Technical Standard

Resource ReSerVation Protocol API (RAPI)

The Open Group

Technical Standard

Resource ReSerVation Protocol API (RAPI)

ISBN: 1-85912-226-4
Document Number: C809

Published in the U.K. by The Open Group, December 1998.

Any 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

List of Figures

Preface

The Open Group

The Open Group is the leading vendor-neutral, international consortium for buyers and suppliers of technology. Its mission is to cause the development of a viable global information infrastructure that is ubiquitous, trusted, reliable, and as easy-to-use as the telephone. The essential functionality embedded in this infrastructure is what we term the *IT DialTone*. The Open Group creates an environment where all elements involved in technology development can cooperate to deliver less costly and more flexible IT solutions.

Formed in 1996 by the merger of the X/Open Company Ltd. (founded in 1984) and the Open Software Foundation (founded in 1988), The Open Group is supported by most of the world's largest user organizations, information systems vendors, and software suppliers. By combining the strengths of open systems specifications and a proven branding scheme with collaborative technology development and advanced research, The Open Group is well positioned to meet its new mission, as well as to assist user organizations, vendors, and suppliers in the development and implementation of products supporting the adoption and proliferation of systems which conform to standard specifications.

With more than 200 member companies, The Open Group helps the IT industry to advance technologically while managing the change caused by innovation. It does this by:

- Consolidating, prioritizing, and communicating customer requirements to vendors
- Conducting research and development with industry, academia, and government agencies to deliver innovation and economy through projects associated with its Research Institute
- Managing cost-effective development efforts that accelerate consistent multi-vendor deployment of technology in response to customer requirements
- Adopting, integrating, and publishing industry standard specifications that provide an essential set of blueprints for building open information systems and integrating new technology as it becomes available
- Licensing and promoting the Open Brand, represented by the ''X'' Device, that designates vendor products which conform to Open Group Product Standards
- Promoting the benefits of the IT DialTone to customers, vendors, and the public

The Open Group operates in all phases of the open systems technology lifecycle including innovation, market adoption, product development, and proliferation. Presently, it focuses on seven strategic areas: open systems application platform development, architecture, distributed systems management, interoperability, distributed computing environment, security, and the information superhighway. The Open Group is also responsible for the management of the UNIX trademark on behalf of the industry.

Development of Product Standards

This process includes the identification of requirements for open systems and, now, the IT DialTone, development of Technical Standards (formerly CAE and Preliminary Specifications) through an industry consensus review and adoption procedure (in parallel with formal standards work), and the development of tests and conformance criteria.

This leads to the preparation of a Product Standard which is the name used for the documentation that records the conformance requirements (and other information) to which a vendor may register a product.

The ''X'' Device is used by vendors to demonstrate that their products conform to the relevant Product Standard. By use of the Open Brand they guarantee, through the Open Brand Trade Mark License Agreement (TMLA), to maintain their products in conformance with the Product Standard so that the product works, will continue to work, and that any problems will be fixed by the vendor.

Open Group Publications

The Open Group publishes a wide range of technical documentation, the main part of which is focused on development of Technical Standards and product documentation, but which also includes Guides, Snapshots, Technical Studies, Branding and Testing documentation, industry surveys, and business titles.

There are several types of specification:

- *Technical Standards* (formerly *CAE Specifications*)

  The Open Group Technical Standards form the basis for our Product Standards. These Standards are intended to be used widely within the industry for product development and procurement purposes.

  Anyone developing products that implement a Technical Standard can enjoy the benefits of a single, widely supported industry standard. Where appropriate, they can demonstrate product compliance through the Open Brand. Technical Standards are published as soon as they are developed, so enabling vendors to proceed with development of conformant products without delay.

- *CAE Specifications*

  CAE Specifications and Developers' Specifications published prior to January 1998 have the same status as Technical Standards (see above).

- *Preliminary Specifications*

  Preliminary Specifications have usually addressed an emerging area of technology and consequently are not yet supported by multiple sources of stable conformant implementations. They are published for the purpose of validation through implementation of products. A Preliminary Specification is as stable as can be achieved, through applying The Open Group's rigorous development and review procedures.

  Preliminary Specifications are analogous to the *trial-use* standards issued by formal standards organizations, and developers are encouraged to develop products on the basis of them. However, experience through implementation work may result in significant (possibly upwardly incompatible) changes before its progression to becoming a Technical Standard. While the intent is to progress Preliminary Specifications to corresponding Technical Standards, the ability to do so depends on consensus among Open Group members.

- *Consortium and Technology Specifications*

  The Open Group publishes specifications on behalf of industry consortia. For example, it publishes the NMF SPIRIT procurement specifications on behalf of the Network Management Forum. It also publishes Technology Specifications relating to OSF/1, DCE, OSF/Motif, and CDE.

  Technology Specifications (formerly AES Specifications) are often candidates for consensus review, and may be adopted as Technical Standards, in which case the relevant Technology Specification is superseded by a Technical Standard.

In addition, The Open Group publishes:

- *Product Documentation*

  This includes product documentation—programmer's guides, user manuals, and so on—relating to the Pre-structured Technology Projects (PSTs), such as DCE and CDE. It also includes the Single UNIX Documentation, designed for use as common product documentation for the whole industry.

- *Guides*

  These provide information that is useful in the evaluation, procurement, development, or management of open systems, particularly those that relate to the Technical Standards or Preliminary Specifications. The Open Group Guides are advisory, not normative, and should not be referenced for purposes of specifying or claiming conformance to a Product Standard.

- *Technical Studies*

  Technical Studies present results of analyses performed on subjects of interest in areas relevant to The Open Group's Technical Program. They are intended to communicate the findings to the outside world so as to stimulate discussion and activity in other bodies and the industry in general.

Versions and Issues of Specifications

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.

Corrigenda

Readers should note that Corrigenda may apply to any publication. Corrigenda information is published on the World-Wide Web at *http://www.opengroup.org/corrigenda*.

Ordering Information

Full catalogue and ordering information on all Open Group publications is available on the World-Wide Web at *http://www.opengroup.org/pubs*.

This Document

This Technical Standard is based on the Internet Engineering Task force (IETF) Internet Draft version 5 for RAPI, which is a specific Application Programming Interface (API) for the Resource ReSerVation Protocol (RSVP). Rather than develop this into an IETF RFC document, the IETF have deferred to The Open Group to develop RAPI into an open systems standard.

The RAPI interface is one realization of the generic API contained in the RSVP Functional Specification (see referenced document **RFC 2205**).

Intended Audience

This specification is intended for programmers who wish to implement RAPI, and those who wish to write applications to use it.

Structure

- **Chapter 1** defines key terminology usage, and describes the compilation environment, namespace use and reserved identifiers.
- **Chapter 2** provides an overview of the RAPI subject area.
- **Chapter 3** gives the reference page definition for Event Upcalls.
- **Chapter 4** gives the reference page definitions for the Client Library Services.
- **Chapter 5** gives the reference page definitions for the RAPI Formatting Routines.
- **Chapter 6** describes flowspecs, filter specs, sender templates, and sender Tspecs, which are encoded as variable-length RAPI objects.
- **Chapter 7** describes the use of RAPI with *select*( ) and/or *poll*( ).
- **Chapter 8** describes Error Handling.
- **Chapter 9** defines what the header file **<rapi.h>** must contain.
- **Appendix A** gives some general information on the implementation of RAPI which is distributed with the ISI release of RSVP code.

A Glossary and Index are also provided.

Typographical Conventions

The following typographical conventions are used throughout this document:

- **Bold** font is used in text for options to commands, filenames, keywords, and type names.
- *Italic* strings are used for emphasis. Italics in text also denote:
  - — command operands, command option-arguments or variable names, for example, substitutable argument prototypes
  - — environment variables, which are also shown in capitals
  - — utility names
  - — external variables, such as *errno*
  - — functions; these are shown as follows: *name*( ); names without parentheses are C external variables, C function family names, utility names, command operands or command option-arguments.

- Normal font is used for the names of constants and literals.
- The notation **<file.h>** indicates a header.
- Names surrounded by braces, for example, {ARG_MAX}, represent symbolic limits or configuration values which may be declared in appropriate headers by means of the C **#define** construct.
- The notation [ABCD] is used to identify an error value ABCD.
- Syntax, code examples and user input in interactive examples are shown in `fixed width` font. Brackets shown in this font, `[ ]`, are part of the syntax and do *not* indicate optional items. In syntax, the `|` symbol is used to separate alternatives, and ellipses (`...`) are used to show that additional arguments are optional. **`Bold fixed width`** font is used to identify brackets that surround optional items in syntax, **`[ ]`**, and to identify system output in interactive examples.

Trademarks

Acknowledgements

The Open Group gratefully acknowledges the permission of the authors—R.Braden (ISI) and D.Hoffman (Sun Microsystems)—of the IETF Internet Draft for RAPI (An RSVP Application Programming Interface, Version 5, June 16, 1997), for their approval to use their draft as the basis for this RAPI Technical Standard.

Referenced Documents

The following documents are referenced in this Technical Standard:

RFC 791
: Internet Protocol, J. Postel, September 1981. Status: Standard.

RFC 1700
: Assigned Numbers, J Reynolds, J. Postel, October 1994. Status: Standard.

RFC 2205
: Resource ReSerVation Protocol (RSVP) — Version 1 Functional Specification, R. Braden, L. Zhang, S. Berson, September 1997. Status: Proposed Standard.

RFC 2210
: The Use of RSVP with IETF Integrated Services, J. Wroclawski, September 1997. Status: Proposed Standard.

RFC 2211
: Specification of the Controlled-Load Network Element Service, J. Wroclawski. September 1997. Status: Proposed Standard.

RFC 2212
: Specification of Guaranteed Quality of Service, S. Shenker, C. Partridge, R. Guerin, September 1997. Status: Proposed Standard.

RFC 2215
: General Characterization Parameters for Integrated Service Network Elements, S. Shenker, J. Wroclawski, September 1997. Status: Proposed Standard.

XNS, Issue 5
: CAE Specification, February 1997, Networking Services, Issue 5 (ISBN: 1-85912-165-9, C523), published by The Open Group.

XSH, Issue 5
: CAE Specification, January 1997, System Interfaces and Headers, Issue 5 (ISBN: 1-85912-181-0, C606), published by The Open Group.

The following referenced documents are not necessary for implementation of the specification, but provide additional information likely to be of value to implementors or application writers:

RFC 1825
: Security Architecture for the Internet Protocol, commonly known as IPSEC, R. Atkinson, August 1995. Status: Proposed Standard.

RFC 1826
: Internet Protocol Authentication Header, R. Atkinson, August 1995. Status: Draft Standard.

RFC 1827
: Internet Protocol Encapsulating Security Payload (ESP), R. Atkinson, August 1995. Status: Draft Standard.

RFC 2207
: RSVP Extensions for IPSEC Data Flows, L. Berger, T. O'Malley, September 1997.

RFC 2208
: Resource ReSerVation Protocol (RSVP) — Version 1 Applicability Statement: Some Guidelines on Deployment, A. Mankin, Ed., F. Baker, B. Braden, S. Bradner, M. O'Dell, A. Romanow, A. Weinrib, L. Zhang, September 1997. Status: Informational.

RFC 2209
: Resource ReSerVation Protocol (RSVP) — Version 1: Message Processing Rules, R. Braden, L. Zhang, September 1997. Status: Informational.

RFC 2216
: Network Element Service Specification Template, S. Shenker, J. Wroclawski, September 1997. Status: Informational.

Chapter 1

Introduction

1.1 Purpose and Scope

RAPI is a specific Application Programming Interface (API) for the Resource ReSerVation Protocol (RSVP).

The RAPI interface is one realization of the generic API contained in the RSVP Functional Specification (see reference **RFC 2205).** This Resource ReSerVation Protocol (RSVP) describes a resource reservation setup protocol designed for an integrated services Internet. RSVP provides receiver-initiated setup of resource reservations for multicast or unicast data flows. See the RSVP applicability statement in reference **RFC 2210**.

RSVP is designed to be used in conjunction with the Internet Protocol (IP, see reference **RFC 791**). It may be used in conjunction with either IP version 4 (IPv4) or IP version 6 (IPv6) of that protocol. The definition of RAPI assumes use of RSVP in conjunction with IP, and contains some provisions that are IP-specific.

The RAPI interface is a set of C language bindings whose calls are defined in this specification.

1.2 Terminology

The following terms are used in this specification:

can
This describes a permissible optional feature or behavior available to the user or application; all systems support such features or behavior as mandatory requirements.

implementation-dependent
The value or behavior is not consistent across all implementations. The provider of an implementation normally documents the requirements for correct program construction and correct data in the use of that value or behavior. When the value or behavior in the implementation is designed to be variable or customizable on each instantiation of the system, the provider of the implementation normally documents the nature and permissible ranges of this variation. Applications that are intended to be portable must not rely on implementation-dependent values or behavior.

may
With respect to implementations, the feature or behavior is optional. Applications should not rely on the existence of the feature. To avoid ambiguity, the reverse sense of *may* is expressed as *need not*, instead of *may not*.

must
This describes a requirement on the application or user.

should
With respect to implementations, the feature is recommended, but it is not mandatory. Applications should not rely on the existence of the feature.

With respect to users or applications, the word means recommended programming practice that is necessary for maximum portability.

undefined
A value or behavior is undefined if this document imposes no portability requirements on applications for erroneous program constructs or erroneous data. Implementations may specify the result of using that value or causing that behavior, but such specifications are not guaranteed to be consistent across all implementations. An application using such behavior is not fully portable to all systems.

unspecified
A value or behavior is unspecified if this document imposes no portability requirements on applications for correct program construct or correct data. Implementations may specify the result of using that value or causing that behavior, but such specifications are not guaranteed to be consistent across all implementations. An application requiring a specific behavior, rather than tolerating any behavior when using that functionality, is not fully portable to all systems.

will
This means that the behavior described is a requirement on the implementation and applications can rely on its existence.

1.3 Namespace Use

Feature test macro _XOPEN_SOURCE is used to control use of the namespace by implementations of this specification.

All identifiers used in the normative sections of this specification are defined in **<rapi.h>** (see Chapter 9 on page 37). In order to use the API defined in this specification, an application should define macro _XOPEN_SOURCE with a value of at least 500 before including header file **<rapi.h>**. When this is done:

- The symbols specified in Chapter 9 on page 37 are made visible to the application.
- identifiers with the prefixes or suffix shown below are reserved for any use by the implementation: applications must not define or declare identifiers beginning with any of these prefixes or ending with this suffix.
- all other identifiers except those reserved to the implementation by the ISO C Standard are available for use by the application or by implementations of services other than RAPI.

Header	**Prefix**	**Suffix**
<rapi.h>	RAPI_, rapi_, RSVP_, rsvp_, IS_, is_	_t

The behavior of the implementation when _XOPEN_SOURCE is not defined, or is defined with a value of less than 500, is unspecified.

Chapter 2

Overview

2.1 Positioning

An Internet application uses some *API* (Application Programming Interface) in order to request enhanced quality-of-service (QoS). The local implementation then uses the RSVP protocol to propagate the QoS request through the routers along the path(s) for the data flow. Each router may accept or deny the request, depending upon its available resources. In the case of failure, the local implementation will return the decision to the requesting application via the API.

This specification describes a particular RSVP API known as *RAPI*. Applications use RSVP to obtain consistent QoS for end systems in packet-switched internetworking.

2.2 Reservation Model

RSVP is a receiver-oriented signalling protocol that enables applications to request quality of service on an IP network. The types of quality of service applications may request are defined by Integrated Services. RSVP signalling applies to simplex unicast or multicast data flows. Although RSVP distinguishes senders from receivers, the same application may act in both roles.

RSVP assigns QoS to specific IP data flows which can be either multipoint-to-multipoint or point-to-point and are known as *sessions*. A session is defined by a particular transport protocol, IP destination address, and destination port. In order to receive data packets for a particular multicast session, a host must have joined the corresponding IP multicast group.

A data source, or *sender*, is defined by an IP source address and a source port or, alternatively, by an IPv6 source address and flowlabel. A given session may have multiple senders S1, S2, ... Sn, and if the destination is a multicast address, multiple *receivers* R1, R2, ... Rm.

Under RSVP, QoS requests are made by the data receivers. A QoS request contains a *flowspec*, together with a *filter spec*. The flowspec includes an *Rspec*, which defines the desired QoS and is used to control the packet scheduling mechanism in the router or host, and also a *Tspec*, which defines the traffic expected by the receiver. The filter spec controls packet classification to determine which senders' data packets receive the corresponding QoS.

The detailed manner in which reservations from different receivers are shared in the Internet is controlled by a reservation parameter known as the *reservation style*. The RSVP Functional Specification (see referenced document **RFC 2205**) contains a definition and explanation of the different reservation styles.

2.3 API Outline

Using the RAPI interface, an application uses the *rapi_session*( ) call to define an *API session* for sending a single simplex data flow and/or receiving such a data flow. The *rapi_sender*( ) call may then be used to register as a data sender, and/or the *rapi_reserve*( ) call may be used to make a QoS reservation as a data receiver.

The *rapi_sender*( ) or *rapi_reserve*( ) calls may be repeated with different parameters to dynamically modify the state at any time or they can be issued in null forms that retract the corresponding registration. The application can call *rapi_release*( ) to close the session and delete all of its resource reservations. The relationship among the RAPI calls is summarized by the RAPI state diagram shown in Figure 2-1. The calls *rapi_sender*( ) and *rapi_reserve*( ) represent null calls in that diagram.

Note that a single API session, defined by a single *rapi_session*( ) call, can define only one sender at a time. More than one API session may be established for the same RSVP session. For example, suppose an application sends multiple UDP data flows, distinguished by source port. It will call *rapi_session*( ) and *rapi_sender*( ) separately for each of these flows.

The *rapi_session*( ) call allows the application to specify an *upcall* (or *callback*) routine that will be invoked to signal RSVP state changes and error events. There are five types of event:

- RAPI_PATH_EVENT signals the arrival or change of path state.
- RAPI_RESV_EVENT signals the arrival or change of reservation state.
- RAPI_PATH_ERROR signals the corresponding *path* error.
- RAPI_RESV_ERROR signals the corresponding *reservation* error.
- RAPI_RESV_CONFIRM signals the arrival of a CONFIRM message.

Each RAPI implementation must provide and document a mechanism to receive the incoming asynchronous events and call the application upcall function. Those systems which provide either the *select*( ) or *poll*( ) functions must provide the mechanism described in Chapter 7.

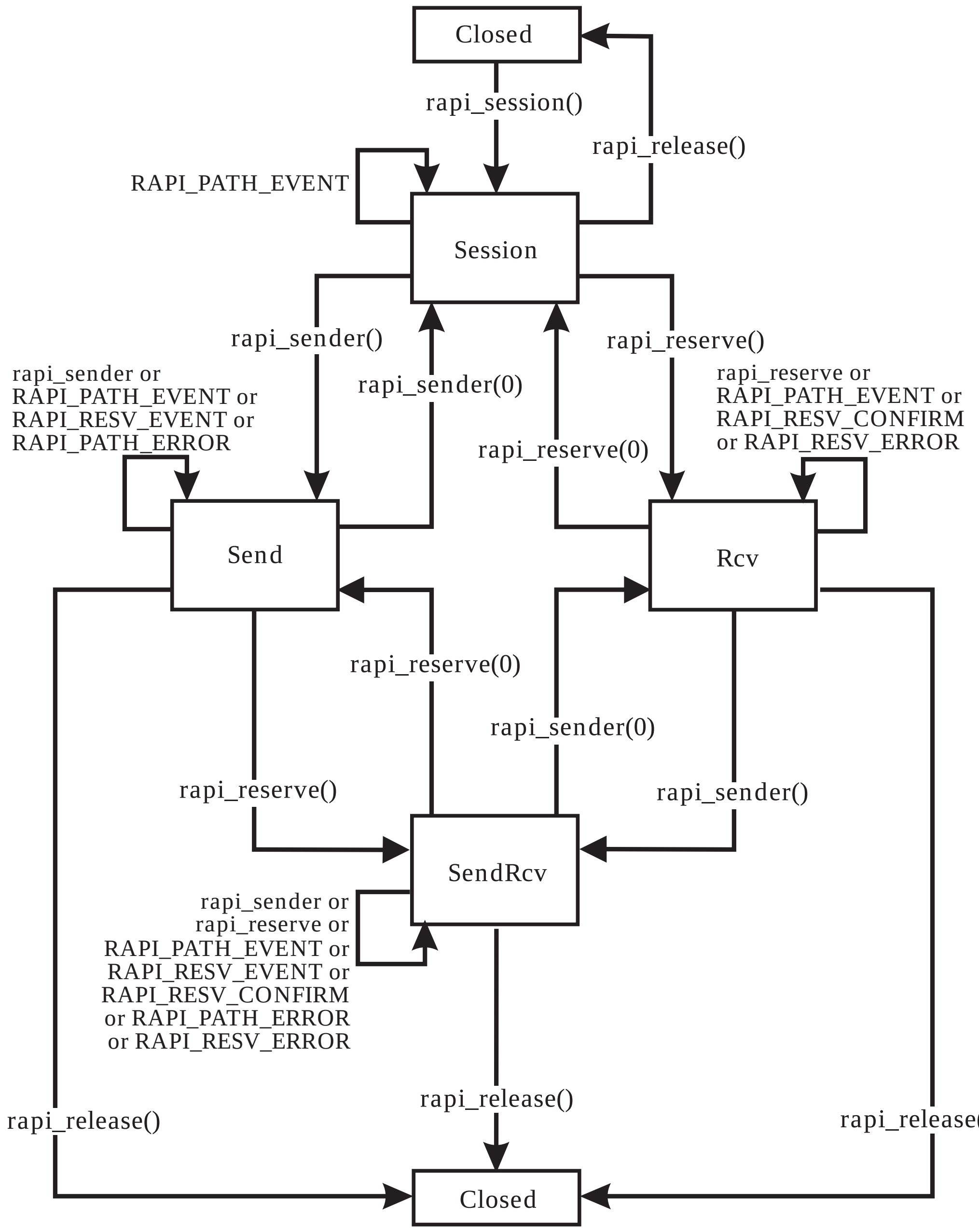
Closed
rapi_session()
rapi_release()
RAPI_PATH_EVENT
Session
rapi_sender()
rapi_sender(0)
rapi_reserve(0)
rapi_reserve()
rapi_sender or RAPI_PATH_EVENT or RAPI_RESV_EVENT or RAPI_PATH_ERROR
rapi_reserve or RAPI_PATH_EVENT or RAPI_RESV_CONFIRM or RAPI_RESV_ERROR
Send
Rcv
rapi_reserve(0)
rapi_sender(0)
rapi_reserve()
rapi_sender()
SendRcv
rapi_sender or rapi_reserve or RAPI_PATH_EVENT or RAPI_RESV_EVENT or RAPI_RESV_CONFIRM or RAPI_PATH_ERROR or RAPI_RESV_ERROR
rapi_release()
rapi_release()
rapi_release()
Closed

Figure 2-1 RAPI State Diagram

A synchronous error in a RAPI routine returns an appropriate error code. Asynchronous RSVP errors are delivered to the application via the RAPI upcall routine.

Error handling is described in Chapter 8.

Chapter 3

Event Upcall

Each RAPI implementation must provide and document a mechanism to receive the incoming asynchronous RSVP events. Those systems which provide either the *select*( ) or *poll*( ) function must provide the mechanism described in Chapter 7 on page 31.

An *upcall* is invoked by the implementation's asynchronous event mechanism. It executes the function whose address was specified in the *Event_rtn* parameter in the *rapi_session*( ) call.

The event upcall function template *rapi_event_rtn_t*( ) is defined in reference manual page format, in the following pages.

NAME

rapi_event_rtn_t — event upcall function template

SYNOPSIS

```
#include <rapi.h>

typedef void    rapi_event_rtn_t(
    rapi_sid_t        Sid,              /* Session ID           */
    rapi_eventinfo_t  EventType,        /* Event type           */
    rapi_styleid_t    Style,            /* Reservation style    */
    int               ErrorCode,        /* Error event: code    */
    int               ErrorValue,       /* Error event: value   */
    rapi_addr_t       *ErrorNode,       /* Node detecting error  */
    unsigned int      ErrorFlags,       /* Error flags          */
    int               FilterSpecNo,     /* number of filterSpecs */
    rapi_filter_t     *FilterSpec_list,
    int               FlowspecNo,       /* number of flowspecs   */
    rapi_flowspec_t   *Flowspec_list,
    int               AdspecNo,         /* number of adspecs     */
    rapi_adspec_t     *Adspec_list,
    void              *Event_arg        /* application argument  */
);
```

DESCRIPTION

This is the template for the function the address of which each application must supply as an argument to *rapi_session*( ) and which the implementation will call when asynchronous RAPI events occur. The event upcall function is invoked from the implementation's asynchronous RSVP event mechanism when an event occurs. See *rapi_session*( ) for more details.

PARAMETERS

Sid

This parameter must be a session ID returned by a successful *rapi_session*( ) call.

EventType

Upcall event type.

Style

This parameter contains the style of the reservation; it is non-zero only for a RAPI_RESV_EVENT or RAPI_RESV_ERROR event.

ErrorCode, ErrorValue

These values encode the error cause, and they are set only for a RAPI_PATH_ERROR or RAPI_RESV_ERROR event. See *rapi_strerror*( ) and Chapter 8 on page 35 (Error Handling) for interpretation of these values.

ErrorNode

This is the IP (V4 or V6) address of the node that detected the error, and it is set only for a RAPI_PATH_ERROR or RAPI_RESV_ERROR event. The format of a *rapi_addr_t* is implementation-dependent. See Chapter 9.

ErrorFlags

These error flags are set only for a RAPI_PATH_ERROR or RAPI_RESV_ERROR event.

RAPI_ERRF_InPlace

The reservation failed, but another (presumably smaller) reservation is still in place on the same interface.

RAPI_ERRF_NotGuilty

The reservation failed, but the request from this client was merged with a larger reservation upstream, so this client's reservation might not have caused the failure.

FilterSpec_list, FilterSpecNo

The *FilterSpec_list* parameter is a pointer to an area containing a sequential vector of RAPI *filter spec* or *sender template* objects. The number of objects in this vector is specified in *FilterSpecNo*. If *FilterSpecNo* is zero, the *FilterSpec_list* parameter will be NULL.

Flowspec_list, FlowspecNo

The *Flowspec_list* parameter is a pointer to an area containing a sequential vector of RAPI *flowspec* or *Tspec* objects. The number of objects in this vector is specified in *FlowspecNo*. If *FlowspecNo* is zero, the *Flowspec_list* parameter will be NULL.

Adspec_list, AdspecNo

The *Adspec_list* parameter is a pointer to an area containing a sequential vector of RAPI *adspec* objects. The number of objects in this vector is specified in *AdspecNo*. If *AdspecNo* is zero, the *Adspec_list* parameter will be NULL.

Event_arg

This is the value supplied in the *rapi_session*( ) call.

RESULT

When the application's upcall function returns, any areas pointed to by *Flowspec_list*, *FilterSpec_list* or *Adspec_list* become invalid for further reference. The upcall function must copy any values it wants to save.

The specific parameters depend upon *EventType*, which may have one of the following values:

RAPI_PATH_EVENT

A *path* event indicates that RSVP sender ("Path") state from a remote node has arrived or changed at the local node. A RAPI_PATH_EVENT event containing the complete current list of senders (or possibly no senders, after a path teardown) in the path state for the specified session will be triggered whenever the path state changes.

FilterSpec_list, *Flowspec_list*, and *Adspec_list* will be of equal length, and corresponding entries will contain *sender templates*, *sender Tspecs*, and *Adspecs*, respectively, for all senders known at this node. In general, a missing object will be indicated by an empty RAPI object.

RAPI_PATH_EVENT events are enabled by the initial *rapi_session*( ) call.

RAPI_RESV_EVENT

A *reservation* event indicates that reservation state has arrived or changed at the node, implying (but not assuring) that reservations have been established or deleted along the entire data path to one or more receivers. RAPI_RESV_EVENT events containing the current reservation state for the API session will be triggered whenever the reservation state changes.

Flowspec_list will either contain one *flowspec* object or be empty (if the state has been torn down), and *FilterSpec_list* will contain zero or more corresponding *filter spec* objects. *Adspec_list* will be empty.

RAPI_RESV_EVENT events are enabled by a *rapi_sender*( ) call; the *sender template* from the latter call will match the *filter spec* returned in the upcall triggered by a *reservation* event.

RAPI_PATH_ERROR

A *path error* event indicates that an asynchronous error has been found in the sender information specified in a *rapi_sender*( ) call.

The *ErrorCode* and *ErrorValue* parameters will specify the error. *FilterSpec_list* and *Flowspec_list* will each contain one object, the *sender template* and corresponding *sender Tspec* (if any) in error, while *Adspec_list* will be empty. If there is no *sender Tspec*, the object in *Flowspec_list* will be an empty RAPI object. The *Adspec_list* will be empty.

RAPI_PATH_ERROR events are enabled by a *rapi_sender*( ) call, and the *sender Tspec* in that call will match the *sender Tspec* returned in a subsequent upcall triggered by a RAPI_PATH_ERROR event.

RAPI_RESV_ERROR
A *reservation error* event indicates that an asynchronous reservation error has occurred.

The *ErrorCode* and *ErrorValue* parameters will specify the error. *Flowspec_list* will contain one *flowspec*, while *FilterSpec_list* may contain zero or more corresponding filter specs. *Adspec_list* will be empty.

RAPI_RESV_ERROR events are enabled by a call to *rapi_reserve*( ).

RAPI_RESV_CONFIRM
A RAPI_RESV_CONFIRM event indicates that a reservation has been made at least up to an intermediate merge point, and probably (but not necessarily) all the way to at least one sender. A RAPI_RESV_CONFIRM event is enabled by a *rapi_reserve*( ) call with the RAPI_REQ_CONFIRM flag set, and at most one *confirmation* event will result from each such call.

The parameters of a RAPI_RESV_CONFIRM event upcall are the same as those for a RAPI_RESV_EVENT event upcall.

The accompanying table summarizes the events. Here, "n" is a non-negative integer.

Event	**Enabled by**	**FilterSpecNo**	**FlowspecNo**	**AdspecNo**
RAPI_PATH_EVENT	rapi_session	n	n	n
RAPI_PATH_ERROR	rapi_sender	1	1	0
RAPI_RESV_EVENT	rapi_sender	n	1 or 0	0
RAPI_RESV_ERROR	rapi_reserve	n	1	0
RAPI_RESV_CONFIRM	rapi_reserve	1	1	0

Chapter 4

Client Library Services

The RSVP API provides the following client library calls:

- *rapi_release*( )
- *rapi_reserve*( )
- *rapi_sender*( )
- *rapi_session*( )
- *rapi_strerror*( )
- *rapi_version*( )

These are defined in the following reference pages.

To use these calls, the application must include the file **<rapi.h>**. See Chapter 9.

NAME

rapi_release — remove a session

SYNOPSIS

```
#include <rapi.h>

int rapi_release (rapi_sid_t Sid)
```

DESCRIPTION

The *rapi_release*( ) call removes the reservation, if any, and the state corresponding to a given session handle. This call will be made implicitly if the application terminates without closing its RSVP sessions.

PARAMETERS

Sid

This parameter must be a session ID returned by a successful *rapi_session*( ) call.

RESULT

If the session handle is invalid, the call returns a corresponding RAPI error code; otherwise, it returns zero.

NAME

rapi_reserve — make, modify, or delete a reservation

SYNOPSIS

```
#include <rapi.h>

int rapi_reserve(
    rapi_sid_t       Sid,             /* Session ID            */
    unsigned int     Flags,           /* Flags                 */
    rapi_addr_t     *RHost,           /* Receive host addr     */
    rapi_styleid_t   StyleId,         /* Style ID              */
    rapi_stylex_t   *Style_Ext,       /* Style extension       */
    rapi_policy_t   *Rcvr_Policy,     /* Receiver policy       */
    int              FilterSpecNo,    /* Number of filter specs */
    rapi_filter_t   *FilterSpec_list, /* List of filter specs   */
    int              FlowspecNo,      /* Number of flowspecs    */
    rapi_flowspec_t *Flowspec_list    /* List of flowspecs      */
)
```

DESCRIPTION

The *rapi_reserve*( ) function is called to make, modify, or delete a resource reservation for a session. The call may be repeated with different parameters, allowing the application to modify or remove the reservation; the latest call will take precedence.

PARAMETERS

Sid

This parameter must be a session ID returned by a successful *rapi_session*( ) call.

Flags

Setting the RAPI_REQ_CONFIRM flag requests confirmation of the reservation, by means of a confirmation upcall (type RAPI_RESV_CONFIRM).

RHost

This parameter may be used to define the interface address on which data will be received for multicast flows. It is useful for a multi-homed host. If it is NULL or the host address is INADDR_ANY, an implementation-defined interface will be chosen. The format of a *rapi_addr_t* is implementation-dependent, see Chapter 9.

StyleId

This parameter specifies the reservation style id (values defined below).

Style_Ext

This parameter is a pointer to a style-dependent extension to the parameter list, or NULL.

Rcvr_Policy

This parameter is a pointer to a policy data structure, or it is NULL.

FilterSpec_list, FilterSpecNo

The *FilterSpec_list* parameter is a pointer to an area containing a sequential vector of RAPI filter spec objects. The number of objects in this vector is specified in *FilterSpecNo*. If *FilterSpecNo* is zero, the *FilterSpec_list* parameter is ignored and can be NULL.

Flowspec_list, FlowspecNo

The *Flowspec_list* parameter is a pointer to an area containing a sequential vector of RAPI flow spec objects. The number of objects in this vector is specified in *FlowspecNo*. If *FlowspecNo* is zero, the *Flowspec_list* parameter is ignored and can be NULL.

If *FlowspecNo* is zero, the *rapi_reserve*( ) call will remove the current reservation(s) for the specified session, and *FilterSpec_list* and *Flowspec_list* will be ignored. Otherwise, the parameters depend upon the style, as follows:

Wildcard Filter (WF)

Use *StyleId* = RAPI_RSTYLE_WILDCARD. The *Flowspec_list* parameter may be NULL (to delete the reservation) or else point to a single flowspec. The *FilterSpec_list* parameter may be empty or it may point to a single filter spec containing appropriate wildcard(s).

Fixed Filter (FF)

Use *StyleId* = RAPI_RSTYLE_FIXED. *FilterSpecNo* must equal *FlowspecNo*. Entries in *Flowspec_list* and *FilterSpec_list* parameters will correspond in pairs.

Shared Explicit (SE)

Use *StyleId* = RAPI_RSTYLE_SE. The *Flowspec_list* parameter should point to a single flowspec. The *FilterSpec_list* parameter may point to a list of any length.

RESULT

Depending upon the parameters, each call may or may not result in new *admission control* calls, which could fail asynchronously.

If there is a synchronous error in this call, *rapi_reserve*( ) returns a RAPI error code; otherwise, it returns zero.

Applications that make use of the RAPI_RESV_CONFIRM flag will normally receive positive acknowledgement that the QoS request succeeded. Otherwise, applications measure success in the form of errors returned when making QoS requests. No final positive acknowledgement will occur.

An *admission control* failure (for example, refusal of the QoS request) is reported asynchronously by an upcall of type RAPI_RESV_ERROR. A RSVP_Err_NO_PATH error code indicates that RSVP state from one or more of the senders specified in *FilterSpec_list* has not (yet) propagated all the way to the receiver; it may also indicate that one or more of the specified senders has closed its API session and that its RSVP state has been deleted from the routers.

NAME

rapi_sender — specify sender parameters

SYNOPSIS

```
#include <rapi.h>

int rapi_sender(
    rapi_sid_t       Sid,            /* Session ID         */
    unsigned int     Flags,          /* Flags              */
    rapi_addr_t     *LHost,          /* Local Host         */
    rapi_filter_t   *SenderTemplate, /* Sender template    */
    rapi_tspec_t    *SenderTspec,    /* Sender Tspec       */
    rapi_adspec_t   *SenderAdspec,   /* Sender Adspec      */
    rapi_policy_t   *SenderPolicy,   /* Sender policy data */
    int              TTL             /* Multicast data TTL */
)
```

DESCRIPTION

An application must issue a *rapi_sender*( ) call if it intends to send a flow of data for which receivers may make reservations. This call defines, redefines, or deletes the parameters of that flow. A *rapi_sender*( ) call may be issued more than once for the same API session; the most recent one takes precedence.

Once a successful *rapi_sender*( ) call has been made, the application may receive upcalls of type RAPI_RESV_EVENT or RAPI_PATH_ERROR.

PARAMETERS

Sid

This parameter must be a session ID returned by a successful *rapi_session*( ) call.

Flags

No flags are currently defined for this call.

LHost

This parameter may point to a *rapi_addr_t* structure specifying the IP (v4 or v6) source address and, if applicable, the source port or flowlabel from which data will be sent, or it may be NULL. The format of a *rapi_addr_t* is implementation-dependent, see Chapter 9.

If the IP source address is INADDR_ANY, the API will use the default IP address of the local host. This is sufficient unless the host is multi-homed. The port number may be zero if the protocol for the session does not have ports.

A NULL *LHost* parameter indicates that the application wishes to withdraw its registration as a sender. In this case, the following parameters will all be ignored.

SenderTemplate

This parameter may be a pointer to a RAPI filter specification structure (see Chapter 6) specifying the format of data packets to be sent, or it may be NULL.

If this parameter is NULL, a sender template will be created internally from the *Dest* and *LHost* parameters. The *Dest* parameter was supplied in an earlier *rapi_session*( ) call. If a *SenderTemplate* parameter is present, the (non-NULL) *LHost* parameter is ignored.

This parameter must be non-NULL in order to declare the sender template for a session using IPSEC, that is, a session created with the RAPI_GPI_SESSION flag set.

SenderTspec

This parameter is a pointer to a *Tspec* that defines the traffic that this sender will create, and must not be NULL.

SenderAdspec

This parameter may point to a RAPI Adspec structure (see Chapter 6), or it may be NULL.

SenderPolicy

This parameter may be a pointer to a sender policy data structure, or it may be NULL.

TTL

This parameter specifies the IP TTL (Time-to-Live) value with which multicast data will be sent. It allows RSVP to send its control messages with the same TTL scope as the data packets.

RESULT

If there is a synchronous error, *rapi_sender*( ) returns a RAPI error code; otherwise, it returns zero.

NAME

rapi_session — create a session

SYNOPSIS

```
#include <rapi.h>

rapi_sid_t rapi_session(
    rapi_addr_t      *Dest,       /* Session: (Dst addr, port)  */
    int               Protid,     /* Protocol Id                */
    unsigned int      Flags,      /* Flags                      */
    rapi_event_rtn_t *Event_rtn,  /* Address of upcall routine  */
    void             *Event_arg,  /* App argument to upcall     */
    int              *Errnop      /* Place to return error code */
)
```

DESCRIPTION

The *rapi_session*( ) call creates an API session.

After a successful *rapi_session*( ) call has been made, the application may receive upcalls of type RAPI_PATH_EVENT for the API session.

PARAMETERS

The parameters are as follows:

Dest

This parameter points to a rapi_addr_t structure defining the destination IP (V4 or V6) address and a port number to which data will be sent. The *Dest* and *Protid* parameters define an RSVP session. If the *Protid* specifies UDP or TCP transport, as specified in the Assigned Numbers **RFC 1700**, the port corresponds to the appropriate transport port number. The format of a *rapi_addr_t* is implementation-dependent, see Chapter 9.

Protid

The IP protocol ID for the session. If it is omitted (that is, zero), 17 (UDP) is assumed.

Flags

RAPI_GPI_SESSION

If set, this flag requests that this API session be defined in the GPI format used by the IPSEC extension of RSVP. If this flag is set, the port number included in *Dest* is considered "virtual" (see the IPSEC specification, reference **RFC 1825** for details), and any sender template and filter specifications must be in GPI format.

RAPI_USE_INTSERV

If set, *IntServ* formats are used in upcalls; otherwise, the *Simplified* format is used (see Chapter 6).

Event_rtn

This parameter is a pointer to an upcall function that will be invoked to notify the application of RSVP errors and state change events. Pending events cause the invocation of the upcall function — see Chapter 7). The application must supply an upcall routine for event processing. See Chapter 3.

Event_arg

This parameter is an argument that will be passed to any invocation of the upcall routine. See Chapter 3.

Errnop

The address of an integer into which a RAPI error code will be returned.

RESULT

If it succeeds, the *rapi_session*( ) call returns an opaque but non-zero session handle for use in subsequent calls related to this API session.

If the call fails synchronously, it returns zero (RAPI_NULL_SID) and stores a RAPI error code into the integer variable pointed to by the *Errnop* parameter.

EXTENDED DESCRIPTION

An application can have multiple API sessions registered for the same or different RSVP sessions at the same time. There can be at most one sender associated with each API session; however, an application can announce multiple senders for a given RSVP session by announcing each sender in a separate API session.

Two API sessions for the same RSVP session, if they are receiving data, are assumed to have joined the same multicast group and will receive the same data packets. An implementation should disallow multiple API sessions for the same sender within one RSVP session (see Section 2.2 on page 3). The behavior of an implementation that allows multiple API sessions for the same sender is unspecified.

NAME

rapi_strerror — get RAPI error message string

SYNOPSIS

```
#include <rapi.h>

const char *rapi_strerror(int ErrorCode, int ErrorValue)
```

DESCRIPTION

This call maps the error code and value to an error message string, and returns a pointer to that string.

RESULT

This call returns NULL if the arguments are out of bounds, otherwise a pointer to an error message string.

NAME

rapi_version — RAPI version

SYNOPSIS

```
#include <rapi.h>

int rapi_version(void)
```

DESCRIPTION

This call obtains the version of the interface. It may be used by an application to adapt to different versions.

RESULT

This call returns a single integer that defines the version of the interface. The returned value is composed of a major number and a minor number, encoded as:

```
100 * major + minor
```

The API described in this specification has major version number 6.

Chapter 5

RAPI Formatting Routines

For convenience of applications, RAPI includes standard routines for displaying the contents of RAPI objects. These functions are optional; a conforming implementation need not provide them.

These standard formatting routines are:

- *rapi_fmt_adspec*( )
- *rapi_fmt_filtspec*( )
- *rapi_fmt_flowspec*( )
- *rapi_fmt_tspec*( )

This Chapter presents the reference pages which define these standard formatting routines.

To use these routines, the application must include the header file **<rapi.h>**. See Chapter 9.

NAME

rapi_fmt_adspec — format an adspec

SYNOPSIS

```
#include <rapi.h>

void rapi_fmt_adspec(
    rapi_adspec_t    *adspecp,  /* Addr of RAPI Adspec */
    char             *buffer,   /* Addr of buffer      */
    int               length    /* Length of buffer    */
)
```

DESCRIPTION

The *rapi_fmt_adspec*( ) call formats a given RAPI *Adspec* into a buffer of given address and length. The output is truncated if the length is too small.

NAME

rapi_fmt_filtspec — format a filter spec

SYNOPSIS

```
#include <rapi.h>

void rapi_fmt_filtspec(
    rapi_filter_t    *filtp,    /* Addr of RAPI Filt Spec */
    char             *buffer,   /* Addr of buffer         */
    int               length    /* Length of buffer       */
)
```

DESCRIPTION

The *rapi_fmt_filtspec*( ) call formats a given RAPI *filter spec* into a buffer of given address and length. The output is truncated if the length is too small.

NAME

rapi_fmt_flowspec — format a flowspec

SYNOPSIS

```
#include <rapi.h>

void rapi_fmt_flowspec(
    rapi_flowspec_t  *specp,   /* Addr of RAPI flowspec */
    char             *buffer,  /* Addr of buffer        */
    int               length   /* Length of buffer      */
)
```

DESCRIPTION

The *rapi_fmt_flowspec*( ) call formats a given RAPI *flowspec* into a buffer of given address and length. The output is truncated if the length is too small.

NAME

rapi_fmt_tspec — format a tspec

SYNOPSIS

```
#include <rapi.h>

void rapi_fmt_tspec(
    rapi_tspec_t    *tspecp,  /* Addr of RAPI Tspec  */
    char            *buffer,  /* Addr of buffer      */
    int              length   /* Length of buffer    */
)
```

DESCRIPTION

The *rapi_fmt_tspec*( ) call formats a given RAPI *Tspec* into a buffer of given address and length. The output is truncated if the length is too small.

Chapter 6

RAPI Objects

Flowspecs, *filter specs*, *sender templates*, and *sender Tspecs* are encoded as variable-length RAPI objects.

Every RAPI object begins with a header of type *rapi_hdr_t*, which contains:

- The total length of the object in bytes.
- The type.

An empty object consists only of a header, with type zero and length *sizeof (rapi_hdr_t)*.

Integrated services data structures are defined in referenced document **RFC 2210**, which describes the use of RSVP with the Controlled-Load and Guaranteed services. RSVP defines several data objects which carry resource reservation information but are opaque to RSVP itself. The usage and data format of those objects is given in the referenced document **RFC 2210**.

6.1 Flowspecs

There are two formats for RAPI *flowspecs*. For further details, see **<rapi.h>** in Chapter 9.

6.1.1 RAPI_FLOWSTYPE_Simplified

This is a *simplified* format. It consists of a simple list of parameters needed for either *Guaranteed* or *Controlled Load* service, using the service type QOS_GUARANTEED or QOS_CNTR_LOAD, respectively.

The RAPI client library routines map this format to/from an appropriate Integrated Services data structure.

6.1.2 RAPI_FLOWSTYPE_Intserv

This *flowspec* must be a fully formatted Integrated Services flowspec data structure.

6.1.3 Upcalls

In an upcall, a *flowspec* is by default delivered in *simplified* format. However, if the RAPI_USE_INTSERV flag was set in the *rapi_session*( ) call, then the *IntServ* format is used in upcalls.

6.2 Sender Tspecs

There are two formats for RAPI *Sender Tspecs*. For further details, see **<rapi.h>** in Chapter 9.

6.2.1 RAPI_TSPECTYPE_Simplified

This is a *simplified* format, consisting of a simple list of parameters with the service type QOS_TSPEC. The RAPI client library routines map this format to/from an appropriate Integrated Services data structure.

6.2.2 RAPI_TSPECTYPE_Intserv

This flowspec must be a fully formatted Integrated Services *Tspec* data structure.

6.2.3 Upcalls

In an upcall, a *sender Tspec* is by default delivered in *simplified* format. However, if the RAPI_USE_INTSERV flag was set in the *rapi_session*( ) call, then the *IntServ* format is used in upcalls.

6.3 Adspecs

There are two formats for RAPI *Adspecs*. For further details, see **<rapi.h>** in Chapter 9.

6.3.1 RAPI_ADSTYPE_Simplified

This is a *simplified* format, consisting of a list of *Adspec* parameters for all possible services. The RAPI client library routines maps this format to/from an appropriate Integrated Services data structure.

6.3.2 RAPI_ADSTYPE_Intserv

This *flowspec* must be a fully formatted Integrated Services *Tspec* data structure.

6.3.3 Upcalls

In an upcall, an *Adspec* is by default delivered in *simplified* format. However, if the RAPI_USE_INTSERV flag was set in the *rapi_session*( ) call, then the *IntServ* format is used in upcalls.

6.4 Filter Specs and Sender Templates

There are two formats for these objects:

- RAPI_FILTERFORM_BASE (RAPI_FILTERFORM_BASE6)
  This object consists of only a socket address structure, defining the IPv4 address and port or, alternatively, the IPv6 address and flowlabel.
- RAPI_FILTERFORM_GPI (RAPI_FILTERFORM_GPI6)
  This object consists of only an address structure, defining the IP V4 (or V6) address and a 32-bit Generalized Port Identifier. It is recommended for all IPSEC applications.

  Other non-TCP/non-UDP transports may also utilize this format in the future.

6.5 Policy Data Objects

Under RSVP, applications may need to supply policy information containing permission rights to obtain the preferential access to the network that they request. The *rapi_policy_t* type is a framework to carry the aforementioned integrity policy object.

Chapter 7

Use with select() or poll()

When a system provides either *select*( ) or *poll*( ) (see the referenced **XSH** specification and the referenced **XNS** specification), the RAPI implementation must provide two additional calls: *rapi_getfd*( ) and *rapi_dispatch*( ).

The upcall routine is invoked indirectly (and synchronously) by the application, using the following mechanism:

- The application issues the RAPI library call *rapi_getfd*( ) to learn the file descriptor of the endpoint (for example, socket) used by the API.
- The application detects read events on this file descriptor, either passing it directly in a select or poll call or passing it to the notifier of another library (such as XLib, tk / tcl, RPC).
- When a read event on the file descriptor is signaled, the application should call *rapi_dispatch*( ). This causes the API to execute the upcall routine if appropriate.

The remainder of this Chapter presents the definitions for:

- *rapi_dispatch*( )
- *rapi_getfd*( )

NAME

rapi_dispatch — dispatch API event

SYNOPSIS

```
#include <rapi.h>

int rapi_dispatch(void)
```

DESCRIPTION

The application should call this routine whenever a read event is signaled on a file descriptor returned by *rapi_getfd*( ). The *rapi_dispatch*( ) routine may be called at any time, but it will generally have no effect unless there is a pending event.

RESULT

Calling this routine may result in one or more upcalls to the application from any of the open API sessions.

If this call encounters an error, *rapi_dispatch*( ) returns a RAPI error code; otherwise, it returns zero. See Chapter 8 for a list of error codes.

NAME

rapi_getfd — get file descriptor

SYNOPSIS

```
#include <rapi.h>

int rapi_getfd(rapi_sid_t Sid)
```

DESCRIPTION

After a *rapi_session*( ) call has completed successfully and before *rapi_release*( ) has been called, the application may call *rapi_getfd*( ) to obtain the file descriptor associated with that session. When a read event is signalled on this file descriptor, the application should call *rapi_dispatch*( ).

Note: Calls to *rapi_getfd*( ) for different RAPI sessions may return the same file descriptor.

PARAMETERS

Sid

This parameter must be a session ID returned by a successful *rapi_session*( ) call.

RESULT

If *Sid* is illegal or undefined, this call returns –1; otherwise, it returns the file descriptor.

Chapter 8

Error Handling

8.1 Introduction

Errors can be detected **synchronously** or **asynchronously**.

When an error is detected synchronously, a *RAPI error code* is returned via the *Errnop* argument of *rapi_session*( ), or as the function return value of *rapi_sender*( ), *rapi_reserve*( ), *rapi_release*( ) or *rapi_dispatch*( ).

When an error is detected asynchronously, it is indicated by a RAPI_PATH_ERROR or RAPI_RESV_ERROR event. An RSVP error code and error value are then contained in the *ErrorCode* and *ErrorValue* arguments of the event upcall function. In case of an *API error* (RSVP error code 20), a RAPI error code is contained in the *ErrorValue* argument.

A description of RSVP error codes and values can be found in Appendix B of the referenced **RFC 2205**.

8.2 RAPI Error Codes

[RAPI_ERR_OK]	No error
[RAPI_ERR_INVAL]	Invalid parameter
[RAPI_ERR_MAXSESS]	Too many sessions
[RAPI_ERR_BADSID]	Session identity out of legal range
[RAPI_ERR_N_FFS]	Wrong filter number or flow number for style
[RAPI_ERR_BADSTYLE]	Illegal reservation style
[RAPI_ERR_SYSCALL]	A system error has occurred; its nature may be indicated by *errno*
[RAPI_ERR_OVERFLOW]	Parameter list overflow
[RAPI_ERR_MEMFULL]	Not enough memory
[RAPI_ERR_NORSVP]	RSVP implementation internal error
[RAPI_ERR_OBJTYPE]	Invalid object type
[RAPI_ERR_OBJLEN]	Invalid object length
[RAPI_ERR_NOTSPEC]	No sender Tspec
[RAPI_ERR_INTSERV]	Invalid Integrated Services parameter format
[RAPI_ERR_GPI_CONFLICT]	IPSEC: Conflicting C-type
[RAPI_ERR_BADPROTO]	IPSEC: Protocol not AH or ESP
[RAPI_ERR_BADVDPORT]	IPSEC: vDstPort is zero
[RAPI_ERR_GPISESS]	IPSEC: invalid parameters for GPI_SESSION flag, or other parameter error

[RAPI_ERR_BADSEND]	Sender address not my interface
[RAPI_ERR_BADRECV]	Receiver address not my interface
[RAPI_ERR_BADSPORT]	Invalid source port: the source port is non-zero when the destination port is zero.
[RAPI_ERR_UNSUPPORTED]	Unsupported feature
[RAPI_ERR_UNKNOWN]	Unknown error

Note that [RAPI_ERR_BADSEND], [RAPI_ERR_BADRECV] and [RAPI_ERR_BADSPORT] occur only asynchronously, as the *ErrorValue* when the *ErrorCode* is 20 (API error).

8.3 RSVP Error Codes

Symbol	Value	Meaning
RSVP_Err_NONE	0	No error (confirmation)
RSVP_Err_ADMISSION	1	Admission control failure
RSVP_Err_POLICY	2	Policy control failure
RSVP_Err_NO_PATH	3	No path information
RSVP_Err_NO_SENDER	4	No sender information
RSVP_Err_BAD_STYLE	5	Conflicting style
RSVP_Err_UNKNOWN_STYLE	6	Unknown style
RSVP_Err_BAD_DSTPORT	7	Conflicting destination port in session
RSVP_Err_BAD_SNDPORT	8	Conflicting source port
	9	reserved
	10	reserved
	11	reserved
RSVP_Err_PREEMPTED	12	Service preempted
RSVP_Err_UNKN_OBJ_CLASS	13	Unknown object class
RSVP_Err_UNKNOWN_CTYPE	14	Unknown object C-Type
	15	reserved
	16	reserved
	17	reserved
	18	reserved
	19	reserved
RSVP_Err_API_ERROR	20	API error
RSVP_Err_TC_ERROR	21	Traffic control error
RSVP_Err_TC_SYS_ERROR	22	Traffic control system error
RSVP_Err_RSVP_SYS_ERROR	23	RSVP system error

Chapter 9

Header File

9.1 Integer and Floating Point Types

Types *uint8_t*, *uint16_t* and *uint32_t* which appear in the **<rapi.h>** header file are unsigned integer types of length 8, 16 and 32 bits, respectively. They may be made available by inclusion of **<inttypes.h>** (see the referenced **XSH** specification).

Type *float32_t* is a floating-point type of length 32 bits. It is defined by including the **<rapi.h>** header file.

9.2 The <rapi.h> Header

This header file contains the definitions of the RSVP API (RAPI) library calls.

Inclusion of this header may make available other symbols in addition to those specified in this section.

9.2.1 General Definitions

When header **<rapi.h>** is included:

i. Macro RAPI_VERSION is defined with value 100 * *major* + *minor*, where *major* is the major version number and *minor* is the minor version number. The value of RAPI_VERSION is returned by *rapi_version*( ).

ii. Type *rapi_addr_t* is defined for protocol addresses. Implementations of RAPI that are intended to be used in conjunction with the sockets interface defined in the referenced **XNS** specification must define *rapi_addr_t* to be *struct sockaddr*. Implementations that are intended to be used in conjunction with other networking APIs may define *rapi_addr_t* differently.

    Note: If they do so, they will need to add further members to the *rapi_format_t* enumeration, and to the *filt_u* union in the *rapi_filter_t* structure, and possibly to make further additions.

iii. Enumeration *rapi_qos_service_t* is defined by typedef and has at least the following members.

Member	**Meaning**
RAPI_QOS_TSPEC	Generic Tspec
RAPI_QOS_CNTR_LOAD	Controlled-load service
RAPI_QOS_GUARANTEED	Guaranteed service

iv. Enumeration *rapi_format_t* is defined by typedef and has at least the following members.

Member	**Meaning**
RAPI_EMPTY_OTYPE	Empty object
RAPI_FLOWSTYPE_Intserv	Int-Serv format flowspec
RAPI_FLOWSTYPE_Simplified	Simplified format flowspec
RAPI_TSPECTYPE_Intserv	Int-Serv format (sndr)Tspec
RAPI_TSPECTYPE_Simplified	Simplified format (sndr)Tspec
RAPI_ADSTYPE_Intserv	Int-Serv format Adspec
RAPI_ADSTYPE_Simplified	Simplified format Adspec
RAPI_FILTERFORM_BASE	Simple V4: Only *sockaddr*
RAPI_FILTERFORM_GPI	IPV4 GPI filter format
RAPI_FILTERFORM_BASE6	Simple V6: Only *sockaddr*
RAPI_FILTERFORM_GPI6	IPV6 GPI filter format

v. Type *rapi_hdr_t* is defined by typedef as a structure to represent a generic RAPI object header. It has the following members, followed by type-specific contents.

Member	**Type**	**Usage**
len	unsigned int	Actual length in bytes
form	rapi_format_t	Format

vi. The following macros are defined with the values given below.

Macro	**Value**
RAPIObj_Size(p)	(((rapi_hdr_t *)(p))→len)
RAPIObj_data(p)	((rapi_hdr_t *)(p)+1)
After_RAPIObj(p)	((char *)(p) + RAPIObj_Size(p))

9.2.2 Tspec Definitions

When header **<rapi.h>** is included:

i. Type *rapi_qos_Tspec_body* is defined by typedef as a structure with at least the following members.

Member	**Type**	**Usage**
spec_Tspec_b	float32_t	Token bucket depth in bytes
spec_Tspec_r	float32_t	Token bucket average rate in bytes per second
spec_Tspec_p	float32_t	Peak data rate in bytes per second
spec_Tspec_m	uint32_t	Minimum policed unit in bytes
spec_Tspec_M	uint32_t	Maximum packet size in bytes

ii. Type *rapi_qos_tspecx_t* is defined by typedef as a structure that contains the generic Tspec parameters, and has at least the following members.

Member	Type	Usage
spec_type	rapi_qos_service_t	QoS_service_type
xtspec_Tspec	rapi_qos_Tspec_body	Tspec

iii. The following macros are defined with the values given below.

Macro	Value
xtspec_r	xtspec_Tspec.spec_Tspec_r
xtspec_b	xtspec_Tspec.spec_Tspec_b
xtspec_p	xtspec_Tspec.spec_Tspec_p
xtspec_m	xtspec_Tspec.spec_Tspec_m
xtspec_M	xtspec_Tspec.spec_Tspec_M

iv. Type *rapi_tspec_t* is defined by typedef as a structure to represent a Tspec descriptor, and has at least the following members.

Member	Type	Member	Type	Usage
len	unsigned int			Actual length in bytes
form	rapi_format_t			tspec format
tspecbody_u	union			
		qosxt	rapi_qos_tspecx_t	Simplified format Tspec
		ISt	IS_tspbody_t	Int-serv format Tspec

v. The following macros are defined with the values given below.

Macro	Value
tspecbody_qosx	tspecbody_u.qosxt
tspecbody_IS	tspecbody_u.ISt

9.2.3 Flowspec Definitions

When header **<rapi.h>** is included:

i. Type *rapi_qos_flowspecx_t* is defined by typedef as a structure that contains the union of the parameters for *controlled-load service* and *guaranteed service* models, and has at least the following members.

Member	Type	Usage
spec_type	rapi_qos_service_t	QoS_service_type
xspec_Tspec	rapi_qos_Tspec_body	Tspec
xspec_R	float32_t	Rate in bytes per second
xspec_S	uint32_t	Slack term in microseconds

ii. The following macros are defined with the values given below.

Macro	Value
xspec_r	xspec_Tspec.spec_Tspec_r
xspec_b	xspec_Tspec.spec_Tspec_b
xspec_p	xspec_Tspec.spec_Tspec_p
xspec_m	xspec_Tspec.spec_Tspec_m
xspec_M	xspec_Tspec.spec_Tspec_M

iii. Type *rapi_flowspec_t* is defined by typedef as a structure to represent a Flowspec descriptor, and has at least the following members.

Member	Type	Member	Type	Usage
len	unsigned int			Actual length in bytes
form	rapi_format_t			flowspec format
specbody_u	union			
		qosx	rapi_qos_flowspecx_t	Simplified format flowspec
		IS	IS_specbody_t	Int-serv format flowspec

iv. The following macros are defined with the values given below.

Macro	Value
specbody_qosx	specbody_u.qosx
specbody_IS	specbody_u.IS

9.2.4 Adspec Definitions

When header **<rapi.h>** is included:

i. Type *rapi_qos_adspecx_t* is defined by typedef as a structure that contains the union of all *adspec* parameters for *controlled-load service* and *guaranteed service* models, and has at least the following members.

Member	Type	Usage
		General path characterization parameters
xaspec_flags	uint8_t	flags(1)
xaspec_hopcnt	uint16_t	
xaspec_path_bw	float32_t	
xaspec_min_latency	uint32_t	
xaspec_composed_MTU	uint32_t	
		Controlled-load service Adspec parameters
xClaspec_flags	uint8_t	Flags
xClaspec_override	uint8_t	See note(2)
xClaspec_hopcnt	uint16_t	
xClaspec_path_bw	float32_t	
xClaspec_min_latency	uint32_t	
xClaspec_composed_MTU	uint32_t	
		Guaranteed service Adspec parameters
xGaspec_flags	uint8_t	Flags
xGaspec_Ctot	uint32_t	
xGaspec_Dtot	uint32_t	
xGaspec_Csum	uint32_t	
xGaspec_Dsum	uint32_t	
xGaspec_override	uint8_t	See note(2)
xGaspec_hopcnt	uint16_t	
xGaspec_path_bw	float32_t	
xGaspec_min_latency	uint32_t	
xGaspec_composed_MTU	uint32_t	

Notes:

(1) FLG_IGN is not allowed; FLG_PARM is assumed.

(2) A value of 1 means "override all generic parameters".

ii. The following macros are defined with bitwise-distinct integral values for use in the *xaspec_flags*, *xClaspec_flags*, and *xGaspec_flags* fields.

Macro	Meaning
RAPI_XASPEC_FLG_BRK	Break bit: service unsupported in some node
RAPI_XASPEC_FLG_IGN	Ignore flag: do not include this service
RAPI_XASPEC_FLG_PARM	Parms-present flag: include service parameters

iii. Type *rapi_adspec_t* is defined by typedef as a structure to represent an Adspec descriptor, and has at least the following members.

Member	Type	Member	Type	Usage
len	unsigned int			Actual length in bytes
form	rapi_format_t			adspec format
adsbody_u	union			
		adsx	rapi_qos_adspecx_t	Simplified format adspec
		ISa	IS_adsbody_t	Int-serv format adspec

iv. The following macros are defined with the values given below.

Macro	Value
adspecbody_qosx	adsbody_u.adsx
adspecbody_IS	adsbody_u.ISa

9.2.5 Filter Spec Definitions

When header **<rapi.h>** is included:

i. Type *rapi_filter_base_t* is defined by typedef as a structure that contains at least the following member:

Member	Type
sender	struct sockaddr_in

ii. Type *rapi_filter_gpi_t* is defined by typedef as a structure that contains at least the following members.

Member	Type
sender	struct in_addr
gpi	uint32_t

iii. Type *rapi_filter_base6_t* is defined by typedef as a structure that contains at least the following member.

Member	Type
sender	struct sockaddr_in6

iv. Type *rapi_filter_gpi6_t* is defined by typedef as a structure that contains at least the following members.

Member	Type
sender	struct in6_addr
gpi	uint32_t

v. Type *rapi_filter_t* is defined by typedef as a structure that contains at least the following members.

Member	Type	Member	Type	Usage
len	unsigned int			Actual length in bytes
form	rapi_format_t			filterspec format
filt_u	union(1)			
		base	rapi_filter_base_t	
		gpi	rapi_filter_gpi_t	
		base6	rapi_filter_base6_t	
		gpi6	rapi_filter_gpi6_t	

Note:

(1) variable length

vi. The following macros are defined with the values given below.

Macro	Value
rapi_filt4	filt_u.base.sender
rapi_filtbase4_addr	rapi_filt4.sin_addr
rapi_filtbase4_port	rapi_filt4.sin_port
rapi_filtgpi4	filt_u.gpi
rapi_filtgpi4_addr	rapi_filtgpi4.sender
rapi_filtgpi4_gpi	rapi_filtgpi4.gpi
rapi_filt6	filt_u.base6.sender
rapi_filtbase6_addr	rapi_filt6.sin6_addr
rapi_filtbase6_port	rapi_filt6.sin6_port
rapi_filtgpi6	filt_u.gpi6
rapi_filtgpi6_addr	rapi_filtgpi6.sender
rapi_filtgpi6_gpi	rapi_filtgpi6.gpi

9.2.6 Policy Definitions

When header **<rapi.h>** is included, type *rapi_policy_t* is defined by typedef as a structure that contains at least the following members:

Member	Type
len	unsigned int
form	rapi_format_t
pol_u	union

9.2.7 Reservation Style Definitions

When header **<rapi.h>** is included:

i. Enumeration *rapi_styleid_t* is defined by typedef for reservation style identifiers, and has at least the following members.

Member	Meaning
RAPI_RSTYLE_WILDCARD	Reservation will be shared among a wildcard selection of senders
RAPI_RSTYLE_FIXED	Reservation will not be shared and will be dedicated to a particular sender
RAPI_RSTYLE_SE	Reservation will be shared among an explicit list of senders

ii. Type *rapi_stylex_t* is defined by typedef as *void*.

Note: It is intended that this identifier be used in future for a reservation style extension structure that will allow passing of style-specific parameters for possible new styles.

9.2.8 Function Interface Definitions

When header **<rapi.h>** is included:

i. Macro RAPI_NULL_SID is defined for error returns from *rapi_session*( ).

ii. The following macros are defined and evaluate to bitwise-distinct integral values.

Constant	Meaning
RAPI_USE_INTSERV	Use Int-Serv fmt in upcalls
RAPI_GPI_SESSION	Use GPI session format
RAPI_REQ_CONFIRM	Request confirmation

Enumeration *rapi_eventinfo_t* is defined by typedef for RAPI event types, and has at least the following members.

Member
RAPI_PATH_EVENT
RAPI_RESV_EVENT
RAPI_PATH_ERROR
RAPI_RESV_ERROR
RAPI_RESV_CONFIRM

Identifiers RAPI_PATH_STATUS and RAPI_RESV_STATUS are reserved for use by the implementation.

iii. The following macros are defined and evaluate to distinct integral values.

Constant	Meaning
RAPI_ERRF_InPlace	Left reservation in place
RAPI_ERRF_NotGuilty	This receiver not guilty

iv. Type *rapi_event_rtn_t* is defined by typedef as a function that conforms to the prototype defined in the reference manual page definition for *event upcall* (see Chapter 3).

v. Prototypes for each function defined in this specification must be provided in **<rapi.h>**.

vi. The following macros are defined and evaluate to distinct integral values for use as RAPI error codes. Macro RAPI_ERR_OK (which indicates that there is no error) evaluates to zero. The meanings of these error codes are described in Section 8.2 on page 35.

```
RAPI_ERR_OK
RAPI_ERR_INVAL
RAPI_ERR_MAXSESS
RAPI_ERR_BADSID
RAPI_ERR_N_FFS
RAPI_ERR_BADSTYLE
RAPI_ERR_SYSCALL
RAPI_ERR_OVERFLOW
RAPI_ERR_MEMFULL
RAPI_ERR_NORSVP
RAPI_ERR_OBJTYPE
RAPI_ERR_OBJLEN
RAPI_ERR_NOTSPEC
RAPI_ERR_INTSERV
RAPI_ERR_GPI_CONFLICT
RAPI_ERR_BADPROTO
RAPI_ERR_BADVDPORT
RAPI_ERR_GPISESS

RAPI_ERR_BADSEND
RAPI_ERR_BADRECV
RAPI_ERR_BADSPORT

RAPI_ERR_UNSUPPORTED
RAPI_ERR_UNKNOWN
```

vii. The following macros are defined and evaluate to the RSVP error code values as defined in Section 8.3 on page 36.

```
RSVP_Err_NONE
RSVP_Err_ADMISSION
RSVP_Err_POLICY
RSVP_Err_NO_PATH
RSVP_Err_NO_SENDER
RSVP_Err_BAD_STYLE
RSVP_Err_UNKNOWN_STYLE
RSVP_Err_BAD_DSTPORT
RSVP_Err_BAD_SNDPORT
RSVP_Err_PREEMPTED
RSVP_Err_UNKN_OBJ_CLASS
RSVP_Err_UNKNOWN_CTYPE
RSVP_Err_API_ERROR
RSVP_Err_TC_ERROR
RSVP_Err_TC_SYS_ERROR
RSVP_Err_RSVP_SYS_ERROR
```

9.3 Integrated Services Data Structures and Macros

Note: This section defines the integrated services (see reference **RFC 2210**) data formats. The designers of the RAPI interface wanted to allow an application to specify either the *int-serv* format of a flowspec, Tspec, or adspec, or a "simplified" version of each. It would certainly be simpler if RAPI supported only one of these formats, but this was not felt to be safe in the long run.

The simplified versions allow almost any *int-serv* version to be generated, but there may be circumstances in which this is not adequate. For example, more general forms of flowspec, containing more than one service, may be defined in future (so that in case the Resv message reaches a node which does not implement service A, it can drop back to service B). Allowing an application to specify the body of an arbitrary *int-serv* data object allows for such contingencies.

Nevertheless, the result is not entirely satisfactory, and future versions of this specification may change the definitions in this section. Application writers are therefore advised not to use these definitions except where absolutely necessary.

These Integrated Services data structures and macros are made available when header **<rapi.h>** is included.

Note that the values in the data structures defined in this section are in host byte order.

Inclusion of this header may make available other symbols in addition to those specified in this section.

9.3.1 General Definitions

When header **<rapi.h>** is included:

i. The following macro is defined with the value given below.

Macro	Value	Usage
wordsof(x)	(((x)+3)/4)	Number of 32-bit words

ii. The following macros are defined with the following integer values for service numbers.

Note: The values are protocol values defined in referenced documents **RFC 2211**, **RFC 2212**, and **RFC 2215**.

Macro	Value
GENERAL_INFO	1
GUARANTEED_SERV	2
CONTROLLED_LOAD_SERV	5

iii. Enumeration *int_serv_wkp* is defined for well-known parameter identities and has at least the following members with the following integer values.

Note: The values are protocol values defined in reference **RFC 2215**.

Member	Value	Meaning
IS_WKP_HOP_CNT	4	Number of network nodes supporting Integrated Services along the flow path
IS_WKP_PATH_BW	6	Available bandwidth in bytes per second throughout the flow path
IS_WKP_MIN_LATENCY	8	Minimum end-to-end latency in microseconds
IS_WKP_COMPOSED_MTU	10	Maximum transmission unit without causing IP fragmentation along the flow path
IS_WKP_TB_TSPEC	127	Token bucket TSPEC parameter

iv. The following macros are defined with the values given below.

Macro	Value
INTSERV_VERS_MASK	0xf0
INTSERV_VERSION0	0
Intserv_Version(x)	(((x)&INTSERV_VERS_MASK)>>4)
Intserv_Version_OK(x)	(((x)→ismh_version&INTSERV_VERS_MASK)== \ INTSERV_VERSION0)

v. Type *IS_main_hdr_t* is defined by typedef as a structure to represent an Integrated Services main header, and has at least the following members.

Member	Type	Usage
ismh_version	uint8_t	Version
ismh_unused	uint8_t	
ismh_len32b	uint16_t	Number of 32-bit words excluding this header

vi. Type *IS_serv_hdr_t* is defined by typedef as a structure to represent an Integrated Services service element header, and has at least the following members.

Member	Type	Usage
issh_service	uint8_t	Service number
issh_flags	uint8_t	Flag byte
issh_len32b	uint16_t	Number of 32-bit words excluding this header

vii. The following macro is defined with the value given below to indicate the *break* bit in the *IS_serv_hdr_t* flag byte.

Macro	Value
ISSH_BREAK_BIT	0x80

viii. Type *IS_parm_hdr_t* is defined by typedef as a structure to represent an Integrated Services parameter element header, and has at least the following members.

Member	Type	Usage
isph_parm_num	uint8_t	Parameter number
isph_flags	uint8_t	Flags
isph_len32b	uint16_t	Number of 32-bit words excluding this header

ix. The following macro is defined with the value given below to indicate the *invalid* bit in the *IS_parm_hdr_t* flag byte.

Macro	Value
ISPH_FLG_INV	0x80

x. The following macros are defined with the values given below.

Macro	Value
Set_Main_Hdr(p, len)	{(p)→ismh_version = INTSERV_VERSION0; \ (p)→ismh_unused = 0; \ (p)→ismh_len32b = wordsof(len); }
Set_Serv_Hdr(p, s, len)	{(p)→issh_service = (s); \ (p)→issh_flags = 0; \ (p)→issh_len32b = wordsof(len); }
Set_Parm_Hdr(p, id, len)	{(p)→isph_parm_num = (id); \ (p)→isph_flags = 0; \ (p)→isph_len32b = wordsof(len); }
Set_Break_Bit(p)	(((IS_serv_hdr_t *)p)→issh_flags | =ISSH_BREAK_BIT)
Next_Main_Hdr(p)	(IS_main_hdr_t *)((uint32_t *)(p)+1+(p)→ismh_len32b)
Next_Serv_Hdr(p)	(IS_serv_hdr_t *)((uint32_t *)(p)+1+(p)→issh_len32b)
Next_Parm_Hdr(p)	(IS_parm_hdr_t *)((uint32_t *)(p)+1+(p)→isph_len32b)
Non_Is_Hop(p)	(((IS_serv_hdr_t *)p)→issh_flags & ISSH_BREAK_BIT)

9.3.2 Generic Tspec format

When header **<rapi.h>** is included:

i. The following macros define constraints on the *token bucket* parameters for both the Controlled-Load and Guaranteed service. These constraints are imposed by the respective service specifications and are not an indication of what minimum or maximum values a RAPI implementation will accept.

The following macros are defined with values of type *float32_t*.

Macro	Usage	Value
TB_MIN_RATE	Minimum token bucket rate	1 byte per second
TB_MAX_RATE	Maximum token bucket rate	40 terabytes per second
TB_MIN_DEPTH	Minimum token bucket depth	1 byte
TB_MAX_DEPTH	Maximum token bucket depth	250 gigabyte

ii. Type *TB_Tsp_parms_t* is defined by typedef as a structure to represent generic Tspec parameters, and has at least the following members.

Member	Type	Usage
TB_Tspec_r	float32_t	Token bucket rate in bytes per second
TB_Tspec_b	float32_t	Token bucket depth in bytes
TB_Tspec_p	float32_t	Peak data rate in bytes per second
TB_Tspec_m	uint32_t	Minimum policed unit in bytes
TB_Tspec_M	uint32_t	Maximum packet size in bytes

iii. Type *gen_Tspec_t* is defined by typedef as a structure to represent a generic Tspec, and has at least the following members.

Member	Type	Usage
gen_Tspec_serv_hdr	IS_serv_hdr_t	(GENERAL_INFO, length)
gen_Tspec_parm_hdr	IS_parm_hdr_t	(IS_WKP_TB_TSPEC,)
gen_Tspec_parms	TB_Tsp_parms_t	

iv. The following macros are defined with the values given below.

Macro	Value
gtspec_r	gen_Tspec_parms.TB_Tspec_r
gtspec_b	gen_Tspec_parms.TB_Tspec_b
gtspec_m	gen_Tspec_parms.TB_Tspec_m
gtspec_M	gen_Tspec_parms.TB_Tspec_M
gtspec_p	gen_Tspec_parms.TB_Tspec_p
gtspec_parmno	gen_Tspec_parm_hdr.isph_parm_num
gtspec_flags	gen_Tspec_parm_hdr.isph_flags
gtspec_len	(sizeof(gen_Tspec_t) - sizeof(IS_serv_hdr_t))

9.3.3 Formats for Controlled-Load Service

When header **<rapi.h>** is included:

i. Type *CL_flowspec_t* is defined by typedef as a structure to represent a controlled-load Flowspec, and has at least the following members.

Member	Type	Usage
CL_spec_serv_hdr	IS_serv_hdr_t	(CONTROLLED_LOAD_SERV, 0, len)
CL_spec_parm_hdr	IS_parm_hdr_t	(IS_WKP_TB_TSPEC,)
CL_spec_parms	TB_Tsp_parms_t	

ii. The following macros are defined with the values given below.

Macro	Value
CLspec_r	CL_spec_parms.TB_Tspec_r
CLspec_b	CL_spec_parms.TB_Tspec_b
CLspec_p	CL_spec_parms.TB_Tspec_p
CLspec_m	CL_spec_parms.TB_Tspec_m
CLspec_M	CL_spec_parms.TB_Tspec_M
CLspec_parmno	CL_spec_parm_hdr.isph_parm_num
CLspec_flags	CL_spec_parm_hdr.isph_flags
CLspec_len32b	CL_spec_parm_hdr.isph_len32b
CLspec_len	(sizeof(CL_flowspec_t) - sizeof(IS_serv_hdr_t))

9.3.4 Formats for Guaranteed Service

When header **<rapi.h>** is included:

i. The following enumeration is defined for service-specific parameter identifiers and has at least the following members with the following values:

Member	Value
IS_GUAR_RSPEC	130
GUAR_ADSPARM_C	131
GUAR_ADSPARM_D	132
GUAR_ADSPARM_Ctot	133
GUAR_ADSPARM_Dtot	134
GUAR_ADSPARM_Csum	135
GUAR_ADSPARM_Dsum	136

ii. Type *guar_Rspec_t* is defined by typedef as a structure for guaranteed Rspec parameters, and has at least the following members.

Member	Type	Usage
Guar_R	float32_t	Guaranteed rate in bytes per second
Guar_S	uint32_t	Slack term in microseconds

iii. Type *Guar_flowspec_t* is defined by typedef as a structure to represent a guaranteed Flowspec, and has at least the following members.

Member	Type	Usage
Guar_serv_hdr	IS_serv_hdr_t	(GUARANTEED_SERV, 0, length)
Guar_Tspec_hdr	IS_parm_hdr_t	(IS_WKP_TB_TSPEC,)
Guar_Tspec_parms	TB_Tsp_parms_t	GENERIC Tspec parameters
Guar_Rspec_hdr	IS_parm_hdr_t	(IS_GUAR_RSPEC,)
Guar_Rspec	guar_Rspec_t	Guaranteed rate in bytes per second

iv. The following macros are defined with the values given below.

Macro	Value
Gspec_r	Guar_Tspec_parms.TB_Tspec_r
Gspec_b	Guar_Tspec_parms.TB_Tspec_b
Gspec_p	Guar_Tspec_parms.TB_Tspec_p
Gspec_m	Guar_Tspec_parms.TB_Tspec_m
Gspec_M	Guar_Tspec_parms.TB_Tspec_M
Gspec_R	Guar_Rspec.Guar_R
Gspec_S	Guar_Rspec.Guar_S
Gspec_T_parmno	Guar_Tspec_hdr.isph_parm_num
Gspec_T_flags	Guar_Tspec_hdr.isph_flags
Gspec_R_parmno	Guar_Rspec_hdr.isph_parm_num
Gspec_R_flags	Guar_Rspec_hdr.isph_flags
Gspec_len	(sizeof(Guar_flowspec_t) - sizeof(IS_serv_hdr_t))

v. Type *Gads_parms_t* is defined by typedef as a structure for guaranteed Adspec parameters, and has the following members which may be followed by override general parameter values.

Member	Type	Usage
Gads_serv_hdr	IS_serv_hdr_t	(GUARANTEED_SERV, x, len)
Gads_Ctot_hdr	IS_parm_hdr_t	(GUAR_ADSPARM_Ctot,)
Gads_Ctot	uint32_t	
Gads_Dtot_hdr	IS_parm_hdr_t	(GUAR_ADSPARM_Dtot,)
Gads_Dtot	uint32_t	
Gads_Csum_hdr	IS_parm_hdr_t	(GUAR_ADSPARM_Csum,)
Gads_Csum	uint32_t	
Gads_Dsum_hdr	IS_parm_hdr_t	(GUAR_ADSPARM_Dsum,)
Gads_Dsum	uint32_t	

9.3.5 Basic Adspec Pieces

When header **<rapi.h>** is included:

i. Type *genparm_parms_t* is defined by typedef as a structure for general path characterization parameters, and has at least the following members.

Member	Type	Usage
gen_parm_hdr	IS_serv_hdr_t	(GENERAL_INFO, len)
gen_parm_hopcnt_hdr	IS_parm_hdr_t	(IS_WKP_HOP_CNT,)
gen_parm_hopcnt	uint32_t	
gen_parm_pathbw_hdr	IS_parm_hdr_t	(IS_WKP_PATH_BW,)
gen_parm_path_bw	float32_t	
gen_parm_minlat_hdr	IS_parm_hdr_t	(IS_WKP_MIN_LATENCY,)
gen_parm_min_latency	uint32_t	
gen_parm_compmtu_hdr	IS_parm_hdr_t	(IS_WKP_COMPOSED_MTU,)
gen_parm_composed_MTU	uint32_t	

ii. Type *Min_adspec_t* is defined by typedef as a structure to represent a minimal Adspec per-service fragment - an empty service header - and has at least the following member.

Member	Type	Usage
mads_hdr	IS_serv_hdr_t	(<service>, 1, len=0)

9.3.6 Integrated Services Flowspec

When header **<rapi.h>** is included:

i. Type *IS_specbody_t* is defined by typedef as a structure to represent an Integrated Services Flowspec, and has at least the following members.

Member	Type	Member	Type	Usage
spec_mh	IS_main_hdr_t			
spec_u	union			
		CL_spec	CL_flowspec_t	Controlled-load service
		G_spec	Guar_flowspec_t	Guaranteed service

ii. The following macros are defined with the values given below.

Macro	Value
ISmh_len32b	spec_mh.ismh_len32b
ISmh_version	spec_mh.ismh_version
ISmh_unused	spec_mh.ismh_unused

9.3.7 Integrated Services Tspec

When header **<rapi.h>** is included:

i. Type *IS_tspbody_t* is defined by typedef as a structure to represent an Integrated Services Tspec, and has at least the following members.

Member	Type	Member	Type	Usage
st_mh	IS_main_hdr_t			
tspec_u	union(1)			
		gen_stspec	gen_Tspec_t	Generic Tspec

Note:

(1) While it is possible that there could be service-dependent Tspecs, there are in fact none.

ii. The following macros are defined with the values given below.

Macro	Value
IStmh_len32b	st_mh.ismh_len32b
IStmh_version	st_mh.ismh_version
IStmh_unused	st_mh.ismh_unused

9.3.8 Integrated Services Adspec

When header **<rapi.h>** is included, type *IS_adsbody_t* is defined by typedef as a structure to represent a (minimal) Integrated Services Adspec, and has the following members, followed by variable-length fragments for some or all services. These can be minimal length fragments.

Member	Type	Usage
adspec_mh	IS_main_hdr_t	Main header
adspec_genparms	genparm_parms_t	General char parameter fragment

Appendix A

Example Implementation

This Appendix contains some general remarks based on ISI's experience in implementing this API with their release of RSVP code.

A.1 Protocols

There are three protocol interfaces involved in invoking RSVP via the API:

- Procedure Call Interface to Application

  The term *RAPI* (RSVP API) is used for the function call interface to applications, and for the data structures (*objects*) used in that interface. This document is primarily concerned with the RAPI interface. This interface is realized by functions included in the library routine **librsvp.a**, which is compiled from **rapi_lib.c** and **rapi_fmt.c**.

- Application—Daemon Protocol

  The term *API* is used in the code for the local protocol across the socket between the **librsvp.a** routines and the RSVP daemon *rsvpd*. This protocol generally uses RSVP object bodies but RAPI object framing.

- RSVP Protocol

  The RSVP protocol is used in the Internet between RSVP daemon programs.

The code is organized to make these three interfaces logically independent, so they can be changed independently. Each of these three protocol interfaces has an independent version number, defined in **<rapi.h>** and **<rsvp.h>**, for RAPI and RSVP, respectively.

The RAPI call library **librsvp.a** includes routines which convert objects between RAPI and API formats. Similarly, the file **rsvp_api.c** included in the RSVP daemon includes routines that convert between the API representation and the RSVP representation. In some cases, these conversion functions are identity transformations (that is, pure copies). However, they provide the structure to allow any of the three interfaces to be changed in the future.

There are two different object framing conventions. RAPI and API objects have a 2-word header — a total length in bytes, and a format code — and a body. RSVP objects have a 1-word header. In general, objects in the API interface (that is, across the socket) carry the 2-word RAPI object header, but their body is that of the corresponding RSVP object. Therefore, the API↔RSVP conversion in **rsvp_api.c** simply maps the framing convention.

In the RAPI interface, the application is given some choice of data formats. For example, QoS control objects (that is, *flowspecs*, *Tspecs*, and *Adspecs*) can be represented in either the RSVP (really *Int-Serv*) format, which has complex packing, or in the more convenient *Simplified* format. The RAPI library routines map between *Simplified* format and *Int-Serv* format, which is used across the API.

A.2 RAPI Sessions

Each instance of the RAPI library routines keeps a local (to the application process) table of open RAPI sessions. The index into this table is the session handle (*a_sid*) used locally.

The RSVP daemon keeps its own table of RAPI sessions. From the daemon's viewpoint, a RAPI session is defined by the triple (*fd*, *pid*, *a_sid*), where *fd* is the file descriptor for the socket, *pid* is the process id, and *a_sid* is an application session id received over *fd* from *pid*.

The first *rapi_session*( ) call in a particular instance of the RAPI library opens a UNIX-domain RAPI socket to the RSVP daemon and passes the session registration request across it. If the application (or the daemon) crashes without properly closing the RAPI socket, the other side will be notified to perform a cleanup. In particular, if the user process terminates without explicitly closing the RAPI session, the daemon will delete the corresponding reservation state from the routers.

A.3 Implementation Restrictions

In the ISI reference implementation of RSVP:

- The RAPI_FILTERFORM_GPI and RAPI_FILTERFORM_GPI objects and the session flag RAPI_GPI_SESSION were implemented in RAPI and the API, but the IPSEC extensions were not fully implemented in RSVP.
- The *SenderAdspec* and *SenderPolicy* parameters in *rapi_sender*( ) were not implemented.
- The *Style_Ext* and *Rcvr_Policy* parameters in *rapi_reserve*( ) were not implemented.

A.4 Implementation Model

Figure A-1 shows RAPI's implementation model.

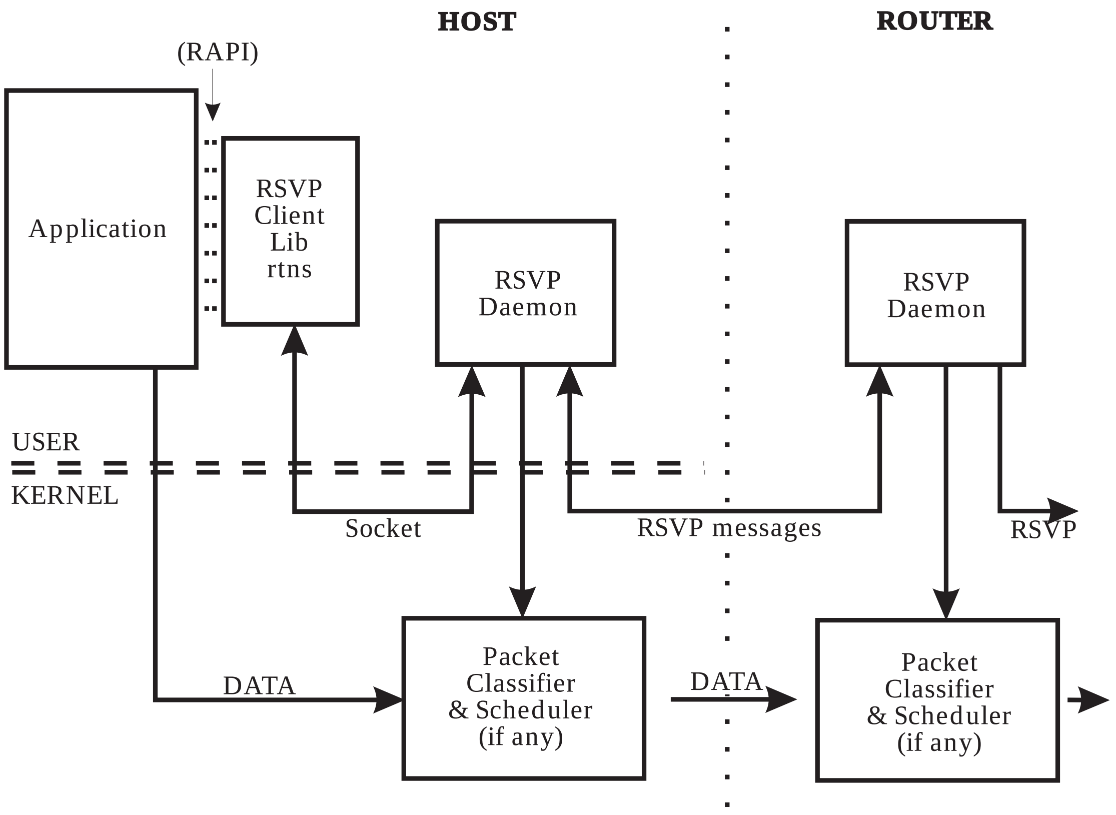

Figure A-1 ISI RAPI Implementation Model

Glossary

The reader is advised to consult the current list of Internet Official Protocol Standards (IETF Std 1) for the status of the RFCs listed below, and to review any RFCs that supersede them.

The reader may also want to check Internet Drafts in relevant IETF working groups for the latest developments.

See reference **RFC 2205** for a more extensive glossary of RSVP terms.

Adspec
A data element (object) in a Path message that carries a package of OPWA advertising information. See "One Pass With Advertising ".

AH
Abbreviation for IP "Authentication Header". See reference **RFC 1826**.

API
Application Programming Interface

C-type
A sub-identifier within an object class. A class identifier and C-type together uniquely identify an RSVP object.

Controlled-Load service
An integrated service that provides a quality of service that closely resembles to "best effort" service under unloaded conditions. See reference **RFC 2211**.

Error term C
The rate-dependent error term of the flow in a network element. It represents the delay a datagram in the flow might incur in a network element that can be directly attributed to the rate parameters of the flow. The error term C is measured in units of bytes.

Error term Csum
The cumulative sum of the error terms C of the flow since the most recent reshaping point upstream from the receiver. See reference **RFC 2212**.

Error term Ctot
The end-to-end sum of the error terms C of the flow. See reference **RFC 2212**.

Error term D
The rate-independent term of the flow in a network element. It represents the worst case non-rate-based transit delay through the network element. The error term D is measured in units of microseconds.

Error term Dsum
The cumulative sum of the error terms D of the flow since the most recent reshaping point upstream from the receiver. See reference **RFC 2212**.

Error term Dtot
The end-to-end sum of the error terms D of the flow. See reference **RFC 2212**.

ESP
Abbreviation for IP "Encapsulation Security Payload". See reference **RFC 1827**.

Filter Spec
Together with the session information, defines the set of data packets to receive the QoS specified in a flowspec. The filter spec is used to set parameters in the packet classifier function.

Flow
The set of data packets defined by a session specification together with a Filter Spec.

Flow descriptor
The combination of a flowspec and a filter spec.

Flowspec
Defines the QoS to be provided for a flow. The flowspec is used to set parameters in the packet scheduling function to provide the requested quality of service.

Guaranteed service
An integrated service that guarantees both bandwidth and firm end-to-end delay bounds. See reference **RFC 2212**.

GPI
Abbreviation for "Generalized Port Identifier". When it is used in the context of IP Security, it refers to the Security Parameter Index, or SPI. See reference **RFC 2207**.

IETF
Internet Engineering Task Force

INTSERV
Integrated Services

IP
Internet Protocol, version 4 (IPv4, see reference **RFC 791**) is in current use at the time of publication of this Technical Standard. Version 6 (IPv6) is the "new generation" Internet Protocol being specified by the IETF to replace IPv4.

IPSEC
Abbreviation for IP Security Protocol. See reference **RFC 1825**.

ISI
Information Sciences Institute of the School of Engineering at the University of Southern California

Minimum policed unit
The size in bytes of the smallest packet in the flow. The packet size includes the application data and all protocol headers at or above the IP level. This is used by network elements to compute maximum bandwidth overhead needed to carry a flow’s packets over a particular link technology.

One Pass With Advertising (OPWA)
Describes a reservation setup model in which (Path) messages sent downstream gather information that the receiver(s) can use to predict the end-to-end service. The information that is gathered is called an advertisement. See also "Adspec".

OPWA
Abbreviation for "One Pass With Advertising".

Override
A flag indicating whether the general path characterization parameters are overridden by parameters specific to a given QoS service. A non-zero value will indicate at least one of the general path characterization parameters is to be overridden.

Peak data rate
Sender’s peak traffic generation rate if it is known and controlled. It may be set to the sender’s outgoing interface link rate if it is known, or it may be set to positive infinity if no better value is available.

QoS
Quality of Service.

RAPI
RSVP API

RAPI Session
A RAPI session is an API session created by a call to *rapi_session*( ) and terminated by a call to *rapi_release*( ).

Rspec
The component of a flowspec that defines a desired QoS.

RSVP
Resource ReSerVation Protocol

RSVP Session
An RSVP session defines one simplex unicast or multicast data flow for which reservations are required. An RSVP session is identified by the destination address, transport-layer protocol, and an optional (generalized) destination port.

Session
See "RAPI Session" and "RSVP Session".

Slack term
The delay delta between the desired delay and the delay that can be obtained by using a reservation bandwidth level R as specified in the Rspec. The slack term is measured in units of microseconds. See reference **RFC 2212**.

Token bucket depth
Size of an abstract linear first-in-first-out buffer for holding a sender's data before the data is transmitted. The token bucket depth is one of the several parameters used to define the traffic specification of a data flow. See reference **RFC 2211**, and reference **RFC 2212**.

Token bucket rate
The average data rate that a sender will be sending into the flow. The token bucket rate is measured in bytes of IP datagrams per second. The IP header and any other upper layer protocol header are included in this measurement.

TSpec
A traffic parameter set that describes a flow.

TTL
An abbreviation for the IP "Time to Live" field. This one-octet field in the IP protocol header controls the maximum distance the payload datagram can be forwarded. The implication of time is historical; in the current Internet, the TTL field is simply a count of hops the datagram may travel before being discarded.

vDstPort
Abbreviation for "Virtual Destination Port". It is used to de-multiplex RSVP sessions beyond the IP destination address for IPSEC flows. See reference **RFC 2207**.

Index