Total Pages 73

# Joint Tactical Radio System (JTRS) Standard Packet Application Program Interface (API)


## Version: 2.0.2
## 02 April 2008

**REVISION HISTORY**

| Version | Authorization | Description | Last Modified Date |
|---|---|---|---|
| 0.1 | | Initial release | 04-May-2006 |
| 0.2 | | Added sequence diagram. | 19-May-2006 |
| 0.3 | | Revised sequence diagram | 02-June-2006 |
| 0.4 | | Updates based on Increment 4d review comments. | 11-July-2006 |
| 1.0 | | Updates based on Increment 4d ICWG.<br>**ICWG Approved** | 07-August-2006 |
| 1.1 | | Added Flow Control, Queue, and Priority Queue Extensions. | 11-October-2006 |
| 1.2 | | Removed Queue and Priority Queue Extensions; Refactored EmptyControl/Signals interfaces. | 26-October-2006 |
| 1.3 | | Updates based on Increment 6 review comments. | 06-December-2006 |
| 2.0 | | Added Empty Signals Extension; Added text to Section A.5.1<br>**ICWG Approved** | 13-December-2006 |
| 2.0.1 | | Preparation for public release | 29-March-2007 |
| 2.0.2 | | Errata: In PacketEmptySignals.idl removed extra "d" from the parameter "streamdId" | 02-April-2008 |

## Table of Contents

# Table of Contents

# Lists of Figures

# Lists of Tables

# A. PACKET API

## A.1 INTRODUCTION

This document defines a common set of *Packet* interfaces to be used by Joint Tactical Radio (JTR) Set Applications and Services. The *Packet* interfaces provide methods to push two-way packets of common data types to the device or service user. It also provides other methods to configure and query the packet payload sizes.

The *Packet* interfaces are documented within to minimize coupling between the device and service interfaces that utilize these *Packet* interfaces.

### A.1.1 Overview

This document contains as follows:

a. Section A.1, *Introduction*, contains the introductory material regarding the overview, service layer description, modes, states, and referenced documents of this document.

b. Section A.2, *Services,* specifies the interfaces for the component, port connections, and sequence diagrams.

c. Section A.3, *Service Primitives and Attributes,* specifies the operations that are provided by the *Packet* interfaces.

d. Section A.4, *IDL*.

e. Section A.5, *UML*.

f. Appendix A.A*Abbreviations and Acronyms*.

g. Appendix A.B*Performance Specification*.

## A.1.2 Service Layer Description

### A.1.2.1 Packet Port Diagram

Figure 1 and port definitions (below) shows a generic bidirectional instantiation of the packet interfaces to be specified by a service or device. A unidirectional instantiation may be achieved by only specifying the "uses" or the "provides" ports. Note: "*[DataType]*" is meant to be replaced by the data type defining the packet stream (i.e. *octet, ushort*). The port names are for reference only.

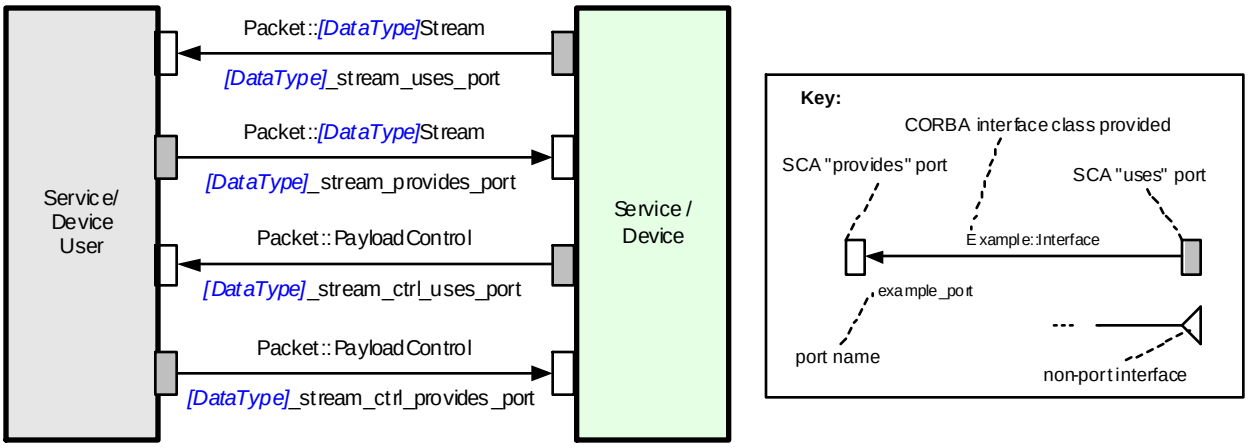


**Figure 1 – Packet Port Diagram**

***Packet* Provides Ports Definitions**

***[DataType]*_stream_provides_port** is provided by the *Device or Service* to consume packets through the *pushPacket* operation.
***[DataType]*_stream_ctrl_provides_port** is provided by the *Device or Service* to set the payload size by the *Device or Service User*.

***Packet* Uses Ports Definitions**

***[DataType]*_stream_uses_port** is used by the *Device or Service* to push packets to the *Device User or Service User*.
***[DataType]*_stream_ctrl_uses_port** is used by the *Device or Service* to set the payload size of the incoming packets from the *Device or Service User*.

## A.1.3 Modes of Service

Not Applicable.

## A.1.4 Service States

Not Applicable.

## A.1.5 Referenced Documents

The following documents of the exact issue shown form a part of this specification to the extent specified herein.

### A.1.5.1 Government Documents

#### A.1.5.1.1 Specifications

##### A.1.5.1.1.1 Federal Specifications

None

##### A.1.5.1.1.2 Military Specifications

None

#### A.1.5.1.2 Other Government Agency Documents

[1] JTRS Standard, "JTRS CORBA Types," JPEO, Version 1.0.2.
[2] JTRS Standard, "Device Message Control API," JPEO, Version 1.1.1.

### A.1.5.2 Commercial Standards

None

# A.2 SERVICES

## A.2.1 Provide Services

The following table describes the generic provides service interfaces to be specified by the service or device. These provide service interfaces correspond to the port diagram in Figure 1.

Note: "*[DataType]*" is meant to be replaced by the data type defining the packet stream (i.e. *octet, ushort*).

**Table 1 – Packet Provides Service Interfaces**

| **Service Group (Port Name)** | **Service (Interface Provided)** | | **Primitives (Provided)** |
|---|---|---|---|
| *[DataType]*_stream _provides_port | Packet:: *[DataType]*Stream | | pushPacket() |
| | | Packet:: PayloadStatus | getMaxPayloadSize() |
| | | | getMinPayloadSize() |
| | | | getDesiredPayloadSize() |
| | | | getMinOverrideTimeout() |
| *[DataType]*_stream_ctrl _provides_port | Packet:: PayloadControl | | setMaxPayloadSize() |
| | | | setMinPayloadSize() |
| | | | setDesiredPayloadSize() |
| | | | setMinOverrideTimeout() |

## A.2.2 Use Services

The following table describes the generic uses service interfaces to be specified by the service or device. These interfaces correspond to the port diagram in Figure 1.

Note: "*[DataType]*" is meant to be replaced by the data type defining the packet stream (i.e. *octet*, *ushort*).

**Table 2 – Packet Uses Service Interface**

| **Service Group (Port Name)** | **Service (Interface Provided)** | | **Primitives (Provided)** |
|---|---|---|---|
| *[DataType]*_stream _uses_port | Packet:: *[DataType]*Stream | | pushPacket() |
| | | Packet:: PayloadStatus | getMaxPayloadSize() |
| | | | getMinPayloadSize() |
| | | | getDesiredPayloadSize() |
| | | | getMinOverrideTimeout() |
| *[DataType]*_stream_ctrl _uses_port | Packet:: PayloadControl | | setMaxPayloadSize() |
| | | | setMinPayloadSize() |
| | | | setDesiredPayloadSize() |
| | | | setMinOverrideTimeout() |

## A.2.3 Interface Modules

### A.2.3.1 Packet

The class diagram for the Packet API is shown in Figure 2.

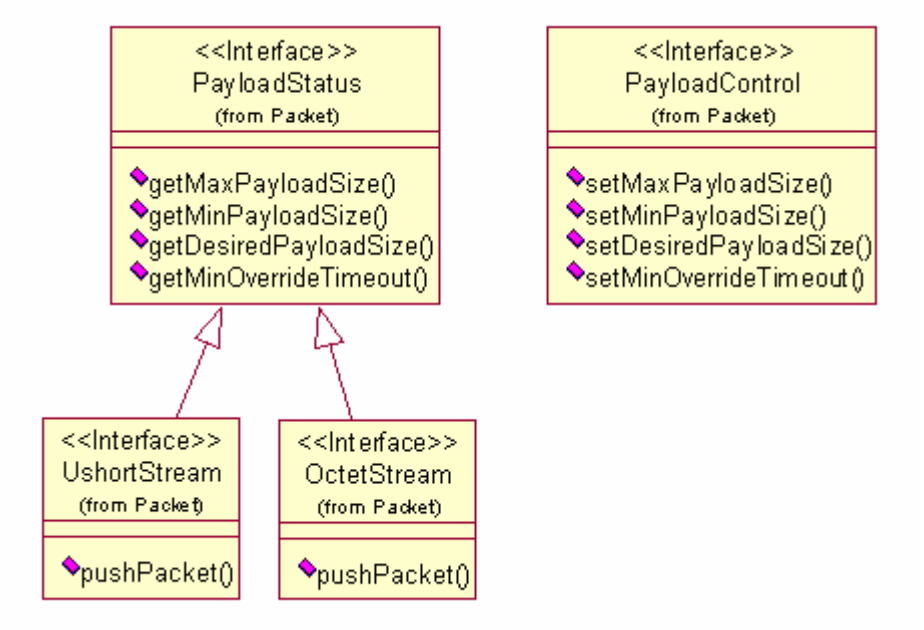


**Figure 2 - Packet Class Diagram**

#### A.2.3.1.1 PayloadStatus Interface Description

The interface design of *PayloadStatus* is shown in Figure 3. It provides methods to get the maximum payload size, the minimum payload size, the desired payload size, and the minimum override timeout value. The *getMinPayloadSize* operation is used for asynchronous modes while the *getDesiredPayloadSize* operation is used for synchronous modes.

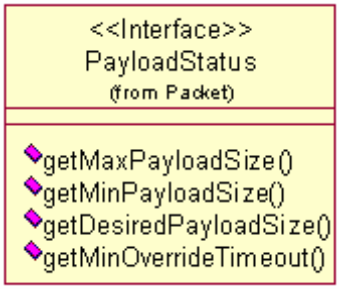


**Figure 3 - PayloadStatus Class Diagram**

### A.2.3.1.2 PayloadControl Interface Description

The interface design of *PayloadControl* is shown in Figure 4. It provides the ability to set the maximum payload size, the minimum payload size, the desired payload size, and the minimum override time out value. The *setMinPayloadSize* operation is used for asynchronous modes while the *setDesiredPayloadSize* operation is used for synchronous modes.

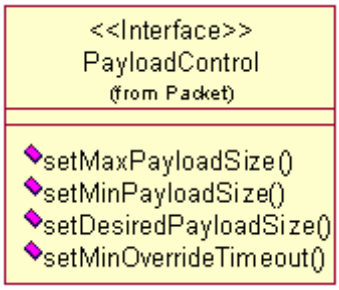


**Figure 4 - PayloadControl Class Diagram**

### A.2.3.1.3 OctetStream Interface Description

The interface design of *OctetStream* is shown in Figure 5. It inherits the *PayloadStatus* interface to query the packet sizes (see A.2.3.1.1). It also adds the capability to transfer octet packets between the provider and user.

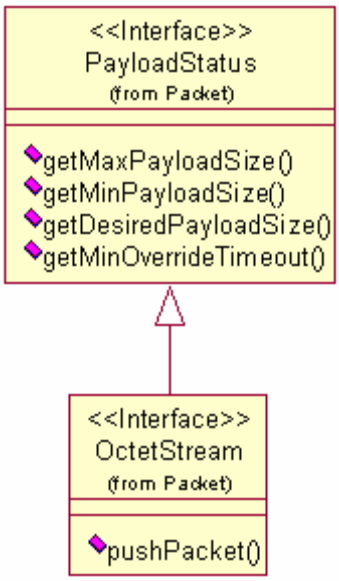


**Figure 5 - OctetStream Class Diagram**

### A.2.3.1.4 UshortStream Interface Description

The interface design of *UshortStream* is shown in Figure 6. It inherits the *PayloadStatus* interface to query the packet sizes (see A.2.3.1.1). It also adds the capability to transfer unsigned short packets between the provider and user.

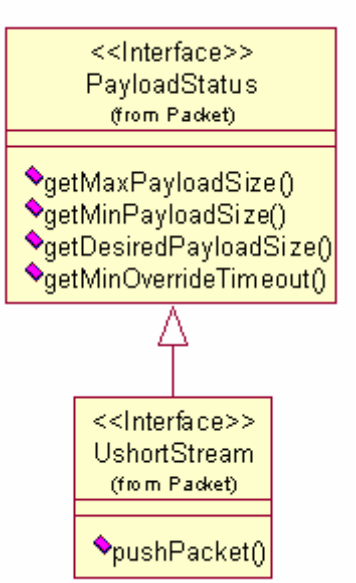


**Figure 6 - UshortStream Class Diagram**

## A.2.4 Sequence Diagrams

### A.2.4.1 MaxPayloadSize Sequence

#### Description

The *Max Payload Sequence Diagram* is shown in Figure 7.

The *Data Producer* in the sequence is the interface inheriting the Packet::PayloadControl interface. The *Data Consumer* in the sequence is the interface inheriting the Packet::*[DataType]*Stream interface.

Upon startup, the *Data Producer* calls *getMaxPayloadSize()*. The *Data Consumer* returns the maximum payload size it can support. The *Data Producer* transfers packets to the *Data Consumer* using the *pushPacket()* operation and the specified payload size. When required, the *Data Consumer* will call the *setMaxPayloadSize()* operation to modify the maximum payload size. The *Data Producer* will provide subsequent packets using the modified size. The *pushPacket()* messaging should produce no more then a single additional packet using the previous payload size.

Note: "*[DataType]*" is meant to be replaced by the data type defining the packet stream (i.e. octet, ushort).

#### Pre-conditions
None

#### Post-conditions
The packets are pushed using the value specified in the setMaxPayloadSize() operation.

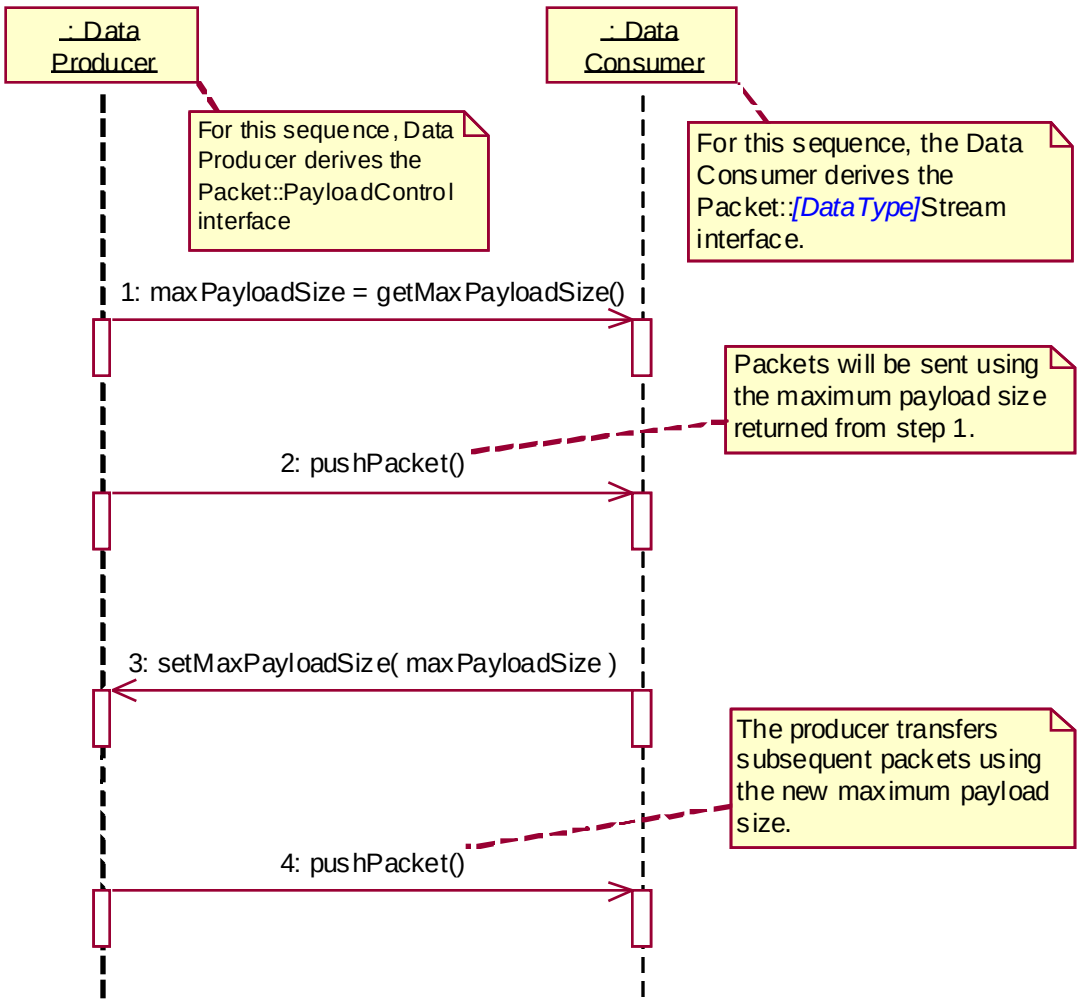


**Figure 7 – Max Payload Size Sequence Diagram**

# A.3 SERVICE PRIMITIVES AND ATTRIBUTES

To enhance the readability of this API document and to avoid duplication of data, the type definitions of all structured types (i.e., data types, enumerations, exceptions, and structures) used by the service primitives and attributes have been co-located in section A.5. This cross-reference of types also includes any nested structures in the event of a structure of structures or an array of structures.

## A.3.1 Packet::PayloadStatus

### A.3.1.1 *getMaxPayloadSize* Operation

The *getMaxPayloadSize* operation returns the absolute maximum payload size allowed for a payload passed to the *pushPacket* operation.

#### A.3.1.1.1 Synopsis

*unsigned long getMaxPayloadSize();*

#### A.3.1.1.2 Parameters

None

#### A.3.1.1.3 State

Not Applicable.

#### A.3.1.1.4 New State

Not Applicable.

#### A.3.1.1.5 Return Value

| Return Value | Type | Units |
|---|---|---|
| The maximum payload size. | unsigned long | bytes |

#### A.3.1.1.6 Originator

Service Provider, Service User.

#### A.3.1.1.7 Exceptions

None

## A.3.1.2 *getMinPayloadSize* Operation

The *getMinPayloadSize* operation is used for asynchronous modes. It returns the minimum payload size allowed for a payload passed to the *pushPacket* operation.

Note that payloads of 0 (zero) size (i.e. control packets) are exempt.

### A.3.1.2.1 Synopsis

*unsigned long getMinPayloadSize();*

### A.3.1.2.2 Parameters

None

### A.3.1.2.3 State

Not Applicable.

### A.3.1.2.4 New State

Not Applicable.

### A.3.1.2.5 Return Value

| Return Value | Type | Units |
|---|---|---|
| The minimum payload size. | unsigned long | bytes |

### A.3.1.2.6 Originator

Service Provider, Service User.

### A.3.1.2.7 Exceptions

None

### A.3.1.3 *getDesiredPayloadSize* Operation

The *getDesiredPayloadSize* operation is used for synchronous modes. It returns the desired payload size allowed for a payload passed to the *pushPacket* operation.

#### A.3.1.3.1 Synopsis

*unsigned long getDesiredPayloadSize();*

#### A.3.1.3.2 Parameters

None

#### A.3.1.3.3 State

Not Applicable.

#### A.3.1.3.4 New State

Not Applicable.

#### A.3.1.3.5 Return Value

| Return Value | Type | Units |
|---|---|---|
| The desired payload size. | unsigned long | bytes |

#### A.3.1.3.6 Originator

Service Provider, Service User.

#### A.3.1.3.7 Exceptions

None

## A.3.1.4 *getMinOverrideTimeout* Operation

The *getMinOverrideTimeout* operation returns the time a payload smaller than the "minPayloadSize" for asynchronous modes or the "desiredPayloadSize" for synchronous modes should be held before passed to the *pushPacket* operation.

### A.3.1.4.1 Synopsis

*unsigned long getMinOverrideTimeout();*

### A.3.1.4.2 Parameters

None

### A.3.1.4.3 State

Not Applicable.

### A.3.1.4.4 New State

Not Applicable.

### A.3.1.4.5 Return Value

| Return Value | Type | Units |
|---|---|---|
| The minimum override timeout period. | unsigned long | milliseconds |

### A.3.1.4.6 Originator

Service Provider, Service User.

### A.3.1.4.7 Exceptions

None

## A.3.2 Packet::PayloadControl

### A.3.2.1 *setMaxPayloadSize* Operation

The s*etMaxPayloadSize* operation sets the absolute maximum payload size allowed for a payload passed to the *pushPacket* operation.

Note: The payload size should not be set during the stream.

#### A.3.2.1.1 Synopsis

*void setMaxPayloadSize(in unsigned long maxPayloadSize) raises(JTRS::InvalidParameter);*

#### A.3.2.1.2 Parameters

| Parameter Name | Description | Type | Units |
|---|---|---|---|
| maxPayloadSize | The absolute maximum payload size allowed for a payload. | unsigned long | bytes |

#### A.3.2.1.3 State

Not Applicable.

#### A.3.2.1.4 New State

Not Applicable.

#### A.3.2.1.5 Return Value

None

#### A.3.2.1.6 Originator

Service Provider, Service User.

#### A.3.2.1.7 Exceptions

| Type | Description |
|---|---|
| JTRS::InvalidParameter<br>*See JTRS CORBA Types* [1]. | The input parameter is not valid. |

## A.3.2.2 *setMinPayloadSize* Operation

The s*etMinPayloadSize* operation is used for asynchronous modes. It sets the minimum payload size allowed for a payload passed to the *pushPacket* operation.

Note: Payloads of 0 (zero) size (i.e. control packets) are exempt. The payload size should not be set during the stream.

### A.3.2.2.1 Synopsis

*void setMinPayloadSize(in unsigned long minPayloadSize) raises(JTRS::InvalidParameter);*

### A.3.2.2.2 Parameters

| Parameter Name | Description | Type | Units |
|---|---|---|---|
| minPayloadSize | The minimum payload size allowed for a payload. | unsigned long | bytes |

### A.3.2.2.3 State

Not Applicable.

### A.3.2.2.4 New State

Not Applicable.

### A.3.2.2.5 Return Value

None

### A.3.2.2.6 Originator

Service Provider, Service User.

### A.3.2.2.7 Exceptions

| Type | Description |
|---|---|
| JTRS::InvalidParameter<br>*See JTRS CORBA Types* [1]. | The input parameter is not valid. |

## A.3.2.3 *setDesiredPayloadSize* Operation

The *setDesiredPayloadSize* operation is used for synchronous modes. It sets the desired payload size allowed for a payload passed to the *pushPacket* operation.

Note: The payload size should not be set during the stream.

### A.3.2.3.1 Synopsis

*void setDesiredPayloadSize(in unsigned long desiredPayloadSize)*
*raises(JTRS::InvalidParameter);*

### A.3.2.3.2 Parameters

| Parameter Name | Description | Type | Units |
|---|---|---|---|
| desiredPayloadSize | The desired payload size allowed for a payload. | unsigned long | bytes |

### A.3.2.3.3 State

Not Applicable.

### A.3.2.3.4 New State

Not Applicable.

### A.3.2.3.5 Return Value

None

### A.3.2.3.6 Originator

Service Provider, Service User.

### A.3.2.3.7 Exceptions

| Type | Description |
|---|---|
| JTRS::InvalidParameter<br>*See JTRS CORBA Types* [1]. | The input parameter is not valid. |

## A.3.2.4 *setMinOverrideTimeout* Operation

The s*etMinOverrideTimeout* operation sets the time a payload smaller than "minPayloadSize" for asynchronous modes or the "desiredPayloadSize" for synchronous modes should be held before passed to the *pushPacket* operation.

### A.3.2.4.1 Synopsis

*void setMinOverrideTimeout(in unsigned long minOverrideTimeout)*
*raises(JTRS::InvalidParameter);*

### A.3.2.4.2 Parameters

| Parameter Name | Description | Type | Units |
|---|---|---|---|
| minOverrideTimeout | The minimum time out value to be overridden. | unsigned long | milliseconds |

### A.3.2.4.3 State

Not Applicable.

### A.3.2.4.4 New State

Not Applicable.

### A.3.2.4.5 Return Value

None

### A.3.2.4.6 Originator

Service Provider, Service User.

### A.3.2.4.7 Exceptions

| Type | Description |
|---|---|
| JTRS::InvalidParameter<br>*See JTRS CORBA Types* [1]. | The input parameter is not valid. |

## A.3.3 Packet::OctetStream

### A.3.3.1 *pushPacket* Operation

The *pushPacket* operation provides the ability to push octet data packets to the packet consumer.

#### A.3.3.1.1 Synopsis

*void pushPacket( in StreamControlType control,*
*in JTRS::OctetSequence payload )*
*raises ( UnableToComplete);*

#### A.3.3.1.2 Parameters

| Parameter Name | Description | Type |
|---|---|---|
| control | A structure containing the elements used to control the packet stream. | StreamControlType *See A.5.1.* |
| payload | A sequence of octet contains the data to be pushed. | JTRS::OctetSequence *See JTRS CORBA Types* [1]. |

#### A.3.3.1.3 State

Not Applicable.

#### A.3.3.1.4 New State

Not Applicable.

#### A.3.3.1.5 Return Value

None

#### A.3.3.1.6 Originator

Service Provider, Service User.

#### A.3.3.1.7 Exceptions

| Type | Description |
|---|---|
| UnableToComplete *See A.5.3.1.* | The *pushPacket* call was not complete. |

## A.3.4 Packet::UshortStream

### A.3.4.1 *pushPacket* Operation

The *pushPacket* operation provides the ability to push unsigned short data packets to the packet consumer.

#### A.3.4.1.1 Synopsis

*void pushPacket( in StreamControlType control,*
*in JTRS::UshortSequence payload )*
*raises ( UnableToComplete);*

#### A.3.4.1.2 Parameters

| Parameter Name | Description | Type |
|---|---|---|
| control | A structure containing the elements used to control the packet stream. | StreamControlType *See A.5.1.* |
| payload | A sequence of unsigned short contains the data to be pushed. | JTRS::UshortSequence *See JTRS CORBA Types* [1]. |

#### A.3.4.1.3 State

Not Applicable.

#### A.3.4.1.4 New State

Not Applicable.

#### A.3.4.1.5 Return Value

None

#### A.3.4.1.6 Originator

Service Provider, Service User.

#### A.3.4.1.7 Exceptions

| Type | Description |
|---|---|
| UnableToComplete *See A.5.3.1.* | The *pushPacket* call was not complete. |

## A.4 IDL

### A.4.1 Packet IDL

```
/*
** Packet.idl - JTRS Packet Types
*/

#ifndef __PACKET_DEFINED
#define __PACKET_DEFINED

#ifndef __JTRSCORBATYPES_DEFINED
#include "JtrsCorbaTypes.idl"
#endif

module Packet
{
	//
	// Base Packet Service Set
	//

	// Payload Status Query
	interface PayloadStatus
	{
		unsigned long getMaxPayloadSize();
		unsigned long getMinPayloadSize();
		unsigned long getDesiredPayloadSize();
		unsigned long getMinOverrideTimeout();
	};

	// Payload Modifiers
	interface PayloadControl
	{
		void setMaxPayloadSize( in unsigned long maxPayloadSize )
			raises (JTRS::InvalidParameter);
```

```
        void setMinPayloadSize( in unsigned long minPayloadSize )
                raises (JTRS::InvalidParameter);

        void setDesiredPayloadSize( in unsigned long desiredPayloadSize )
                raises (JTRS::InvalidParameter);

        void setMinOverrideTimeout( in unsigned long minOverrideTimeout )
                raises (JTRS::InvalidParameter);
};

//
// Streaming Services
//
typedef unsigned short      Stream;
typedef octet               SeqNum;
struct StreamControlType
{
        boolean       endOfStream;   // Indicates whether it is end of stream or not.
        Stream        streamId;            // Identification number for the stream.
        SeqNum        sequenceNumber;      // The sequence number for the stream.
        boolean       purge;               // Indicates whether to purge the stream or not.
};

// Stream Exceptions
typedef JTRS::ExtEnum PushError;

const PushError PPKT_UNKNOWN               = 1;
const PushError PPKT_ERROR_BAD_SIZE        = 2;
const PushError PPKT_ERROR_STREAM_BLOCKED  = 3;

exception UnableToComplete
{
        unsigned short payloadSent;  // Count of payload elements sent
        PushError errorCode;
};

// Packet Stream 'Consumers' types
interface OctetStream : PayloadStatus
{
```

```
                void pushPacket( in StreamControlType control, in JTRS::OctetSequence payload )
                        raises( UnableToComplete );
        };

        interface UshortStream : PayloadStatus
        {
                void pushPacket( in StreamControlType control, in JTRS::UshortSequence payload )
                   raises( UnableToComplete );
        };
};

#endif // __PACKET_DEFINED
```

## A.5 UML

This section contains Packet Component UML Diagram and the definitions of all data types referenced (directly or indirectly) by the Service Primitives and Attributes in Section A.3.

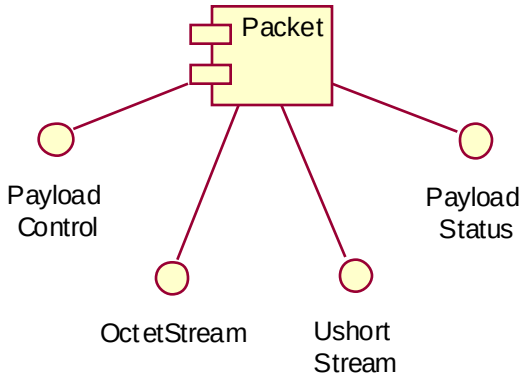


**Figure 8 – Packet Component Diagram**

## A.5.1 Data Types

### A.5.1.1 Packet::SeqNum

*SeqNum* is a data type of type *octet*.

```
typedef octet SeqNum;
```

### A.5.1.2 Packet::Stream

*Stream* is a data type of type *unsigned short*.

```
typedef unsigned short Stream;
```

## A.5.2 Enumerations

### A.5.2.1 Packet::PushError

The *PushError* type definition is a JTRS extension enumeration (see *JTRS CORBA Types* [1]). It enumerates the type of error that occurred in the *UnableToComplete* exception (see A.5.3.1).

```
// Stream Exceptions
typedef JTRS::ExtEnum PushError;

const PushError PPKT_UNKNOWN = 1;
const PushError PPKT_ERROR_BAD_SIZE =2 ;
const PushError PPKT_ERROR_STREAM_BLOCKED = 3;
```

| JTRS::ExtEnum | Element | Value | Description |
|---|---|---|---|
| PushError | PPKT_UNKNOWN | 1 | Error is unknown |
| | PPKT_ERROR_BAD_SIZE | 2 | Packet is of the wrong size |
| | PPKT_ERROR_STREAM_BLOCKED | 3 | Stream is blocked |

## A.5.3 Exceptions

### A.5.3.1 Packet::UnableToComplete

The *UnableToComplete* exception indicates that the *pushPacket* call was not complete (see A.3.3.1 and A.3.4.1). This exception indicates an implementation's inability to complete the contract of the interface. It should not be used as a flow control mechanism.

```
exception UnableToComplete
{
```

```
        unsigned short payloadSent;  // Count of payload elements sent
        PushError      errorCode;
    };
```

| Exception | Element | Type | Description |
|---|---|---|---|
| UnableToComplete | payloadSent | unsigned short | Count of payload elements sent |
| | errorCode | PushError<br>*See A.5.2.1.* | Type of error that occurred |

## A.5.4 Structures

### A.5.4.1 **Packet::StreamControlType**

The *StreamControlType* structure defines the attributes used for stream control in a *pushPacket* operation (see A.3.3.1 and A.3.4.1).

```
structure StreamControlType
{
        boolean  endOfStream;
        Stream   streamId;
        SeqNum   sequenceNumber;
        boolean  purge;
};
```

| Struct | Attributes | Description | Type | Valid Range |
|---|---|---|---|---|
| StreamControlType | endOfStream | Indicates whether it is end of stream or not. | boolean | TRUE = enable; FALSE = disable. |
| | streamId | The streamId field identifies the stream membership of the packet. A change in streamId indicates the start of a new stream. The streamId may rollover but should never be reset. | Stream *See A.5.1.2* | 0 – 65535 |
| | sequenceNumber | The sequenceNumber field is the sequence number of the packet in the current stream. The sequenceNumber starts at zero for each stream and increments for each packet. A change in the sequence indicates recovery/error checking may need to be performed. | SeqNum *See A.5.1.1* | 0 – 255 |

| Struct | Attributes | Description | Type | Valid Range |
|---|---|---|---|---|
| | purge | The purge field indicates whether the stream should be purged. A stream will be purged in response to a *DeviceMessageControl::abortTx* call from an external source. (See *DeviceMessageControl* [2]). Note: a purge condition is accomplished by sending a 0 length packet with "purge" and "endOfStream" set to TRUE. | boolean | TRUE = enable; FALSE = disable. |

# Appendix A.A Abbreviations and Acronyms

| | |
|---|---|
| **API** | Application Program Interface |
| **CORBA** | Common Object Request Broker Architecture |
| **ICWG** | Interface Control Working Group |
| **IDL** | Interface Definition Language |
| **JPEO** | Joint Program Executive Office |
| **JTR** | Joint Tactical Radio |
| **JTRS** | Joint Tactical Radio System |
| **UML** | Unified Modeling Language |

# Appendix A.B Performance Specification

Table 3 provides a template for the generic performance specification for the Packet API which will be documented in the service or device using the interface. This performance specification corresponds to the port diagram in Figure 1.

Note: "*[DataType]*" is meant to be replaced by the data type defining the packet stream (i.e. *octet*, *ushort*).

**Table 3 – Packet Performance Specification**

| Specification | Description | Units | Value |
|---|---|---|---|
| Worst Case Command Execution Time for *pushPacket()* on *[DataType]*_stream_provides_port | * | * | * |
| Worst Case Command Execution Time for pushPacket() on *[DataType]*_stream_uses_port | * | * | * |
| Worst Case Command Execution Time for *[DataType]*_stream_provides_port | * | * | * |
| Worst Case Command Execution Time for *[DataType]*_stream_uses_port | * | * | * |
| Worst Case Command Execution Time for *[DataType]*_stream_ctrl_uses_port | * | * | * |
| Worst Case Command Execution Time for *[DataType]*_stream_ctrl_provides_port | * | * | * |

*Note this template will be filled in by individual developers.

# B. FLOW CONTROL EXTENSION

## B.1 INTRODUCTION

The *Flow Control Extension* provides the ability to send flow controlled data to/from the device or service user. Data is controlled by either polling for space availability or waiting for a signal to resume. The *Flow Control Extension* supports pushing two-way packets of common data types to the device or service user. This *pushPacket* returns a *boolean* indicating whether or not space is available for additional packets.

The *Flow Control Extension* interfaces may be used in conjunction with other interfaces to create the device/service *Data Producer* and *Data Consumer* interfaces. They are documented independently to minimize coupling between the devices and services that utilize these interfaces.

### B.1.1 Overview

This extension contains as follows:

a. Section B.1, *Introduction*, contains the introductory material regarding the overview, service layer description, modes, states, and referenced documents of this document.

b. Section B.2, *Services*, specifies the interfaces for the component, port connections, and sequence diagrams.

c. Section B.3, *Service Primitives and Attributes*, specifies the operations that are provided by the *Flow Control Extension*.

d. Section B.4, *IDL*.

e. Section B.5, *UML*.

f. Appendix B.A, *Abbreviations and Acronyms*.

g. Appendix B.B, *Performance Specification*.

## B.1.2 Service Layer Description

### B.1.2.1 Flow Control Extension Port Diagram

The following figure, port definitions, and port names are for reference only. The service/device will specify all port definitions and interfaces comprising the *Data Producer* and *Data Consumer*. This diagram specifies a generic bidirectional instantiation of the *Flow Control Extension* interfaces to be specified by a service or device. A unidirectional instantiation may be achieved by only specifying the "uses" or the "provides" ports. Note: "*[DataType]*" is meant to be replaced by the data type defining the packet stream (i.e. *octet, ushort*).

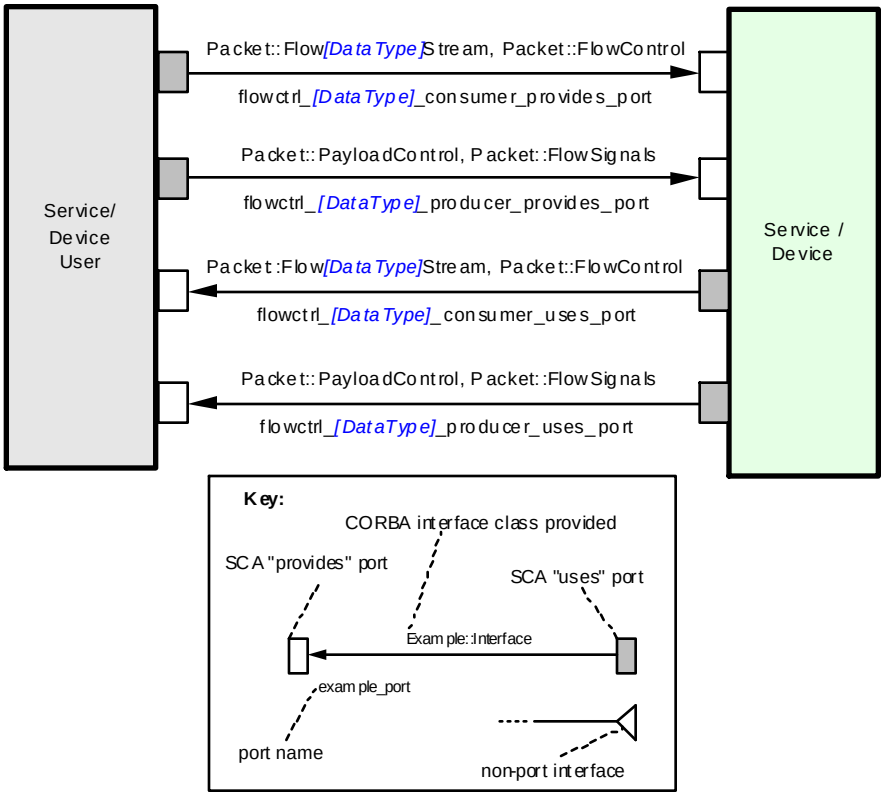


**Figure 9 – Flow Control Extension Port Diagram**

***Packet* Flow Control Extension Provides Ports Definitions**

**flowctrl_*[DataType]*_consumer_provides_port** is provided by the *Device or Service* to consume packets through the *pushPacket* operation and activate the flow control signals. **flowctrl_ *[DataType]*_producer_provides_port** is provided by the *Device or Service* to produce packets and provide flow control signals.

***Packet* Flow Control Extension UsesPorts Definitions**

**flowctrl_ *[DataType]*_consumer_uses_port** is used by the *Device or Service* to consume through the *pushPacket* operation and activate the flow control signals.

**flowctrl_ *[DataType]*_producer_uses_port** is used by the *Device or Service* to produce packets and provide flow control signals.

## B.1.3 Modes of Service

Not Applicable.

## B.1.4 Service States

Not Applicable.

## B.1.5 Referenced Documents

There are no changes from the base API.

# B.2 SERVICES

## B.2.1 Provide Services

Table 4 describes the generic provides service interfaces to be specified by the service or device. These provide service interfaces correspond to the port diagram in Figure 9.

Note: "*[DataType]*" is meant to be replaced by the data type defining the packet stream (i.e. *octet*, *ushort*). Interfaces and services shaded in *grey* have already been defined in the section indicated.

**Table 4 – Flow Control Extension Provides Service Interfaces**

| **Service Group (Port Name)** | **Service (Interface Provided)** | | **Primitives (Provided)** |
|---|---|---|---|
| flowctrl_*[DataType]*_ consumer_provides_port | Packet:: Flow*[DataType]* Stream | | pushPacket() |
| | | Packet:: PayloadStatus *See A.3.1.* | getMaxPayloadSize() |
| | | | getMinPayloadSize() |
| | | | getDesiredPayloadSize() |
| | | | getMinOverrideTimeout() |
| | Packet:: FlowControl | | enableFlowResumeSignals() |
| | | | spaceAvailable() |
| flowctrl_*[DataType]*_ producer_provides_port | Packet:: PayloadControl *See A.3.2.* | | setMaxPayloadSize() |
| | | | setMinPayloadSize() |
| | | | setDesiredPayloadSize() |
| | | | setMinOverrideTimeout() |
| | Packet:: FlowSignals | | signalResume() |

## B.2.2 Use Services

The following table describes the generic uses service interfaces to be specified by the service or device. These provide service interfaces correspond to the port diagram in Figure 9. Note: "*[DataType]*" is meant to be replaced by the data type defining the packet stream (i.e. *octet, ushort*). Interfaces and services shaded in *grey* have already been defined in the section indicated.

**Table 5 – Flow Control Extension Uses Service Interfaces**

| **Service Group (Port Name)** | **Service (Interface Provided)** | | **Primitives (Provided)** |
|---|---|---|---|
| flowctrl_*[DataType]*_ consumer_uses_port | Packet:: Flow*[DataType]* Stream | | pushPacket() |
| | | Packet:: PayloadStatus *See A.3.1.* | getMaxPayloadSize() |
| | | | getMinPayloadSize() |
| | | | getDesiredPayloadSize() |
| | | | getMinOverrideTimeout() |
| | Packet:: FlowControl | | enableFlowResumeSignals() |
| | | | spaceAvailable() |
| flowctrl_*[DataType]*_ producer_uses_port | Packet:: PayloadControl *See A.3.2.* | | setMaxPayloadSize() |
| | | | setMinPayloadSize() |
| | | | setDesiredPayloadSize() |
| | | | setMinOverrideTimeout() |
| | Packet:: FlowSignals | | signalResume() |

# B.2.3 Interface Modules

## B.2.3.1 Packet

### B.2.3.1.1 Flow Control Extension

The class diagram for the *Flow Control Extension* is shown in *yellow* in Figure 10. Interfaces shown in *grey* are defined in section A.

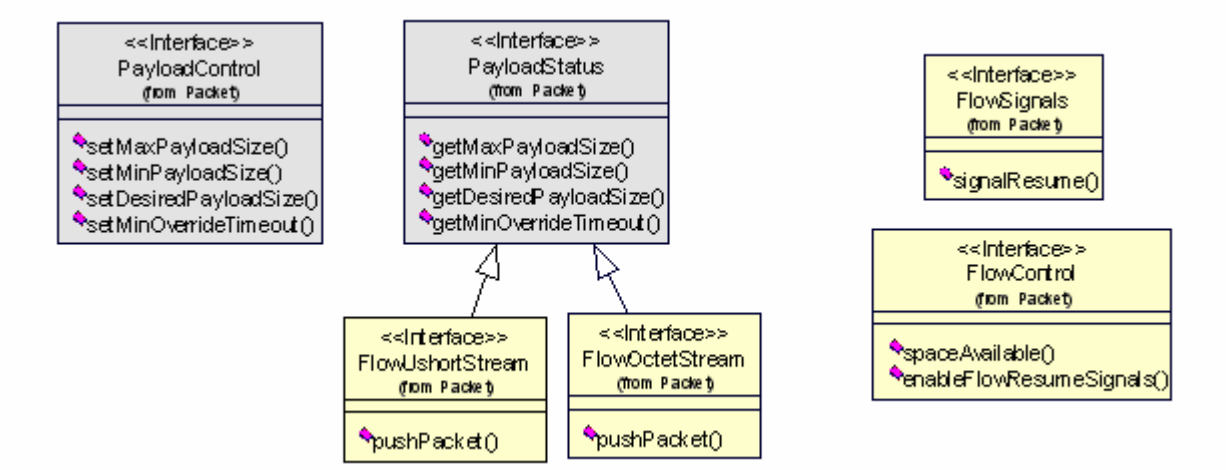


**Figure 10 – Flow Control Extension Class Diagram**

### B.2.3.1.2 Flow Control Extension Data Producer and Data Consumer Interface Descriptions

The *Flow Control Extension Data Producer and Data Consumer* interfaces are shown in Figure 11 and Figure 12. The *Data Producer* and *Data Consumer* interfaces shown in *green* are not defined in this API. They have been provided to illustrate the collection of *Flow Control Extension* interfaces which will be specified by the service/device or service/device user.

Note: Additional interfaces may be included to complete the *Data Producer/Consumer* interfaces (e.g. *DeviceIo*). Interfaces shown in *grey* are defined in section A.

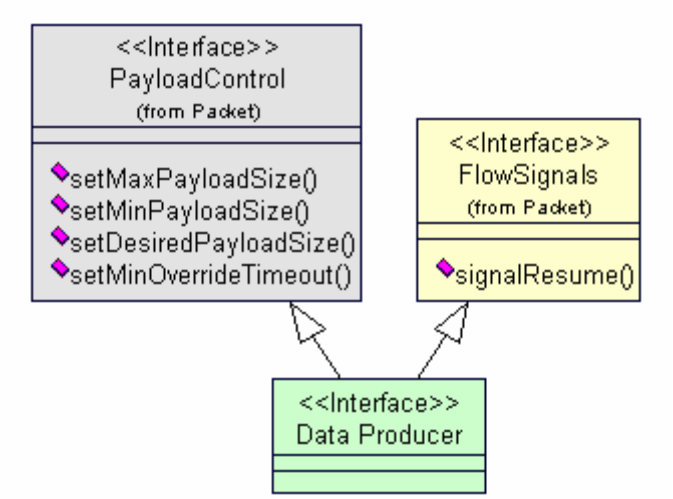


**Figure 11 – Flow Control Extension Data Producer Class Diagram**

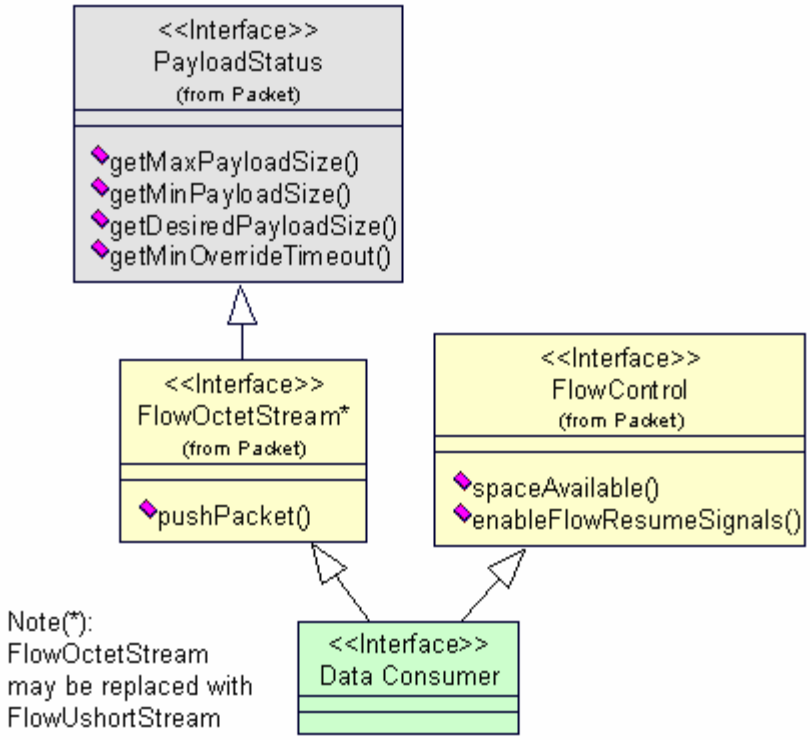


**Figure 12 – Flow Control Extension Data Consumer Class Diagram**

## B.2.3.1.3 FlowControl Interface Description

The interface design of *FlowControl* is shown in Figure 13. It provides methods to activate or deactivate the flow resume signals (*enableFlowResumeSignals*). It also provides an operation (*spaceAvailable*) to poll whether space is available.

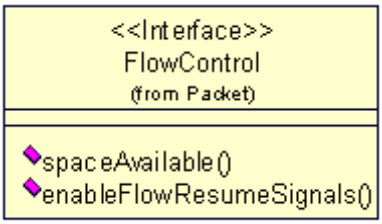


**Figure 13 – FlowControl Interface**

### B.2.3.1.4 FlowSignals Interface Description

The interface design of *FlowSignals* is shown in Figure 14. It provides methods to call or signal the flow resume signal (*signalResume*) that was enabled by the *FlowControl* interface.

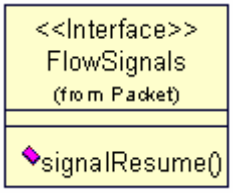


**Figure 14 – FlowSignals Interface**

### B.2.3.1.5 FlowOctetStream Interface Description

The interface design of *FlowOctetStream* is shown in Figure 15. It provides the capability to transfer *octet* packets between the provider and user. The *pushPacket* returns a *boolean* indicating whether or not space is available for additional packets.

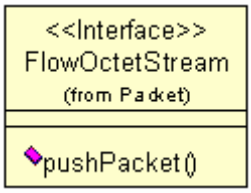


**Figure 15 – FlowOctetStream Interface**

### B.2.3.1.6 FlowUshortStream Interface Description

The interface design of *FlowUshortStream* is shown in Figure 16. It provides the capability to transfer *unsigned short* packets between the provider and user. The *pushPacket* returns a *boolean* indicating whether or not space is available for additional packets.

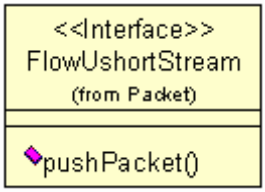


**Figure 16 – FlowUshortStream Interface**

## B.2.4 Sequence Diagrams

### B.2.4.1 Signal Resume Sequence Diagram

**Description**
The *Signal Resume Sequence Diagram* is shown in Figure 17.

The *Data Producer* and *Data Consumer* in the sequence inherit those interfaces shown in Figure 11 and Figure 12.

The *Data Producer* activates the flow resume signals by calling *enableFlowResumeSignals(TRUE)* and pushes packets to the *Data Consumer*. The *Data Consumer* will return a *boolean* value indicating whether or not there is space available for at least one more packet of the maximum size. If the *Data Consumer* returns a FALSE, the *Data Producer* will wait until it receives a s*ignalResume* call indicating that space is now available for more packets.

**Pre-conditions**
None

**Post-conditions**
Data flow is controlled by the *signalResume* signal.

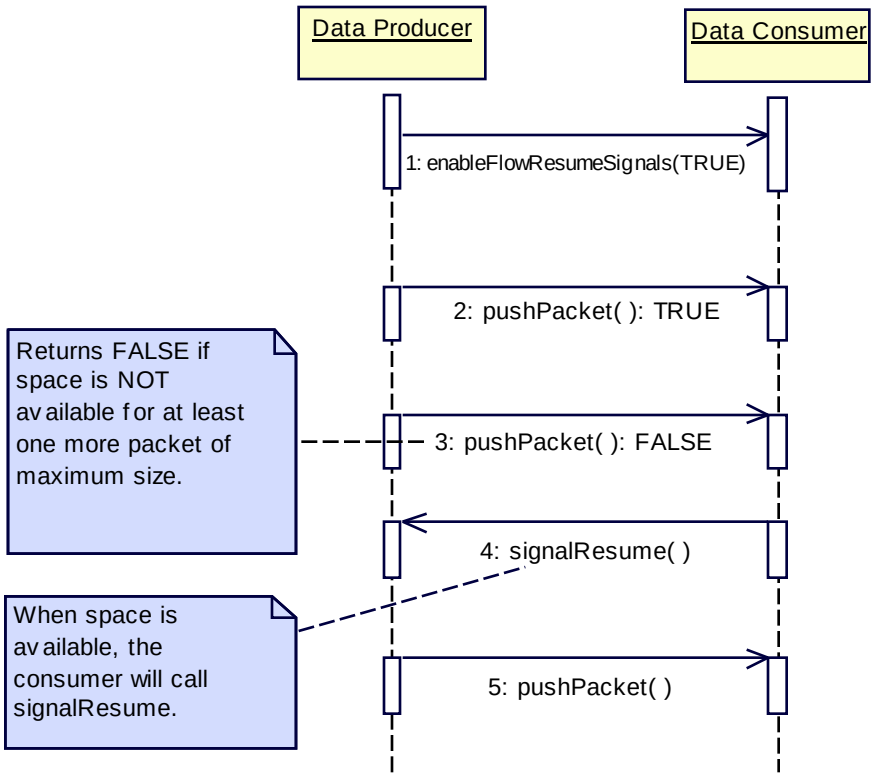


**Figure 17 – Signal Resume Sequence Diagram**

### B.2.4.2 Space Available Sequence Diagram

**Description**
The *Space Available Sequence Diagram* is shown in Figure 18.

The *Data Producer* and *Data Consumer* in the sequence inherit those interfaces shown in Figure 11 and Figure 12.

The *Data Producer* will poll the *Data Consumer* to determine if space is not available for at least one more packet of maximum size. When the *Data Consumer* returns TRUE, it will continue pushing packets.

**Pre-conditions**
None

**Post-conditions**
Data flow is controlled by polling for space availability.

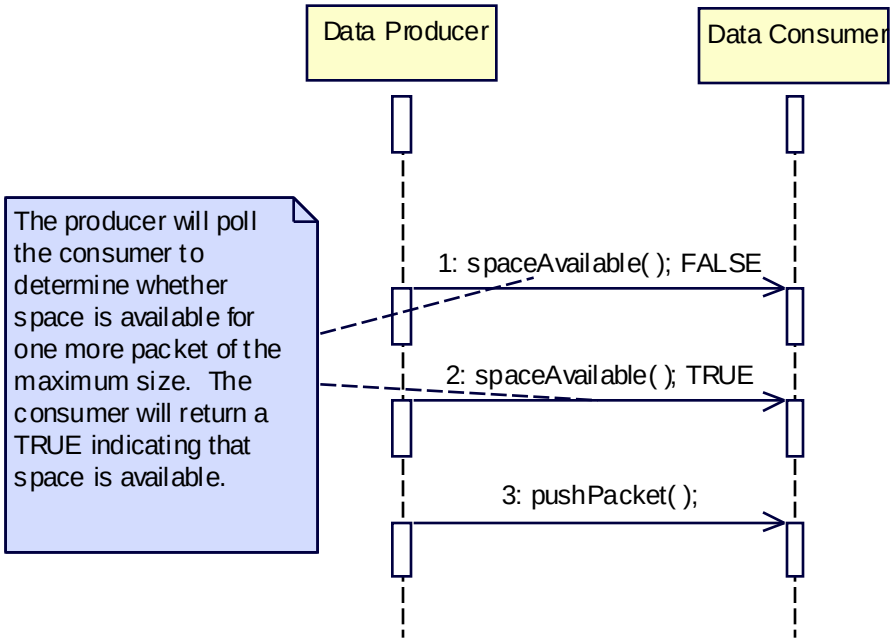


**Figure 18 – Space Available Sequence Diagram**

## B.3 SERVICE PRIMITIVES AND ATTRIBUTES

To enhance the readability of this API document and to avoid duplication of data, the type definitions of all structured types (i.e., data types, enumerations, exceptions, and structures) used by the service primitives and attributes have been co-located in Section B.5. This cross-reference of types also includes any nested structures in the event of a structure of structures or an array of structures.

## B.3.1 Packet::FlowControl

### B.3.1.1 *enableFlowResumeSignals* Operation

The *enableFlowResumeSignals* operation is used to activate and deactivate the use of the *signalResume* callback operation by the *Data Consumer*. By default, the flow resume signaling will be disabled and the producer is responsible for polling the consumer to determine when additional space is available. Enabling flow resume via this operation, instructs the *Data Consumer* to inform the *Data Producer* (using *signalResume*) when additional space has been freed.

#### B.3.1.1.1 Synopsis

*oneway void enableFlowResumeSignals( in boolean setEnableState );*

#### B.3.1.1.2 Parameters

| Parameter Name | Type | Description | Valid Range |
|---|---|---|---|
| setEnableState | boolean | Determines whether *signalResume* is used. | TRUE = enable; FALSE= disable |

#### B.3.1.1.3 State

Not Applicable

#### B.3.1.1.4 New State

Not Applicable

#### B.3.1.1.5 Return Value

None

#### B.3.1.1.6 Originator

Service Provider, Service User.

#### B.3.1.1.7 Exceptions

None

### B.3.1.2 *spaceAvailable* Operation

The *spaceAvailable* operation is used to poll the *Data Consumer* to determine when space is available for one more packet of the maximum size.

#### B.3.1.2.1 Synopsis

*boolean spaceAvailable( );*

#### B.3.1.2.2 Parameters

None

#### B.3.1.2.3 State

Not Applicable

#### B.3.1.2.4 New State

Not Applicable

#### B.3.1.2.5 Return Value

| Type | Description | Valid Range |
|---|---|---|
| boolean | Indicates whether space is available for one more packet of the maximum size. | TRUE = space is available; FALSE = space is not available |

#### B.3.1.2.6 Originator

Service Provider, Service User.

#### B.3.1.2.7 Exceptions

None

## B.3.2 Packet::FlowSignals

### B.3.2.1 *signalResume* Operation

The *signalResume* operation is used to inform the *Data Producer* when additional space has been freed (i.e. the consumer has transitioned from a "no space available" to a "space available" condition). This operation is activated by the *enableFlowResumeSignals* operation.

#### B.3.2.1.1 Synopsis

*oneway void signalResume( );*

#### B.3.2.1.2 Parameters

None

#### B.3.2.1.3 State

Not Applicable

#### B.3.2.1.4 New State

Not Applicable

#### B.3.2.1.5 Return Value

None

#### B.3.2.1.6 Originator

Service Provider, Service User.

#### B.3.2.1.7 Exceptions

None

## B.3.3 Packet::FlowOctetStream

### B.3.3.1 *pushPacket* Operation

The *pushPacket* operation provides the ability to transfer *octet* data packets to the *Data Consumer*. It returns a *boolean* indicating whether or not space is available for additional packets.

#### B.3.3.1.1 Synopsis

*boolean pushPacket( in StreamControlType control,*
*in JTRS::OctetSequence payload)*
*raises( UnableToComplete );*

#### B.3.3.1.2 Parameters

| Parameter Name | Description | Type |
|---|---|---|
| Control | A structure containing the elements used to control the packet stream. | StreamControlType *See A.5.1.* |
| Payload | A sequence of *octet* containing the data to be pushed. | JTRS::OctetSequence *See JTRS CORBA Types* [1]. |

#### B.3.3.1.3 State

Not Applicable.

#### B.3.3.1.4 New State

Not Applicable.

#### B.3.3.1.5 Return Value

| Type | Description | Valid Range |
|---|---|---|
| boolean | Indicates whether space is available for at least one more packet of the maximum size. | TRUE = space is available; FALSE = space is not available |

#### B.3.3.1.6 Originator

Service Provider, Service User.

#### B.3.3.1.7 Exceptions

| Type | Description |
|---|---|
| UnableToComplete *See A.5.3.1.* | The *pushPacket* call was not complete. |

## B.3.4 Packet::FlowUshortStream

### B.3.4.1 *pushPacket* Operation

The *pushPacket* operation provides the ability to push *unsigned short* data packets to the *Data Consumer*. It returns a *boolean* indicating whether or not space is available for additional packets.

#### B.3.4.1.1 Synopsis

*boolean pushPacket( in StreamControlType control,*
*in JTRS::UshortSequence payload )*
*raises ( UnableToComplete);*

#### B.3.4.1.2 Parameters

| Parameter Name | Description | Type |
|---|---|---|
| control | A structure containing the elements used to control the packet stream. | StreamControlType *(See A.5.1)* |
| payload | A sequence of unsigned short contains the data to be pushed. | JTRS::UshortSequence *See JTRS CORBA Types* [1]. |

#### B.3.4.1.3 State

Not Applicable.

#### B.3.4.1.4 New State

Not Applicable.

#### B.3.4.1.5 Return Value

| Type | Description | Valid Range |
|---|---|---|
| boolean | Indicates whether space is available for at least one more packet of the maximum size. | TRUE = space is available; FALSE = space is not available |

#### B.3.4.1.6 Originator

Service Provider, Service User.

#### B.3.4.1.7 Exceptions

| Type | Description |
|---|---|
| UnableToComplete *See A.5.3.1.* | The *pushPacket* call was not complete. |

## B.4 IDL

### B.4.1 PacketFlowControl IDL

```
/*
** PacketFlowControl.idl - Packet Flow Control Interfaces
*/

#ifndef __PACKETFLOWCONTROL_DEFINED
#define __PACKETFLOWCONTROL_DEFINED

#ifndef __PACKET_DEFINED
#include "Packet.idl"
#endif

module Packet
{
	// Packet Flow Control -- Consumer provides
	interface FlowControl
	{
		boolean spaceAvailable();

		// Signaling Controls (see: FlowSignals)
		oneway void enableFlowResumeSignals( in boolean setEnableState   );
	};

	// Packet Flow Signal -- Producer provides
	interface FlowSignals
	{
		oneway void signalResume();
	};

	// Packet 'Consumers' types
	interface FlowOctetStream : PayloadStatus
	{
		boolean pushPacket(
```

```
                    in StreamControlType  control,
                    in JTRS::OctetSequence       payload
                    ) raises( UnableToComplete );
        };

        interface FlowUshortStream : PayloadStatus
        {
                boolean pushPacket(
                    in StreamControlType  control,
                    in JTRS::UshortSequence      payload
                    ) raises( UnableToComplete );
        };
};

#endif // __PACKETFLOWCONTROL_DEFINED
```

## B.5 UML

This section contains Packet Component UML Diagram and the definitions of all data types referenced (directly or indirectly) by the Service Primitives and Attributes in Section B.3.

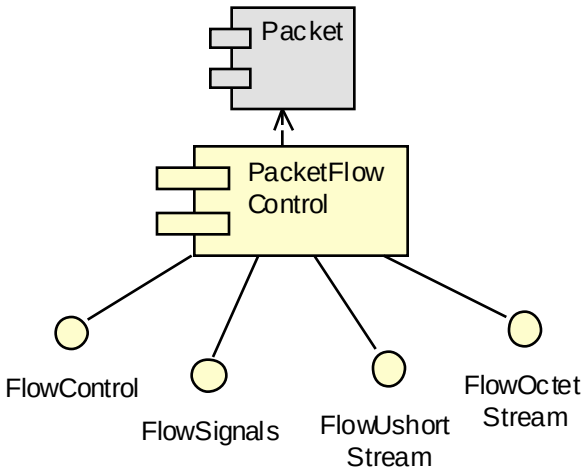


**Figure 19 – Flow Control Extension Component Diagram**

### B.5.1 Data Types

None

### B.5.2 Enumerations

None

### B.5.3 Exceptions

None

### B.5.4 Structures

None

# Appendix B.A Abbreviations and Acronyms

There are no changes from Appendix A.A.

# Appendix B.B Performance Specification

The following table provides a template for the generic performance specification for the *Flow Control Extension* which will be documented in the service or device using the interface. This performance specification corresponds to the port diagram in Figure 9.

Note: "*[DataType]*" is meant to be replaced by the data type defining the packet stream (i.e. *octet*, *ushort*).

**Table 6 – Flow Control Extension Performance Specification**

| Specification | Description | Units | Value |
|---|---|---|---|
| Worst Case Command Execution Time for *pushPacket()* on flowctrl_*[DataType]*_consumer_provides_port | * | * | * |
| Worst Case Command Execution Time for *pushPacket()* on flowctrl_ *[DataType]*_consumer_uses_port | * | * | * |
| Worst Case Command Execution Time for flowctrl_ *[DataType]*_consumer_provides_port | * | * | * |
| Worst Case Command Execution Time for flowctrl_ *[DataType]*_consumer_uses_port | * | * | * |
| Worst Case Command Execution Time for flowctrl_ *[DataType]*_producer_provides_port | * | * | * |
| Worst Case Command Execution Time for flowctrl_ *[DataType]*_producer_uses_port | * | * | * |

*Note this template will be filled in by individual developers.

# C. EMPTY SIGNALS EXTENSION

## C.1 INTRODUCTION

This extension defines a common set of *Empty Signals* interfaces to be used by JTR Set Applications and Services. The *Empty Signals Extension* provides the ability to indicate when a specified stream has been processed.

The *Empty Signals Extension* interfaces may be used in conjunction with the other interfaces (e.g. Packet Flow Control Extensions) to create the device/service *Data Producer* and *Data Consumer* interfaces.

### C.1.1 Overview

This extension contains as follows:

a. Section C.1, *Introduction*, contains the introductory material regarding the overview, service layer description, modes, states, and referenced documents of this document.

b. Section C.2, *Services*, specifies the interfaces for the component, port connections, and sequence diagrams.

c. Section C.3, *Service Primitives and Attributes,* specifies the operations that are provided by the *Empty Signals Extension*.

d. Section C.4, *IDL*.

e. Section C.5, *UML*.

f. Appendix C.A, *Abbreviations and Acronyms*.

g. Appendix C.B, *Performance Specification*.

## C.1.2 Service Layer Description

### C.1.2.1 Empty Signals Extension Port Diagram

In Figure 20, port definitions, and port names (below) are for reference only. The service/device will specify all interfaces comprising the *Data Producer* and *Data Consumer*. This diagram specifies a generic bidirectional instantiation of the *Empty Signals* interfaces to be specified by a service or device. A unidirectional instantiation may be achieved by only specifying the "uses" or the "provides" ports.

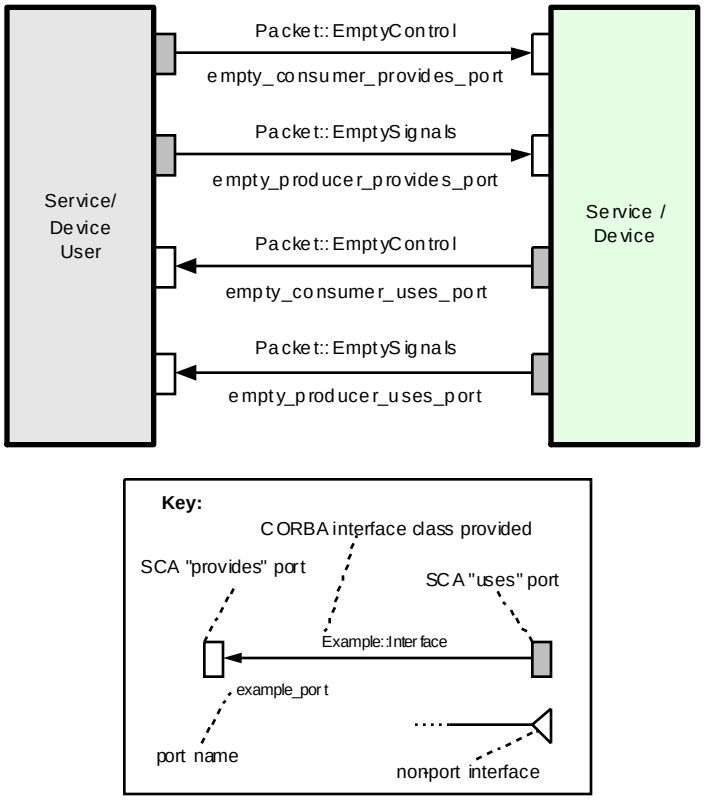


**Figure 20 – Empty Signals Extension Port Diagram**

***Packet* Empty Signals Extension Provides Ports Definitions**

**empty_consumer_provides_port** is provided by the *Device or Service* to enable the empty signals callback.
**empty_producer _provides_port** is provided by the *Device or Service* to signal an empty condition.

***Packet* Empty Signals Extension Uses Ports Definitions**

**empty_consumer_uses_port** is used by the *Device or Service* to enable the empty signals callback.
**empty_producer_uses_port** is used by the *Device or Service* to signal an empty condition.

## C.1.3 Modes of Service

Not Applicable.

## C.1.4 Service States

Not Applicable.

## C.1.5 Referenced Documents

There are no changes from the base API.

# C.2 SERVICES

## C.2.1 Provide Services

Table 7 describes the generic provides service interfaces to be specified by the service or device. These provide service interfaces correspond to the port diagram in Figure 9.

**Table 7 – Empty Signals Extension Provides Service Interfaces**

| Service Group (Port Name) | Service (Interface Provided) | Primitives (Provided) |
|---|---|---|
| empty_consumer_ provides_port | Packet:: EmptyControl | enableEmptySignals() |
| empty_producer_ provides_port | Packet:: EmptySignals | signalEmpty() |

## C.2.2 Use Services

Table 8 describes the generic uses service interfaces to be specified by the service or device. These provide service interfaces correspond to the port diagram in Figure 9.

**Table 8 – Empty Signals Extension Uses Service Interfaces**

| Service Group (Port Name) | Service (Interface Provided) | Primitives (Provided) |
|---|---|---|
| empty_consumer _uses_port | Packet:: EmptyControl | enableEmptySignals() |
| empty_producer _uses_port | Packet:: EmptySignals | signalEmpty() |

## C.2.3 Interface Modules

### C.2.3.1 **Packet**

#### C.2.3.1.1 Empty Signals Extension

The class diagram for the *Empty Signals Extension* is shown in Figure 21.

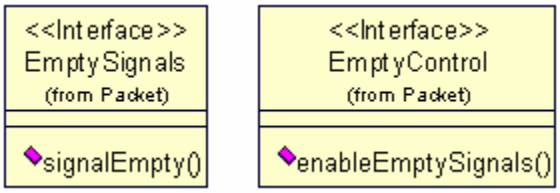


**Figure 21 – Empty Signals Class Diagram**

The *Empty Signals Data Producer and Data Consumer* interfaces are shown in Figure 22. The *Data Producer* and *Data Consumer* interfaces shown in *green* are not defined in this API. They have been provided to illustrate the collection of interfaces which will be specified by the service/device or service/device user. Additional interfaces may be required to complete the *Data Producer/Consumer* interfaces (e.g. Flow Control Extension, Section B).

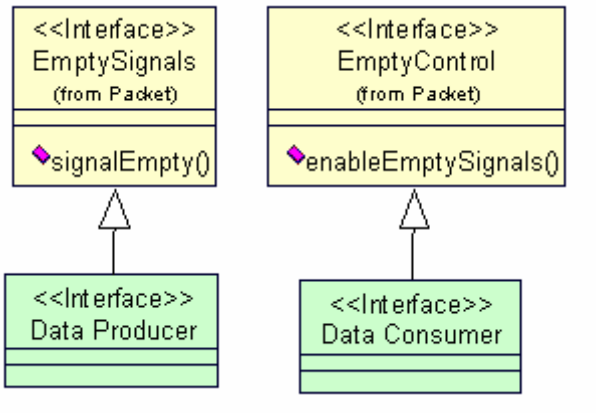


**Figure 22 – Empty Signals Data Consumer and Data Producer Interfaces**

### C.2.3.1.2 EmptyControl Interface Descriptions

The interface design of *EmptyControl* is shown in the diagram below. It provides a method to activate or deactivate the empty signal (*enableEmptySignals*).

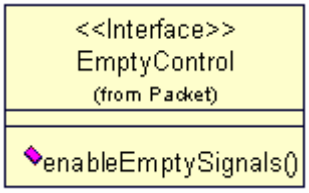


**Figure 23 – EmptyControl Interface**

### C.2.3.1.3 EmptySignals Interface Descriptions

The interface design of *EmptySignals* is shown in the diagram below. It provides methods to call or signal the empty signal (*signalEmpty*) that was enabled by the *Empty Control* interface.

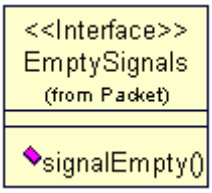


**Figure 24 – EmptySignals Interface**

# C.2.4 Sequence Diagrams

## C.2.4.1 Empty Signals Sequence Diagram

**Description**
The *Empty Signals Sequence Diagram* is shown in Figure 25 – Empty Signals Sequence Diagram
.

The *Data Producer* and *Data Consumer* in the sequence inherit those interfaces shown in Figure 22.

The *Data Producer* activates the *signalEmpty* signal by calling *enableSignalEmpty()* with input value set to TRUE and pushes packets to the *Data Consumer*. The packet received will contain information regarding whether it is the last packet in a specified stream. When the *endOfStream* condition is TRUE, the *Data Consumer* will return *signalEmpty* when the specified stream is empty.

**Pre-conditions**
None

**Post-conditions**
An empty condition is signaled by *signalEmpty*.

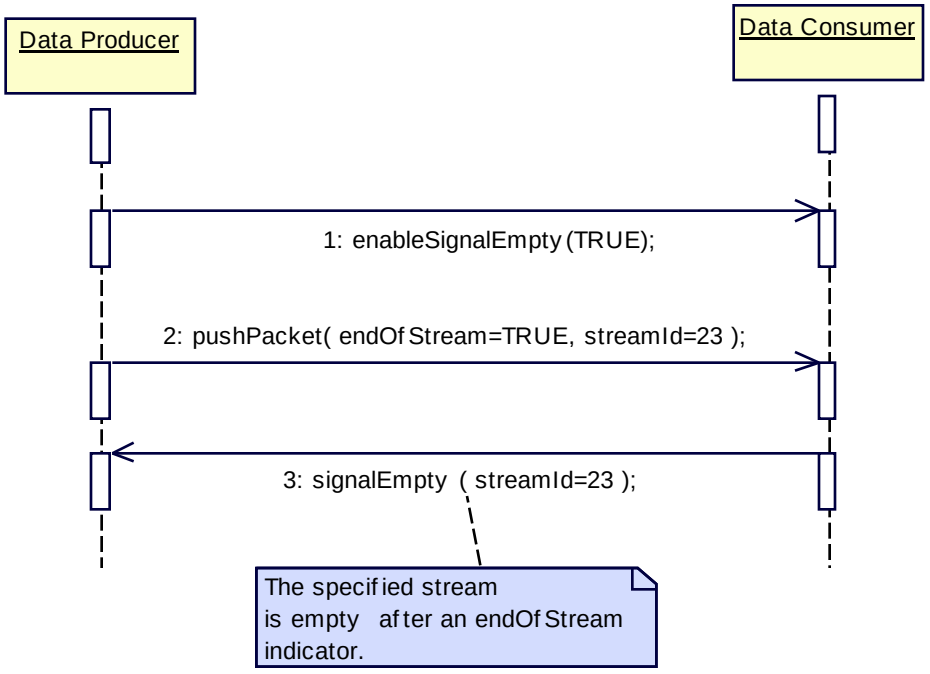


**Figure 25 – Empty Signals Sequence Diagram**

# C.3 SERVICE PRIMITIVES AND ATTRIBUTES

To enhance the readability of this API document and to avoid duplication of data, the type definitions of all structured types (i.e., data types, enumerations, exceptions, and structures) used by the service primitives and attributes have been co-located in Section C.5. This cross-reference of types also includes any nested structures in the event of a structure of structures or an array of structures.

## C.3.1 Packet::EmptyControl

### C.3.1.1 *enableEmptySignals* Operation

The *enableEmptySignals* operation is used to activate and deactivate the use of the *signalEmpty* callback operation by the *Data Consumer*. When enabled, the *Data Consumer* will signal an empty condition (using *signalEmpty*) when the stream is empty after an *endOfStream* condition was signaled.

#### C.3.1.1.1 Synopsis

*oneway void enableEmptySignals(in boolean setEnableState);*

#### C.3.1.1.2 Parameters

| Parameter Name | Type | Description | Valid Range |
|---|---|---|---|
| setEnableState | Boolean | Determines whether to generate a signal to indicate when the stream is empty. | TRUE=enable; FALSE=disable |

#### C.3.1.1.3 State

Not Applicable

#### C.3.1.1.4 New State

Not Applicable

#### C.3.1.1.5 Return Value

None

#### C.3.1.1.6 Originator

Service Provider, Service User.

#### C.3.1.1.7 Exceptions

None

## C.3.2 Packet::EmptySignals

### C.3.2.1 *signalEmpty* Operation

The *signalEmpty* operation is used to signal an empty condition to the *Data Producer* when the specified stream is empty after an *endOfStream* condition. This operation is activated by the *enableEmptySignals* operation.

#### C.3.2.1.1 Synopsis

*oneway void signalEmpty(in Stream streamId );*

#### C.3.2.1.2 Parameters

| Parameter Name | Type | Description |
|---|---|---|
| streamId | Stream<br>*See A.5.1.2.* | Indicates the stream identifier that was emptied. |

#### C.3.2.1.3 State

Not Applicable

#### C.3.2.1.4 New State

Not Applicable

#### C.3.2.1.5 Return Value

None

#### C.3.2.1.6 Originator

Service Provider, Service User.

#### C.3.2.1.7 Exceptions

None

## C.4 IDL

### C.4.1 PacketEmptySignals IDL

```
/*
** PacketEmptySignals.idl - Packet Empty Signals Interfaces
*/

#ifndef __PACKETEMPTYSIGNALS_DEFINED
#define __PACKETEMPTYSIGNALS_DEFINED

#ifndef __PACKET_DEFINED
#include "Packet.idl"
#endif

module Packet
{
	// Packet Empty Control -- Consumer provides
	interface EmptyControl
	{
		oneway void enableEmptySignals( in boolean setEnableState );
	};

	// Packet Empty Signals -- Producer provides
	interface EmptySignals
	{
		oneway void signalEmpty( in Stream streamId );
	};

};

#endif // __PACKETEMPTYSIGNALS_DEFINED
```

## C.5 UML

This section contains Packet Component UML Diagram and the definitions of all data types referenced (directly or indirectly) by the Service Primitives and Attributes in Section C.3.

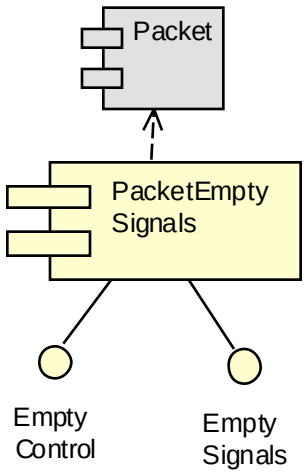


**Figure 26 – Empty Signals Extension Component Diagram**

### C.5.1 Data Types

None

### C.5.2 Enumerations

None

### C.5.3 Exceptions

None

### C.5.4 Structures

None

# Appendix C.A Abbreviations and Acronyms

There are no changes from Appendix A.A.

# Appendix C.B Performance Specification

Table 9 provides a template for the generic performance specification for the *Empty Signals Extension* which will be documented in the service or device using the interface. This performance specification corresponds to the port diagram in Figure 20.

**Table 9 – Empty Signals Extension Performance Specification**

| Specification | Description | Units | Value |
|---|---|---|---|
| Worst Case Command Execution Time for empty_consumer_provides_port | * | * | * |
| Worst Case Command Execution Time for empty_consumer_uses_port | * | * | * |
| Worst Case Command Execution Time for empty_producer_provides_port | * | * | * |
| Worst Case Command Execution Time for empty_producer_uses_port | * | * | * |

*Note this template will be filled in by individual developers.