Consortium Specification

Interconnect Transport API (IT-API)

Issue 1.0

The Interconnect Software Consortium

in association with

THE Open GROUP

Copyright © 2004, 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 NONINFRINGEMENT. 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.

Consortium Specification

Interconnect Transport API (IT-API) Issue 1.0

ISBN: 1-931624-37-2

Document Number: C040

Published by The Open Group, February 2004.

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

The Open Group
Apex Plaza, Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by electronic mail to:

ogspecs@opengroup.org

Contents

Figures

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, a vendor and technology-neutral consortium, has a vision of Boundaryless Information Flow achieved through global interoperability in a secure, reliable, and timely manner. The Open Group mission is to drive the creation of Boundaryless Information Flow by:

Working with customers to capture, understand, and address current and emerging requirements, establish policies, and share best practices

Working with suppliers, consortia, and standards bodies to develop consensus and facilitate interoperability, to evolve and integrate open specifications and open source technologies

Offering a comprehensive set of services to enhance the operational efficiency of consortia

Developing and operating the industry's premier certification service and encouraging procurement of certified products

The Open Group provides opportunities to exchange information and shape the future of IT. The Open Group members include some of the largest and most influential organizations in the world. The flexible structure of The Open Group membership allows for almost any organization, no matter what their size, to join and have a voice in shaping the future of the IT world.

More information 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.

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:

A new *Version* indicates there is no change to the definitive information contained in the previous publication of that title, but additions/extensions are included. As such, it *replaces* the previous publication.

A new *Issue* indicates there is substantive change to the definitive information contained in the previous publication of that title, and there may also be additions/extensions. As such, both previous and new documents are maintained as current publications.

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 Consortium Specification for the Interconnect Transport API. It has been developed and approved by The Interconnect Software Consortium in association with The Open Group.

Typographical Conventions

The following typographical conventions are used throughout this document:

Bold font is used in text for filenames, type names, and data structures

Italic strings are used for emphasis. Italics in text also denote variable names and functions.

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 in Chapter 1. IT-API objects are capitalized throughout the document (e.g. Interface Adapter, Endpoint, etc).

Trademarks

The Open Group® is a registered trademark 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 IEEE.

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 the following people in the development of this document:

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

Referenced Documents

The following documents are referenced in this document:

Infiniband Architecture Release 1.1 specification
Infiniband Trade Association

1 Introduction

The IT-API defines interfaces for direct interaction with RDMA-capable transports. The Phase 1 Specification covers VIA networks and the ***Reliable Connection*** and ***Unreliable Datagram*** services of InfiniBand networks. The IT-API Phase 1 Specification documentation set includes this introduction, a glossary, a global behaviors section, manual pages for 62 APIs and their supporting data type definitions, an implementation guide section, and two sample header files. The introduction and implementation guide and the 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 man 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 man pages. It is an introduction to how to use the API. A separate Implementation Guide discusses issues related to implementing the API.

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 see to it that the named instances of RDMA-capable transports available within a system can be listed using 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* is returned. The *it_ia_handle* is an opaque type reference ***Handle*** used by the Consumer to refer to a specific instance of an Implementation created ***IT Object***. The *it_ia_handle* 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* 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* 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 for message buffers. 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* that is used in subsequent IT-API calls to manage the IA's use of the LMR. Remote access privileges for the LMR can be set when the LMR is created. Attributes of the LMR can be queried and modified using the *it_lmr_query* and *it_lmr_modify* calls, respectively. The LMR is released using the *it_lmr_free* call.

A ***Protection Zone***, also called a ***PZ***, is used to control access to memory when messages are transferred. Many IT-API objects are associated with a PZ when they are created. IT-API 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 using the *it_pz_create* call, which returns an *it_pz_handle*. Attributes of the PZ can be queried using the *it_pz_query* call. A PZ is released using the *it_pz_free* call.

The Consumer may create a Remote Memory Region, also called an RMR, using the *it_rmr_create* call, which returns an *it_rmr_handle*. An RMR can be used in subsequent RDMA ***Data Transfer Operations*** to describe a local data ***Destination*** buffer. RMR attributes can be queried using the *it_rmr_query* call, and can be released using the *it_rmr_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 Endpoint by calling either *it_ep_rc_create*, for use with Reliable Connection communications, or *it_ep_ud_create*, for use with Unreliable Datagram communications. The EP can be queried and modified using the *it_ep_query* and *it_ep_modify* calls, respectively. It can be released using the *it_ep_free* call.

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 Handle object that is used to await Connection Requests. The Listen Handle is created using the *it_listen_create* call, which returns an *it_listen_handle*. Attributes of the Listen Handle can be queried using the *it_listen_query* and the Listen Handle can be released using the *it_listen_free* call.

When a Connection Request is received it can be accepted or rejected using the *it_ep_accept* and *it_reject* calls, respectively. An existing Connection can be terminated using 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*.

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 using the *it_address_handle_create* call, which returns an *it_address_handle*. Attributes of the Address Handle can be queried and modified using the *it_address_handle_query* and *it_address_handle_modify* calls. The Address Handle can be released using 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 using the *it_ud_service_request_handle_create* call, which returns an *it_ud_svc_req_handle*. Attributes of the Service Request Handle can be queried using the *it_ud_service_request_handle_query* and the Service Request Handle can be released using the *it_ud_service_request_handle_free* call. A Service Request Handle is used in the *it_ud_service_request* to provide addressing information for use in sending the reply message sent by the *it_ud_service_reply* call.

1.4 Data Transfer Operations

The Consumer can queue different kinds of Data Transfer Operations, also called ***DTO***s, to an Endpoint. DTOs include sending and receiving messages, issuing RDMA requests, and associating and disassociating Remote Memory Regions with Local Memory Regions.

The Consumer can associate an RMR with a sub-region of memory within an LMR using the *it_rmr_bind* call. The *it_rmr_bind* call returns an *it_rmr_context* identifier, which can be used in subsequent RDMA transfer requests to define the message buffer to be used for the RDMA operation. The *it_rmr_unbind* call removes the binding currently associated with a RMR. These calls provide request-specific access control for IA memory accesses, in addition to the region-based access control offered by LMRs.

The Consumer may issue requests to send messages using either the *it_post_send* or *it_post_sendto* interfaces, depending on whether the Endpoint used for communication is of the Connected or Datagram type, respectively. The Consumer issues requests to receive messages using either the *it_post_recv* or *it_post_recvfrom* calls. RDMA operations are initiated using the *it_post_rdma_read* and *it_post_rdma_write* calls. Completions of the posted operations 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 the request has been executed successfully, while for other calls it indicates only that the request has been accepted by the Implementation for later execution. The Consumer is notified of the completion of a request via an asynchronous event mechanism. Each type of Event generated by the IT-API has an associated IT Event object that contains information about the Event. Common IT Event types include DTO completion, Connection establishment-related Events, and transport error conditions. 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*. Attributes of an EVD can be queried and modified using the *it_evd_query* and *it_evd_modify* calls, respectively. An EVD is released using the *it_evd_free* call. The Consumer may access 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 Events.

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.

Most Events are associated with Endpoints. These Events correspond to DTO completions or other communication 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 DTO requests. This includes RMR association and disassociation, messages sent, and RDMA requests completed. A second EVD enqueues Events generated as a result of messages being received at the Endpoint. For EPs using connected communications, a third EVD enqueues Events related to Connection management. EVDs may be shared across multiple Endpoints.

Some Events are not associated with a particular Endpoint, but are associated with the 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.

1.6 Event Notification

The IT-API provides the Consumer with control over a number of aspects of Event Notification. IT-API interfaces used to initiate Data Transfer Operations include flags to support Event and Event Notification Suppression, and to request remote-side Endpoint Notification of message delivery. The details of these features are described in *it_dto_flags*. The *it_evd_wait* call provides a blocking interface for waiting for the next Event to occur, along with a timeout value. EVDs provide a thresholding attribute used to batch the delivery of Event Notification.

2 Definitions

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 on in this section.)

AEVD

See Aggregation Event Dispatcher.

Affiliated Asynchronous Event

An Event associated with a specific Endpoint or EVD.

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.

Bind

See RMR Bind.

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 Event

The set of Event types that indicate that a previously posted operation has completed.

Completion Suppression

An optional DTO behavior specifying that no Event is to be generated upon successful completion of the 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 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.

Data Transfer Operation (DTO)

A request submitted by the Consumer to the Implementation to move data between two Endpoints.

Destination

The Endpoint where a message is received.

DTO

See Data Transfer Operation.

DTO Cookie

A Consumer-supplied identifier for a Data Transfer Operation, Bind, or Unbind operation that allows the Consumer to uniquely identify the operation when it completes.

Endpoint (EP)

The object to which DTOs and RMR operations are posted. An Endpoint is associated with a single Spigot.

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.

Endpoint Key

A construct that some transports require to be associated with an outgoing datagram to allow the Receiver to validate that the sender of the datagram has permission to access the Receiver's Endpoint.

Endpoint Protection Zone

The Protection Zone associated with an Endpoint.

EP

See Endpoint.

EVD

See Event Dispatcher.

Event

A structure or record that is delivered to the Consumer through an Event Dispatcher to provide notice of some kind. Types of Events include DTO completions, Connection state changes, asynchronous errors, and information passed through the *it_evd_post_se* interface that is generated by the Consumer.

Event Dispatcher (EVD)

An IT Object that conceptually merges Event completion Notifications for the Consumer. The IT-API defines two types of EVDs: a Simple Event Dispatcher, and an Aggregation Event Dispatcher.

Event Stream

A source of Events for the Simple Event Dispatcher: DTO completions, Connection Requests, Connection reject Notifications, Connection establishment completion Notifications, disconnect Notifications, Connection errors, Connection Request timeouts, asynchronous errors, RMR Bind and Unbind completion Notifications, and Consumer-generated Notifications. An Event Stream is the conduit between IT-API objects that generate Events and Simple Event Dispatchers that consume Events.

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.

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.

InfiniBand (IB)

One of the transports that the IT-API supports. The host interface portion of InfiniBand is defined in "InfiniBand Architecture Specification Volume 1", available at http://www.infinibandta.org.

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 Native Transport

Transport services defined by 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.

IP

The IETF Internet Protocol.

IPv4

The IETF Internet Protocol version 4.

IPv6

The IETF Internet Protocol version 6.

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

LMR

See Local Memory Region.

LMR Triplet

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

Local Memory Region (LMR)

A virtually contiguous area of arbitrary size within a Consumer's address space that has been registered using the *it_lmr_create* routine, enabling local access and optional remote access.

Network Address

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

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.

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.

OUI

See Organization Unique Identifier.

Outstanding Operation

An operation is "Outstanding" until the Event for the operation completes, or for an operation whose completion has been suppressed, until an operation posted subsequent to it completes.

Path

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

Port Number

See IANA Port Number.

Private Data

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

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 Read

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

RDMA Write

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

Receive

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

Receive Queue

An internal queue associated with an Endpoint on which Receive DTOs are posted.

Reliable Connected (RC)

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

Reliable Connection

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

Remote Direct Memory Access (RDMA)

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

Remote Memory Region (RMR)

A window that can be bound to a section of a Local Memory Region to enable remote accesses.

Request Queue

An internal queue of an Endpoint on which DTOs and RMR Binds and Unbinds are posted. The Request Queue to which RMR Bind, RMR Unbind, Send, RDMA Read, and RDMA Write operations are posted is commonly called the Send Queue. The Request Queue to which Receive operations are posted is commonly called the Receive Queue.

RMR

See Remote Memory Region.

RMR Bind

An operation that associates an RMR with a section of an LMR and thereby enables remote access to that section.

RMR Context

An opaque identifier generated by the Implementation to represent a contiguous memory region. Used by remote Consumers in RDMA operations that target this region.

RMR Unbind

An operation that destroys the association of an RMR with a section of an LMR.

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

An internal queue of an Endpoint on which Receive DTOs are posted.

Service Reply

See UD Service Reply.

Service Request

See UD Service Request.

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.

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.

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.

The Open Group

The Open Group standards organization.

Transport Service Type

See Service Type.

UD

See Unreliable Datagram.

Unaffiliated Asynchronous Event

An Event that is not associated with a specific Endpoint or EVD, but is only associated with an Interface Adapter.

Unbind

See RMR Unbind.

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.

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.

VIA

See Virtual Interface Architecture.

Virtual Interface Architecture (VIA)

One of the transports that the IT-API supports. VIA is defined by "The Virtual Interface Architecture Specification", which is available at http://developer.intel.com/design/servers/vi/the_spec/specification.htm.

3 Global Behavior

The IT-API Global Behavior 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 manual pages.

3.1 Non-Blocking APIs

Nearly all IT-API routine invocations return program control to the issuing Consumer without blocking the caller's execution indefinitely. The call may return success or failure, with call success indicating that the request was completed successfully, or that the request has been accepted for later execution. The Consumer is notified of the subsequent completion of a request via an asynchronous Event mechanism. Each Event generated by the IT-API has an associated IT Event object that contains information about the disposition and status of the Event.

A few IT-API interfaces may block the caller's execution pending some Event or condition. These exceptions are noted explicitly in the appropriate manual pages.

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

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, and 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 *rmr_bind* the *lmr_handle*, *rmr_handle*, and *ep_handle* arguments must be treated as primary call arguments for thread-safety purposes. For *rmr_unbind* the *rmr_handle* and *ep_handle* arguments, and the *lmr_handle* that is bound to the rmr must be treated as primary call arguments for thread-safety purposes.)

This definition of thread-safe and not thread-safe routines allow 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.

766

Non-Performance Critical Routines	Strongly Thread-Safe Model	Efficiently Thread-Safe Model	Not Thread-Safe Model
it_address_handle_create *it_convert_net_addr* *it_ep_rc_create* *it_ep_ud_create* *it_evd_create* *it_get_handle_type* *it_get_pathinfo* *it_hton64, it_ntoh64* *it_ia_create* *it_ia_query* *it_interface_list* *it_listen_create* *it_listen_query* *it_lmr_create* *it_make_rdma_addr* *it_pz_create* *it_pz_query* *it_rmr_create* *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 *it_evd_post_se* *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_rdma_read *it_post_rdma_write* *it_post_recv* *it_post_recvfrom*	Thread-safe	Not thread-safe	

it_post_send *it_post_sendto* *it_rmr_bind* *it_rmr_unbind* *it_ud_service_reply* *it_ud_service_request*		

767

768

Object Management Routines	Strongly Thread-Safe Model	Efficiently Thread-Safe Model	Not Thread-Safe Model
it_address_handle_modify	Thread-safe	Not thread-safe	
it_address_handle_query			
it_ep_connect			
it_ep_modify			
it_ep_query			
it_ep_reset			
it_evd_modify			
it_evd_query			
it_get_consumer_context			
it_lmr_modify			
it_lmr_query			
it_lmr_sync_rdma_read			
it_lmr_sync_rdma_write			
it_rmr_query			
it_set_consumer_context			
it_ud_service_request_handle_query			
Object Destructor Routines	**Strongly Thread-Safe Model**	**Efficiently Thread-Safe Model**	**Not Thread-Safe Model**
it_address_handle_free	Not thread-safe		
it_ep_accept			
it_ep_disconnect			
it_ep_free			
it_evd_free			
it_handoff			
it_ia_free			
it_ia_info_free			
it_listen_free			
it_lmr_free			
it_reject			

it_ud_service_request_handle_free *it_pz_free* *it_rmr_free*	

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

3.3 Signal Handlers

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

3.4 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.

3.5 Exec Semantics

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

3.6 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.

3.7 Error Handling

Error Notification is provided to Consumers of the IT-API in two ways: as error return values to interface calls, and as asynchronous Events containing error status information for the earlier request. In general, interface calls return an 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 asynchronous Event.

3.8 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 error whenever possible. However, it may not always be possible for the Implementation to detect improper use of a Handle, and improper use may result in data corruption or fatal errors for the Consumer.

4 API Manual 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 - disconnects an existing Endpoint–to-Endpoint Connection.

it_ep_free – destroys 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 – resets a Reliable Connected Endpoint into the initial state

it_ep_ud_create – Create an Endpoint for Unreliable Datagram.

it_evd_create – create Simple or Aggregate Event Dispatcher

it_evd_dequeue – Dequeue Events from Event Dispatcher

it_evd_free – destroys 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* - hands-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 – destroys 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_free – destroy a 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_sync_rdma_read – make memory changes visible to an incoming RDMA Read op

it_lmr_sync_rdma_write – make effects of an incoming RDMA Write operation visible

it_make_rdma_addr – make a platform independent RDMA address

it_post_rdma_read – post an RDMA Read DTO to a Reliable Connected Endpoint

it_post_rdma_write – post an RDMA Write DTO to a connected Endpoint

it_post_recv – post a Receive DTO to a connected Endpoint

it_post_recvfrom – post a Receive DTO to a datagram Endpoint

it_post_send – post a Send DTO to a connected Endpoint

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_bind – post request to Bind a Remote Memory Region to a memory range

it_rmr_create – create a Remote Memory Region (RMR)

it_rmr_free – destroy a Remote Memory Region

it_rmr_query – get attributes of a Remote Memory Region

it_rmr_unbind – post operation to Unbind a Remote Memory Region from its memory range

it_set_consumer_context – associate a Consumer Context with an IT Object Handle

it_ud_service_reply – returns UD communication information

it_ud_service_request – request the recipient to return UD communication information.

it_ud_service_request_handle_create – creates 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 – returns *it_ud_svc_req_handle_t* information.

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;
```

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 logical 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 the "Create Address Handle" section 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.

ERRORS

None.

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
);
```

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 in DTO operations. If an Address Handle is freed while there is still a Send DTO outstanding that references the Address Handle, whether that Send completes successfully is implementation-dependent. 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.

ERRORS

None.

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
);
```

DESCRIPTION

addr_handle	Address Handle to modify.
mask	Logical 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 can not be modified.

```
typedef struct {
   ...
   it_path_t   path; /* IT_ADDR_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_address_handle_modify* with the resulting Path. See the *it_address_handle_create* man 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_SGID	The Source Global ID (*ib.local_port_gid*) member of the supplied Path was invalid
IT_ERR_INVALID_SLID	The Source Local ID (*ib.local_port_lid*) member 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.

ERRORS

None.

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;
```

DESCRIPTION

addr_handle — Address Handle to query.

mask — Logical 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 that this Address Handle is associated with.
pz	The Handle for the Protection Zone that this Address Handle is associated with.
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_address_handle_create* man page for details.

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.

ERRORS

None.

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.

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 format of the Network Address was not recognized.
IT_ERR_INVALID_CONVERSION	The requested Network Address conversion was not supported by the Implementation.

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.

ERRORS

None.

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 type more to its liking.

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;
```

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*) manual 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. The delivery of Private Data to the Remote Endpoint 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*) manual 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. 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*.

On the Passive side, the Endpoint will transition into the IT_EP_STATE_PASSIVE_CONNECTION_PENDING state when the Consumer calls *it_ep_accept*. The Passive side successfully calling *it_ep_accept* will cause the Active Endpoint to eventually transition into the IT_EP_STATE_ACTIVE2_CONNECTION_PENDING state .

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 can not 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* manual 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_inflight_incoming* or the number of *rdma_read_inflight_outgoing* 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.

ERRORS

None.

APPLICATION USAGE

1. The Consumer should be aware that the delivery of Private Data to the Remote Endpoint may be unreliable and should be used accordingly. For some transports the Passive side's *it_ep_accept* Private Data is delivered reliably.
2. 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.
3. 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.
4. 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 */
```

```
} it_via_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_conn_attributes_t;

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

DESCRIPTION

ep_handle Handle for an instance 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 that the Consumer is initiating a Connection establishment request to. See *it_conn_qual_t* for more information on Connection Qualifiers.

cn_est_flags 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*.

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.

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 implementation-independent code should pass a NULL value for the *conn_attr* parameter. If the Consumer passes a NULL 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 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 a NULL value for the *conn_attr* parameter.

The passive side Consumer can either choose to either 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 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 transitions into the (IT_EP_STATE_CONNECTED) connected state. See the *it_ep_state_t* manual 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* manual 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 can not 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_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.

ERRORS

None.

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.

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
);
```

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 established 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. The Endpoints will transition into the IT_EP_STATE_NONOPERATIONAL state. See the *it_ep_reset* manual 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 transitions 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 state.

Once the Endpoint is in the IT_EP_STATE_NONOPERATIONAL state any pending Data Transfer Operations or Bind or Unbind 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* manual page for a complete description of the Endpoint state behavior and transitions.

The delivery of Private Data to the Remote Endpoint is unreliable.

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.

ERRORS

See above.

APPLICATION USAGE

The Consumer should be aware that the delivery of Private Data to the Remote Endpoint is unreliable and should be used accordingly.

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

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 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.

The Endpoint can be destroyed in any state. The freeing of an Endpoint also terminates the generation of Events to any of the EVDs associated with the Endpoint.

Use of the Handle *ep_handle* of the destroyed Endpoint in any subsequent operation fails.

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*.

Freeing an Endpoint in the IT_EP_STATE_CONNECTED state while DTOs are in progress causes incoming DTOs to be ignored. The outstanding DTOs and RMRs may be flushed to the *request_sevd_handle* and *recv_sevd_handle*. 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 Endpoint in the IT_EP_STATE_ACTIVE1_CONNECTION_PENDING, IT_EP_STATE_ACTIVE2_CONNECTION_PENDING, 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_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.

ERRORS

None.

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.

If the Consumer wants to ensure that all Completion Events are dequeued prior to calling *it_ep_free*, the following method will work for both Request and Receive Queues:

The Consumer should call *it_ep_disconnect* first. Then post a DTO or RMR operation set up as a "marker" that is flushed by the Implementation to *recv_evd_handle* or *request_evd_handle*. The DTO or RMR is made a "marker" operation by setting the IT_COMPLETION_FLAG on the operation. Now, when the Consumer dequeues the completion of the "marker" from the EVD, it is guaranteed that all previously posted DTO and RMR completions (including those posted with IT_COMPLETION_FLAG cleared) for the Endpoint were dequeued from that EVD for the Request or Receive Queue of the Endpoint to which the "marker" had been posted.

For *connect_evd_handle*, Consumer should dequeue the disconnect or broken Connection Event for this Endpoint.

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 the *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 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_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	Logical 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 can not 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

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 logical OR of the Endpoint attributes flag values (above) that 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_inflight_incoming* resources at this time.
IT_ERR_RESOURCE_ORD	The underlying transport could not allocate the requested *rdma_read_inflight_outgoing* 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_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.

ERRORS

None.

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                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	Logical 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 Endpoint valid only if not in IT_EP_STATE_UNCONNECTED state otherwise, value is undefined.
ep_state	State of the Endpoint.

service_type	Endpoint Service Type.
dst_path	For RC it is invalid and contents are undefined if the Endpoint is in the IT_EP_STATE_UNCONNECTED state. Otherwise, on the active side of a Connection, it is the Path that was specified at Connection Request time; on the passive side of a Connection, it is the Path used by the Implementation to reach the requesting remote Endpoint. Invalid for UD Endpoint.
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.

ERRORS

None.

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_rc_creation_flags_t;
```

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 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 Bind and RMR Unbind are reported in *request_sevd_handle*. It is permissible for *recv_sevd_hand*le 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 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. Use of the *flags* value IT_EP_REUSEADDR 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.

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 as well. This interval can be on 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 specifies IT_EP_REUSEADDR in *flags* for the *it_ep_rc_create* routine.

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 Consumer requested in *ep_attr*.

The Consumer can find the actual allocated resources by using *it_ep_query*. For detailed Endpoint attributes see man 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_inflight_incoming* resources at this time.
IT_ERR_RESOURCE_ORD	The underlying transport could not allocate the requested *rdma_read_inflight_outgoing* 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* for a description of the disabled state.

ERRORS

None.

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_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
);
```

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 can not be reset.

Upon return of this operation any Completions for the Endpoint not yet harvested by 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 reuse 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.

ERRORS

None.

SEE ALSO

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

it_ep_ud_create()

NAME

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

 );
```

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_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_hand*le 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 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 Consumer requested in *ep_attr*. Consumer can find the actual allocated resources by using *it_ep_query*. For detailed Endpoint attributes see man 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_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.

ERRORS

None.

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_create()

NAME

it_evd_create – create Simple or Aggregate Event Dispatcher

SYNOPSIS

```
#include <it_api.h>

it_status_t    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 created Event Dispatcher belongs.
event_number	Identifier for Event Stream type that can be enqueued to the EVD.
evd_flag	Logical OR of flag values for creation operation.
sevd_queue_size	Minimum size of the Simple EVD Event queue. This parameter is ignored for 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 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 optional *file descriptor* corresponding to Event Dispatcher. Only valid if return value is IT_SUCCESS and 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:

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.

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 Aggregate EVD this means that no Events are fed to *evd_handle* until *evd_handle* is associated with a Simple EVD. For Simple EVD this means that no Events are fed to *evd_handle* until *evd_handle* is associated with an Endpoint, Listen Handle, or UD Service Request Handle depending on the stream type.

For IT_ASYNC_AFF_EVENT_STREAM Event Stream type, the Simple EVD receives the Async Affilated Events for the *ia_handle*. For 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 exception of IT_ASYNC_AFF_EVENT_STREAM, IT_ASYNC_UNAFF_EVENT_STREAM and IT_SOFTWARE_EVENT_STREAM Event Stream types. For 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 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 can not overflow.

Once a SEVD overflows, subsequent Events from the Event Stream will be dropped. For all Event Streams, with the exception of the 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 a SEVD either by dequeueing an Event, or by using *it_evd_modify* to increase the queue size of the SEVD. For the IT_DTO_EVENT_STREAM, however, Events will continue to be dropped; the overflow can not be corrected.

The behavior of a 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 man page associated with each Event Stream type provides details of the default overflow behavior. The Consumer specifies they desire 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 a 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 a 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 a 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* if they so choose. 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 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	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 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 man 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.

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 Consumer make 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 Consumer must manually reset the *evd_overflowed* state of the EVD (see *it_evd_modify*) and 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 can not 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 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 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 greater than or equal to one. 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 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 SEVD the a*evd_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*; specifying both an *aevd_handle* not equal to IT_NULL_HANDLE and the IT_EVD_CREATE_FD *evd_flag* in a call to *it_evd_create* will fail and return 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 a*evd_handle* or the *fd* will be notified by the Implementation when a Notification Event arrives or *sevd_threshold* value is reached when EVD is enabled.

When a 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 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 can not 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 IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* bit cleared. The order of Events of 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 an lower-level Event from the same SEVD that caused Notification.

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

SEVD and AEVD can support multiple waiters. For SEVD the *sevd_threshold* value must be 1 for multiple waiters to be supported.

An SEVD waiter will block when SEVD queue is empty. An AEVD waiter will block when all associated SEVDs are empty. An SEVD waiter may block when 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 number of Events larger or equal to its *sevd_threshold*.

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 may be unblocked. If an arriving Notification Event causes SEVD to reach Notification criteria and the SEVD is enabled then Notification will be generated for the associated AEVD or *fd*. As many Notifications can be generated as there are Events available on all SEVDs associated with the AEVD.

it_evd_dequeue from SEVD will return an Event, if one exists, from SEVD queue regardless if there are waiters except when the SEVD is associated with AEVD with IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* bit cleared. In the latter case dequeue from the SEVD is not allowed. For AEVD with 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 AEVD with 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.

ERRORS

None.

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*. 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 DESCRIPTION section above for exceptions) and also may be prohibited from dequeueing from the underlying SEVDs (again see 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 Consumer via Events if there is no Simple EVD for IT_ASYNC_AFF_EVENT_STREAM or for IT_ASYNC_UNAFF_EVENT_STREAM. Additionally, an IA can enter catastrophic state and not notify 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 IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* bit clear, the operation dequeues the first Event from one of its associated SEVDs.

For AEVD with 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 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 Bind or RMR Unbind operation that was posted to an Endpoint, the Implementation guarantees that all Send, RDMA Read, RDMA Write, RMR Bind and RMR Unbind 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.

ERRORS

None.

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 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 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.

Use of the Handle *evd_handle* in any subsequent operation fails.

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_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.

ERRORS

None.

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	Logical 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 can not 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.

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 IT_FALSE) to reset an overflow condition on the EVD. See *it_evd_create* for more details.

AEVD can only be changed for the SEVD that is disabled, and IT_EVD_DEQUEUE_NOTIFICATIONS is set in the *evd_flag* for the current *aevd* (i.e. for the AEVD), and there is no *fd* associated with the SEVD (i.e., for the SEVD). 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 can not 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 can not 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_CREAT_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 *sevd_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, 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 or *fd*, if either of them exists. If an associated AEVD has the IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* cleared the AEVD can dequeue Events from the SEVD. The Consumer can not wait on or dequeue from the SEVD which is associated with the AEVD has the IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* cleared even when SEVD is disabled. The Consumer can wait on or dequeue from the SEVD which is associated with the AEVD has the IT_EVD_DEQUEUE_NOTIFICATIONS *evd_flag* set when SEVD is disabled.

The Consumer can disable 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_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.

ERRORS

None.

SEE ALSO

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

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.

ERRORS

None.

APPLICATION USAGE

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 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_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_evd_param_t;
```

DESCRIPTION

evd_handle	Event Dispatcher.
mask	Logical 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 below 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.

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.

ERRORS

None.

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 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).

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 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.

Consumer should not wait on an EVD that is associated with and enabled for Notification to higher level objects. An attempt by 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).

An Implementation can support one or more simultaneous waiters on the same EVD (for thread safety models see Section 3.2) 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 supported for the SEVD.

A waiter can be blocked. An SEVD waiter will block when 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 equal or larger than *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 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 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 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 Bind or RMR Unbind operation that was posted to an Endpoint, the Implementation guarantees that all Send, RDMA Read, RDMA Write, RMR Bind and RMR Unbind 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.

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_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.

ERRORS

None.

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 Consumer provides is sufficient to hold a returned Event. As a result a segmentation fault or memory corruption may occur if the Implementation over-runs 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 one, 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 Consumer should not have multiple threads calling on the same EVD Handle simultaneously. Consumer should choose an Implementation that supports multithreaded applications if they want to have multiple waiters. Consumer should set the *sevd_threshold* to one 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 that it is currently waiting on. 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_send*(), *it_post_sendto*(), *it_post_rdma_read*(), *it_post_rdma_write*(), *it_rmr_bind*(), *it_rmr_unbind*(), *it_dto_events*, *it_dto_flags_t*

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_context 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.

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.

ERRORS

None.

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.

ERRORS

None.

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 to communicate with.
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 man 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 one 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 man page for *it_path_t* for details on the internal structure of a Path.

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_NETADDR	The format of the Network Address was not recognized.
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.

ERRORS

None.

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. 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 can not 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.

ERRORS

None.

APPLICATION USAGE

Calls to *it_reject*, *it_ep_accept*, and *it_handoff* that 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.

ERRORS

None.

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(), *ntohl*()

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.

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 0. When a new version of the IT-API is released, a unique combination of major and minor version numbers is associated with it. If the new release is source code compatible with the previous release the major version number of the new release will be the same as that of the previous release, and the minor version number will be incremented by one. If the new release is not source code compatible with the previous release the major version number of the new release will be incremented by one, and the minor 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.

ERRORS

None.

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* 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.

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.

ERRORS

None.

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.

RETURN VALUE

None.

ERRORS

None.

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 man 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 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.

ERRORS

None.

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 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.

RETURN VALUE

None.

ERRORS

None.

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, NULL);
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 isn't available anymore. */

            }

            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 isn't available anymore.  (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_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	Specifies whether the Connection Qualifer is an input or output parameter.
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.

A backlog for the incoming Connection Request Events is provided by the size of the Simple Event Dispatcher that the Events are directed to. 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.

ERRORS

None.

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*.

RETURN VALUE

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

IT_ERR_INVALID_LISTEN	The Listen 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.

ERRORS

None.

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 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.

ERRORS

None.

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_length_t       length,
  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_PRIV_NONE           = 0x0001,
   IT_PRIV_READ_ONLY      = 0x0002,
   IT_PRIV_REMOTE_READ    = 0x0004,
   IT_PRIV_REMOTE_WRITE   = 0x0008,
   IT_PRIV_REMOTE         = 0x0010,
   IT_PRIV_ALL            = 0x0020,
   IT_PRIV_DEFAULT        = 0x0040
} it_mem_priv_t;

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

DESCRIPTION

pz_handle	Protection Zone in which to create memory region.
addr	Virtual address for start of memory region.
length	Length of memory region in bytes.
privs	Logical OR of access privilege flags for region.

flags	Logical OR of modifier flags.
shared_id	Optional identifier for sharing Interface Adapter translation resources.
lmr_handle	Returned Handle for created memory region.
rmr_context	Optionally returned Context allowing remote access to this memory.

The *it_lmr_create* routine allows an Interface Adapter to access a contiguous Local Memory Region in a process' virtual address space; memory that is to be the Source or Destination of a DTO must first be registered using this call. The region starts at virtual address *addr* and extends for *length* bytes, and this address range must already be valid in the process's virtual 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.

The type of access granted is specified by the *privs* argument as the logical OR of one or more of the following flags. Unless otherwise noted, all combinations are allowed.

IT_PRIV_READ_ONLY	Specifies that only read accesses will be allowed to the region. If this flag is omitted, read and write access will be allowed. Note that this flag does not by itself enable remote access; one or more of the remote access flags must also be specified for that purpose.
IT_PRIV_REMOTE_READ	Grants remote read access to the IA, enabling the LMR to be used as an RMR, as a Source buffer for remote RDMA Read DTOs.
IT_PRIV_REMOTE_WRITE	Grants remote write access to the IA, enabling the LMR to be used as an RMR, as a Destination buffer for remote RDMA Write DTOs.
IT_PRIV_REMOTE	Grant remote read and write access to the IA.
IT_PRIV_ALL	Grant all types of access to the IA.

The special value IT_PRIV_DEFAULT or 0 may be used to grant default access, which includes local read and write access. The value IT_PRIV_NONE is invalid here. It is invalid to request remote write access in combination with IT_PRIV_READ_ONLY. 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 logical 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 re-use resources from a matching LMR, which is defined as an existing LMR that was created using the same value of the *shared_id* argument, refers to the same physical memory pages as the new LMR, and has the same coherency mode. If any of these conditions are not met, the LMRs do not match. If a matching LMR is not found, then a new LMR is created and *shared_id* is associated with it. The value of *shared_id* is only an efficiency aid for the matching process and need not be unique. For example, if unrelated callers 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. If IT_LMR_FLAG_SHARED is not specified, then *shared_id* is ignored.

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_sync_rdma_write* and *it_lmr_sync_rdma_read* calls.

The coherency mode of an LMR is inherited by any RMR that is bound 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 NULL. 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. Modifying or destroying the LMR may revoke remote access using this 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. 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 boundaries 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 an RMR using *it_rmr_bind*, which guarantees byte level registration granularity.

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 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.

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_PRIVS	The requested memory privileges (*privs*) contained an invalid flag.
IT_ERR_INVALID_FLAGS	The *flags* value was invalid.
IT_ERR_FAULT	Part or all of the supplied address range was invalid.
IT_ERR_ACCESS	The Consumer was not allowed to have the requested memory privileges.
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_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.

ERRORS

None.

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 binding the RMR to the desired region of the LMR using *it_rmr_bind*. This operation returns an RMR Context.

SEE ALSO

it_lmr_free(), *it_lmr_query*(), *it_lmr_modify*(), *it_lmr_sync_rdma_read*(), *it_lmr_sync_rdma_write*()

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* may no longer be used. 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. *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 DTO's, Bind, and Unbind 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.

ERRORS

None.

SEE ALSO

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

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 Logical 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 can not 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_lmr_create* for flag definitions and restrictions.

On successful return, the previous RMR Context (if any) is invalidated. If remote access privileges are specified in *privs*, then a new RMR Context is created and associated with the LMR. The new RMR Context may be retrieved using *it_lmr_query*.

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 DTO's, Bind, or Unbind 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 modify operation fails because the caller was not allowed to have the requested memory privileges, or fails due to insufficient resources, then the old RMR Context (if any) is invalidated. The *lmr_handle* is also invalidated and may no longer be used in any calls. Any resources that were associated with the LMR are freed in this case.

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_PZ	The Protection Zone Handle (*pz_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_ACCESS	The Consumer was not allowed to have the requested memory privileges.
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.

ERRORS

None.

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 binding an RMR to an LMR using *it_rmr_bind*.

SEE ALSO

it_lmr_create(), *it_lmr_free*(), *it_lmr_query*(), *it_lmr_param_t*, *it_lmr_param_mask_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_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_lmr_param_t;
```

DESCRIPTION

lmr_handle Local Memory Region.

mask Logical 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 of the LMR.
length	The requested length in bytes of the LMR.
privs	The memory access privileges specified to create the LMR.
flags	The flags specified to create the LMR.
shared_id	The *shared_id* specified to create the LMR, if *flags* included IT_LMR_FLAG_SHARED. Otherwise, undefined.
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.
actual_length	The actual length for which bounds checking is done for data transfers.

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.

ERRORS

None.

SEE ALSO

it_lmr_create(), *it_lmr_modify*(), *it_lmr_free*()

it_lmr_sync_rdma_read()

NAME

it_lmr_sync_rdma_read – make memory changes visible to an incoming RDMA Read operation

SYNOPSIS

```
#include <it_api.h>

it_status_t it_lmr_sync_rdma_read(
  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_sync_rdma_read* 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_sync_rdma_read* after modifying data in a memory range in this region that will be the target of an *incoming* RDMA Read operation. *it_lmr_sync_rdma_read* must be called after the Consumer has modified the memory range but before the RDMA Read operation starts, and the memory range that will be accessed by the RDMA Read must be supplied by the caller in the *local_segments* array. After this call returns, the RDMA Read 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 operation on an LMR created in non-coherent mode attempts to read from a memory range that is not properly synchronized using *it_lmr_sync_rdma_read*, the returned contents are undefined.

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.

ERRORS

None.

APPLICATION USAGE

Determining when an RDMA Read 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 to call *it_lmr_sync_rdma_read* 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 can receive this message and thus know when it is safe to initiate the RDMA Read operation.

SEE ALSO

it_lmr_create(), *it_lmr_sync_rdma_write*(), *it_lmr_triplet_t*

it_lmr_sync_rdma_write()

NAME

it_lmr_sync_rdma_write – make effects of an incoming RDMA Write operation visible to Consumer

SYNOPSIS

```
#include <it_api.h>

it_status_t it_lmr_sync_rdma_write(
  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_sync_rdma_write* 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_sync_rdma_write* before reading data from a memory range in this region that was the target of an *incoming* RDMA Write operation. *it_lmr_sync_rdma_write* 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_sync_rdma_write* 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_sync_rdma_write* before the RDMA Write is initiated. Conversely, after an RDMA Write completes, the Consumer must call *it_lmr_sync_rdma_write* 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_sync_rdma_write*, 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.

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.

ERRORS

None.

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_sync_rdma_write*.

SEE ALSO

it_lmr_create(), *it_lmr_sync_rdma_read*(), *it_lmr_triplet_t*

it_make_rdma_addr()

NAME

it_make_rdma_addr – make a platform independent RDMA address

SYNOPSIS

```
#include <it_api.h>

typedef uint64_t it_rdma_addr_t;

it_rdma_addr_t it_make_rdma_addr(
                    void        *addr
);
```

DESCRIPTION

addr Local address.

The *it_make_rdma_addr* 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 called an RDMA address. A network peer may use this RDMA address in RDMA Read and Write operations. *it_make_rdma_*addr performs no validity checking on *addr*, so *addr* is not required to lie within a currently registered LMR when *it_make_rdma_addr* is called.

RETURN VALUE

This function always succeeds and returns a 64-bit RDMA address.

ERRORS

None.

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_post_rdma_read()

NAME

it_post_rdma_read – post an RDMA Read DTO to a Reliable Connected 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
);
```

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 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 remote buffer to read from.
rmr_context	The RMR Context for the remote buffer to read from.

it_post_rdma_read requests a transfer of the data from a remote buffer into the local buffer specified by *num_segments* and *local_segments* over the reliable Connection of the *ep_handle* Endpoint. The size of the data transferred is specified by the sum of sizes of the *local_segments*. A zero-sized message may be transferred over the Connection. The *it_post_rdma_read* is only applicable to reliable Connections.

num_segments specifies the number of segments in the *local_segments* vector.

The Implementation allows the buffer segments described by the *local_segments* to overlap but the resulting content of the local buffer is undefined.

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 remote buffer unless there is local overlap. If there is local overlap, the byte order in 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 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.

The completion of the posted RDMA Read is reported asynchronously to the Consumer according to the rules defined in *it_dto_flags_t*. Any generated DTO Completion Event manifests on the EVD associated with the Endpoint. See *it_ep_rc_create*, *it_dto_status_t* and *it_dto_events*. A completion status other than IT_DTO_SUCCESS will break the Connection. 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 *dto_flags* value is used as specified in *it_dto_flags_t*.

The *cookie* allows the Consumer to associate an identifier with each DTO. This identifier is completely under Consumer control and is opaque to the Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted RDMA Read. See *it_dto_cookie_t*.

The Implementation ensures that an LMR Triplet supports byte alignment for Data Transfer Operations.

Data corruption (at local and/or at remote) and data loss will be reported to the Consumer in the DTO Completion Event. These conditions will cause the Connection to be broken. Once the Connection is broken, all outstanding and in-progress operations on the Connection will complete with a failure reported in their corresponding DTO Completion Events.

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 all RDMA Read operations 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.

The Implementation ensures that all data for a given RDMA Read operation is transferred from the remote buffers and into the local buffers before a RDMA Read completion is generated with the status of IT_DTO_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 error.

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.

IT_SUCCESS returned from the *it_post_rdma_read* call means that the RDMA Read operation was successfully posted to the transport layer in use.

RETURN VALUE

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

IT_ERR_TOO_MANY_POSTS	The operation failed due to an overflow of a work queue.
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_NUM_SEGMENTS	The requested number of segments (*num_segments*) was larger than the Endpoint supports.
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

ERRORS

None.

APPLICATION USAGE

This function is used after a Connection has been established to transfer data from a Consumer-specified remote buffer to 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 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*.

SEE ALSO

it_post_send(), *it_post_sendto()*, *it_post_recv()*, *it_post_recvfrom()*, *it_post_rdma_write()*, *it_rmr_bind()*, *it_rmr_unbind()*, *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_rdma_write()

NAME

it_post_rdma_write – post an RDMA Write DTO to a connected 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
);
```

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 for a zero-sized message.

num_segment 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 remote buffer to write to.

rmr_context The RMR Context for the remote buffer to write to.

it_post_rdma_write requests a transfer of all the data from *local_segments* into a remote buffer on the other side of the Connection. The Connection is implemented on a reliable transport. A zero-sized message may be transferred over the Connection.

num_segments specifies the number of segments in the *local_segments* vector. The Completion Event for the *it_post_rdma_write* call indicates to the Consumer that the local buffer is under Consumer control again. It does not guarantee that the contents of the local buffer have been successfully delivered into the memory of the remote Consumer. 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*. Prior to the Completion Event being generated, the content of the remote 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 ownership of the *num_segments* and *local_segments* arguments (but not the local buffer identified by them) when *it_post_rdma_write* returns and is free to use the *num_segments* and *local_segments* arguments for other calls, to modify them, or to destroy them.

The completion of the posted RDMA Write is reported asynchronously to the Consumer according to the rules defined in *it_dto_flags_t*. Any generated DTO Completion Event manifests on the EVD associated with the Endpoint. See *it_ep_rc_create*, *it_dto_status_t* and *it_dto_events*. A completion status other than IT_DTO_SUCCESS will break the Connection. If the reported status of the completion DTO Event corresponding to the posted RDMA Write DTO is not IT_DTO_SUCCESS, then the contents of the remote buffer are not defined.

The *dto_flags* value is used as specified in *it_dto_flags_t.*

The *cookie* allows the Consumer to associate an identifier with each DTO. This identifier is completely under Consumer control and is opaque to the Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted RDMA Write.

The buffer segments described by *local_segments* can overlap.

The Implementation ensures that an LMR Triplet supports byte alignment for Data Transfer Operations.

Data corruption (at local and/or at remote) and data loss will be reported to the Consumer in the DTO Completion Event. These conditions will cause the Connection to be broken. Once the Connection is broken, all outstanding and in-progress operations on the Connection will complete with an error status.

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 all RDMA Write operations start and complete in the order posted.

If the RDMA Write operation exceeds the bounds of the remote buffer, the completion status will be IT_DTO_ERR_REMOTE_ACCESS.

The Implementation ensures that each RDMA Write Data Transfer Operation posted on a Connection prior to a Send Data Transfer Operation posted to the same Connection has its complete data payload delivered to the remote memory prior to the completion of the Receive Data Transfer Operation at the remote side matching that Send.

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 that 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.

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 error.

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.

IT_SUCCESS returned from the *it_post_rdma_write* call means that the RDMA Write operation was successfully posted to the transport layer in use.

EXTENDED DESCRIPTION

See *it_lmr_sync_rdma_write* for a discussion of ramifications of RDMA Write use on non-coherent systems.

RETURN VALUE

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

IT_ERR_TOO_MANY_POSTS	The operation failed due to an overflow of a work queue.
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_NUM_SEGMENTS	The requested number of segments (*num_segments*) was larger than the Endpoint supports.
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.

ERRORS

None.

APPLICATION USAGE

This function is used after Connection has been established to transfer data from Consumer specified local buffer to 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 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 peer to reply to that message. By requiring the remote Consumer to reap the Receive Completion for the Send from the local Consumer, the payload of the RDMA Write is delivered into the remote memory.

SEE ALSO

it_post_send(), *it_post_sendto*(), *it_post_recv*(), *it_post_recvfrom*(), *it_post_rdma_read*(), *it_rmr_bind*(),*it_rmr_unbind*(), *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_recv()

NAME

it_post_recv – post a Receive DTO to a connected Endpoint

SYNOPSIS

```
#include <it_api.h>

it_status_t it_post_recv(
   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
);
```

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 to contain the data to be received. Can be NULL 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 supplies the local Receive buffer specified by *local_segments* and *num_segments* to the *ep_handle* Endpoint. A single incoming message from a single corresponding Send over the Connection is deposited into the local Receive buffer. Zero-sized messages are supported and will consume a posted Receive.

num_segments specifies the number of segments in the *local_segments* vector.

The Implementation allows the buffer segments described by the *local_segments* vector to overlap but the resulting Receive behavior is undefined.

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 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 ownership of the *num_segments* and *local_segments* arguments (but not the local buffer identified by them) when *it_post_recv* returns and is free to use the *num_segments* and *local_segments* arguments for other calls, to modify them, or to destroy them.

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 DTO Completion Event is always generated and manifests on the EVD associated with the Endpoint. See *it_ep_rc_create*, *it_dto_status_t* and *it_dto_events*. A completion status other than IT_DTO_SUCCESS will break the Connection. If the reported *dto_status* of the Completion DTO Event corresponding to the posted Receive DTO is not IT_DTO_SUCCESS, the content of the local buffer is not defined.

The *dto_flags* value is used as specified in *it_dto_flags_t*.

The *cookie* allows the Consumer to associate an identifier with each DTO. This identifier is completely under Consumer control and is opaque to the Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted Receive.

The Implementation ensures that an LMR Triplet supports byte alignment for Data Transfer Operations.

Data corruption (at local and/or at remote) and data loss will be reported to the Consumer in the DTO Completion Event. These conditions will cause the Connection to be broken. Once the Connection is broken, all outstanding and in-progress operations on the Connection will complete with an error status.

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 a RDMA Write DTO from the remote connected Endpoint preceding a Send from the remote connected Endpoint has fully delivered its payload prior to the completion of the Receive corresponding to the Send.

The Implementation ensures that all Receives start and complete in the order posted.

Receive Data Transfer Operations on a Connection are completed in the order of posting of their corresponding Sends at the remote Endpoint. Since Sends start and complete in order, the Recvs complete in order.

There is no order relationship between completions of Receive Data Transfer Operations and all other Data Transfer Operations (including RMR operations) on the same Connection.

The Implementation ensures that all data from a given Send operation is transferred from the remote buffers and into the local buffers before a Receive completion is generated with IT_DTO_SUCCESS.

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.

IT_SUCCESS returned from the *it_post_recv* call means that the Receive operation was successfully posted to the transport layer for use.

RETURN VALUE

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

IT_ERR_TOO_MANY_POSTS	The operation failed due to an overflow of a work queue.

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_DTO_FLAGS	The Data Transfer Operation flags (*dto_flags*) value was invalid.
IT_ERR_INVALID_NUM_SEGMENTS	The requested number of segments (*num_segments*) was larger than the Endpoint supports.
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.

ERRORS

None.

APPLICATION USAGE

This function is used after a Connection has been established to transfer data into a Consumer-specified local buffer from a buffer specified by the corresponding Send 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 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*.

SEE ALSO

it_post_send(), *it_post_sendto*(), *it_post_recvfrom*(), *it_post_rdma_read*(), *it_post_rdma_write*(), *it_rmr_bind*(), *it_rmr_unbind*(), *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_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
);
```

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 supplies the local Receive buffer specified by *local_segments* and *num_segments* to the *ep_handle* datagram Endpoint. A single incoming message from a single corresponding Send from a remote datagram Endpoint is deposited into the local Receive buffer.

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.

num_segments specifies the number of segments in the *local_segments* vector.

The Implementation allows the buffer segments described by the *local_segments* vector to overlap but the resulting Receive behavior is undefined.

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.

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 ownership of the *num_segments* and *local_segments* arguments (but not the local buffer identified by them) when *it_post_recvfrom* returns and is free to use the *num_segments* and *local_segments* arguments for other calls, to modify them, or to destroy them.

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 DTO Completion Event is always generated and manifests on the EVD associated with the Endpoint Receive Queue. See *it_ep_ud_create*, *it_dto_status_t* and *it_dto_events*. If the reported *dto_status* of the Completion DTO Event corresponding to the posted Receive DTO is not IT_DTO_SUCCESS, the content of the local buffer is not defined.

The *dto_flags* value is used as specified in *it_dto_flags_t*.

The *cookie* allows the Consumer to associate an identifier with each DTO. This identifier is completely under Consumer control and is opaque to the Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted Receive.

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, the reported status of the posted Receive DTO in the corresponding Completion DTO Event is IT_DTO_ERR_LOCAL_LENGTH.

The Implementation ensures that an LMR Triplet supports byte alignment for Data Transfer Operations.

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.

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.

IT_SUCCESS returned from the *it_post_recvfrom* call means that the Receive operation was successfully posted to the transport layer for use.

RETURN VALUE

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

IT_ERR_TOO_MANY_POSTS	The operation failed due to an overflow of a work queue.
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_DTO_FLAGS	The Data Transfer Operation flags (*dto_flags*) value was invalid.
IT_ERR_INVALID_NUM_SEGMENTS	The requested number of segments (*num_segments*) was larger than the Endpoint supports.
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.

ERRORS

None.

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 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*.

SEE ALSO

it_post_send(), *it_post_sendto*(), *it_post_recv*(), *it_post_rdma_read*(), *it_post_rdma_write*(), *it_rmr_bind*(), *it_rmr_unbind*(), *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
);
```

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 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 requests a transfer of all the data from *local_segments* into a remote buffer specified by a single corresponding Receive on the other side of the Connection. The Connection is implemented on a reliable transport. A zero-sized message may be transferred over the Connection and will consume a buffer specified by the corresponding Receive.

num_segments specifies the number of segments in the *local_segments* vector.

The Completion Event for the *it_post_send* call indicates to the Consumer that the local buffer is under Consumer control again. It does not guarantee that the contents of the local buffer have been successfully received by the remote Consumer. 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.

Once the local buffer has reached 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. Prior to the Completion Event being generated, the contents of the receiver’s *local_segments* are 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 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.

The completion of the posted Send is reported asynchronously to the Consumer according to the rules defined in *it_dto_flags_t*. Any generated DTO Completion Event manifests on the EVD associated with the Endpoint. See *it_ep_rc_create*, *it_dto_status_t* and *it_dto_events*. A completion status other than IT_DTO_SUCCESS will break the Connection.

The *dto_flags* value is used as specified in *it_dto_flags_t*.

The *cookie* allows the Consumer to associate an identifier with each DTO. This identifier is completely under Consumer control and is opaque to the Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted Send.

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* to overlap.

Data corruption (at local and/or at remote) and data loss will be reported to the Consumer in the DTO Completion Event. These conditions will cause the Connection to be broken. Once the Connection is broken, all outstanding and in-progress operations on the Connection will complete with an error status.

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. If no Receive resources are ever posted at the remote end, then a Send will eventually abort with a completion error and the Connection will be broken. In order to avoid this scenario the remote Consumer should post Receive resources prior to the local Consumer posting the Send.

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.

The Implementation ensures that all data from the given Send operation is transferred from the local buffer and to the remote buffer before a Send completion is generated with the status of IT_DTO_SUCCESS. If the corresponding remote Receive buffer is not sufficient in size for the Send data buffer then the operation will complete with an error status. In order to avoid this scenario the remote Consumer should post a buffer large enough for the incoming Send data.

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.

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 error.

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.

IT_SUCCESS returned from the *it_post_send* call means that the Send operation was successfully posted to the transport layer.

RETURN VALUE

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

IT_ERR_TOO_MANY_POSTS	The operation failed due to an overflow of a work queue.
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_NUM_SEGMENTS	The requested number of segments (*num_segments*) was larger than the Endpoint supports.
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.

ERRORS

None.

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 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*.

SEE ALSO

it_post_sendto(), *it_post_recv*(), *it_post_recvfrom*(), *it_post_rdma_read*(), *it_post_rdma_write*(), *it_rmr_bind*(), *it_rmr_unbind*(), *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
);
```

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 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 requests a transfer of all the data from *local_segments* via the local datagram *ep_handle* into a remote buffer specified by a single corresponding Receive at the remote datagram Endpoint as specified by *remote_ep_addr*. No guarantee of delivery is provided.

num_segments specifies the number of segments in the *local_segments* vector.

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. Prior to the Completion Event being generated, the contents of the receiver's *local_segments* are 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 *num_segments* and *local_segments* arguments (but not the local buffer identified by them) when *it_post_sendto* returns and is free to use the *num_segments* and *local_segments* arguments for other calls, to modify them, or to destroy them.

The completion of the posted Send is reported asynchronously to the Consumer according to the rules defined in *it_dto_flags_t*. Any generated DTO Completion Event manifests on the EVD associated with the Endpoint. See *it_ep_ud_create*, *it_dto_status_t*, and *it_dto_events*.

The *dto_flags* value is used as specified in *it_dto_flags_t*.

The *cookie* allows the Consumer to associate an identifier with each DTO. This identifier is completely under Consumer control and is opaque to the Implementation. The *cookie* is returned to the Consumer in the Completion Event for the posted Send.

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.

The Implementation ensures that an LMR Triplet supports byte alignment for Data Transfer Operations.

The Implementation allows the buffer segments described by *local_segments* to overlap.

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.

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.

IT_SUCCESS returned from the *it_post_sendto* call means that the Send operation was successfully posted to the transport layer.

When *it_post_sendto* completes with IT_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_TOO_MANY_POSTS	The operation failed due to an overflow of a work queue.
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_DTO_FLAGS	The Data Transfer Operation flags (*dto_flags*) value was invalid.
IT_ERR_INVALID_NUM_SEGMENTS	The requested number of segments (*num_segments*) was larger than the Endpoint supports.
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_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.

ERRORS

None.

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 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.

SEE ALSO

it_post_send(), *it_post_recv*(), *it_post_recvfrom*(), *it_post_rdma_read*(), *it_post_rdma_write*(), *it_rmr_bind*(), *it_rmr_unbind*(), *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.

ERRORS

None.

APPLICTION USAGE

An LMR, RMR, Endpoint or Address Handle can not 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, Bind, and Unbind 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 Bind and Unbind 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.

ERRORS

None.

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	Logical 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.

ERRORS

None.

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;
```

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. The delivery of Private Data to the Remote Endpoint is unreliable.
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* manual 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 Bind or Unbind 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.

ERRORS

None.

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. The Consumer should be aware that the delivery of Private Data to the Remote Endpoint is unreliable.

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_bind()

NAME

it_rmr_bind – post operation to Bind a Remote Memory Region to a memory range

SYNOPSIS

```
#include <it_api.h>

it_status_t it_rmr_bind(
   IN                  it_rmr_handle_t    rmr_handle,
   IN                  it_lmr_handle_t    lmr_handle,
   IN                  void               *addr,
   IN                  it_length_t        length,
   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
);
```

DESCRIPTION

rmr_handle	Handle of RMR that will be bound.
lmr_handle	LMR to which RMR will be bound.
addr	Starting address of region to be bound.
length	Length in bytes of region to be bound. Must not be 0.
privs	Logical OR of requested remote access privilege flags for bound region.
ep_handle	Endpoint on which to post the Bind operation.
cookie	Consumer-provided cookie that is returned to the Consumer in the Completion Event corresponding to the RMR Bind operation.
dto_flags	Logical OR of options for operation handling.
rmr_context	Returned Context allowing remote access to the bound region.

The *it_rmr_bind* routine posts to Endpoint *ep_handle* an operation to Bind 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 *ep_handle* should be a Reliable Connected Endpoint; if it is not, an immediate error will be returned. The Protection Zones of the *lmr_handle*, *rmr_handle*, and *ep_handle* must match; if they do not, a completion error will be generated with completion status (*it_dto_status_t*) set to IT_DTO_ERR_LOCAL_PROTECTION. Like DTOs, the Bind operation completes asynchronously, and its completion is reported to the Consumer through a Completion Event based on the specified *dto_flags* value. The Consumer defined *cookie* argument is opaque to the

Implementation and is returned in the Completion Event. A Bind operation will only complete successfully if it is posted to an Endpoint in the IT_EP_STATE_CONNECTED state. 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. A Bind operation may be submitted for an RMR that has never been bound, is currently bound, or has been unbound using *it_rmr_unbind*.

The starting virtual address and length of the region to be bound is specified by *addr* and *length*, respectively. Remote access to the region is enforced with byte level granularity, unlike an LMR. The specified address range must fall within the LMR given by *lmr_handle*.

The type of remote access to be allowed is specified by the *privs* argument as a logical OR of zero or more of the following values:

IT_PRIV_REMOTE_READ	Enable access for RDMA Read operations.
IT_PRIV_REMOTE_WRITE	Enable access for RDMA Write operations.
IT_PRIV_REMOTE	Enable access for both remote RDMA Read and RDMA Write.
IT_PRIV_ALL	Equivalent to IT_PRIV_REMOTE.

Pass 0 or the value IT_PRIV_NONE to disallow remote access to the RMR. The flags IT_PRIV_READ_ONLY and IT_PRIV_DEFAULT are invalid in this Context. It is invalid to request remote write access if the memory access flags for *lmr_handle* include IT_PRIV_READ_ONLY.

Request handling is specified by the *dto_flags* argument and is the logical 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_bind* automatically fences all DTO, Bind, and Unbind operations subsequently submitted on the Endpoint *ep_handle* such that none of these operations starts until the currently posted Bind operation completes.

The value of *rmr_context* is immediately available when *it_rmr_bind* returns, but it may not be used by a remote host for an RDMA operation until the Bind 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 Bind Completion Event, any previous binding for the RMR is invalidated. 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_bind* is called. Completions for such operations should be dequeued prior to calling *it_rmr_bind*.

The new binding remains valid until the next Bind or Unbind operation completes successfully, or until the RMR is destroyed. A Bind 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 binding.

If *it_rmr_bind* returns successfully, but the Bind Completion Event status indicates failure, then the previous binding and RMR Context remains valid. If *ep_handle* is part of a Reliable Connection, then the Connection is broken, the Endpoint transitions into the IT_EP_STATE_NONOPERATIONAL state, and an IT_CM_MSG_CONN_BROKEN_EVENT Event is delivered to the Connect EVD of *ep_handle*.

The Bind Completion Event is defined by *it_dto_cmpl_event_t*. The Event Stream type is IT_RMR_BIND_CMPL_EVENT.

RETURN VALUE

A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below, and the previous binding for the RMR remains valid. It is possible for *it_rmr_bind* to return success but for the Completion Event to indicate failure.

IT_ERR_INVALID_PRIVS	The requested memory privileges (*privs*) contained an invalid flag.
IT_ERR_INVALID_DTO_FLAGS	The Data Transfer Operation flags (*dto_flags*) value was invalid.
IT_ERR_RESOURCES	The requested operation failed due to insufficient resources.
IT_ERR_ADDRESS	The address (*addr*) fell outside the boundaries specified by the Local Memory Region.
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_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_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_EP	The Endpoint Handle (*ep_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.

ERRORS

None.

APPLICATION USAGE

The *it_rmr_bind* 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 bound 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 can be used.

The local Consumer has several options for ensuring that the remote Consumer does not use *rmr_ context* before the Bind 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 Bind the RMR. The barrier-fencing behavior of *it_rmr_bind* ensures that the DTO does not start until the Bind Completion Event has occurred. If the Bind 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 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.

FUTURE DIRECTIONS

Currently the Consumer is allowed to call *it_rmr_bind* on an RMR that is already in the bound state. A future version of the IT-API may require the Consumer on some transports to first Unbind a bound RMR using *it_rmr_unbind* before performing a Bind operation.

SEE ALSO

it_lmr_create(), *it_rmr_unbind()*, *it_rmr_create()*, *it_dto_flags_t*, *it_dto_events*

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,
  OUT                 it_rmr_handle_t  *rmr_handle
);
```

DESCRIPTION

pz_handle	Protection Zone in which the Remote Memory Region will be created.
rmr_handle	Handle of new Remote Memory Region.

The *it_rmr_create* routine creates a Remote Memory Region that may be used as the target for Data Transfer Operations over the Interface Adapter that is implicitly identified by the *pz_handle* argument. The returned RMR must be bound to a Local Memory Region using *it_rmr_bind* before it can be used as a target, however.

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_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.

ERRORS

None.

APPLICATION USAGE

Creating an RMR is a relatively expensive operation. Once created, however, an RMR may be bound repeatedly to different LMR address ranges using the more efficient *it_rmr_bind* call, as long as the Protection Zone of the RMR matches that of the LMR. Binding an RMR is much more efficient than granting and changing remote access privileges using *it_lmr_create* and *it_lmr_modify*.

SEE ALSO

it_rmr_bind(), *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
);
```

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 bound to an LMR, then the RMR binding 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. Note also that the number of possible RMR Context values is finite, and the Implementation will eventually reuse previously freed values in a new binding. If a DTO using an RMR Context is posted after that Context is freed, it is theoretically possible for the Context to be reused before the DTO completes, and for the DTO to complete under the new binding 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.

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.

ERRORS

None.

SEE ALSO

it_rmr_create(), *it_rmr_query*()

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_BOUND       = 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_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     bound;  /* IT_RMR_PARAM_BOUND */
  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_param_t;
```

DESCRIPTION

rmr_handle	Remote Memory Region
mask	logical 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 bound to an LMR as a result of using *it_rmr_bind*:

ia — The Interface Adapter Handle specified to create the RMR.

pz — The Protection Zone Handle specified to create the RMR.

bound — IT_TRUE if the RMR is currently bound, IT_FALSE otherwise.

lmr — The Local Memory Region to which the RMR is currently bound, or undefined if RMR is not bound.

addr — The currently bound starting address of the RMR, or undefined if not bound.

length — The currently bound length in bytes of the RMR, or undefined if not bound.

privs — The currently bound memory access privileges of the RMR, or undefined if not bound.

rmr_context — The currently bound RMR Context associated with the RMR, or undefined if not bound. Returned in network byte order.

If the Consumer calls *it_rmr_query* after posting a Bind or Unbind operation, and before dequeueing the Completion Event of such an operation, then the returned *bound, lmr, addr, length, privs,* and *rmr_context* 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_RMR — The Remote Memory Region Handle (*rmr_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.

ERRORS

None.

SEE ALSO

it_rmr_create(), *it_rmr_free()*, *it_rmr_bind()*, *it_rmr_context_t*

it_rmr_unbind()

NAME

it_rmr_unbind – post operation to Unbind a Remote Memory Region from its memory range

SYNOPSIS

```
#include <it_api.h>

it_status_t it_rmr_unbind(
   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
);
```

DESCRIPTION

rmr_handle	Handle of RMR that will be unbound.
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	Logical OR of options for operation handling.

The *it_rmr_unbind* routine posts to Endpoint *ep_handle* an Unbind operation to Unbind the Remote Memory Region *rmr_handle*. The *ep_handle* should be a Reliable Connected Endpoint; if it is not, an immediate error will be returned. The Protection Zones of the *rmr_handle* and *ep_handle* must match; if they do not, a completion error will be generated with completion status (*it_dto_status_t*) set to IT_DTO_ERR_LOCAL_PROTECTION. The operation completes asynchronously, and its completion is reported to the Consumer through a Completion Event based on the specified *dto_flags* value. The Consumer defined *cookie* argument is opaque to the Implementation and is returned in the Completion Event. An Unbind operation will only complete successfully if it is posted to an Endpoint in the IT_EP_STATE_CONNECTED state. 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.

Request handling is specified by the *dto_flags* argument as the logical 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_unbind* automatically fences all DTO, Bind, and Unbind operations subsequently submitted on the Endpoint *ep_handle* such that none of these operations starts until the currently posted Unbind operation completes.

After a successful Unbind Completion Event, any previous binding 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_unbind* is called. The Consumer must ensure that such operations have completed prior to calling *it_rmr_unbind* if successful completions are desired. An Unbind 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 binding.

If *it_rmr_unbind* returns successfully but the Completion Event status indicates failure, then the previous binding and RMR Context remains valid. If *ep_handle* is part of a Reliable Connection, then the Connection is broken, the Endpoint transitions into the IT_EP_STATE_NONOPERATIONAL state, and an IT_CM_MSG_CONN_BROKEN_EVENT Event is delivered to the Connect EVD of *ep_handle*.

The Unbind operation generates an *it_dto_compl_event_t* Completion Event. The Event Stream type is IT_RMR_BIND_CMPL_EVENT.

RETURN VALUE

A successful call returns IT_SUCCESS. Otherwise, an error code is returned as described below, and the previous binding for the RMR remains valid. It is possible for *it_rmr_unbind* to return success but for the Completion Event to indicate failure.

IT_ERR_INVALID_DTO_FLAGS	The Data Transfer Operation flags (*dto_flags*) value was invalid.
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_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_EP	The Endpoint Handle (*ep_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.

ERRORS

None.

APPLICATION USAGE

it_rmr_unbind may be used to revoke remote Consumer access to an RMR that was previously granted. In addition, the Consumer must Unbind all RMRs that refer to an LMR in order to destroy or modify the LMR. Note that the RMR is not considered unbound 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 Bind the RMR has become disconnected, because an Unbind operation can only be posted to a connected Endpoint. One solution is for the Consumer to create a special pair of Endpoints to be used in this situation that are connected in loopback mode to each other, created using the same Protection Zone as the RMR. Another solution is to destroy the RMR by calling *it_rmr_free.*

For reasons already described, the Unbind 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 Unbind operation may be inferred by other means. For example, completion of a subsequently posted operation of any type indicates that the Unbind operation has completed successfully. If the Unbind operation fails, a Completion Event is generated regardless.

SEE ALSO

it_rmr_create(), *it_rmr_bind*(), *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 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 experiences 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.

ERRORS

None.

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_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;
```

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. The delivery of Private Data to the Remote Endpoint is unreliable.
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* can not be reused.

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 can not 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 if 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 can not 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.

ERRORS

None.

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
);
```

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.

ERRORS

None.

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
);
```

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. The delivery of Private Data to the Remote Endpoint is unreliable.
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 if 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_identifer_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.

ERRORS

None.

APPLICATION USAGE

The resulting *ud_svc_handle* (*it_ud_service_request_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 the 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
);
```

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.

ERRORS

None.

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_LEN */
} it_ud_svc_req_param_t;
```

DESCRIPTION

ud_svc_handle	Unreliable Datagram (UD) Service Request Handle previously created by a call to *it_ud_service_request_handle_create*.
mask	Logical 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.

ERRORS

None.

SEE ALSO

it_ud_service_request_handle_create(), *it_ud_service_request_handle_free*(), *it_ud_service_request*()

5 Data Type Manual Pages

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 Bind & Unbind

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_lmr_triplet_t – structure describing a DTO buffer in a Local Memory Region

it_net_addr_t – encapsulates all supported Network Address types

it_path_t – describes the Path between a pair of Spigots

it_software_event_t – Software Event type

it_unaffiliated_event_t – Unaffiliated Asynchronous Event type

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. 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.

RETURN VALUE

None.

ERRORS

None.

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;
   } u;
} it_affiliated_event_t;
```

DESCRIPTION

event_number	Identifier of the Event type. Valid values: IT_ASYNC_AFF_SEVD_ENQUEUE_FAILURE, IT_ASYNC_AFF_EP_FAILURE, IT_ASYNC_AFF_EP_BAD_TRANSPORT_OPCODE, IT_ASYNC_AFF_EP_REQ_DROPPED, IT_ASYNC_AFF_EP_RDMAW_ACCESS_VIOLATION, IT_ASYNC_AFF_EP_RDMAW_CORRUPT_DATA, IT_ASYNC_AFF_EP_RDMAR_ACCESS_VIOLATION
evd	Handle for the Event Dispatcher where the Event was queued.
sevd	The Handle for the SEVD that the Implementation failed to enqueue an Event for. 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.

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 used *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_LOCAL_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_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.

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 Volume 1, Chapter 11 of the Infiniband specification.

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_LOCAL_ACCESS_VIOLATION	Local Access Violation 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.)

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" man page in the Appendix of the VIA specification.

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_LOCAL_ACCESS_VIOLATION	(Not applicable to the VIA transport.)
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

RETURN VALUE

None.

ERRORS

None.

SEE ALSO

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

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).

RETURN VALUE

None.

ERRORS

None.

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_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_inflight_incoming;
   uint32_t                      rdma_read_inflight_outgoing;
   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. 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: IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT, IT_CM_MSG_CONN_ESTABLISHED_EVENT, 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
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_inflight_incoming	Maximum number of incoming simultaneous RDMA Read operations supported. Only valid if the *it_ia_info.ird_support* value is IT_TRUE and as described under Application Usage below.
rdma_read_inflight_outgoing	Maximum number of outgoing simultaneous RDMA Read operations supported. Only valid if the *it_ia_info.ord_support* value is IT_TRUE and as described under Application Usage below.
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 **Error! Reference source not found.**:

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 both two-way and three-way Connection establishment. 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 both two-way and three-way Connection establishment. 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 both two-way and three-way Connection establishment. 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 the remote side not establishing a Connection that are not related to the remote Consumer issuing *it_reject*. Such reasons include overflow of the remote EVD for Connection Events, timeouts of the Connection attempt, and the passive side rejecting the proposed Path for the Connection attempt.	Applies to both two-way and three-way Connection establishment. The Endpoint is in IT_EP_ STATE_NONOPERATIONAL state. All preposted DTOs and RMRs are flushed.
IT_CM_MSG_CONN_BROKEN_EVENT	The Connection identified has been disconnected by the Implementation. Causes include transport errors.	Applies to both two-way and three-way Connection establishment. The Endpoint is in IT_EP_ STATE_NONOPERATIONAL state. All posted DTOs and RMRs are flushed.

Table 3: Connection Management Event Definitions

Error! Reference source not found. identifies which fields are valid in each of the Connection management 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*, *private_data*, *private_data_ present*.*cn_est_id* may not be valid if Consumer called *it_ep_disconnect* of the associated Endpoint at any time before the *cn_est_id* is used. The following are valid only on active side when three-way Connection establishment is used: *rdma_read_inflight_incoming*, *rdma_read_inflight_outgoing*.
IT_CM_MSG_CONN_ESTABLISHED_EVENT	*ep*, *private_data*, *private_data_present*
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* If the *reject_reason_code* is IT_CN_REJ_ BAD_PATH, then *dst_path* is also valid.
IT_CM_MSG_CONN_BROKEN_EVENT	*ep*

Table 4: Event Management Event Fields

Error! Reference source not found. describes the meaning of the various reject_reason_code values that can be 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.
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.

reject_reason_code value	Description
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_inflight_incoming* limit in the passive-side Endpoint.
IT_CN_REJ_RESOURCES	The remote peer was unable to allocate resources necessary to establish the Connection.

Table 5: reject_reason_code Descriptions

The UD Service Resolution Reply Event is described in **Error! Reference source not found.**:

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 **Error! Reference source not found.**:

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, **Error! Reference source not found.** 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
IT_CN_REJ_RESOURCES	1, 3

Table 8: InfiniBand reject_reason_code Mapping

For the VIA transport, **Error! Reference source not found.** maps the values in the reject_reason_code field to their corresponding return values from the man pages for "VipConnectRequest" and "VipConnectAccept" in the Appendix of the VIA specification. Return values that are not listed in the table below are manifest to Consumers of the IT-API through a mechanism other than a IT_CM_MSG_CONN_NONPEER_REJECT_EVENT.

reject_reason_code value	**VIA Return Code**
IT_CN_REJ_OTHER	VipConnectAccept – VIP_INVALID_RELIABILITY_LEVEL, VIP_INVALID_QOS, VIP_TIMEOUT, VIP_ERROR_RESOURCE VipConnectRequest – VIP_NO_MATCH
IT_CN_REJ_TIMEOUT	VipConnectAccept – VIP_NOT_REACHABLE VipConnectRequest – VIP_TIMEOUT, VIP_NOT_REACHABLE

reject_reason_code value	VIA Return Code
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

Table 9: VIA reject_reason_code Mapping

For the Infiniband transport, **Error! Reference source not found.** 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 10: ep_info Element Mapping

RETURN VALUE

None.

ERRORS

None.

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_inflight_incoming* and *rdma_read_inflight_outgoing* 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.

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 the Event was dequeued before or after the call was made. If the Consumer does use the *cn_est_id* then the Implementation generate 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* man 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.

FUTURE DIRECTIONS

When an Endpoint gets connected, the Endpoint moves to the IT_EP_STATE_CONNECTED state and an IT_CM_MSG_CONN_ESTABLISHED_EVENT Event is generated on the Connection Event Stream for that Endpoint. An Active-side Endpoint can get connected without any DTO needing to be processed by either of the Endpoints in the Connection. A Passive-side Endpoint can also usually (but not always) get connected without any DTO needing to be processed. (A Passive-side Endpoint can always get connected if a Receive DTO posted to the Passive-side Endpoint completes.) A future version of the API may not allow a Passive-side Endpoint to get connected unless a Receive DTO posted to the Passive-side Endpoint first completes.

Currently when *it_reject* is called by the remote side during a Connection establishment attempt an IT_CM_MSG_CONN_PEER_REJECT_EVENT Event is generated on the local side to let the local Consumer know that the attempt was rejected. A future version of the API may on some transports generate an IT_CM_MSG_CONN_NONPEER_REJECT_EVENT Event instead of an IT_CM_MSG_CONN_PEER_REJECT_EVENT in this circumstance.

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_inflight_incoming;
   uint32_t                    rdma_read_inflight_outgoing;
   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_req_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 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_inflight_incoming	Maximum number of incoming simultaneous RDMA Read operations of requesting remote EP supported. Only valid if *it_ia_info.ird_support* is IT_TRUE.
rdma_read_inflight_outgoing	Maximum number of outgoing simultaneous RDMA Read operations of requesting remote EP supported. Only valid if *it_ia_info.ord_support* is IT_TRUE.
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.

Event type	Description
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 11: 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.

RETURN VALUE

None.

ERRORS

None.

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 = 0x1,

   /* InfiniBand Service ID, as described in section 12.7.3 of
      Volume 1 of the InfiniBand specification. */
   IT_IB_SERVICEID = 0x2,

   /* VIA Connection Discriminator */
   IT_VIA_DISCRIMINATOR = 0x4

} 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;

/* 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;

        } 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.)

RETURN VALUE

None.

ERRORS

None.

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.

RETURN VALUE

None.

ERRORS

None.

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;
```

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.

RETURN VALUE

None.

ERRORS

None.

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.

RETURN VALUE

None.

ERRORS

None.

SEE ALSO

it_dto_events, *it_dto_flags_t*, *it_post_send*(), *it_post_sendto*(), *it_post_recv*(), *it_post_recvfrom*(), *it_post_rdma_read*(), *it_post_rdma_write*(), *it_rmr_bind*(), *it_rmr_unbind*()

it_dto_events

NAME

DTO and RMR Bind/Unbind Completion Event types

SYNOPSIS

```
#include <it_api.h>

typedef enum {
   IB_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_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_dto_ud_flags_t       flags;
         it_ud_ep_id_t           ud_ep_id;
         it_path_t               src_path;
} it_all_dto_cmpl_event_t;
```

DESCRIPTION

The DTO Completion Event Stream, IT_DTO_EVENT_STREAM, generates Events for all Data Transfer Operations completions as well as RMR Bind and Unbind 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 Binds and Unbinds, 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: IT_DTO_SEND_CMPL_EVENT, IT_DTO_RC_RECV_CMPL_EVENT, IT_DTO_RDMA_WRITE_CMPL_EVENT, IT_DTO_RDMA_READ_CMPL_EVENT, IT_RMR_BIND_CMPL_EVENT
evd	Handle for the Event Dispatcher where the Event was queued.
ep	Handle for the Endpoint on which the DTO was posted.
cookie	Cookie that the Consumer associated with the DTO at the post time. See *it_dto_cookie_t* for details.
dto_status	Status of completed DTO.
transferred_length	Length of transferred message.

See *it_dto_status_t* for values and definition of *dto_status*.

The transferred_length field indicates the amount of data transferred in Receive operations. The content of this field is undefined for Send, RDMA Read, RDMA Write, RMR Bind, and RMR Unbind 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 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 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 the Infiniband Architecture Release 1.1 specification.

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. 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 can not 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.

RETURN VALUE

None.

ERRORS

None.

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.

SEE ALSO

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_dto_flags_t

NAME

it_dto_flags_t – DTO flags for Send, Receive, RDMA Read, RDMA Write, RMR Bind and RMR Unbind operations

SYNOPSIS

```
#include <it_api.h>

typedef enum
{
   /* If flag set, completion generates a local event */
         IT_COMPLETION_FLAG                  = 0x01,

   /* If flag set, completion cause 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, DTO processing will not start if
      previously posted RDMA Reads are not complete. */
         IT_BARRIER_FENCE_FLAG               = 0x08,
} it_dto_flags_t;
```

DESCRIPTION

it_dto_flags Flags for posted DTOs: Send, Receive, RDMA Read, RDMA Write, RMR Bind and RMR Unbind.

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 DTO, else do not generate a Completion Event unless there is an error. If there is an error, the Completion Event will be generated with Notification regardless of *IT_NOTIFY_FLAG* value.

If not set, then the completion of a subsequent DTO on the same work queue of the same Endpoint with this flag set or with error completion will indicate the successful completion of prior DTO(s) with this flag cleared.

Restrictions

IT_COMPLETION_FLAG may be set or cleared only for Send, RDMA Write, RDMA Read, RMR Bind and RMR Unbind 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 DTO operations on all Service Types (Reliable Connection and Unreliable Datagram). Posting a Receive DTO operation with *IT_COMPLETION_FLAG* cleared is an error.

IT_NOTIFY_FLAG

If set, generate Notification of completion of the DTO/RMR.

Restrictions

IT_ NOTIFY_FLAG may be set or cleared on all DTO and RMR operations 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 DTO if the matching received Send DTO had been posted at the remote with the *IT_SOLICITED_WAIT_FLAG* set regardless of the *IT_NOTIFY_FLAG* of the posted Receive DTO.

IT_SOLICITED_WAIT_FLAG

If set, the Send DTO operation 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_SOLICTED_WAIT_FLAG* on other operations.

If set, requests Notification of completion of the matching remote Receive DTO regardless of the value of the *IT_NOTIFY_FLAG* on the Receive DTO.

IT_BARRIER_FENCE_FLAG

If set, then the DTO/RMR operation will not be started until all previously posted RDMA Read requests to the Endpoint have been 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 DTO on UD service.

IT_BARRIER_FENCE_FLAG must be cleared for all Receive DTO operations on any Service Types. Posting a Receive DTO operation with *IT_BARRIER_FENCE_FLAG* set is an error.

EXTENDED DESCRIPTION

The following table lists all DTO and RMR operations and details the legal *it_dto_flags* values on each.

DTO or RMR operation	Legal it_dto_flags combinations
it_post_send	All possible combinations subject to the constraint that IT_NOTIFY_FLAG may only be set if IT_COMPLETION_FLAG is also set. All flags cleared is a legal value.
it_post_sendto	IT_BARRIER_FENCE_FLAG may not be used. All other possible combinations of the remaining flags are legal subject to the constraint that IT_NOTIFY_FLAG may only be set if IT_COMPLETION_FLAG is also set. All flags cleared is a legal value.
it_post_recv	IT_COMPLETION_FLAG must be specified. IT_BARRIER_FENCE_FLAG may not be used. IT_SOLICITED_WAIT_FLAG may not be used. All other possible combinations of the remaining flags are legal.
it_post_recvfrom	IT_COMPLETION_FLAG must be specified. IT_BARRIER_FENCE_FLAG may not be used. IT_SOLICITED_WAIT_FLAG may not be used. All other possible combinations of the remaining flags are legal.
it_post_rdma_read	IT_SOLICITED_WAIT_FLAG 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_rdma_write	IT_SOLICITED_WAIT_FLAG 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_rmr_bind	IT_SOLICITED_WAIT_FLAG 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_rmr_unbind	IT_SOLICITED_WAIT_FLAG 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.

The following table lists the DTOs on each Service Type on which each flag value is supported.

it_dto_flags_t Value	Supported DTO on RC	Supported DTO on UD
IT_COMPLETION_FLAG	Send, RDMA Read, RDMA Write, RMR Bind, RMR Unbind	Recvfrom
IT_NOTIFY_FLAG	Send, Recv, RDMA Read, RDMA Write, RMR Bind, RMR Unbind	Sendto, Recvfrom

IT_SOLICITED_WAIT_FLAG	Send	Sendto
IT_BARRIER_FENCE_FLAG	Send, RDMA Read, RDMA Write, RMR Bind, RMR Unbind	N/A

As stated in the DESCRIPTION section, *IT_COMPLETION_FLAG* must be set on Recv and Recvfrom DTOs.

As stated in the DESCRIPTION section, *IT_BARRIER_FENCE_FLAG* must be cleared on Recv DTO for RC and must be cleared on Sendto DTO as well as cleared on Recvfrom DTO 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 Volume 1 of the Infiniband specification.

it_dto_flags_t Value	**IB Concept**
IT_COMPLETION_FLAG	May be implemented using Unsignalled Completions concept
IT_NOTIFY_FLAG	None but can be supported by Implementation.
IT_SOLICITED_WAIT_FLAG	Solicited Event
IT_BARRIER_FENCE_FLAG	Fence Indicator

For the VIA transport, the following table maps the values in the *it_dto_flags_t* enumeration to their corresponding concepts as documented in the VIA specification.

it_dto_flags_t Value	**VIA Concept**
IT_COMPLETION_FLAG	May be implemented by associating completion queues with work queues
IT_NOTIFY_FLAG	VipSendNotify, VipRecvNotify, VipCQNotify
IT_SOLICITED_WAIT_FLAG	Not applicable
IT_BARRIER_FENCE_FLAG	Queue Fence Bit

RETURN VALUE

None.

ERRORS

None.

APPLICATION USAGE

See Application Usage in *it_ep_state_t* for discussion of flushing DTO completions when the DTOs have IT_COMPLETION_FLAG cleared.

When posting Send DTOs with Completion Suppression (IT_COMPLETION_FLAG cleared) to an Endpoint, the Consumer is advised to enqueue at least one DTO 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_send(), *it_post_sendto*(), *it_post_recv*(), *it_post_recvfrom*(), *it_post_rdma_read*(), *it_post_rdma_write*(), *it_rmr_bind*(), *it_rmr_unbind*(), *it_dto_status_t*, *it_dto_events*, *it_ep_state_t*

it_dto_status_t

NAME

it_dto_status_t – definition of DTO and RMR 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_status_t;
```

DESCRIPTION

Any successfully initiated Data Transfer Operation (i.e. Send, Receive, RDMA Read, or RDMA Write) or RMR operation (i.e. RMR Bind or RMR Unbind) can return its completion status asynchronously via an Event enqueued on an SEVD. For some DTOs, 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 RC Operations**	**Applicable UD Operations**
IT_DTO_SUCCESS	The DTO completed successfully.	Send Recv RDMA Read RDMA Write RMR Bind RMR Unbind	Sendto Recvfrom
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 RDMA Read RDMA Write	Sendto
IT_DTO_ERR_LOCAL_LENGTH	The total length of the buffers associated with a Receive DTO was too small to hold all the incoming data from a Send DTO	Recv	Recvfrom
IT_DTO_ERR_LOCAL_EP	An internal local Endpoint consistency error was detected while processing a DTO.	Send Recv RDMA Read RDMA Write RMR Bind RMR Unbind	Sendto Recvfrom

it_dto_ status_t Value	**Description**	**Applicable RC Operations**	**Applicable UD Operations**
IT_DTO_ERR_LOCAL_PROTECTION	One of the segments in the DTO caused a protection violation when the DTO was processed. Possible causes for this error include the LMR in the segment wasn't valid, the range specified by the *addr* and *length* in the segment was outside the bounds of the LMR, the Protection Zone associated with the LMR didn't match the Protection Zone of the Endpoint that the DTO was posted to, or an attempt was made to access the LMR in a way that conflicted with its access permissions.	Send Recv RDMA Read RDMA Write	Sendto Recvfrom
IT_DTO_ERR_FLUSHED	The Endpoint entered the IT_EP_STATE_NONOPERATIONAL state before processing of the DTO could begin.	Send Recv RDMA Read RDMA Write RMR Bind RMR Unbind	Sendto Recvfrom

it_dto_ status_t Value	**Description**	**Applicable RC Operations**	**Applicable UD Operations**
IT_ RMR_OPERATION_FAILED	An RMR operation failed due to a protection violation. Possible causes for this error include LMR specified in the *it_rmr_bind* (or *it_rmr_unbind*) call was invalid, the range specified by the address and length in the call was outside the bounds of the LMR, the Protection Zones associated with the LMR, RMR and Endpoint to which the RMR operation was posted didn't match, or an attempt was made to grant access through the RMR that conflicted with the access allowed by either the LMR or the Endpoint.	RMR Bind RMR Unbind	N/A
IT_DTO_ERR_BAD_RESPONSE	The DTO operation that was posted to the Request Queue was responded to with an unexpected transport opcode	Send RDMA Read RDMA Write	N/A

it_dto_ status_t Value	Description	Applicable RC Operations	Applicable UD Operations
IT_DTO_ERR_REMOTE_ACCESS	A protection violation was detected at the remote end when processing an RDMA DTO operation. Possible causes include a Protection Zone mismatch between the RMR and the Endpoint that is responding to the RDMA DTO operation, an attempt being made to do an RDMA Read or Write using an RMR that doesn’t have those permissions enabled, or an attempt being made to do an RDMA Read or Write when the responding Endpoint doesn’t have those permissions enabled.	RDMA Read RDMA Write	N/A
IT_DTO_ERR_REMOTE_RESPONDER	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 error to be returned.	Send RDMA Read RDMA Write	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, or the network fabric being used by the IA is broken.	Send Receive RDMA Read RDMA Write	N/A

it_dto_ status_t Value	Description	Applicable RC Operations	Applicable UD Operations
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 RDMA Read RDMA Write	N/A
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

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 Volume 1, Chapter 11 of the Infiniband specification.

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.)

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 the VIA specification.

it_dto_status_t Value	VIA "Status Bit" Name
IT_DTO_SUCCESS	Done

it_dto_status_t Value	**VIA “Status Bit” Name**
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 Bind or RMR Unbind 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

RETURN VALUE

None.

ERRORS

None.

SEE ALSO

it_post_send(), *it_post_sendto()*, *it_post_recv()*, *it_post_recvfrom()*, *it_post_rdma_read()*, *it_post_rdma_write()*, *it_rmr_bind()*, *it_rmr_unbind()*

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_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_inflight_incoming;
                     /* IT_EP_PARAM_MAX_IRD */
```

```
        uint32_t   rdma_read_inflight_outgoing;
                        /* IT_EP_PARAM_MAX_ORD */
    } it_rc_only_attributes_t;

    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_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* man 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. 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.

Attribute	**Description**	**Service Type**	**Modifiable?**
max_request_dtos	Maximum number of outstanding Send, Sendto, RDMA Read, RDMA Write DTOs, RMR Bind and RMR Unbind 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 *resizable_work_queue* 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.	UD and RC	Subject to the setting of the *resizable_work_queue* 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.
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.	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	In any state.
rdma_write_enable	Flag allowing Consumer to enable or disable incoming RDMA Write operations on this Endpoint.	RC only	In any state.

Attribute	Description	Service Type	Modifiable?
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.
rdma_read_inflight_incoming	Maximum number of incoming RDMA Reads from the remote side of the connected Endpoint that can be outstanding simultaneously.	RC only	When Endpoint is in the IT_EP_STATE_UNCONNECTED or IT_EP_STATE_ACTIVE2_CONNECTION_PENDING state.
rdma_read_inflight_outgoing	Maximum number of outgoing RDMA Reads of the connected Endpoint that can be outstanding simultaneously.	RC only	When Endpoint is in the IT_EP_STATE_UNCONNECTED or IT_EP_STATE_ACTIVE2_CONNECTION_PENDING state.
ep_state	The current state of the Endpoint.	UD and RC	Never – this is a READ-ONLY attribute.

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_inflight_outgoing* is not an error and will have no adverse effects.

RETURN VALUE

None.

ERRORS

None.

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_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. 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_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_ep_rc_create it is in this state.	
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.	*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_CONNECTED	The Endpoint is Connected and is ready for all types of Data Transfer and Bind Operations.	*it_ep_disconnect*, *it_ep_free*, *it_ep_modify*, *it_ep_query*, *it_get_consumer_context*, *it_get_handle_type*, *it_post_rdma_read*, *it_post_rdma_write*, *it_post_recv*, *it_post_send*, *it_rmr_bind*, *it_rmr_unbind*, *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	*it_ep_disconnect*, *it_ep_free*, *it_ep_modify*, *it_ep_query*, *it_ep_reset*, *it_get_consumer_context*,

	subsequently posted in this state will complete with Flushed or error Status.	*it_get_handle_type*, *it_post_rdma_read*, *it_post_rdma_write*, *it_post_recv*, *it_post_send*, *it_rmr_bind*, *it_rmr_unbind*, *it_set_consumer_context*

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
	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 two-way 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_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* for three-way Connection Establishment. In this state only Receive Data Transfer Operations can be posted. 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 Bind 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_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

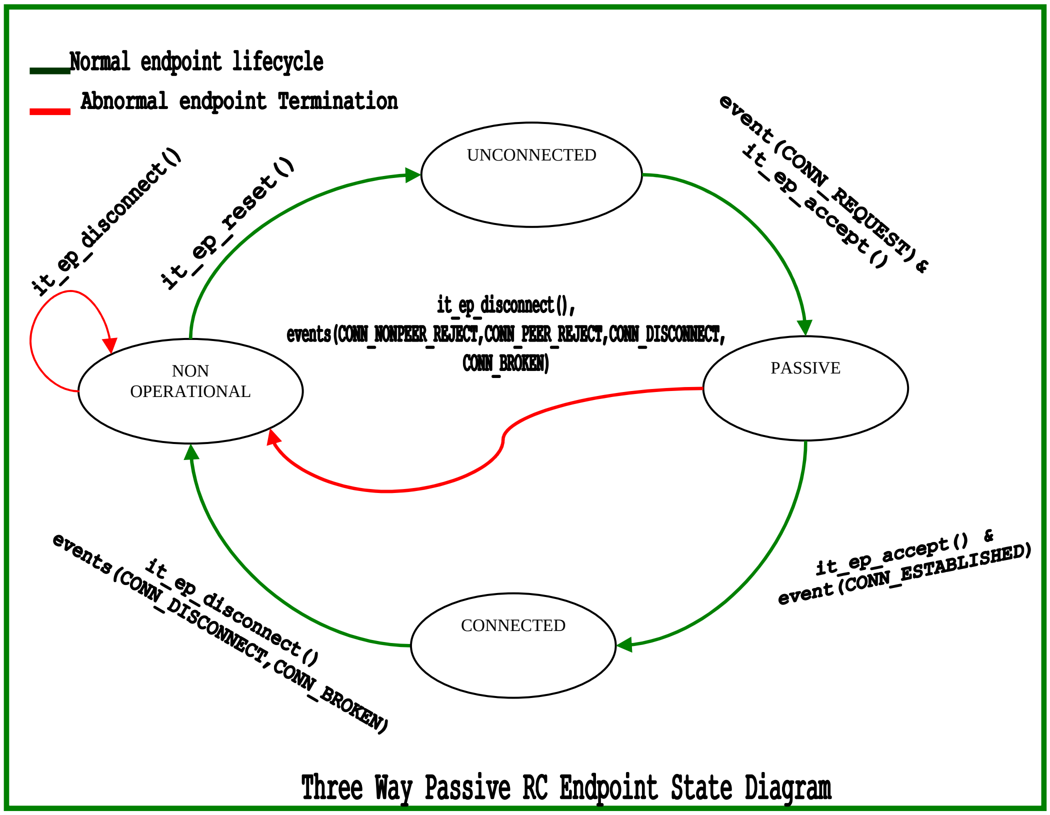

Figure 1 : Three Way Passive RC Endpoint State Diagram

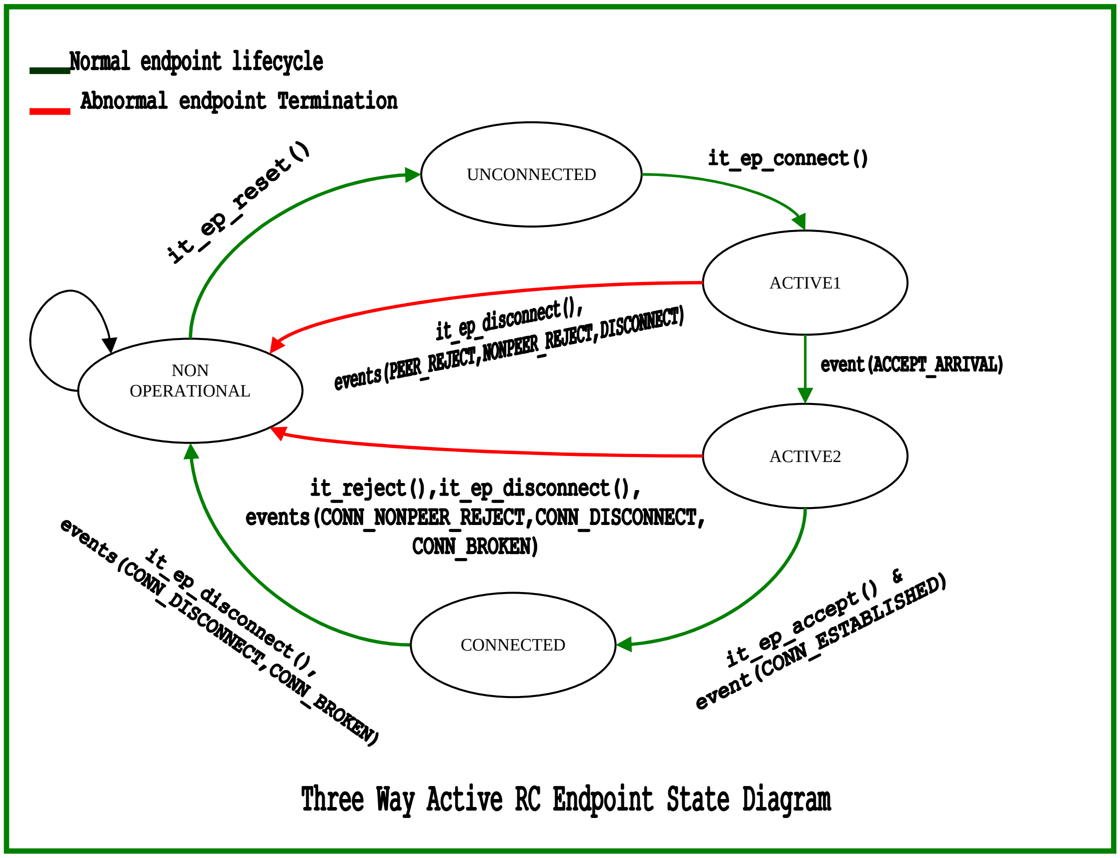

Figure 2 : Three Way Active RC Endpoint State Diagram

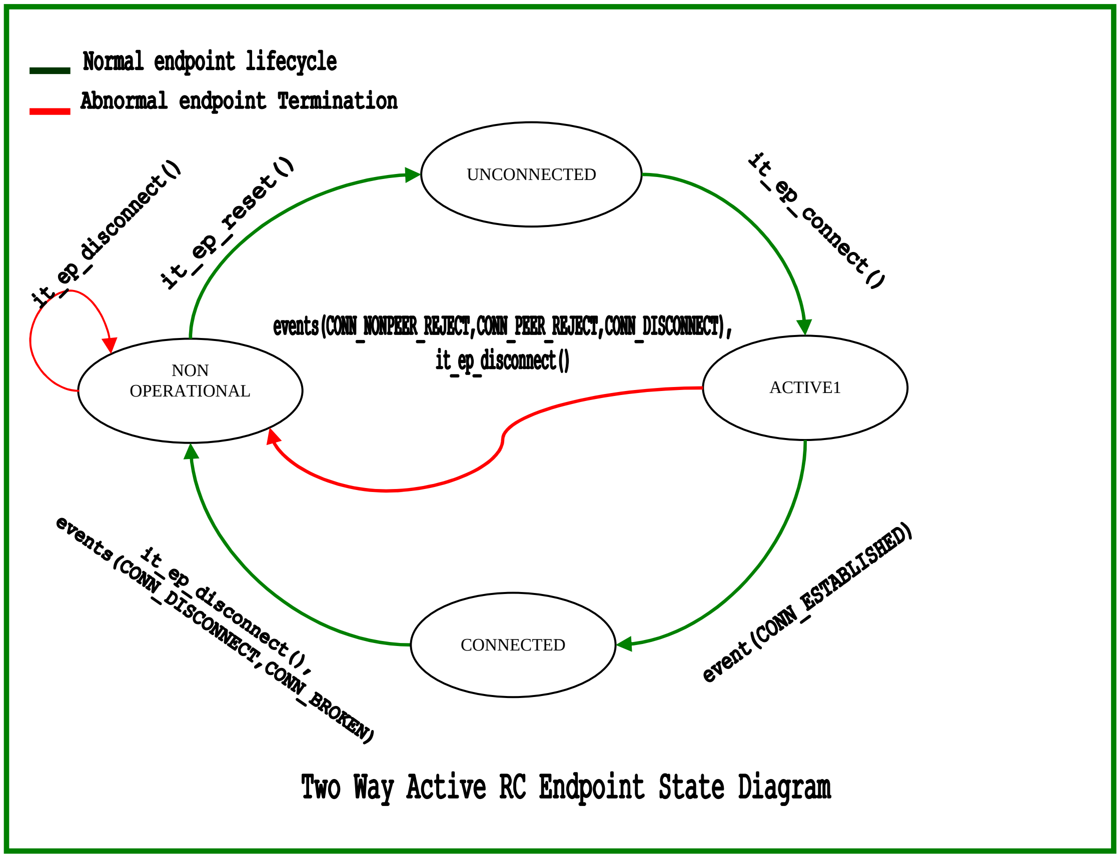

Figure 3 : Two Way Active RC Endpoint State Diagram

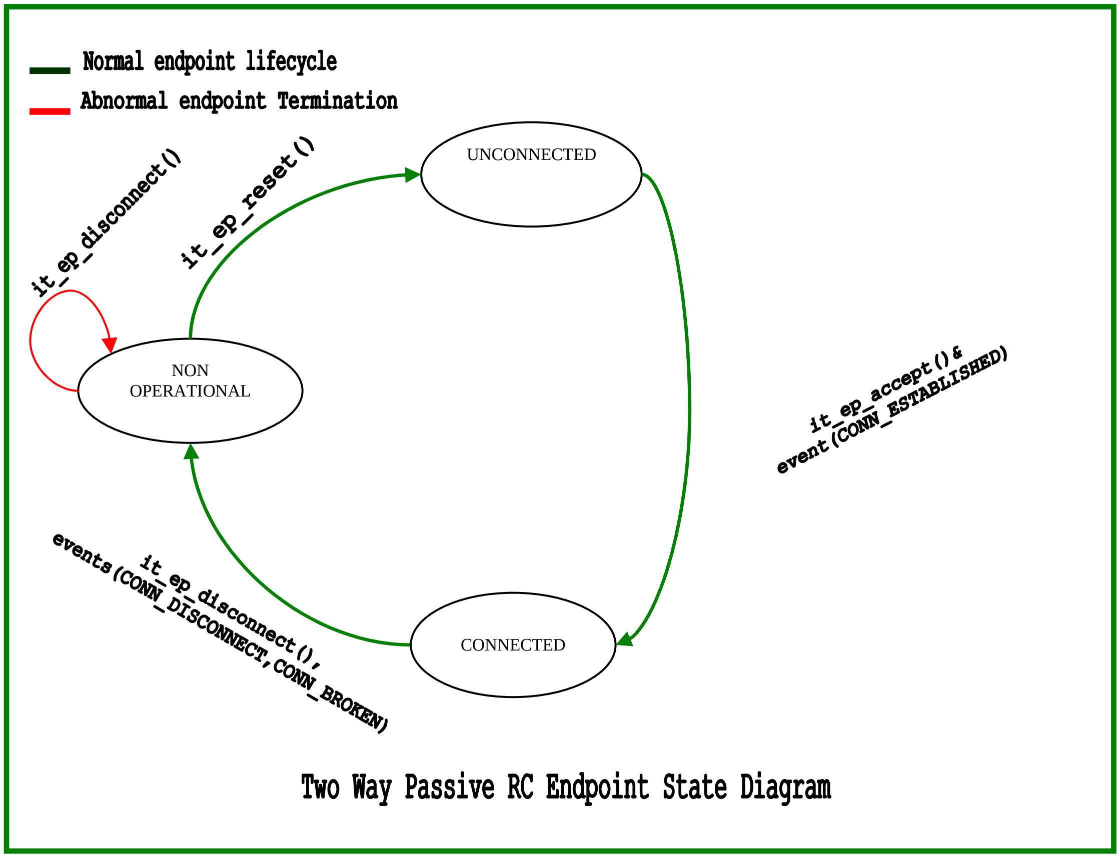

Figure 4 : Two Way Passive RC Endpoint State Diagram

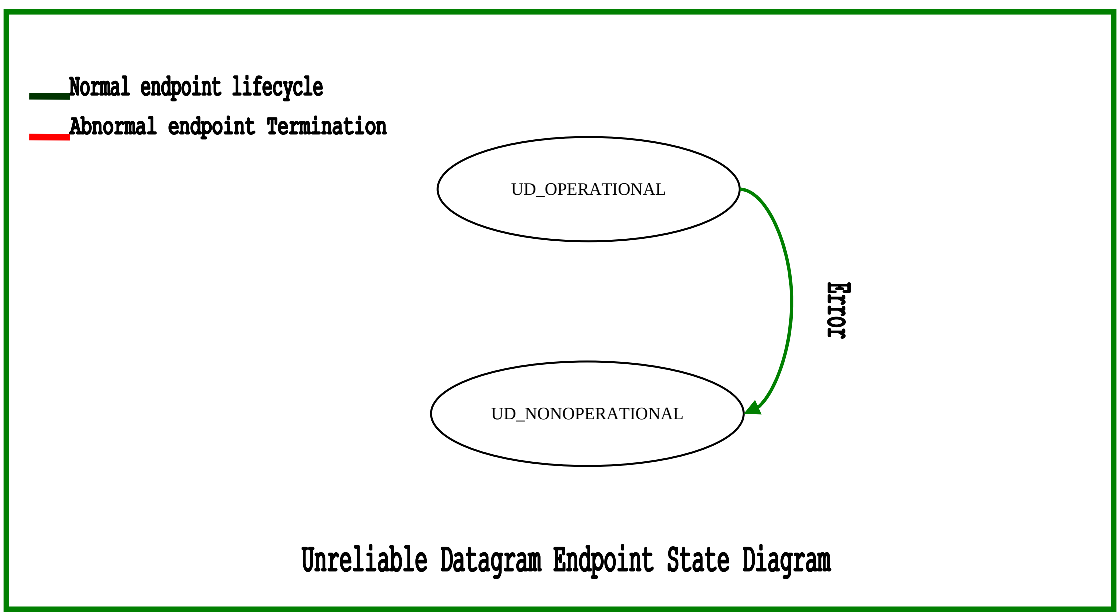

Figure 5 : Unreliable Datagram Endpoint State Diagram

RETURN VALUE

None.

ERRORS

None.

APPLICATION USAGE

1. If the Consumer cares about the Completion Status of posted Data Transfer or Bind Operations after an Endpoint transitions into the IT_EP_STATE_NONOPERATIONAL state then the Consumer can Post Send and Receive DTOs to the Endpoint to serve as markers in the Endpoint associated EVD. The API guarantees that any posting in IT_EP_STATE_NONOPERATIONAL state will be immediately flushed to the EVDs. The Consumer can reap Completions from the associated EVDs until Completions for the marker DTOs are returned. This way the Consumer can be guaranteed that all Completions associated with the Endpoint have been reaped.
2. The Consumer is responsible for coordinating the use of functions that free a Connection Establishment Identifier (cn_est_id) such as *it_ep_accept*, *it_reject*, *it_ep_disconnect* and *it_handoff*. The behavior of functions that are passed as invalid Connection Establishment Identifier is indeterminate.
3. The Consumer should be aware that the delivery of Private Data to the Remote Endpoint specified in calls to *it_ep_accept*, *it_reject* and *it_ep_disconnect* is unreliable and should be used accordingly.

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
{
    /* DTO Completion Event Stream */
    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,

    /*
     * Communication Management Request Event Stream
     */
    IT_CM_REQ_EVENT_STREAM                   = 0x01000,
    IT_CM_REQ_CONN_REQUEST_EVENT             = 0x01001,
    IT_CM_REQ_UD_SERVICE_REQUEST_EVENT       = 0x01002,

    /*
     * Communication Management Message Event Stream
     */
    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,

    /* Asynchronous Affiliated Event Stream */
    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_LOCAL_ACCESS_VIOLATION   = 0x04004,
    IT_ASYNC_AFF_EP_REQ_DROPPED              = 0x04005,
    IT_ASYNC_AFF_EP_RDMAW_ACCESS_VIOLATION   = 0x04006,
    IT_ASYNC_AFF_EP_RDMAW_CORRUPT_DATA       = 0x04007,
    IT_ASYNC_AFF_EP_RDMAR_ACCESS_VIOLATION   = 0x04008,

    /* Asynchronous Non-Affiliated Event Stream */
```

```
        IT_ASYNC_UNAFF_EVENT_STREAM               = 0x08000,
        IT_ASYNC_UNAFF_IA_CATASTROPHIC_ERROR      = 0x08001,
        IT_ASYNC_UNAFF_SPIGOT_ONLINE              = 0x08002,
        IT_ASYNC_UNAFF_SPIGOT_OFFLINE             = 0x08003,
        IT_ASYNC_UNAFF_SEVD_ENQUEUE_FAILURE       = 0x08004,

        /* Software Event Stream */
        IT_SOFTWARE_EVENT_STREAM                  = 0x10000,
        IT_SOFTWARE_EVENT                         = 0x10001,

        /* AEVD Notification Event Stream */
        IT_AEVD_NOTIFICATION_EVENT_STREAM         = 0x20000,
        IT_AEVD_NOTIFICATION_EVENT                = 0x20001
    } it_event_type_t;

    typedef struct {
         it_event_type_t event_number;
         it_evd_handle_t evd;
    } it_any_event_t;

    typedef union
    {
        /*
         * The following two union elements are
         * available for programming convenience.
         *
         * The event_number may be used to determine the
         * it_event_type_t of any Event. it_any_event_t
         * allows the EVD to be determined as well.
         */
         it_event_type_t      event_number;
         it_any_event_t       any;

        /*
         * The remaining union elements correspond to
         * the various it_event_type_t types.
         */

        /*
         * The following two Event structures
         * support the IT_DTO_EVENT_STREAM Event Stream.
         *
         * it_dto_cmpl_event_t supports
         * only the following events:
         *     IT_DTO_SEND_CMPL_EVENT
         *     IT_DTO_RC_RECV_CMPL_EVENT
         *     IT_DTO_RDMA_WRITE_CMPL_EVENT
         *     IT_DTO_RDMA_READ_CMPL_EVENT
         *     IT_RMR_BIND_CMPL_EVENT
         *
         * it_all_dto_cmpl_event_t supports all
         * possible DTO and RMR events:
         *     IT_DTO_SEND_CMPL_EVENT
         *     IT_DTO_RC_RECV_CMPL_EVENT
```

```
         *     IT_DTO_UD_RECV_CMPL_EVENT
         *     IT_DTO_RDMA_WRITE_CMPL_EVENT
         *     IT_DTO_RDMA_READ_CMPL_EVENT
         *     IT_RMR_BIND_CMPL_EVENT
         */
         it_dto_cmpl_event_t         dto_cmpl;
         it_all_dto_cmpl_event_t     all_dto_cmpl;

        /*
         * The following two Event structures
         * support the IT_CM_REQ_EVENT_STREAM Event
         * stream:
         *
         * it_conn_request_event_t supports:
         *     IT_CM_REQ_CONN_REQUEST_EVENT
         *
         * it_ud_svc_request_event_t supports:
         *     IT_CM_REQ_UD_SERVICE_REQUEST_EVENT
         */
        it_conn_request_event_t      conn_req;
        it_ud_svc_request_event_t    ud_svc_request;

        /*
         * The following two Event structures
         * support the IT_CM_MSG_EVENT_STREAM Event
         * stream:
         *
         * it_connection_event_t supports:
         *     IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT
         *     IT_CM_MSG_CONN_ESTABLISHED_EVENT
         *     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 man page(s) specific to each Event group.

RETURN VALUE

None.

ERRORS

None.

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_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;
```

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_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

RETURN VALUE

None.

ERRORS

None.

SEE ALSO

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 Native 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,

   /* 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. */
   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;

/* The following structure is used to identify the vendor associated
   with an IA that uses the IB transport*/
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
      IB spec. */
   uint32_t                      revision;
} it_vendor_ib_t;

/* The following structure is used to identify the vendor associated
   with an IA that uses a VIA transport*/
typedef struct {
   /* The “Name” member of the VIP_NIC_ATTRIBUTES structure, as
      described in the VIA spec. */
   char                          name[64];
```

```
      /* The “HardwareVersion” member of the VIP_NIC_ATTRIBUTES structure,
         as described in the VIA spec. */
      unsigned long                 hardware;

      /* The “ProviderVersion” member of the VIP_NIC_ATTRIBUTES structure,
          as described in the VIA spec. */
      unsigned long                 provider;
   } it_vendor_via_t;

   /* The following structure is returned by the it_ia_query function  */
   typedef struct {

      /* Interface Adapter name, as specified in it_ia_create */
      char*                         ia_name;

      /* The major version number of the latest version of the IT-API that
          this IA supports. */
      uint32_t                      api_major_version;

      /* The minor version number of the latest version of the IT-API that
          this IA supports. */
      uint32_t                      api_minor_version;

      /* The major version number for the software being used to control
          this IA.  The IT-API imposes no structure whatsoever on this
          number; its meaning is completely IA-dependent. */
      uint32_t                      sw_major_version;

      /* The minor version number for the software being used to control
          this IA.  The IT-API imposes no structure whatsoever on this
          number; its meaning is completely IA-dependent. */
      uint32_t                      sw_minor_version;

      /* The vendor associated with the IA.  This information is useful
          if the Consumer wishes to do device-specific programming.  This
          union is discriminated by transport_type.  No vendor
          identification is provided for transports not listed below. */
      union {

            /* Used if transport_type is IT_IB_TRANSPORT */
            it_vendor_ib_t          ib;

            /* Used if transport_type is IT_VIA_IP_TRANSPORT or
   IT_VIA_FC_TRANSPORT */
            it_vendor_via_t         via;

      } vendor;

      /* The Interface Adapter and platform provide a data alignment hint
          to the Consumer to help the Consumer align their data transfer
          buffers in a way that is optimal for the performance of the IA.
          For example, if the best throughput is obtained by aligning
          buffers to 128-byte boundaries, dto_alignment_hint will have the
```

```
        value 128.  The Consumer may choose to ignore the alignment hint
        without any adverse functional impact.  (There may be an adverse
        performance impact.) */
    uint32_t                        dto_alignment_hint;

    /* The transport type (e.g. InfiniBand) supported by Interface
        Adapter.  An Interface Adapter supports precisely one transport
        type. */
    it_transport_type_t             transport_type;

    /* The Transport Service Types supported by this IA.  This is
        constructed by doing an inclusive OR of the Transport Service
        Type values.*/
    it_transport_service_type_t     supported_service_types;

    /* Indicates whether work queues are resizable */
    it_boolean_t                    resizable_work_queue;

    /* Indicates whether the underlying transport used by this IA uses a
        three-way handshake for doing Connection establishment.  Note
        that if the underlying transport supports a three-way handshake
        the Consumer can choose whether to use two handshakes or three
        when establishing the Connection.  If the underlying transport
        supports a two-way handshake for establishing a Connection, the
        Consumer can only use two handshakes when establishing the
        Connection. */
    it_boolean_t                    three_way_handshake_support;

    /* Indicates whether Private Data is supported on Connection
        establishment or UD service resolution operations. */
    it_boolean_t                    private_data_support;

    /* Indicates whether the max_message_size field in the
        IT_CM_REQ_CONN_REQUEST_EVENT is valid for this IA. */
    it_boolean_t                    max_message_size_support;

    /* Indicates whether the rdma_read_inflight_incoming field in the
       IT_CM_REQ_CONN_REQUEST_EVENT is valid for this IA. */
    it_boolean_t                    ird_support;

    /* Indicates whether the rdma_read_inflight_outgoing field in the
        IT_CM_REQ_CONN_REQUEST_EVENT is valid for this IA. */
    it_boolean_t                    ord_support;

    /* Indicates whether the IA generates IT_ASYNC_UNAFF_SPIGOT_ONLINE
        Events.  See it_unaffiliated_event_t for details. */
    it_boolean_t                    spigot_online_support;

    /* Indicates whether the IA generates IT_ASYNC_UNAFF_SPIGOT_OFFLINE
        Events.  See it_unaffiliated_event_t for details. */
    it_boolean_t                    spigot_offline_support;

    /* The maximum number of bytes of Private Data supported for the
        it_ep_connect routine.  This will be less than or equal to
```

```
            IT_MAX_PRIV_DATA. */
        size_t                          connect_private_data_len;

        /* The maximum number of bytes of Private Data supported for the
             it_ep_accept routine. This will be less than or equal to
             IT_MAX_PRIV_DATA. */
        size_t                          accept_private_data_len;

        /* The maximum number of bytes of Private Data supported for the
             it_reject routine. This will be less than or equal to
             IT_MAX_PRIV_DATA. */
        size_t                          reject_private_data_len;

        /* The maximum number of bytes of Private Data supported for the
             it_ep_disconnect routine. This will be less than or equal to
             IT_MAX_PRIV_DATA. */
        size_t                          disconnect_private_data_len;

        /* The maximum number of bytes of Private Data supported for the
             it_ud_service_request_handle_create routine. This will be
    less than or
             equal to IT_MAX_PRIV_DATA. */
        size_t                          ud_req_private_data_len;

        /* The maximum number of bytes of Private Data supported for the
            it_ud_service_reply routine. This will be less than or equal to
             IT_MAX_PRIV_DATA. */
        size_t                          ud_rep_private_data_len;

        /* Specifies the number of Spigots associated with this Interface
             Adapter */
        size_t                          num_spigots;

        /* An array of Spigot information data structures.  The array
             contains num_spigots elements.                    */
        it_spigot_info_t*         spigot_info;

        /* The Handle for the EVD that contains the affiliated async Event
             Stream.  If no EVD contains the Affiliated Async Event Stream,
             this member will have the distinguished value IT_NULL_HANDLE */
        it_evd_handle_t                 affiliated_err_evd;

        /* The Handle for the EVD that contains the Unaffiliated Async Event
             Stream.  If no EVD contains the Unaffiliated Async Event Stream,
             this member will have the distinguished value IT_NULL_HANDLE */
        it_evd_handle_t                 unaffiliated_err_evd;

    } 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.

RETURN VALUE

None.

ERRORS

None.

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_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;
   void                 *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.

length Length in bytes of the local buffer segment.

RETURN VALUE

None.

ERRORS

None.

APPLICATION USAGE

The LMR Triplet is an input parameter to all of the DTO operations, such as *it_post_send*.

SEE ALSO

it_post_send(), *it_post_sendto*(), *it_post_recv*(), *it_post_recvfrom*(), *it_post_rdma_write*(), *it_post_rdma_read*().

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.)

RETURN VALUE

None.

ERRORS

None.

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 remote component of 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;
```

```
    /* SL, as defined in the REQ message for the IB CM
       protocol.  This is also the “Service Level” to be used
       when creating an Address Handle. */
    uint8_t                       sl : 4;

    /* Subnet Local, as defined in the REQ message for the IB
       CM protocol.  When creating an Address Handle, setting
       this bit causes a GRH to be included as part of any
       Unreliable Datagram sent using the Address Handle. */
    uint8_t                       subnet_local : 1;

    /* Flow Label, as defined in the REQ message for the IB CM
       protocol.  This is also the “Flow Label” to be used when
       creating an Address Handle.  This is only valid if
      subnet_local is clear. */
    uint32_t                      flow_label : 20;

    /* Traffic Class, as defined in the REQ message for the IB
       CM protocol.  This is also the “Traffic Class” to be
       used when creating an Address Handle.  This is only
       valid if subnet_local is clear. */
    uint8_t                       traffic_class;

    /* Hop Limit, as defined in the REQ message for the IB CM
       protocol.  This is also the “Hop Limit” to be used when
       creating an Address Handle. This is only valid if
       subnet_local is clear. */
    uint8_t                       hop_limit;

} it_ib_net_endpoint_t;

/* This is the remote component of the Path information for the
   VIA transport */
typedef it_via_net_addr_t it_via_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 can not 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;

        } 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.

RETURN VALUE

None.

ERRORS

None.

SEE ALSO

it_get_pathinfo(), *it_ep_connect*(), *it_address_handle_create*(), *it_ud_service_request_handle_create*()

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 just 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.

RETURN VALUE

None.

ERRORS

None.

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_CANNOT_RESET,
   IT_ERR_CONN_QUAL_BUSY,
   IT_ERR_EP_TIMEWAIT,
   IT_ERR_EVD_BUSY,
   IT_ERR_EVD_QUEUE_FULL,
   IT_ERR_FAULT,
   IT_ERR_IA_CATASTROPHE,
   IT_ERR_INTERRUPT,
   IT_ERR_INVALID_ADDRESS,
   IT_ERR_INVALID_AEVD,
   IT_ERR_INVALID_AH,
   IT_ERR_INVALID_ATIMEOUT,
   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_LENGTH,
   IT_ERR_INVALID_LISTEN,
   IT_ERR_INVALID_LMR,
   IT_ERR_INVALID_LTIMEOUT,
   IT_ERR_INVALID_MAJOR_VERSION,
   IT_ERR_INVALID_MASK,
   IT_ERR_INVALID_MINOR_VERSION,
```

```
    IT_ERR_INVALID_NAME,
    IT_ERR_INVALID_NETADDR,
    IT_ERR_INVALID_NUM_SEGMENTS,
    IT_ERR_INVALID_PDATA_LENGTH,
    IT_ERR_INVALID_PRIVS,
    IT_ERR_INVALID_PZ,
    IT_ERR_INVALID_QUEUE_SIZE,
    IT_ERR_INVALID_RECV_EVD,
    IT_ERR_INVALID_RECV_EVD_STATE,
    IT_ERR_INVALID_REQ_EVD,
    IT_ERR_INVALID_REQ_EVD_STATE,
    IT_ERR_INVALID_RETRY,
    IT_ERR_INVALID_RMR,
    IT_ERR_INVALID_RNR_RETRY,
    IT_ERR_INVALID_RTIMEOUT,
    IT_ERR_INVALID_SGID,
    IT_ERR_INVALID_SLID,
    IT_ERR_INVALID_SOFT_EVD,
    IT_ERR_INVALID_SOURCE_PATH,
    IT_ERR_INVALID_SPIGOT,
    IT_ERR_INVALID_THRESHOLD,
    IT_ERR_INVALID_UD_STATUS,
    IT_ERR_INVALID_UD_SVC,
    IT_ERR_INVALID_UD_SVC_REQ_ID,
    IT_ERR_LMR_BUSY,
    IT_ERR_MISMATCH_FD,
    IT_ERR_NO_CONTEXT,
    IT_ERR_NO_PERMISSION,
    IT_ERR_PAYLOAD_SIZE,
    IT_ERR_PDATA_NOT_SUPPORTED,
    IT_ERR_PZ_BUSY,
    IT_ERR_QUEUE_EMPTY,
    IT_ERR_RANGE,
    IT_ERR_RESOURCES,
    IT_ERR_RESOURCE_IRD,
    IT_ERR_RESOURCE_LMR_LENGTH,
    IT_ERR_RESOURCE_ORD,
    IT_ERR_RESOURCE_QUEUE_SIZE,
    IT_ERR_RESOURCE_RECV_DTO,
    IT_ERR_RESOURCE_REQ_DTO,
    IT_ERR_RESOURCE_RRSEG,
    IT_ERR_RESOURCE_RSEG,
    IT_ERR_RESOURCE_RWSEG,
    IT_ERR_RESOURCE_SSEG,
    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 man 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 if an asynchronous operation was successfully completed, the Completion Event for the asynchronous operation should be examined.

RETURN VALUE

None.

ERRORS

None.

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_sync_rdma_read*(), *it_lmr_sync_rdma_write*(), *it_post_rdma_read*(), *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_bind*(), *it_rmr_create*(), *it_rmr_free*(), *it_rmr_query*(), *it_rmr_unbind*(), *it_set_consumer_context*(), *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:
IT_ASYNC_UNAFF_SPIGOT_ONLINE,
IT_ASYNC_UNAFF_SPIGOT_OFFLINE,
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_event_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_event_support* value is

	IT_TRUE.
IT_ASYNC_UNAFF_SEVD_ENQUEUE_FAILURE	The API Implementation was unable to enqueue an entry into an Affiliated Asynchronous Event SEVD.

EXTENDED DESCRIPTION

For the Infiniband transport, the following table maps the values in the Unaffiliated Asynchronous Errors *it_event_type_t* enumeration to their corresponding “Unaffiliated Asynchronous Events” and “Unaffiliated Asynchronous Errors” as specified in Volume 1, Chapter 11 of the Infiniband specification.

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

For the VIA transport, the following table maps the values in the Unaffiliated Asynchronous Errors *it_event_type_t* enumeration to their corresponding descriptions in the “VipErrorCallback” man page in the Appendix of the VIA specification.

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.)

All Events on an IT_ ASYNC_UNAFF_EVENT_STREAM SEVD cause Notification. See *it_evd_create* for details of Notification.

Overflow of an IT_ASYNC_UNAFF_EVENT_STREAM SEVD is not visible to the Consumer; all subsequent IT_ASYNC_UNAFF_EVENT_STREAM Events are silently dropped until the Consumer dequeues at least one Event from the EVD that contains the IT_ASYNC_UNAFF_EVENT_STREAM.

When a Consumer has created more than one IA corresponding to an underlying physical adapter (say in different processes), then every Unaffiliated Event is replicated to every IA instance.

RETURN VALUE

None.

ERRORS

None.

SEE ALSO

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 doesn’t 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 doesn’t 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 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.

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 time frame 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_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 case 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 attribute. In general, the Implementation can allocate more resources than requested by 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 can not fail because resources requested by Consumer at Endpoint creation time are not available. Connection establishment may fail when remote Endpoint of the Connection does not have enough resources to match local Endpoint ones. For example, if the local Endpoint's attributes have an rdma_read_inflight_incoming that is less than the remote Endpoint's rdma_read_inflight_outgoing, 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_inflight_outgoing

it_ep_attributes_t.max_dto_payload_size

The reason *rdma_read_inflight_outgoing* is listed above is that for InfiniBand you can't establish a Connection unless the ORD value for one side of the Connection is less than or equal to the IRD value for other side.

The reason *max_dto_payload_size* is listed above is that for VIA you can't 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 resizable_work_queue* is Clear must not be successful and IT_ERR_INVALID_EP_STATE is the return value for this case.

When running over the InfiniBand transport, 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 non-operation 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, the Implementation should send CM DREQ (Disconnect Request).

it_ep_free

The *it_ep_free* 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. 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_rc_creation_flag_t parameter* on transports where the timewait state is not applicable.

it_ep_reset

This operation must hide any internal Implementation waiting for timeout expiration that Endpoint may be in due to *it_ep_disconnect* call during Connection set up.

it_ep_state_t

In all case 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.

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. is an Event for a DTO with IT_NOTIFY_FLAG set when posted.
    b. is an Event for a Recv DTO where matching Send DTO was originally posted with IT_SOLICITED_WAIT_FLAG set (regardless of IT_NOTIFY_FLAG on Recv DTO).
    c. is the Nth Event to arrive where threshold is set to N.
    d. is an Event for a DTO completing in error regardless of IT_NOTIFY_FLAG and IT_COMPLETION_FLAG.
    e. 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*[1] 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[2] when:
    a. for SEVD - queue is empty.
    b. for AEVD - all associated SEVDs are empty.
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 event*[3] or the number of Events is greater than or equal

[1] 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 its own Event Stream types and its own SEVDs. Hence, Implementation can retain the *notification status* for them based on SEVD Event Stream type.

[2] This is really an Implementation issue. Semantically, Consumer does not care if it is blocked or not.

to the *threshold*:

a. for SEVD - on the SEVD queue.

b. for AEVD - on any associated SEVDs.

4. If threshold > 1, evd_wait **should** block[4] if less than threshold number of Events and no *notification events* are[5]

   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[6], 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[7].

<u>Heuristic for wakeup of multiple waiters:</u>

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[8], 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());
```

[3] 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).

[4] 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.

[5] with sevd_threshold = IT_THRESHOLD_DISABLED (= infinity), it_evd_wait will block if no notification event are on 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.

[6] Events that arrive when there are no blocked evd_wait *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. Implementation *can* be *over-eager* and return an it_evd_wait without checking the *notification status* of events on EVD

[7] 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.

[8] The queue length function is not Verbs compliant, but will be commonly available.

```
                    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 SEVD.

If SEVD is enabled, then an arriving SEVD Event that causes 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.

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 Un-affiliated and Affiliated Events that happened before Consumer created SEVDs for 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 Implementation.

Implementation is not required to check that the *Event* structure that 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 Implementation.

Implementation is not required to check that the *event* structure that 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 that 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.

it_ia_create

The Implementation shall not provide to the Consumer Un-affiliated and Affiliated Events that happened before Consumer created SEVDs for Unaffiliated and Affiliated Event Stream. The Implementation can provide Events that happened during SEVD creation time.

it_ia_free

The Implementation should free the IT Objects in the reverse order of their construction in order to guarantee that all underlying transport resources will be successfully freed.

it_ia_info_t

The 1000+ number range in the *it_transport_type_t* is intended to facilitate the short-term prototyping efforts of vendors who are developing new transports that work with the IT-API. A vendor that wishes to productize their prototyping effort should contact the ICSC in order to be assigned a permanent transport number in the < 1000 range. The Implementation is not responsible for ensuring that two different vendors utilizing the 1000+ range of transport numbers do not collide.

For the IB transport, when *it_ep_disconnect* is called there are two possible underlying CM messages that the Private Data could travel with: DREQ, or REJ. The value of disconnect_private_data_len for IB should be the lesser of the maximum Private Data supported in a DREQ message and the maximum Private Data supported in a REJ message.

it_lmr_create

If the EVD where the CM Request Events stream (IT_CM_REQ_EVENT_STREAM) is routed on the passive side is full, then the Active side shall receive an IT_CM_MSG_CONN_NONPEER_REJECT_EVENT Event with a reason code of IT_CN_REJ_TIMEOUT.

it_lmr_create

it_lmr_create can be implemented on the InfiniBand transport by using the Register Memory Region Verb. To allow subsequent calls to *it_rmr_bind* using this LMR, the Implementation must include the Enable Memory Window Binding input modifier, and also the Enable Local Write Access modifier. The latter is necessary because the subsequent *it_rmr_bind* call may request remote write access, and the InfiniBand Architecture specifies that local access is a pre-requisite for remote access. Note that the Register Memory Region Verb implicitly enables local access; so on InfiniBand transport the Consumer's setting of the IT_PRIV_LOCAL_READ flag in the privs argument is effectively ignored.

The IT_LMR_FLAG_SHARED option can be implemented using the InfiniBand Register Memory Region and Register Shared Memory Region Verbs. 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 Memory Region Verb. The Implementation should protect against race conditions between multiple Consumers and avoid calling Register Memory Region multiple times for the same memory region.

See the advice under *it_rmr_bind* for comments concerning *rmr_context* and byte order.

it_lmr_free

Beware that determining when it is permissible to unlock 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_query

it_lmr_query can be implemented on the InfiniBand transport by using the Query Memory Region Verb. 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. In addition, the Actual Remote Protection Bounds will be contained within the Actual Local Protection Bounds, because the remote bounds are rounded to 4K byte boundaries, and local bounds are rounded to page boundaries. This means the Consumer may safely use the returned actual_addr and actual_length as both local and remote bounds.

it_lmr_sync_rdma_read
it_lmr_sync_rdma_write

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_rmr_bind

The *rmr_context* returned by *it_rmr_bind* is defined to be returned in network byte order, e.g. 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_bind* 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

In order to return correct values for the *bound*, *lmr*, *addr*, *length*, and *privs* attributes of params, the Implementation must track the completion status of Bind and Unbind operations, and the parameters associated with those operations. To do so, the Implementation can replace the Consumer's cookie passed to *it_rmr_bind* or *it_rmr_unbind* with an Implementation cookie that points to a structure. This structure stores the original Consumer cookie and all relevant parameters to the Bind or Unbind operation. In *it_evd_dequeue*, the Implementation would peek at the operation type of the dequeued Completion Event. If the type is a Bind operation (note that InfiniBand does not distinguish this from an Unbind operation), and the completion status is successful, 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.

The *rmr_context* attribute could be handled similarly, or the Query Memory Region Verb could be used for InfiniBand transport.

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 can not 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.

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 can not 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 can not 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 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 can not 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 can not 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 can not 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 (IBTA 1.1 vol 1).

The Implementation should support zero-copy data transfers and kernel bypass for the SendTo 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 Affiliated 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. Header Files

B.1 it_api.h

```
#include "it_api_os_specific.h"

#define IN
#define OUT

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_CANNOT_RESET,
   IT_ERR_CONN_QUAL_BUSY,
   IT_ERR_EP_TIMEWAIT,
   IT_ERR_EVD_BUSY,
   IT_ERR_EVD_QUEUE_FULL,
   IT_ERR_FAULT,
   IT_ERR_IA_CATASTROPHE,
   IT_ERR_INTERRUPT,
   IT_ERR_INVALID_ADDRESS,
   IT_ERR_INVALID_AEVD,
   IT_ERR_INVALID_AH,
   IT_ERR_INVALID_ATIMEOUT,
   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_LENGTH,
                IT_ERR_INVALID_LISTEN,
                IT_ERR_INVALID_LMR,
                IT_ERR_INVALID_LTIMEOUT,
                IT_ERR_INVALID_MAJOR_VERSION,
                IT_ERR_INVALID_MASK,
                IT_ERR_INVALID_MINOR_VERSION,
                IT_ERR_INVALID_NAME,
                IT_ERR_INVALID_NETADDR,
                IT_ERR_INVALID_NUM_SEGMENTS,
                IT_ERR_INVALID_PDATA_LENGTH,
                IT_ERR_INVALID_PRIVS,
                IT_ERR_INVALID_PZ,
                IT_ERR_INVALID_QUEUE_SIZE,
                IT_ERR_INVALID_RECV_EVD,
                IT_ERR_INVALID_RECV_EVD_STATE,
                IT_ERR_INVALID_REQ_EVD,
                IT_ERR_INVALID_REQ_EVD_STATE,
                IT_ERR_INVALID_RETRY,
                IT_ERR_INVALID_RMR,
                IT_ERR_INVALID_RNR_RETRY,
                IT_ERR_INVALID_RTIMEOUT,
                IT_ERR_INVALID_SGID,
                IT_ERR_INVALID_SLID,
                IT_ERR_INVALID_SOFT_EVD,
                IT_ERR_INVALID_SOURCE_PATH,
                IT_ERR_INVALID_SPIGOT,
                IT_ERR_INVALID_THRESHOLD,
                IT_ERR_INVALID_UD_STATUS,
                IT_ERR_INVALID_UD_SVC,
                IT_ERR_INVALID_UD_SVC_REQ_ID,
                IT_ERR_LMR_BUSY,
                IT_ERR_MISMATCH_FD,
                IT_ERR_NO_CONTEXT,
                IT_ERR_NO_PERMISSION,
                IT_ERR_PAYLOAD_SIZE,
                IT_ERR_PDATA_NOT_SUPPORTED,
                IT_ERR_PZ_BUSY,
                IT_ERR_QUEUE_EMPTY,
                IT_ERR_RANGE,
                IT_ERR_RESOURCES,
                IT_ERR_RESOURCE_IRD,
                IT_ERR_RESOURCE_LMR_LENGTH,
                IT_ERR_RESOURCE_ORD,
                IT_ERR_RESOURCE_QUEUE_SIZE,
                IT_ERR_RESOURCE_RECV_DTO,
                IT_ERR_RESOURCE_REQ_DTO,
                IT_ERR_RESOURCE_RRSEG,
                IT_ERR_RESOURCE_RSEG,
                IT_ERR_RESOURCE_RWSEG,
                IT_ERR_RESOURCE_SSEG,
                IT_ERR_TIMEOUT_EXPIRED,
                IT_ERR_TOO_MANY_POSTS,
                IT_ERR_WAITER_LIMIT
```

```
} it_status_t;

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_PRIV_NONE           = 0x0001,
   IT_PRIV_READ_ONLY      = 0x0002,
   IT_PRIV_REMOTE_READ    = 0x0004,
   IT_PRIV_REMOTE_WRITE   = 0x0008,
   IT_PRIV_REMOTE         = 0x0010,
   IT_PRIV_ALL            = 0x0020,
   IT_PRIV_DEFAULT        = 0x0040
} it_mem_priv_t;

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

typedef uint64_t it_ud_svc_req_identifier_t;

typedef uint64_t it_cn_est_identifier_t;


/* it_boolean_t.txt */

typedef enum {
  IT_FALSE = 0,
  IT_TRUE = 1
} it_boolean_t;

/* it_handle_t.txt */

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_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;


/* it_conn_qual_t.txt */

/* Enumerates all the possible Connection Qualifier types
   supported by the API. */
typedef enum {

   /* IANA (TCP/UDP) Port Number */
   IT_IANA_PORT = 0x1,

   /* InfiniBand Service ID, as described in section 12.7.3 of
          Volume 1 of the InfiniBand specification. */
   IT_IB_SERVICEID = 0x2,

   /* VIA Connection Discriminator */
   IT_VIA_DISCRIMINATOR = 0x4

} 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;
```

```
/* 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;

   } conn_qual;

} it_conn_qual_t;


/* it_context_t.txt */

typedef union {
   void *                        ptr;
   uint64_t                      index;
} it_context_t;

/* it_dto_cookie_t.txt */

typedef uint64_t it_dto_cookie_t;

/* it_dto_status_t.txt */

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_status_t;


/* it_dto_flags_t.txt */
```

```
typedef enum
{
    /* If flag set, completion generates a local Event */
    IT_COMPLETION_FLAG                = 0x01,

    /* If flag set, completion cause 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, DTO processing will not start if
        previously posted RDMA Reads are not complete. */
    IT_BARRIER_FENCE_FLAG             = 0x08,
} it_dto_flags_t;

/* it_net_addr_t.txt */

/* 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;


/* it_ia_info_t.txt */

/*  Enumerates all the transport types supported by the API. */
typedef enum {

   /* InfiniBand Native 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,

   /* 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. */
   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.  A 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;
```

```
/* The following structure is used to identify the vendor associated
   with an IA that uses the IB transport*/
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 IB spec. */
   uint32_t          revision;
} it_vendor_ib_t;

/* The following structure is used to identify the vendor
   associated with an IA that uses a VIA transport*/
typedef struct {
   /* The "Name" member of the VIP_NIC_ATTRIBUTES structure,
      as described in the VIA spec. */
   char              name[64];

   /* The "HardwareVersion" member of the VIP_NIC_ATTRIBUTES
      structure, as described in the VIA spec. */
   unsigned long           hardware;

   /* The "ProviderVersion" member of the VIP_NIC_ATTRIBUTES
      structure, as described in the VIA spec. */
   unsigned long           provider;
} it_vendor_via_t;

/* The following structure is returned by the it_ia_query function */
typedef struct {

   /* Interface Adapter name, as specified in it_ia_create */
   char*                   ia_name;

   /* The major version number of the latest version of the
      IT-API that this IA supports. */
   uint32_t                api_major_version;

   /* The minor version number of the latest version of the
      IT-API that this IA supports. */
   uint32_t                api_minor_version;

   /* The major version number for the software being used to
     control this IA.  The IT-API imposes no structure whatsoever
     on this number; its meaning is completely IA-dependent. */
   uint32_t                sw_major_version;

   /* The minor version number for the software being used to
     control this IA.  The IT-API imposes no structure whatsoever
```

```
        on this number; its meaning is completely IA-dependent. */
    uint32_t                sw_minor_version;

    /* The vendor associated with the IA.  This information is useful
       if the Consumer wishes to do device-specific programming.  This
       union is discriminated by transport_type.  No vendor
       identification is provided for transports not listed below. */
    union {

          /* Used if transport_type is IT_IB_TRANSPORT */
          it_vendor_ib_t          ib;

          /* Used if transport_type is IT_VIA_IP_TRANSPORT or
             IT_VIA_FC_TRANSPORT */
          it_vendor_via_t         via;

    } vendor;

    /* The Interface Adapter and platform provide a data alignment hint
       to the Consumer to Help the Consumer align their data transfer
       buffers in a way the is optimal for the performance of the IA.
       For example, if the best throughput is obtained by aligning
       buffers to 128-byte boundaries, dto_alignment_hint will have the
       value 128.  The Consumer may choose to ignore the alignment hint
       without any adverse functional impact.  (There may be an adverse
       performance impact.) */
    uint32_t                dto_alignment_hint;

    /* The transport type (e.g. InfiniBand) supported by Interface
       Adapter.  An Interface Adapter supports precisely one transport
       type. */
    it_transport_type_t           transport_type;

    /* The Transport Service Types supported by this IA.  This is
       constructed by doing an inclusive OR of the Transport Service
       Type values.*/
    it_transport_service_type_t   supported_service_types;

    /* Indicates whether work queues are resizable */
    it_boolean_t                  resizable_work_queue;

    /* Indicates whether the underlying transport used by this IA uses
       a three-way handshake for doing Connection establishment.  Note
       that if the underlying transport supports a three-way handshake
       the Consumer can choose whether to use two handshakes or three
       when establishing the Connection.  If the underlying transport
       supports a two-way handshake for establishing a Connection, the
       Consumer can only use two handshakes when establishing the
       Connection. */
    it_boolean_t                  three_way_handshake_support;

    /* Indicates whether Private Data is supported on Connection
       establishment or UD service resolution operations. */
    it_boolean_t                  private_data_support;
```

```

        /* Indicates whether the max_message_size field in the
            IT_CM_REQ_CONN_REQUEST_EVENT is valid for this IA. */
        it_boolean_t                  max_message_size_support;

        /* Indicates whether the rdma_read_inflight_incoming field
           in the IT_CM_REQ_CONN_REQUEST_EVENT is valid for this IA. */
        it_boolean_t                  ird_support;

        /* Indicates whether the rdma_read_inflight_outgoing field
           in the IT_CM_REQ_CONN_REQUEST_EVENT is valid for this IA. */
        it_boolean_t                  ord_support;

        /* Indicates whether the IA generates IT_ASYNC_UNAFF_SPIGOT_ONLINE
           Events.  See it_unaffiliated_event_t for details. */
        it_boolean_t                  spigot_online_support;

        /* Indicates whether the IA generates IT_ASYNC_UNAFF_SPIGOT_OFFLINE
           Events.  See it_unaffiliated_event_t for details. */
        it_boolean_t                  spigot_offline_support;

        /* The maximum number of bytes of Private Data supported for the
           it_ep_connect routine.  This will be less than or equal to
           IT_MAX_PRIV_DATA. */
        size_t                        connect_private_data_len;

        /* The maximum number of bytes of Private Data supported for the
           it_ep_accept routine. This will be less than or equal to
           IT_MAX_PRIV_DATA. */
        size_t                        accept_private_data_len;

        /* The maximum number of bytes of Private Data supported for the
           it_reject routine. This will be less than or equal to
           IT_MAX_PRIV_DATA. */
        size_t                        reject_private_data_len;

        /* The maximum number of bytes of Private Data supported for the
           it_ep_disconnect routine. This will be less than or equal to
           IT_MAX_PRIV_DATA. */
        size_t                        disconnect_private_data_len;

        /* The maximum number of bytes of Private Data supported for the
           it_ud_service_request_handle_create routine. This will be less
           than or equal to IT_MAX_PRIV_DATA. */
        size_t                        ud_req_private_data_len;

        /* The maximum number of bytes of Private Data supported for the
          it_ud_service_reply routine. This will be less than or equal to
          IT_MAX_PRIV_DATA. */
        size_t                        ud_rep_private_data_len;

        /* Specifies the number of Spigots associated with this Interface
           Adapter */
        size_t                        num_spigots;
```

```
    /* An array of Spigot information data structures.  The array
       contains num_spigots elements.                    */
    it_spigot_info_t*       spigot_info;

    /* The Handle for the EVD that contains the affiliated async Event
       Stream.  If no EVD contains the Affiliated Async Event Stream,
       this member will have the distinguished value IT_NULL_HANDLE */
    it_evd_handle_t               affiliated_err_evd;

    /* The Handle for the EVD that contains the unaffiliated async Event
       Stream.  If no EVD contains the Unaffiliated Async Event Stream,
       this member will have the distinguished value IT_NULL_HANDLE */
    it_evd_handle_t               unaffiliated_err_evd;

} it_ia_info_t;


/* it_lmr_triplet_t.txt */

typedef struct {
  it_lmr_handle_t      lmr;
  void                 *addr;
  it_length_t          length;
} it_lmr_triplet_t;

/* it_path_t.txt */

/* This is the remote component of 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;

            /* SL, as defined in the REQ message for the IB CM
               protocol.  This is also the "Service Level" to be used
               when creating an Address Handle. */
            uint8_t                       sl : 4;

            /* Subnet Local, as defined in the REQ message for the IB
               CM protocol.  When creating an Address Handle, setting
               this bit causes a GRH to be included as part of any
               Unreliable Datagram sent using the Address Handle. */
            uint8_t                       subnet_local : 1;

            /* Flow Label, as defined in the REQ message for the IB CM
               protocol.  This is also the "Flow Label" to be used when
               creating an Address Handle.  This is only valid if
                subnet_local is clear. */
            uint32_t                flow_label : 20;

            /* Traffic Class, as defined in the REQ message for the IB
               CM protocol.  This is also the "Traffic Class" to be
               used when creating an Address Handle.  This is only
               valid if subnet_local is clear. */
            uint8_t                       traffic_class;

            /* Hop Limit, as defined in the REQ message for the IB CM
               protocol.  This is also the "Hop Limit" to be used when
               creating an Address Handle. This is only valid if
               subnet_local is clear. */
            uint8_t                       hop_limit;

        } it_ib_net_endpoint_t;

        /* This is the remote component of the Path information for the
           VIA transport */
        typedef it_via_net_addr_t it_via_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 can not 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;

   } remote;

} it_path_t;


/* it_ep_attributes_t.txt */

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_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_inflight_incoming;
         /* IT_EP_PARAM_MAX_IRD */
    uint32_t  rdma_read_inflight_outgoing;
         /* IT_EP_PARAM_MAX_ORD */
} it_rc_only_attributes_t;

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_ep_attributes_t;
```

```
/* it_event_t.txt */

#define IT_EVENT_STREAM_MASK     0xff000
#define IT_TIMEOUT_INFINITE      ((uint64_t)(-1))

typedef enum
{
   /* DTO Completion Event Stream */
   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,

   /*
    * Communication Management Request Event Stream
    */
   IT_CM_REQ_EVENT_STREAM              = 0x01000,
   IT_CM_REQ_CONN_REQUEST_EVENT        = 0x01001,
   IT_CM_REQ_UD_SERVICE_REQUEST_EVENT  = 0x01002,

   /*
    * Communication Management Message Event Stream
    */
   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,

   /* Asynchronous Affiliated Event Stream */
   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_LOCAL_ACCESS_VIOLATION    = 0x04004,
   IT_ASYNC_AFF_EP_REQ_DROPPED               = 0x04005,
   IT_ASYNC_AFF_EP_RDMAW_ACCESS_VIOLATION    = 0x04006,
   IT_ASYNC_AFF_EP_RDMAW_CORRUPT_DATA        = 0x04007,
   IT_ASYNC_AFF_EP_RDMAR_ACCESS_VIOLATION    = 0x04008,

   /* Asynchronous Non-Affiliated Event Stream */
   IT_ASYNC_UNAFF_EVENT_STREAM               = 0x08000,
   IT_ASYNC_UNAFF_IA_CATASTROPHIC_ERROR      = 0x08001,
   IT_ASYNC_UNAFF_SPIGOT_ONLINE              = 0x08002,
   IT_ASYNC_UNAFF_SPIGOT_OFFLINE             = 0x08003,
   IT_ASYNC_UNAFF_SEVD_ENQUEUE_FAILURE       = 0x08004,
```

```

    /* Software Event Stream */
    IT_SOFTWARE_EVENT_STREAM                  = 0x10000,
    IT_SOFTWARE_EVENT                         = 0x10001,

    /* AEVD Notification Event Stream */
    IT_AEVD_NOTIFICATION_EVENT_STREAM   = 0x20000,
    IT_AEVD_NOTIFICATION_EVENT          = 0x20001
  } it_event_type_t;


  /* it_aevd_notification_event_t.txt */

  typedef struct {
    it_event_type_t   event_number;
    it_evd_handle_t   aevd;

    it_evd_handle_t   sevd;
  } it_aevd_notification_event_t;

  /* it_affiliated_event_t.txt */

  typedef struct {
    it_event_type_t   event_number;
    it_evd_handle_t   evd;

    union {
    it_evd_handle_t   sevd;
          it_ep_handle_t    ep;
    } u;
  } it_affiliated_event_t;

  /* it_cm_msg_events.txt */

  #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_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_inflight_incoming;
    uint32_t                rdma_read_inflight_outgoing;
    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;

    /* it_cm_req_events.txt */

    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_inflight_incoming;
        uint32_t                      rdma_read_inflight_outgoing;
        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;


    /* it_dto_events.txt */

    typedef enum {
```

```
      IB_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_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_dto_ud_flags_t      flags;
      it_ud_ep_id_t          ud_ep_id;
      it_path_t              src_path;
   } it_all_dto_cmpl_event_t;

   /* it_software_event_t.txt */

   typedef struct {
      it_event_type_t  event_number;
      it_evd_handle_t  evd;
      void             * data;
   } it_software_event_t;

   /* it_unaffiliated_event_t.txt */

   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;

   /* it_event_t.txt */

   typedef struct {
      it_event_type_t  event_number;
      it_evd_handle_t  evd;
   } it_any_event_t;

   typedef union
   {
      /*
       * The following two union elements are
       * available for programming convenience.
       *
```

```
     * The event_number may be used to determine the
     * it_event_type_t of any event. it_any_event_t
     * allows the evd to be determined as well.
     */
    it_event_type_t         event_number;
    it_any_event_t          any;

    /*
     * The remaining union elements correspond to
     * the various it_event_type_t types.
     */

    /*
     * The following two Event structures
     * support the IT_DTO_EVENT_STREAM Event Stream.
     *
     * it_dto_cmpl_event_t supports
     * only the following events:
     *     IT_DTO_SEND_CMPL_EVENT
     *     IT_DTO_RC_RECV_CMPL_EVENT
     *     IT_DTO_RDMA_WRITE_CMPL_EVENT
     *     IT_DTO_RDMA_READ_CMPL_EVENT
     *     IT_RMR_BIND_CMPL_EVENT
     *
     * it_all_dto_cmpl_event_t supports all
     * possible DTO and RMR events:
     *     IT_DTO_SEND_CMPL_EVENT
     *     IT_DTO_RC_RECV_CMPL_EVENT
     *     IT_DTO_UD_RECV_CMPL_EVENT
     *     IT_DTO_RDMA_WRITE_CMPL_EVENT
     *     IT_DTO_RDMA_READ_CMPL_EVENT
     *     IT_RMR_BIND_CMPL_EVENT
     */
    it_dto_cmpl_event_t           dto_cmpl;
    it_all_dto_cmpl_event_t all_dto_cmpl;

    /*
     * The following two Event structures
     * support the IT_CM_REQ_EVENT_STREAM Event
     * stream:
     *
     * it_conn_request_event_t supports:
     *     IT_CM_REQ_CONN_REQUEST_EVENT
     *
     * it_ud_svc_request_event_t supports:
     *     IT_CM_REQ_UD_SERVICE_REQUEST_EVENT
     */
    it_conn_request_event_t conn_req;
    it_ud_svc_request_event_t     ud_svc_request;

    /*
     * The following two Event structures
     * support the IT_CM_MSG_EVENT_STREAM Event
     * stream:
```

```
 *
 * it_connection_event_t supports:
 *     IT_CM_MSG_CONN_ACCEPT_ARRIVAL_EVENT
 *     IT_CM_MSG_CONN_ESTABLISHED_EVENT
 *     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;


/* it_ep_state_t.txt */
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_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;


/* it_dg_remote_ep_addr_t.txt */

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;

typedef enum {
  IT_AH_PATH_COMPLETE = 0x1
} it_ah_flags_t;

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;

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;

    typedef struct {

        /* VIA currently has no transport-specific connection
           attributes */

    } it_via_conn_attributes_t;

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

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

    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;

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

#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;

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_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_evd_param_t;

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;

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

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;

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_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_lmr_param_t;

typedef uint64_t it_rdma_addr_t;

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;

typedef enum {
    IT_RMR_PARAM_ALL         = 0x000001,
    IT_RMR_PARAM_IA          = 0x000002,
    IT_RMR_PARAM_PZ          = 0x000004,
    IT_RMR_PARAM_BOUND       = 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_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     bound;        /* IT_RMR_PARAM_BOUND */
   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_param_t;

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;

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_LEN */
  } it_ud_svc_req_param_t;

  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

  );

  it_status_t it_address_handle_free(
    IN                it_addr_handle_t        addr_handle
  );

  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
  );

  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
  );

  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
  );

  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
  );

   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
```

```
        );

        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
        );

        it_status_t it_ep_free(
          IN                it_ep_handle_t          ep_handle
        );

        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
        );

        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
        );

        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_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,
    IN           const void                    *event
 );

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
);

 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

 );

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

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

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
  );

  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
  );


  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
  );

  it_status_t it_ia_free(
    IN                it_ia_handle_t          ia_handle

  );

  void  it_ia_info_free(
    IN                it_ia_info_t            *ia_info
  );

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

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

  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
  );

  it_status_t it_listen_free(
    IN                 it_listen_handle_t     listen_handle
  );

  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
);

it_status_t it_lmr_create(
  IN                 it_pz_handle_t          pz_handle,
  IN                 void                    *addr,
  IN                 it_length_t             length,
  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
);

it_status_t it_lmr_free(
  IN                 it_lmr_handle_t   lmr_handle
);

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
);

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
);

it_status_t it_lmr_sync_rdma_read(
  IN           const it_lmr_triplet_t        *local_segments,
  IN                 size_t                  num_segments
);

it_status_t it_lmr_sync_rdma_write(
  IN           const it_lmr_triplet_t        *local_segments,
  IN                 size_t                  num_segments
);


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
);

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
  );

  it_status_t it_post_recv(
    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
  );

  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
  );

  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
  );

  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
  );

  it_status_t it_pz_create(
    IN              it_ia_handle_t          ia_handle,
    OUT             it_pz_handle_t          *pz_handle
  );

  it_status_t it_pz_free(
    IN              it_pz_handle_t          pz_handle
  );

  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
);

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
);

it_status_t it_rmr_bind(
    IN                  it_rmr_handle_t         rmr_handle,
    IN                  it_lmr_handle_t         lmr_handle,
    IN                  void                    *addr,
    IN                  it_length_t             length,
    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
);

it_status_t it_rmr_create(
    IN                  it_pz_handle_t          pz_handle,
    OUT                 it_rmr_handle_t         *rmr_handle
);

it_status_t it_rmr_free(
    IN                  it_rmr_handle_t         rmr_handle
);

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
);

it_status_t it_rmr_unbind(
    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
);

it_status_t it_set_consumer_context(
    IN                  it_handle_t             handle,
    IN                  it_context_t            context
);

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
);
```

B.2 it_api_os_specific.h

```
#include "/usr/include/sys/types.h"
#include "/usr/include/netinet/in.h"

/* the following should have arrived from types.h */
typedef unsigned char uint8_t;
typedef unsigned short uint16_t;
typedef unsigned int uint32_t;
typedef unsigned long uint64_t;

/* defines */

#define IT_INTERFACE_NAME_SIZE   128
```