratum GN14=SUN2.

Resolved GN21=SUN9 as defined in Global Behavior section *Output Parameters*.

**it_affiliated_event_t**

Added Affiliated Asynchronous Events for a finer discrimination of locally detected errors.

Added Affiliated Asynchronous Events for a finer discrimination of remotely detected errors, supporting interpretation of iWARP Terminate messages.

Added Affiliated Asynchronous Events supporting the use of S-RQs.

Provided mappings between iWARP Affiliated Asynchronous Events and IT-API Affiliated Asynchronous Events.

**it_cm_msg_events**

Added new reject reason code for iWARP (IT_CN_REJ_BAD_CONN_PARMS).

Added text describing manifestation of capability to disable IRD/ORD.

Replaced broken cross-references, resolving Erratum GN60=HP17.

Added text clarifying that Events apply both to the TII (two-way and three-way) as well as the TDI.

Revised text describing non-peer reject Events to reflect that they can apply both to remotely detected and locally detected events, and also that a possible cause of a non-peer reject Event is a non-interoperable RNIC combination.

Removed text specifying that IRD/ORD values in an accept arrival Event are valid only on the active side in three-way Connection establishment – they are now valid also on the passive side.

Added text specifying that IRD/ORD values are valid in the connection established Event and may differ from the actual IRD/ORD in use on the connection under certain circumstances.

Added table specifying the mapping of reject reason codes on the iWARP Transport.

Added text defining the lack of ordering relationship between IT_CM_MSG_EVENT_STREAM Events and IT_DTO_EVENT_STREAM Events, resolving Erratum GN62=HP19.

**it_cm_req_events**

Fixed typo, resolving Erratum GN38=OG18.

**it_convert_net_addr**

Added text to the IT_ERR_INVALID_CONVERSION error description stating this can also be returned when the Implementation cannot convert the particular source address that was input, resolving Erratum GN22=SUN10.

Clarified the text of IT_ERR_INVALID_NETADDR to mean that the type of the Network Address specified in *source_addr* was not recognized (as opposed to IT_ERR_INVALID_ADDRESS which means the address was a valid type but an invalid value), resolving Erratum GN23=SUN11.

Text added to reflect that this call may block.

**it_conn_qual_t**

Removed assigned enumerated values to *it_conn_qual_type_t*. IT-API Version 1.0 specified these as bit values but they are all mutually-exclusive.

Added a new Connection Qualifier type for iWARP that allows specification of local and remote IANA ports.

**it_dto_events**

Modified the definition for the *ep* member of *it_dto_cmpl_event_t* so that for Receive completions it refers to the Endpoint on which the Receive completed, rather than to the Endpoint on which the Receive was posted. The only time this is different from the IT-API Version 1.0 behavior is when the Receive operation is posted to an S-RQ rather than to an Endpoint.

Typos fixed resolving Erratum GN37=OG17 and GN40=OG20.

Added text defining the lack of ordering relationship between IT_CM_MSG_EVENT_STREAM Events and IT_DTO_EVENT_STREAM Events, resolving Erratum GN62=HP19.

**it_dto_flags_t**

Moved a sentence in the IT_COMPLETION_FLAG description to instead be in the IT_NOTIFY_FLAG description. This was not a functional change; it was done only to enhance readability. This resolves Erratum GN20=SUN8.

Fixed typo. IT_SOLICTED_WAIT_FLAG was changed to IT_SOLICITED_WAIT_FLAG. This resolves Erratum GN36=OG16.

**it_dto_status_t**

Added a new DTO status for memory management operation errors.

Added additional causes for most of the existing DTO status values. A DTO status other than IT_DTO_SUCCSS represents a category of Completion Errors.

Provided mappings between iWARP Completion Errors and IT-API DTO status values.

**it_ep_accept**

Added text describing use in two-way and three-way Connection Establishment, resolving Erratum GN6 = HP5.

Removed text erroneously indicating that Private Data delivery is always unreliable, clarifying a corner case.

Asynchronous Errors:

- Specified how lack of interoperability between RDMAC RNIC and Non-permissive IETF RNIC is manifested.

**it_ep_attributes_t**

Added the following RC attributes for Shared Receive Queue (S-RQ) support:

- *srq*
- *soft_hi_watermark*
- *hard_hi_watermark*

Modified description of *max_recv_dtos* and *max_recv_segments* attributes for S-RQ support.

Modified description of *rdma_read_ord* to reflect new capability of IT-API Version 2.0 to allow Consumer to change ORD after Endpoint is connected.

**it_ep_connect**

Replaced erroneous description implying that Private Data is delivered in an IT_CM_MSG_CONN_PEER_REJECT_EVENT and corrected stating that it appears in the IT_CM_REQ_CONN_REQUEST_EVENT, resolving Erratum GN7=HP6.

Removed text erroneously indicating that Private Data delivery is unreliable, resolving Erratum GN8=HP7.

Added text explaining that the connected event occurs both on Active and Passive sides, resolving Erratum GN9=HP8.

Added dummy entries to attributes structures to aid in ANSI compilation.

Added IT_CONNECT_SUPPRESS_IRD_ORD flags to *it_cn_est_flags_t*. This flag allows the Consumer to suppress use of IRD/ORD negotiation by the Implementation.

Flags now identified as bitwise rather than logically ORed value, resolving Erratum GN63=IBM1.

Clarified the mutual exclusivity of two-way *versus* three-way connection flags.

Asynchronous Errors:

- Specified how lack of interoperability between RDMAC RNIC and Non-permissive IETF RNIC is manifested.

#### it_ep_disconnect

Added text clarifying the relationship between Private Data delivery and generation of disconnect events.

Fixed bad grammar, resolving Erratum GN61=HP18.

#### it_ep_free

Clarified that when *it_ep_free* returns, the handle can no longer be used; the previous statement was that an error was guaranteed to be returned if an attempt was made to use a handle that had been freed, which is not necessarily true. This resolves Erratum GN52=HP12.

Reorganized and expanded the paragraph in the Application Usage section that talks about marker operations to make it clearer how they are used. This resolves Erratum GN54=HP14.

Clarified that the Events that can be lost from the EVDs associated with the Endpoint are only those Events that pertain to the Endpoint. This resolves Erratum GN55=HP15.

#### it_ep_modify

Added the Endpoint Hard High Watermark and the Endpoint Soft High Watermark to the set of Endpoint parameters that can be modified. These new parameters are only used if the Endpoint is associated with a Shared Receive Queue.

Made it illegal to use this routine to modify the total number of entries in the Receive Queue if the Endpoint is associated with a Shared Receive Queue.

#### it_ep_query

Reworded description for the *spigot_id* field to help clarify that this field is not valid if the Endpoint is in the IT_EP_STATE_UNCONNECTED state. This resolves Erratum GN25=OG5.

Reworded description for the *dst_path* field to help clarify that this field is not valid if the Endpoint is in the IT_EP_STATE_UNCONNECTED state, and that it is not valid if this Endpoint is of the UD Service Type. This resolves Erratum GN26=OG6.

Resolved GN21=SUN9 as defined in Global Behavior section *Output Parameters*.

**it_ep_rc_create**

Added the IT_EP_SRQ bit to the set of flags passed to this routine to allow the Consumer to specify that they want to associate a Shared Receive Queue with the Endpoint that they are creating.

**it_ep_reset**

No functional changes.

**it_ep_state_t**

Added the IT_EP_STATE_PASSIVE_WAIT_RDMA_TRANS_REQ state. This new state is used by the TDI (*it_socket_convert*).

Removed text that erroneously implies Private Data delivery is always unreliable.

**it_ep_ud_create**

Fixed typo in description text. IT_EP_STATE_OPERATIONAL was changed to IT_EP_STATE_UD_OPERATIONAL. This resolves Erratum GN33=OG13.

Fixed typo resolving Erratum GN40=OG20.

**it_evd_create**

Flags now identified as bitwise rather than logical, resolving Erratum GN63=IBM1.

IT_SOLICITED_WAIT changed to IT_SOLICITED_WAIT_FLAG, resolving Erratum GN35=OG15.

Invalid threshold value defined (must be strictly less than or equal to the actual Implementation-allocated queue size), resolving Erratum GN27=OG7.

Revised text describing unblocking, resolving Erratum GN57=SUN16.

Numerous grammatical fixes applied to text.

**it_evd_dequeue**

Typo fixed resolving Erratum GN28=OG8.

**it_evd_free**

Removed bad advice regarding freed handles, resolving Erratum GN53=HP13.

**it_evd_modify**

Clarified text regarding conditions under which the AEVD associated with an SEVD may be modifed.

Flags now identified as bitwise rather than logical, resolving Erratum GN63=IBM1.

IT_EVD_CREAT_FD changed to IT_EVD_CREATE_FD, resolving Erratum GN34=OG14.

**it_evd_post_se**

No functional changes.

**it_evd_wait**

Corrected grammar, resolving Erratum GN29=OG9.

Text added to reflect that this call may block.

**it_event_t**

The Event Stream for RMR Link completions is now called IT_RMR_LINK_CMPL_EVENT. The old name IT_RMR_BIND_CMPL_EVENT remains valid for source backwards-compatibility.

Added Event Streams for the new Affiliated Asynchronous Events defined in *it_affiliated_event_t.*

Removed (and marked deprecated) IT_ASYNC_UNAFF_IA_CATASTROPHIC_ERROR resolving Erratum GN4=HP4.

**it_get_consumer_context**

Added text documenting that an error is returned if the Consumer Context was never set, resolving Erratum GN15=SUN3.

**it_get_pathinfo**

Added IT_ERR_INTERRUPT as a return value, resolving Erratum GN1=HP1.

Clarified the text of IT_ERR_INVALID_NETADDR to mean the type of *it_net_addr_t* passed in is not recognized.

Added IT_ERR_INVALID_ADDRESS to mean the address was of a legal type but was invalid.

Text added to reflect that this call may block.

**it_handle_t**

No functional changes.

**it_handoff**

Typos fixed resolving Erratum GN10=OG2 and GN11=OG3.

Clarifying text added resolving Erratum GN56=HP16.

**it_ia_create**

Documented that the second release of the IT-API shall have a major version number of 2 and a minor version number of 0.

Typo fixed resolving Erratum GN5=OG1.

**it_ia_free**

No functional changes.

**it_ia_info_t**

Added iWARP Transport type to *it_transport_type_t*.

Added an iWARP vendor data structure definition.

Added booleans indicating support or not for features as follows:

- S-RQ
- Hard high watermark (S-RQ)
- Soft high watermark (S-RQ)
- Resizable S-RQ
- Extended iWARP state machine
- Support for socket conversion (TDI)
- IRD modifiable
- ORD increasable
- DTO Overflow detection
- RDMA Read local extensions

**it_listen_create**

Flags now identified as bitwise rather than logically ORed value, resolving Erratum GN63=IBM1.

Added functional capability to suppress use of IRD/ORD.

**it_listen_free**

Added text constraining use of *listen_handle* after call returns, resolving Erratum GN51 (HP11).

**it_lmr_create**

Added input modifier to select addressing mode (absolute addressing or relative addressing).

*privs* and *flags* now identified as bitwise rather than logically ORed values, resolving Erratum GN63=IBM1.

*it_mem_priv_t* is now specified on a separate reference page.

Use of 0 for *privs* is no longer permitted.

Clarified restrictions on and use of access privileges, resolving Erratum GN49=SUN15.

Deprecated the use of IT_PRIV_DEFAULT (value remains in header file only), resolving Erratum GN3=HP3.

Clarified the semantics of IT_LMR_FLAG_SHARED, resolving Erratum GN16=SUN4.

Clarified potential issue with stale RMR Contexts, resolving Erratum GN46=SUN12.

New return value IT_ERR_INVALID_ADDR_MODE.

**it_lmr_free**

Specified that any RMR Context associated with the LMR may no longer be used after freeing an LMR, resolving Erratum GN46=SUN12.

Added missing statement that a failing call due to linked RMR(s) will neither affect an associated RMR Context, nor any linked RMR, resolving Erratum GN47=SUN13.

While not a functional change, the discussion on stale RMR Contexts was moved to the Application Usage section, resolving Erratum GN18=SUN6.

**it_lmr_modify**

Added input modifier to select addressing mode (absolute addressing or relative addressing).

*mask* now identified as bitwise rather than logically ORed value, resolving Erratum GN63=IBM1.

Clarified potential issue with stale RMR Contexts, resolving Erratum GN46=SUN12.

Clarified in which cases an LMR remains unaffected when the operation fails and that the operation fails if the LMR has any RMRs linked to it, resolving Erratum GN48=SUN14.

**it_lmr_query**

*mask* now identified as bitwise rather than logically ORed value, resolving Erratum GN63=IBM1.

Added addressing mode (*addr_mode*) LMR attribute.

Specified in which cases the LMR must have byte-level granularity. In particular, Implementations supporting Relative Addressing must provide byte-level granularity.

Resolved GN21=SUN9 as defined in Global Behavior section *Output Parameters*.

**it_lmr_flush_to_mem**

Added statement that when both normal and non-coherent LMRs are input to this routine that only the non-coherent segments are affected, resolving Erratum GN17=SUN5.

**it_lmr_refresh_from_mem**

Added statement that when both normal and non-coherent LMRs are input to this routine that only the non-coherent segments are affected, resolving Erratum GN17=SUN5.

**it_lmr_triplet_t**

The *addr* member of LMR Triplets must now be interpreted depending on the addressing mode (Absolute Addressing or Relative Addressing) of the LMR.

Additionally, the *addr* member has been changed from a simple void pointer to a union of absolute (void pointer) and relative (*it_length_t*) elements.

**it_make_rdma_addr**

The IT-API 1.0 routine *it_make_rdma_addr* has been replaced with two new routines, *it_make_rdma_addr_absolute* and *it_make_rdma_addr_relative*. A backward-compatibility macro is recommended to define *it_make_rdma_addr* as *it_make_rdma_addr_absolute*.

**it_mem_priv_t**

This is a new reference page for a data type that was specified under *it_lmr_create* in IT-API Version 1.0.

*privs* now identified as bitwise rather than logically ORed value, resolving Erratum GN63=IBM1.

Use of 0 for *privs* is no longer permitted.

Clarified restrictions on and use of access privileges, resolving Erratum GN49=SUN15.

Deprecated the use of IT_PRIV_DEFAULT (value remains in header file only), resolving Erratum GN3=HP3.

**it_path_t**

Removed erroneous assertion that *it_ib_net_endpoint* is the remote component of an InfiniBand path.

Added Path data structures for the iWARP Transport.

**it_post_rdma_read**

The starting address of the remote source buffer, *rdma_addr*, must be chosen according to the addressing mode of the remote buffer.

Specified required access privileges for remote source buffer.

Specified required access privileges for local sink buffer, which are transport-dependent.

Clarified that Consumer retains ownership of the array specified by *local_segments* and *num_segments*, resolving Erratum GN31=OG11.

Added ordering rules for subsequent DTOs targeting overlapping sections of local sink buffers.

Added ordering rule for Work Requests posted after RMR Link/Unlink, resolving Erratum GN58=SUN17.

Asynchronous Errors:

- Described scenarios causing an access violation or transport error at the remote/local Endpoint.
- Specified that zero-sized data transfers never cause an access violation.
- Specified how remotely detected errors will manifest remotely and locally.
- Specified how locally detected errors will manifest.

**it_post_rdma_write**

The starting address of the remote sink buffer, *rdma_addr*, must be chosen according to the addressing mode of the remote buffer.

Specified required access privileges for source and sink buffer.

Clarified that the Consumer retains ownership of the array specified by *local_segments* and *num_segments*, resolving Erratum GN31=OG11.

Clarified the lack of end-to-end completions.

Clarified the difference between operation ordering and local/remote completion ordering.

Added ordering rules for subsequent DTOs targeting overlapping sections of remote sink buffers.

Added ordering rule for Work Requests posted after RMR Link/Unlink, resolving Erratum GN58=SUN17.

Asynchronous Errors:

- Described scenarios causing an access violation or transport error at the remote Endpoint.
- Specified that zero-sized data transfers never cause an access violation.
- Change in how remotely detected errors will manifest remotely and locally (transport-dependent).

**it_post_recv**

Receive DTOs can now be posted to an Endpoint or Shared Receive Queue.

Specified required access privileges for the sink buffer.

Clarified that the Consumer retains ownership of the array specified by *local_segments* and *num_segments*, resolving Erratum GN31=OG11.

Added ordering rules for subsequent incoming Send DTOs matching with Receive DTOs whose Receive buffers overlap.

Asynchronous Errors:

- Described scenarios causing an access violation, length error, or transport error at the local Endpoint.
- Change in how locally detected errors will manifest locally and remotely (transport-dependent).

Clarified that a Receive DTO can be posted to an EP prior to reaching the IT_EP_STATE_CONNECTED state.

**it_post_recvfrom**

Specified required access privileges for the sink buffer.

Clarified that the Consumer retains ownership of the array specified by *local_segments* and *num_segments*, resolving Erratum GN31=OG11.

Asynchronous Errors:

- Described scenarios causing an access violation or length error at the local Endpoint.

**it_post_send**

Specified required access privileges for the source buffer.

Clarified that the Consumer retains ownership of the array specified by *local_segments* and *num_segments*, resolving Erratum GN31=OG11.

Clarified the lack of end-to-end completions.

Clarified the difference between operation ordering and local/remote completion ordering.

Added ordering rules for subsequent Send DTOs matching with Receive DTOs whose Receive buffers overlap.

Added ordering rule for Work Requests posted after RMR Link/Unlink, resolving Erratum GN58=SUN17.

Asynchronous Errors:

- Described scenarios causing an access violation, length error, or transport error at the remote Endpoint.
- Change in how remotely detected errors will manifest remotely and locally (transport-dependent).

**it_post_sendto**

Specified required access privileges for the source buffer.

Clarified that the Consumer retains ownership of the array specified by *local_segments* and *num_segments*, resolving Erratum GN31=OG11.

Asynchronous Errors:

- Described scenarios causing an access violation or length error at the remote Endpoint.

**it_pz_query**

Flags now identified as bitwise rather than logical, resolving Erratum GN63=IBM1.

Resolved GN21=SUN9 as defined in Global Behavior section *Output Parameters*.

**it_reject**

Removed text that erroneously implies Private Data delivery is always unreliable and replaced it with a clarification, resolving Erratum GN12=HP9.

**it_rmr_bind**

This routine was replaced by *it_rmr_link*. 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 13026 *privs* and *dto_flags* now identified as bitwise rather than logically ORed values, resolving 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 13030 Added ordering rule for Work Requests posted after RMR Link, resolving Erratum GN58=SUN17.

13031 New return value IT_ERR_INVALID_ADDR_MODE.

13032 Detailed description of Asynchronous Errors.

13033 13034 Deprecated the use of 0 for *privs* and recommended using *it_rmr_unlink* instead, resolving Erratum GN19=SUN7.

13035 13036 Erratum GN2=HP2 (reference to a inexistant IT_DTO_DEFAULT_FLAG) is resolved – in fact it was already resolved in Version 1.0.

13037 13038 Clarified ways to infer completion of an RMR Link operation, resolving Erratum GN59=SUN18.

13039 Clarified potential issue with stale RMR Contexts.

13040 **it_rmr_query**

13041 13042 *mask* now identified as bitwise rather than logically ORed value, resolving Erratum GN63=IBM1.

13043 Added RMR type (*type*) RMR attribute.

13044 Added addressing mode (*addr_mode*) RMR attribute.

13045 13046 Replaced the RMR attribute *bound* by *linked*. If ITAPI_ENABLE_V20_BINDINGS is not 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 | *it_get_pathinfo*: should return IT_ERR_ INTERRUPT | *it_get_pathinfo* | *it_get_pathinfo* 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 | *it_rmr_bind*: IT_DTO_ DEFAULT_FLAG needs to be removed | *it_rmr_bind* | Page 2 line 54 of *it_rmr_bind* 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 | *it_lmr_create*: confusing use of IT_PRIV_ DEFAULT | *it_lmr_create* | On the *it_lmr_create* 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 **<it_api.h>** header file. Either the header file should be fixed, or else we should fix the documentation on this reference page. | 3 |
| 4 | HP4 | *it_event_t*: IT_ASYNC_ UNAFF_IA_ CATASTROPHIC _ERROR needs to be removed | *it_event_t* | In the *it_event_t* 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 | *it_ia_create*: delete an extra "also" | *it_ia_create* | 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 | *it_ep_accept*: description of when this is called could lead to confusion | *it_ep_accept* | Lines 39-44 of the *it_ep_accept* reference page currently state: "Calling *it_ep_accept* 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 *it_ep_accept* 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 | *it_ep_connect*: IT_CM_MSG_ CONN_PEER_ REJECT_EVENT should be IT_CM_REQ_ CONN_ REQUEST_ EVENT | *it_ep_connect* | The description for the *private_data* field for *it_ep_connect* reads as follows: "*private_data*: Opaque Private Data to be sent in the IT_CM_MSG_CONN_ PEER_REJECT_EVENT Event delivered to the Remote Consumer. If the IA does not support Private Data, *private_data_length* must be zero. 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 | *it_ep_connect*: private data delivery IS reliable | *it_ep_connect* | The description for the *private_data* parameter to *it_ep_connect* states: "*private_data*: Opaque Private Data to be sent in the IT_CM_MSG_CONN_ PEER_REJECT_EVENT Event delivered to the Remote Consumer. If the IA does not support Private Data, *private_data_length* must be zero. 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 enequeued without its associated Private Data. The last sentece above should be removed. | 5 |
| 9 | HP8 | *it_ep_connect*: connection established event is generated on both sides, not just the remote side | *it_ep_connect* | Lines 129-134 of the *it_ep_connect* 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 Comsumer accepts the connection and the Endpoint transitions into the (IT_EP_STATE_CONNECTED) connected state. See the **<it_ep_state_t.doc>** reference page for a complete description on the Endpoint state diagram for both the three-way and two-way connection estalishment." 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 | *it_handoff*: typo ("Cn_est_id") | *it_handoff* | Typo: In the Description section, the argument *Cn_est_id* has the first letter in uppercase; it should be lowercase. | 1 |
| 11 | OG3 | *it_handoff*: Typo ("conn_qual)was") | *it_handoff* | 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 | *it_reject*: private data delivery isn't as unreliable as this makes it sound | *it_reject* | There is a note in the Application Usage section of the *it_reject* 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 *it_reject* will be a part of it. That's not the case. The unreliability here is that just because you called *it_reject* 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 *it_reject*. This needs to be clarified. | 3 |
| 13 | SUN1 | *it_address_handle_modify*: obsolete error codes need to be removed | *it_address_handle_modify* | 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 *it_address_handle_create*(). These error codes should be removed from *it_ah_modify.pdf*, as well as *it_status_t.pdf* and from wherever else they appear. | 2 |
| 14 | SUN2 | *it_address_handle_query*: what should be returned for an incomplete path | *it_address_handle_query* | 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 *it_address_handle_create.pdf* be returned for all calls; but a “lazy” implementation might only return *ib.sl* and *ib.remote_port_lid* 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 *it_address_handle_query.pdf* to be more explicit? | 3 |

| Global No. | Company No. | Title | Filename | Problem Description | Suggested Severity |
|---|---|---|---|---|---|
| 15 | SUN3 | *it_get_consumer_ context*: murky description of what happens when no context has been set | *it_get_consumer_ context* | The testable assertion folks make a good point in the specification issues: The error code IT_ERR_NO_CONTEXT has been added to *it_get_consumer_context*() for the situation where “the handle does not have an associated context”. 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:<br>1. Handle has a context, context is non zero: return the context and IT_SUCCESS.<br>2. Handle has a context, context is zero: return the (zero) context and IT_SUCCESS.<br>3a. Handle has no context at all: return IT_ERR_NO_CONTEXT, set context to 0 (current method).<br>3b. Handle has no context at all: return IT_ERR_NO_CONTEXT, don't set context at all (suggested fix).<br>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. | 3 |
| 16 | SUN4 | *it_lmr_create*: confusing description of IT_LMR_FLAG_ SHARED behavior | *it_lmr_create* | 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 ...” | 3 |

| Global No. | Company No. | Title | Filename | Problem Description | Suggested Severity |
|---|---|---|---|---|---|
| 17 | SUN5 | *it_lmr_sync_ rdma_**: behavior when both coherent and incoherent LMRs are passed is unspecified | *it_lmr_sync_rdma_ read* *it_lmr_sync_rdma_ write* | 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 *rdma* reads, while the coherent LMRs should be silently ignored. The lack of clarity here caused some confusion in the testable assertions. | 4 |
| 18 | SUN6 | *it_rmr_free*: description needs to move to Application Usage section | *it_rmr_free* | 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 | *ir_rmr_bind*: privs of 0 should not be allowed | *it_rmr_bind* | 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 *rmr_unbind*() instead. | 5 |
| 20 | SUN8 | *it_dto_flags_t*: sentence should be moved | *it_dto_flags_t* | 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 | *it_*_query*: behavior not defined if params is NULL | *it_lmr_query* other query routines | The behaviour of *it_lmr_query*() 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 | *it_convert_net_addr*: need new error code | *it_convert_net_addr* | 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 *it_convert_net_addr*() 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 IT_ERR_INVALID_CONVERSION. 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, IT_ERR_NO_CONVERSION, to cover this case? | 3 |
| 23 | SUN11 | *it_convert_net_addr*: want to rename an error code | *it_convert_net_addr* | There are two errors here that have very similar names. IT_ERR_INVALID_ADDRESS and IT_ERR_INVALID_NETADDR. Is it possible to rename the latter to a more specific error code such as IT_ERR_INVALID_ADDR_TYPE? (Perhaps it is too late to do this.) | 2 |
| 24 | OG4 | it_ud_service_request_handle_create: typo | *it_ud_service_request_handle_create* | On Line 41 in the *it_ud_service_request_handle_query* reference page, the comment against the *private_data_length* structure member is IT_UD_PARAM_PRIV_DATA_LEN, whereas the actual constant name is IT_UD_PARAM_PRIV_DATA_LENGTH. | 1 |
| 25 | OG5 | *it_ep_query*: what spigot do you get back in unconnected state? | *it_ep_query* | What value does *it_ep_query(ep_handle, mask, params)* return when IT_EP_PARAM_SPIGOT is set in *mask* and *ep_handle* specifies an RC endpoint in the IT_EP_STATE_UNCONNECTED state? | 2 |
| 26 | OG6 | *it_ep_query*: what path do you get back in unconnected state? | *it_ep_query* | What value does *it_ep_query*(*ep_handle, mask, params*) return when IT_EP_PARAM_PATH is set in *mask* and *ep_handle* specifies an RC endpoint in the IT_EP_STATE_UNCONNECTED state, or specifies a UD endpoint? | 2 |

| Global No. | Company No. | Title | Filename | Problem Description | Suggested Severity |
|---|---|---|---|---|---|
| 27 | OG7 | *it_evd_create*: what value of *sevd_threshold* is invalid? | *it_evd_create* | The specification does not say which values of *sevd_threshold* are invalid and would therefore cause an IT_ERR_INVALID_THRESHOLD error. | 2 |
| 28 | OG8 | *it_evd_dequeue*: typo | *it_evd_dequeue* | The text in the 5th paragragh: "...returns an notification event..." should read: "...returns a notification event...". | 1 |
| 29 | OG9 | *it_evd_wait*: typo | *it_evd*_wait | The sentence: "If *sevd_threshold* value of *evd_handle* is 1, then one or more simultaneous waiters can supported for the SEVD." should read: "If *sevd_threshold* value of *evd_handle* is 1, then one or more simultaneous waiters can be supported for the SEVD." | 1 |
| 30 | OG10 | *it_events*: typo | *it_events* | In the *it_unaffiliated_event_t* reference page, in the table at the end of the 'DESCRIPTION' section, it refers to the fields *spigot_online_event_support* and *spigot_offline_event_support*. These should be *spigot_online_support* and *spigot_offline_support* respectively. See the *it_ia_info_t* reference page. | 1 |
| 31 | OG11 | *it_post_**: segment ownership language is confusing | *it_post_rdma_read*<br>*it_post_rdma_write*<br>*it_post_send*<br>*it_post_sendto*<br>*it_post_recv*<br>*it_post_recvfrom* | The Description says: "A Consumer does get back the ownership of the *num_segments* and *local_segments* arguments (but not the local buffer identified by them) when *it_post_rdma_read* returns and is free to use the *num_segments* and *local_segments* 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 *num_segments* and *local_segments* arguments. Therefore it is not meaningful to say that it gets back the ownership of these arguments. | 2 |
| 32 | OG12 | *it_address_handle_modify*: ypo | *it_address_handle_modify* | "IT_ADDR_PATH" should be "IT_ADDR_PARAM_PATH". | 1 |
| 33 | OG13 | *it_ep_ud_create*: typo | *it_ep_ud_create* | "IT_EP_STATE_OPERATIONAL" should be IT_EP_STATE_UD_OPERATIONAL". | 1 |
| 34 | OG14 | *it_evd_modify*: typo | *it_evd_modify* | "IT_EVD_CREAT_FD" should be "IT_EVD_CREATE_FD". | 1 |
| 35 | OG15 | *it_evd_create*: typo | *it_evd_create* | "IT_SOLICITED_WAIT" should be "IT_SOLICITED_WAIT_FLAG". | 1 |

| Global No. | Company No. | Title | Filename | Problem Description | Suggested Severity |
|---|---|---|---|---|---|
| 36 | OG16 | *it_dto_flags_t*: typo | *it_dto_flags_t* | "IT_SOLICTED_WAIT_FLAG" should be "IT_SOLICITED_WAIT_FLAG". | 1 |
| 37 | OG17 | *it_dto_events*: typo | *it_dto_events* | "IB_UD_IB_GRH_PRESENT" should be "IT_UD_IB_GRH_PRESENT". | 1 |
| 38 | OG18 | *it_cm_req_events*: typo | *it_cm_req_events* | "*it_conn_req_event_t*" should be "*it_conn_request_event_t*". | 1 |
| 39 | OG19 | *it_rmr_unbind*: typo | *it_rmr_unbind* | "*it_dto_compl_event_t*" should be "*it_dto_cmpl_event_t*". | 1 |
| 40 | OG20 | *it_dto_events*: typo | *it_dto_events* *it_ep_ud_create* | "*it_ep_states_t*" should be "*it_ep_state_t*". | 1 |
| 41 | OG21 | *it_ud_service_ request_handle_ create*: typo | *it_ud_service_ request_handle_ create* | "*it_ud_service_request_handle_t*" should be "*it_ud_svc_req_handle_t*". | 1 |
| 42 | OG22 | *it_ud_service_ request*: typo | *it_ud_service_ request* *it_ud_service_ request_handle_ create* | "*it_ud_svc_req_identifer_t*" should be "*it_ud_svc_req_identifier_t*". | 1 |
| 43 | OG23 | Can functions be implemented as macros? | *itapdx_implementors _guide* | 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 | **<itapdx_it_api.h>** typo | **<itapdx_it_api.h>** | In the file **<itapdx_it_api.h>**, the comment against the *private_data_length* member of *it_ud_svc_req_param_t* mentions "IT_UD_PARAM_PRIV_ DATA_LEN"; it should say "IT_UD_PARAM_PRIV_DATA_ LENGTH". | 1 |
| 45 | OG25 | **<itapdx_it_api.h>** missing prototypes | **<itapdx_it_api.h>** | The file **<itapdx_it_api.h>** is missing prototypes for the functions *it_make_rdma_addr*() and *it_hton64*(). | 5 |
| 46 | SUN12 | *it_lmr_free*: what happens to the RMR context? | *it_lmr_free* | Nowhere in *it_lmr_free.pdf* do we describe what happens to the associated RMR Context (if one exists) when an *lmr_handle* is freed. It is an error to use it afterwards, much like it's an error to use the *rmr* context of an unbound MW. We should say something to this effect in *it_lmr_free.pdf*. | 3 |

| Global No. | Company No. | Title | Filename | Problem Description | Suggested Severity |
|---|---|---|---|---|---|
| 47 | SUN13 | *it_lmr_free*: RMR context isn't affected by failure of this routine | *it_lmr_free* | 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 | *it_lmr_modify*: confusing text | *it_lmr_modify* | If you call *it_lmr_modify*() and an error occurs:<br>• IT_ERR_RESOURCES: your LMR blows up (33-36).<br>• IT_ERR_ACCESS: your LMR also blows up (33-36).<br>• IT_ERR_LMR_BUSY: your LMR is guaranteed fine (26-28).<br>• IT_ERR_INVALID_{PZ,MASK, PRIVS}: who knows???.<br>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.<br>We should patch these gaps as follows (according to the IB Spec 1.1 (Page 536):<br>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).<br>C11-20 if the CI returns any other error, the CI shall invalidate both “old” and “new” registrations, and release any associated resources.<br>IT_ERR_INVALID_PRIVS corresponds to the IB “Invalid Acess Control specifier” error. Thisis covered by C11-20.<br>IT_ERR_INVALID_PZ corresponds to the “Invalid Protection Domain” error. That is definitely a C11-20 error.<br>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. | 3 |

| Global No. | Company No. | Title | Filename | Problem Description | Suggested Severity |
|---|---|---|---|---|---|
| 49 | SUN15 | *it_lmr_create*: missing flags | *it_lmr_create* | "An RMR Context allowing remote access to the memory region will be created if the *privs* 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 | *it_address_ handle_free*: bad advice regarding freed handles | *it_address_handle_ free* | *it_address_handle_free* says two conflicting things about what happens to a Consumer who uses a freed handle. The current spec text is: "...Once *it_address_handle_free* returns, *addr_handle* 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 | *it_listen_free*: nothing said about freed handles | *it_listen_free* | 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 | *it_ep_free*: bad advice regarding freed handles | *it_ep_free* | *it_ep_free* has an incorrect statement about what happens if the Consumer attempts to use a freed handle. Currently the spec states: "Use of the handle *ep_handle* 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 | *it_evd_free*: bad advice regarding freed handles | *it_evd_free* | *it_evd_free* has an incorrect statement about what happens if the Consumer attempts to use a freed handle. Currently the spec states: “Use of the handle *evd_handle* 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 | *it_ep_free*: disconnect sequence is too terse | *it_ep_free* | *it_ep_free.pdf* recommends doing *it_ep_disconnect* followed by posting/waiting for the marker DTO. It should instead recommend the following sequence:<br>1. First,. *it_ep_disconnect*<br>2. Wait for DISCONNECT event or CONN_BROKEN or REJECT event<br>3. Post a marker *send*/*rmr_bind* DTO if there have been *send*/*rmr_bind* DTOs with COMPLETION_FLAG off<br>4. Wait for the last posted *recv* DTO and wait for the last posted *send*/*rmr_bind* DTO or marker DTO | 2 |
| 55 | HP15 | it_ep_free: shouldn't allow arbitrary events to be lost | *it_ep_free* | Lines 21-24 of the *it_ep_free* reference page currently state: “Freeing an Endpoint potentially means Events might be lost on the *recv_sevd_handle* or *request_sevd_handle* SEVDs associated with the Endpoint. There is also potential to lose Events on the *connect_sevd_ handle* SEVD associated with the Endpoint. The Consumer should first drain these EVDs before calling *it_ep_free*.” 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 | *it_handoff*: add clarifying sentence | *it_handoff* | In the *it_handoff* reference page Description section, add a line after the sentence: "*it_handoff* 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 *conn_qual* or *spigot_id* 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 | *it_evd_create*: unblocking behavior language unclear | *it_evd_create* | Re *it_evd_create*(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 *evd_flag* 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:<br><br>"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 *evd_flag* 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 *evd_flag* 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 *it_evd_modify*). Attempting to wait on the | 3 |

| Global No. | Company No. | Title | Filename | Problem Description | Suggested Severity |
|---|---|---|---|---|---|
| | | | | SEVD when disallowed will return IT_ERR_INVALID_EVD_STATE."<br><br>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):<br><br>"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 *evd_flag* 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 *fd*. 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 *fd*, then notification will be generated for that AEVD or *fd*. In the AEVD case, as many notifications may be generated as there are events available on all SEVDs of the AEVD." | |

| Global No. | Company No. | Title | Filename | Problem Description | Suggested Severity |
|---|---|---|---|---|---|
| 58 | SUN17 | *it_rmr_bind*: ordering rules have gone missing | *it_post_send*<br>*it_rmr_bind*<br>*it_rmr_unbind* | The reference page for *it_post_send*() 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.”<br>There is no discussion of the following IBTA ordering rule:<br>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.<br>Worse, the reference pages for *it_rmr_*[*un*]*bind*() do not contain any text regarding the ordering of various operations. | 3 |

| Global No. | Company No. | Title | Filename | Problem Description | Suggested Severity |
|---|---|---|---|---|---|
| 59 | SUN18 | *it_rmr_bind*: completion suppression advice is confusing | *it_rmr_bind* | The following text in *it_rmr_bind*() 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.”<br>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. | 2 |
| 60 | HP17 | *it_cm_msg_events*: broken cross-reference | *it_cm_msg_events* | Broken cross-references appear in the *it_cm_msg_events* reference page. Search for “Error! Reference source not found.” in the PDF file. | 1 |
| 61 | HP18 | *it_ep_disconnect*: bad grammar | *it_ep_disconnect* | Replace “*cn_est_id* then the Implementation generate an IT_ERR_INVALID_ CN_EST_ID error,” with: “*cn_est_id* 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 | *it_cm_msg_events*<br>*it_dto_events* | 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 *it_lmr_create* or *it_rmr_bind_calls*, 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

# G Header Files

## G.1 it_api.h

```
/*
 * it_api.h
 *
 * Copyright © 2005 The Open Group
 * All rights reserved.
 * The copyright owner hereby grants permission for all or part of this
 * publication to be reproduced, stored in a retrieval system, or
 * transmitted, in any form or by any means, electronic, mechanical,
 * photocopying, recording, or otherwise, provided that it remains
 * unchanged and that this copyright statement is included in all
 * copies or substantial portions of the publication.
 *
 * For any software code contained within this specification,
 * permission is hereby granted, free-of-charge, to any person
 * obtaining a copy of this specification (the "Software"), to deal in
 * the Software without restriction, including without limitation the
 * rights to use, copy, modify, merge, publish, distribute, sublicense,
 * and/or sell copies of the Software, and to permit persons to whom
 * the Software is furnished to do so, subject to the above copyright
 * notice and this permission notice being included in all copies or
 * substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND
 * NON-INFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS
 * BE LIABLE FOR ANY CLAIM, DAMAGES, OR OTHER LIABILITY, WHETHER IN AN
 * ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF, OR IN
 * CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
 * SOFTWARE.
 *
 * Permission is granted for implementers to use the names, labels,
 * etc. contained within the specification. The intent of publication
 * of the specification is to encourage implementations of the
 * specification. This specification has not been verified for
 * avoidance of possible third-party proprietary rights. In
 * implementing this specification, usual procedures to ensure the
 * respect of possible third-party intellectual property rights should
 * be followed.
 */

#ifndef _ITAPI_H_
#define _ITAPI_H_
#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_TIMEWAIT,
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_ATIMEOUT,
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 nunbers, in network byte order. */
13423               it_iana_lr_port_t lr_port;
13424
13425             } conn_qual;
13426
13427           } it_conn_qual_t;
13428
13429           typedef union
13430           {
13431             void *ptr;
13432             uint64_t index;
13433           } it_context_t;
13434
13435           typedef uint64_t it_dto_cookie_t;
13436
13437           typedef enum
13438           {
13439             IT_DTO_SUCCESS = 0,
13440             IT_DTO_ERR_LOCAL_LENGTH = 1,
13441             IT_DTO_ERR_LOCAL_EP = 2,
13442             IT_DTO_ERR_LOCAL_PROTECTION = 3,
13443             IT_DTO_ERR_FLUSHED = 4,
13444             IT_RMR_OPERATION_FAILED = 5,
13445             IT_DTO_ERR_BAD_RESPONSE = 6,
13446             IT_DTO_ERR_REMOTE_ACCESS = 7,
13447             IT_DTO_ERR_REMOTE_RESPONDER = 8,
13448             IT_DTO_ERR_TRANSPORT = 9,
13449             IT_DTO_ERR_RECEIVER_NOT_READY = 10,
13450             IT_DTO_ERR_PARTIAL_PACKET = 11,
13451             IT_DTO_ERR_LOCAL_MM_OPERATION = 12
13452           } it_dto_status_t;
13453
13454           typedef enum
13455           {
13456             /* If flag set, completion generates a local event. */
13457             IT_COMPLETION_FLAG = 0x01,
13458
13459             /* If flag set, completion causes local Notification. */
13460             IT_NOTIFY_FLAG = 0x02,
13461
13462             /* If flag set, receipt of DTO at remote will cause Notification
13463                at remote. */
13464             IT_SOLICITED_WAIT_FLAG = 0x04,
```

```
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
```

```
{

  /* Spigot identifier */
  size_t spigot_id;

  /* Maximum sized Send operation for the RC service
     on this Spigot. */
  size_t max_rc_send_len;

  /* Maximum sized RDMA Read/Write operation for the RC service on
     this Spigot. */
  size_t max_rc_rdma_len;

  /* Maximum sized Send operation for the UD service
     on this Spigot. */
  size_t max_ud_send_len;

  /* Indicates whether the Spigot is online or offline.
     An IT_TRUE value means online. */
  it_boolean_t spigot_online;

  /* A mask indicating which Connection Qualifier types this
     IA supports for input to it_ep_connect and
     it_ud_service_request_handle_create. The bits in the mask are
     an inclusive OR of the values for Connection Qualifier types
     that this IA supports. */
  it_conn_qual_type_t active_side_conn_qual;

  /* A mask indicating which Connection Qualifier types this to
     it_listen_create. The bits in the mask are an inclusive OR of
     the values for Connection Qualifier types that this IA
     supports. */
  it_conn_qual_type_t passive_side_conn_qual;

  /* The number of Network Addresses associated with Spigot. */
  size_t num_net_addr;

  /* Pointer to array of Network Address addresses. */
  it_net_addr_t *net_addr;

} it_spigot_info_t;

typedef struct
{

  /* The NodeInfo:VendorID as described in chapter 14 of the
     IB spec. */
  uint32_t vendor:24;

  /* The NodeInfo:DeviceID as described in chapter 14 of the
     IB spec. */
  uint16_t device;

  /* The NodeInfo:Revision as described in chapter 14 of the
```

```
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
13697             /* Interface Adapter name, as specified in it_ia_create. */
13698             char *ia_name;
13699
13700             /* The major version number of the latest version of the
13701                IT-API that this IA supports. */
13702             uint32_t api_major_version;
13703
13704             /* The minor version number of the latest version of the
13705                IT-API that this IA supports. */
13706             uint32_t api_minor_version;
13707
13708             /* The major version number for the software being used to control
13709                this IA. The IT-API imposes no structure whatsoever on this
13710                number; its meaning is completely IA-dependent. */
13711             uint32_t sw_major_version;
13712
13713             /* The minor version number for the software being used to control
13714                this IA. The IT-API imposes no structure whatsoever on this
13715                number; its meaning is completely IA-dependent. */
13716             uint32_t sw_minor_version;
13717
13718             /* The vendor associated with the IA. This information is useful
13719                if the Consumer wishes to do device-specific programming. This
13720                union is discriminated by transport_type. No vendor
13721                identification is provided for transports not listed below. */
13722             union
13723             {
13724
13725               /* Used if transport_type is IT_IB_TRANSPORT. */
13726               it_vendor_ib_t ib;
13727
13728               /* Used if transport_type is IT_VIA_IP_TRANSPORT or
13729                  IT_VIA_FC_TRANSPORT. */
13730               it_vendor_via_t via;
13731
13732               /* Used if transport_type is IT_IWARP_TCP_TRANSPORT. */
13733               it_vendor_iwarp_tcp_t iwarp;
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_iobl_type_t iobl_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
13973             /* Partition Key, as defined in the REQ message for the IB
13974                CM protocol. */
13975             uint16_t partition_key;
13976
13977             /* Path Packet Payload MTU, as defined in the REQ message
13978                for the IB CM protocol. */
13979             uint8_t path_mtu:4;
13980
13981             /* PacketLifeTime, as defined in the PathRecord in IB
13982                specification. This field is useful for Consumers that
13983                wish to use timeout values other than the default ones
13984                for doing Connection establishment. */
13985             uint8_t packet_lifetime:6;
13986
13987             /* Local Port LID, as defined in the REQ message for the IB
13988                CM protocol. The low-order bits of this value also
13989                constitute the Source Path Bits that are used to
13990                create an Address Handle. */
13991             uint16_t local_port_lid;
13992
13993             /* Remote Port LID, as defined in the REQ message for the
13994                IB CM protocol. This is also the Destination LID used
13995                to create an Address Handle. */
13996             uint16_t remote_port_lid;
13997
13998             /* Local Port GID in network byte order, as defined in the
13999                REQ message for the IB CM protocol. This is also used to
14000                determine the appropriate Source GID Index to be used
14001                when creating an Address Handle. */
14002             it_ib_gid_t local_port_gid;
14003
14004             /* 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
```

```
{
  IT_EP_PARAM_ALL = 0x00000001,
  IT_EP_PARAM_IA = 0x00000002,
  IT_EP_PARAM_SPIGOT = 0x00000004,
  IT_EP_PARAM_STATE = 0x00000008,
  IT_EP_PARAM_SERV_TYPE = 0x00000010,
  IT_EP_PARAM_PATH = 0x00000020,
  IT_EP_PARAM_PZ = 0x00000040,
  IT_EP_PARAM_REQ_SEVD = 0x00000080,
  IT_EP_PARAM_RECV_SEVD = 0x00000100,
  IT_EP_PARAM_CONN_SEVD = 0x00000200,
  IT_EP_PARAM_RDMA_RD_ENABLE = 0x00000400,
  IT_EP_PARAM_RDMA_WR_ENABLE = 0x00000800,
  IT_EP_PARAM_MAX_RDMA_READ_SEG = 0x00001000,
  IT_EP_PARAM_MAX_RDMA_WRITE_SEG = 0x00002000,
  IT_EP_PARAM_MAX_IRD = 0x00004000,
  IT_EP_PARAM_MAX_ORD = 0x00008000,
  IT_EP_PARAM_EP_ID = 0x00010000,
  IT_EP_PARAM_EP_KEY = 0x00020000,
  IT_EP_PARAM_MAX_PAYLOAD = 0x00040000,
  IT_EP_PARAM_MAX_REQ_DTO = 0x00080000,
  IT_EP_PARAM_MAX_RECV_DTO = 0x00100000,
  IT_EP_PARAM_MAX_SEND_SEG = 0x00200000,
  IT_EP_PARAM_MAX_RECV_SEG = 0x00400000,
  IT_EP_PARAM_SRQ = 0x00800000,
  IT_EP_PARAM_SOFT_HI_WATERMARK = 0x01000000,
  IT_EP_PARAM_HARD_HI_WATERMARK = 0x02000000,
  IT_EP_PARAM_ATOMICS_ENABLE = 0x04000000,
  IT_EP_PARAM_PRIV_OPS_ENABLE = 0x08000000
} it_ep_param_mask_t;

typedef struct
{
  it_boolean_t rdma_read_enable;
  /* IT_EP_PARAM_RDMA_RD_ENABLE */
  it_boolean_t rdma_write_enable;
  /* IT_EP_PARAM_RDMA_WR_ENABLE */
  size_t max_rdma_read_segments;
  /* IT_EP_PARAM_MAX_RDMA_READ_SEG */
  size_t max_rdma_write_segments;
  /* IT_EP_PARAM_MAX_RDMA_WRITE_SEG */
  uint32_t rdma_read_ird;
  /* IT_EP_PARAM_MAX_IRD */
  uint32_t rdma_read_ord;
  /* IT_EP_PARAM_MAX_ORD */
  it_srq_handle_t srq;
  /* IT_EP_PARAM_SRQ */
  size_t soft_hi_watermark;
  /* IT_EP_PARAM_SOFT_HI_WATERMARK */
  size_t hard_hi_watermark;
  /* IT_EP_PARAM_HARD_HI_WATERMARK */
  it_boolean_t atomics_enable;
  /* IT_EP_PARAM_ATOMICS_ENABLE */
```

```
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_recv_dtos;  /* IT_EP_PARAM_MAX_RECV_DTO */
14193             size_t max_send_segments;  /* IT_EP_PARAM_MAX_SEND_SEG */
14194             size_t max_recv_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 sevd_queue_size;  /* IT_EVD_PARAM_QUEUE_SIZE */
14718             size_t sevd_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
14730             /* Most recent major version number of the IT-API supported by the
14731                Interface. */
14732             uint32_t major_version;
14733
14734             /* Most recent minor version number of the IT-API supported by the
14735                Interface. */
14736             uint32_t minor_version;
14737
14738             /* The transport that the Interface uses, as defined in
14739                it_ia_info_t. */
14740             it_transport_type_t transport_type;
14741
14742             /* The name of the Interface, suitable for input to it_ia_create.
14743                The name is a string of maximum length IT_INTERFACE_NAME_SIZE,
14744                including the terminating NULL character. */
14745             char name[IT_INTERFACE_NAME_SIZE];
14746           } it_interface_t;
14747
14748           typedef enum
14749           {
14750             IT_LISTEN_NO_FLAG = 0x0000,
14751             IT_LISTEN_CONN_QUAL_INPUT = 0x0001,
14752             IT_LISTEN_SUPPRESS_IRD_ORD = 0x0002
14753           } it_listen_flags_t;
14754
14755           typedef enum
14756           {
14757             IT_LISTEN_PARAM_ALL = 0x0001,
14758             IT_LISTEN_PARAM_IA_HANDLE = 0x0002,
14759             IT_LISTEN_PARAM_SPIGOT_ID = 0x0004,
14760             IT_LISTEN_PARAM_CONNECT_EVD = 0x0008,
```

```
  IT_LISTEN_PARAM_CONN_QUAL = 0x0010
} it_listen_param_mask_t;

typedef struct
{
  it_ia_handle_t ia_handle;  /* IT_LISTEN_PARAM_IA_HANDLE */
  size_t spigot_id;  /* IT_LISTEN_PARAM_SPIGOT_ID */
  it_evd_handle_t connect_evd;  /* IT_LISTEN_PARAM_CONNECT_EVD */
  it_conn_qual_t connect_qual;  /* IT_LISTEN_PARAM_CONN_QUAL */
} it_listen_param_t;

typedef enum
{
  IT_LMR_PARAM_ALL = 0x000001,
  IT_LMR_PARAM_IA = 0x000002,
  IT_LMR_PARAM_PZ = 0x000004,
  IT_LMR_PARAM_ADDR = 0x000008,
  IT_LMR_PARAM_LENGTH = 0x000010,
  IT_LMR_PARAM_MEM_PRIV = 0x000020,
  IT_LMR_PARAM_FLAG = 0x000040,
  IT_LMR_PARAM_SHARED_ID = 0x000080,
  IT_LMR_PARAM_RMR_CONTEXT = 0x000100,
  IT_LMR_PARAM_ACTUAL_ADDR = 0x000200,
  IT_LMR_PARAM_ACTUAL_LENGTH = 0x000400,
  IT_LMR_PARAM_ADDR_MODE = 0x000800,
  IT_LMR_PARAM_LINKED = 0x001000,
  IT_LMR_PARAM_IOBL_NUM_ELTS = 0x002000,
  IT_LMR_PARAM_IOBL_TYPE = 0x004000
} it_lmr_param_mask_t;

typedef struct
{
  it_ia_handle_t ia;  /* IT_LMR_PARAM_IA */
  it_pz_handle_t pz;  /* IT_LMR_PARAM_PZ */
  void *addr;  /* IT_LMR_PARAM_ADDR */
  it_length_t length;  /* IT_LMR_PARAM_LENGTH */
  it_mem_priv_t privs;  /* IT_LMR_PARAM_MEM_PRIV */
  it_lmr_flag_t flags;  /* IT_LMR_PARAM_FLAG */
  uint32_t shared_id;  /* IT_LMR_PARAM_SHARED_ID */
  it_rmr_context_t rmr_context;  /* IT_LMR_PARAM_RMR_CONTEXT */
  void *actual_addr;  /* IT_LMR_PARAM_ACTUAL_ADDR */
  it_length_t actual_length;  /* IT_LMR_PARAM_ACTUAL_LENGTH */
  it_addr_mode_t addr_mode;  /* IT_LMR_PARAM_ADDR_MODE */
  it_boolean_t linked;  /* IT_LMR_PARAM_LINKED */
  size_t iobl_num_elts;  /* IT_LMR_PARAM_IOBL_NUM_ELTS */
  it_iobl_type_t iobl_type;  /* IT_LMR_PARAM_IOBL_TYPE */
} it_lmr_param_t;

typedef uint64_t it_rdma_addr_t;

typedef enum
{
  IT_PZ_PARAM_ALL = 0x01,
  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
```

```
typedef void (*it_evd_callback_rtn_t)(
  IN it_evd_handle_t evd,
  IN void *arg
);

typedef enum
{
  IT_COMPARE_AND_SWAP,
  IT_FETCH_AND_ADD
} it_atomic_op_t;

/* prototypes */

#ifdef ITAPI_ENABLE_V10_BINDINGS
/*
    Backwards compatibility mode:
    For functions whose signature changed from v1.0 to v2.0,
    <it_api.h> converts v1.0 function names to explicit v1.0
    function names. Functions such as it_lmr_create continue
    to use v1.0 signatures.
*/
#define it_lmr_create  it_lmr_create10
#define it_rmr_create  it_rmr_create10
#define it_rmr_bind    it_rmr_bind10

#elif (defined(ITAPI_ENABLE_V20_BINDINGS))

/*
    Full v2.0 functionality:
    For functions whose signature changed from v1.0 to v2.0,
    <it_api.h> converts v1.0 function names to explicit v2.0
    function names. Functions such as it_lmr_create use the
    new v2.0 signatures.
*/
#define it_lmr_create  it_lmr_create20
#define it_rmr_create  it_rmr_create20
#define it_rmr_bind    it_rmr_link

#elif (defined(ITAPI_ENABLE_V21_BINDINGS))

/*
    Full v2.1 functionality:
    For functions whose signature changed relative to an earlier
    Version, <it_api.h> converts the function name to an explicit
    function name carrying as a suffix the IT-API major and minor
    version number in which the signature changed most recently.
*/
#define it_lmr_create  it_lmr_create21
#define it_rmr_create  it_rmr_create20
#define it_rmr_bind    it_rmr_link

#endif /* ifdef ITAPI_ENABLE_V10_BINDINGS */

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_identifier_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,
```

```
  OUT it_ep_param_t     *params
);
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
);
it_status_t it_ep_reset (
  IN  it_ep_handle_t ep_handle
);
it_status_t it_ep_ud_create (
  IN        it_pz_handle_t      pz_handle,
  IN        it_evd_handle_t     request_sevd_handle,
  IN        it_evd_handle_t     recv_sevd_handle,
  IN  const it_ep_attributes_t *ep_attr,
  IN        size_t              spigot_id,
  OUT       it_ep_handle_t     *ep_handle
);
it_status_t it_evd_callback_attach (
  IN  it_evd_handle_t       evd_handle,
  IN  it_evd_callback_rtn_t callback,
  IN  void                 *arg
);
it_status_t it_evd_callback_detach (
  IN  it_evd_handle_t evd_handle
);
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
);
it_status_t it_evd_dequeue (
  IN  it_evd_handle_t evd_handle,
  OUT it_event_t     *event
);
it_status_t it_evd_free (
  IN  it_evd_handle_t evd_handle
);
it_status_t it_evd_modify (
  IN        it_evd_handle_t     evd_handle,
  IN        it_evd_param_mask_t mask,
  IN  const it_evd_param_t     *params
);
it_status_t it_evd_post_se (
  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           /*
15182              it_lmr_create20 provides the v2.0 functionality
15183              and may be renamed to it_lmr_create in a future IT-API version.
15184           */
15185           it_status_t it_lmr_create20 (
15186             IN  it_pz_handle_t        pz_handle,
15187             IN  void                 *addr,
15188             IN  it_length_t           length,
15189             IN  it_addr_mode_t        addr_mode,
15190             IN  it_mem_priv_t         privs,
15191             IN  it_lmr_flag_t         flags,
15192             IN  uint32_t              shared_id,
```

```
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,
```

```
  OUT it_srq_param_t     *params
);
it_status_t it_ud_service_reply (
  IN        it_ud_svc_req_identifier_t ud_svc_req_id,
  IN        it_ud_svc_req_status_t     status,
  IN        it_remote_ep_info_t        ep_info,
  IN  const unsigned char             *private_data,
  IN        size_t                     private_data_length
);
it_status_t it_ud_service_request (
  IN  it_ud_svc_req_handle_t ud_svc_handle
);
it_status_t it_ud_service_request_handle_create (
  IN  const it_conn_qual_t         *conn_qual,
  IN        it_evd_handle_t         reply_evd,
  IN  const it_path_t              *destination_path,
  IN  const unsigned char          *private_data,
  IN        size_t                  private_data_length,
  OUT       it_ud_svc_req_handle_t *ud_svc_handle
);
it_status_t it_ud_service_request_handle_free (
  IN  it_ud_svc_req_handle_t ud_svc_handle
);
it_status_t it_ud_service_request_handle_query (
  IN  it_ud_svc_req_handle_t     ud_svc_handle,
  IN  it_ud_svc_req_param_mask_t mask,
  OUT it_ud_svc_req_param_t     *ud_svc_handle_info
);
#endif /* _ITAPI_H_ */
```

## G.2 it_api_os_specific.h

```
/*
 * it_api_os_specific.h
 *
 * Copyright © 2005 The Open Group
 * All rights reserved.
 * The copyright owner hereby grants permission for all or part of this
 * publication to be reproduced, stored in a retrieval system, or
 * transmitted, in any form or by any means, electronic, mechanical,
 * photocopying, recording, or otherwise, provided that it remains
 * unchanged and that this copyright statement is included in all
 * copies or substantial portions of the publication.
 *
 * For any software code contained within this specification,
 * permission is hereby granted, free-of-charge, to any person
 * obtaining a copy of this specification (the "Software"), to deal in
 * the Software without restriction, including without limitation the
 * rights to use, copy, modify, merge, publish, distribute, sublicense,
 * and/or sell copies of the Software, and to permit persons to whom
 * the Software is furnished to do so, subject to the above copyright
 * notice and this permission notice being included in all copies or
 * substantial portions of the Software.
```

```
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND
 * NON-INFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS
 * BE LIABLE FOR ANY CLAIM, DAMAGES, OR OTHER LIABILITY, WHETHER IN AN
 * ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF, OR IN
 * CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
 * SOFTWARE.
 *
 * Permission is granted for implementers to use the names, labels,
 * etc. contained within the specification. The intent of publication
 * of the specification is to encourage implementations of the
 * specification. This specification has not been verified for
 * avoidance of possible third-party proprietary rights. In
 * implementing this specification, usual procedures to ensure the
 * respect of possible third-party intellectual property rights should
 * be followed.
 */

#ifndef _ITAPI_OS_SPECIFIC_H_
#define _ITAPI_OS_SPECIFIC_H_
#include "/usr/include/netinet/in.h"
#include "/usr/include/sys/types.h"

#define IT_NO_ADDR              NULL
#define IT_INTERFACE_NAME_SIZE  128
#endif /* _ITAPI_OS_SPECIFIC_H_ */
```