Chapter 1

Look Inside

Overview

1.1 Introduction

This document defines requirements that must be met by gaming devices, hosts, and messaging protocols to be compliant with the GSA Point-to-Point WebSocket Transport Specification.

The GSA Point-to-Point WebSocket Transport Specification builds upon the GSA Network & Security Specification. The GSA Network & Security Specification provides the underlying foundation for the GSA Point-to-Point WebSocket Transport Specification. An implementation MUST be compliant with the GSA Network & Security Specification to be compliant with the GSA Point-to-Point WebSocket Transport Specification.

The GSA Transport Negotiation Specification provides a convenient method for clients to determine the services supported by service hosts. An implementation MUST be compliant with the GSA Transport Negotiation Specification to be compliant with the GSA Point-to-Point WebSocket Transport Specification.

1.2 Acronyms

Most of the standards and technologies on which the GSA Point-to-Point WebSocket Transport Specification is based are referred to using acronyms. These acronyms appear throughout this document. Refer to Section I.IV, Acronyms, for descriptions.

1.3 Terminology

The distinctions between the following terms can lead to confusion:

- **server** (a program providing a service),
- **client** (a program using a service),
- **service** (work performed by a server for a client), and
- **service host** (a computer on which one or more servers reside).
- **host entity** (a role defined by a protocol or profile).
- **client entity** (a role defined by a protocol or profile).

Refer to Section I.III, Definitions, for further information on these and other terms.

Chapter 2

Look Inside

WebSocket Transport

2.1 Introduction

This chapter defines a communications mechanism that uses The WebSocket Protocol for full-duplex communications over a single TCP connection. Although initially designed to serve as a communication protocol between web browsers and web servers, The WebSocket Protocol is well suited for communication between any client and any server application.

Within the GSA Point-to-Point WebSockets Specification, a variety of payloads can be communicated within a message, including:

- XML messages (xml),
- GZIP-compressed XML messages (xml-gzip), and
- EXI-encoded messages (exi).

The GSA Transport Negotiation Specification MUST be used to determine which types of payloads are actually supported by a service host as well as the URIs and security schemes of the associated services. See the GSA Transport Negotiation Specification for more details.

Per the GSA Network & Security Specification, clients and service hosts must support secure TLS communications as well as unsecure communications. The client MUST use the security scheme specified for a specific service when communicating with that service. The service host specifies the security scheme for the service as part of the URI for the service. The URI for the service is reported by the service host in the `gsaService` attribute of the response to the client’s transport option request. The token "ws" MUST be used to specify unsecure WebSocket communications; the token "wss" MUST be used to specify secure WebSocket communications. The following table provides examples of URIs for services that use the GSA Point-to-Point WebSocket Transport Specification.

Table 2.1 gsaService URI Examples

Type	URI Example
Unsecure	ws://config.casino.com:80/g2s
Secure	wss://config.casino.com:443/g2s

An HTTP GET operation is used by the client to establish the WebSocket connection with the service host. Once the WebSocket connection has been established, a GSA-defined frame structure is used by the client and the service host to communicate messages in full-duplex mode over the WebSocket connection; both the client and the service host can initiate messages over the same connection. See Chapter 3, WebSocket Transport, for more details.

2.2 Relevant Standards

The following standards are relevant to the generic negotiation mechanism. Clients and service hosts MUST conform to these standards, and their supporting standards, as specified herein.

Table 2.2 Relevant Standards

Standard	Description
EXI	Efficient XML Interchange
HTTP	RFC 7230 Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing RFC 7231 Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content
URI	RFC 3986 Uniform Resource Identifier (URI): Generic Syntax
WebSockets	RFC 6455 The WebSocket Protocol
XML	eXtensible Markup Language

2.3 Base URI

The client MUST use the value specified in the `gsaService` attribute as the base URI for the WebSocket connection to the service host. Certain message protocols may require that additional information be appended to the base URI. If no additional information is specified for a message protocol, the client MUST use the base URI to establish the WebSocket connection to the service host.

2.3.1 G2S URI

When a service supports the G2S message protocol—that is, the `gsaProtocol` attribute is set to "G2S"—the client MUST identify the EGM ID and Host ID of the G2S communications association in the URI used to make the WebSocket connection. This helps reduce the amount of bandwidth required for subsequent application-layer messages. The following table identifies the format that the client MUST use to form the URI when value of the `gsaProtocol` attribute is set to "G2S".

Table 2.3 G2S URI Format

Type	URI Format
Unsecure	[gsaService]?EGMID=[URI-encoded EGM ID]&HOSTID=[URI-encoded Host Id]
Secure	[gsaService]?EGMID=[URI-encoded EGM ID]&HOSTID=[URI-encoded Host Id]

The EGM ID and Host ID MUST be properly encoded for use in a URI. See RFC 3986, Uniform Resource Identifier (URI): Generic Syntax, for details.

Upon connection, the service host MUST verify the Host ID to ensure that the client is connecting to the correct service. If the Host ID is not correct, the service host MUST NOT accept the connection and MUST include HTTP status code "400 Bad Request" in its response.

If the Host ID is correct, the service host MUST store the Host ID and EGM ID as properties of the WebSocket connection and, subsequently, the service host MUST pass the stored values to the application layer with each application-layer message. See the G2S Message Protocol for more details on passing values to the application layer and validating those values. The Host ID and EGM ID are not provided at the transport layer in subsequent application-layer messages.

The client MUST ensure that the Host ID and EGM ID conform to the following data types and restrictions.

Table 2.4 Data Types & Restrictions

Identifier	Data Type	Restrictions
Host ID	`xs:int`	`1 <= Host ID <= 2147483647`
EGM ID	`xs:string`	`[A-Z0-9]{3}_[ -~]{1,28}`

The following table contains examples of fully-formed URIs for services that support the G2S message protocol using the WebSocket transport.

Table 2.5 G2S URI Examples

Type	URI Format
Unsecure	ws://config.casino.com:80/g2s?EGMID=ABC_1234&HOSTID=42
Secure	wss://config.casino.com:443/g2s?EGMID=ABC_1234&HOSTID=42

2.4 Opening Handshake

The client MUST initiate WebSocket communications to the service host using an HTTP GET operation. There are several key HTTP headers that the client MUST include in that initial operation. The following table identifies those headers and describes what values MUST be included for a successful WebSocket connection to be negotiated. This is not an exhaustive list. The full definition of the opening handshake and all headers is available in RFC 6455, The WebSocket Protocol.

Table 2.6 WebSocket Connection Headers

HTTP Header	Expected Value
Upgrade	websocket
Connection	Upgrade
Sec-WebSocket-Key	See RFC 6455, The WebSocket Protocol, for the definition of acceptable keys.
Sec-WebSocket-Version	13
Sec-WebSocket-Protocol	The value reported in the `gsaEncoding` attribute for the service prepended to ".gamingstandards.com"; for example, exi.gamingstandards.com.

2.4.1 Successful Connection

If the WebSocket connection request is successful, the service host will include HTTP status code "101 Switching Protocols" in its response. This indicates that the WebSocket connection has been successfully negotiated and that the client may commence full-duplex WebSocket communications with the service host.

2.4.2 Unsuccessful Connection

If the WebSocket connection request is not successful, the service host will include an HTTP status code other than "101 Switching Protocols" in its response. For example, the service host might include status code "400 Bad Request" in its response. See RFC 6455, The WebSocket Protocol, for additional details.

If the WebSocket connection request is not successful, after waiting at least 60 seconds but not more than 5 (five) minutes, the client MUST request transport options once again and, as described in the GSA Transport Negotiation Specification, try to establish a connection with the service host.

2.5 WebSocket Frame Header

Once the WebSocket connection has been established, clients and service hosts MUST include a standard WebSocket frame header, as described in RFC 6455, in every subsequent message sent over the connection.

The payloads of the WebSocket frames MUST be binary; that is, clients and service hosts MUST set the opcode field of the WebSocket frame header to 0x2.

Clients and service hosts MUST NOT utilize WebSocket payload masking. The frame-masked bit MUST be set to 0 (zero). Per RFC 6455, if the frame-masked bit is set to 0 (zero), the subsequent 4-byte frame-masking-key is omitted.

2.6 Message Flow

Once the WebSocket connection has been established, both the client and the service host may send messages over the connection. Full asynchronous communications takes place over the single WebSocket connection. The service host MUST NOT establish a second connection back to the client.

2.7 Persistent Connections

Clients and service hosts MUST support HTTP 1.1 persistent connections. Clients MUST specify persistent connections when establishing connections. Clients and service hosts MUST maintain persistent connections for at least five minutes unless the connection is no longer required at the application level.

The intent of these requirements is to avoid frequent use of "close" tokens within HTTP headers as well as "close" frames within WebSocket headers. These requirements will cause the HTTP session to be kept open between the client and the service host so that TLS and TCP connections do not have to be recreated more often than necessary.

2.8 Minimum Message Size

Clients and service hosts MUST be able to receive and process a 4 megabyte or smaller XML message. Senders do not have a mandated maximum message size requirement but should be aware that messages, which are larger than 4 megabytes, may not be processed by receivers. This requirement is intended to promote interoperability by providing senders and receivers with a known limit on the size of messages.

In some cases, such as a WAN environment with limited bandwidth capabilities, a 4 megabyte message may not be appropriate or desired. In such cases, an alternate message size requirement MAY be established with a value less than 4 megabytes, agreed upon by the GSA, and clearly identified in the GSA certification requirements. The alternate message size value MAY override the general requirement of 4 megabytes for transport certification.

Note that the WebSocket specification includes required features that allow large messages to be divided into smaller pieces and sent as fragments. Implementations must be prepared to receive large messages in fragments. See the WebSockets specification for full details.