[MS-ASP]:
ASP.NET State Server Protocol Specification

Revision Summary

Date	Revision History	Revision Class	Comments
12/18/2006	0.1		MCPP Milestone 2 Initial Availability
03/02/2007	1.0		MCPP Milestone 2
04/03/2007	1.1		Monthly release
05/11/2007	1.2		Monthly release
06/01/2007	1.2.1	Editorial	Revised and edited the technical content.

Date	Revision History	Revision Class	Comments
07/03/2007	1.2.2	Editorial	Revised and edited the technical content.
07/20/2007	1.2.3	Editorial	Revised and edited the technical content.
08/10/2007	1.2.4	Editorial	Revised and edited the technical content.
09/28/2007	2.0	Major	Updated and revised the technical content.
10/23/2007	2.0.1	Editorial	Revised and edited the technical content.
11/30/2007	2.0.2	Editorial	Revised and edited the technical content.
01/25/2008	3.0	Major	Updated and revised the technical content.

Table of Contents

1 Introduction

The ASP.NET State Server Protocol is a contract for transmitting session state data between a client and a state server. This protocol is used for interaction between a client application that requires persistent session state storage, and an out-of-process state server responsible for storing session state. The data that flows between the client application and a state server is transmitted using the Hypertext Transfer Protocol (HTTP). The ASP.NET State Server Protocol is a Microsoft-proprietary protocol.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

Hypertext Transfer Protocol (HTTP)

The following terms are specific to this document:

Application Domain: A virtual process space within which **ASP.NET** is hosted and executed. On a Web server, it is possible to have multiple **ASP.NET** Web applications running inside a single process. Each **ASP.NET** application runs within its own **application domain**, and is isolated from other **ASP.NET** applications that are running in separate **application domains**. An **application domain** has a unique identifier used as part of the identifying key on a state server when storing and retrieving session data.

ASP.NET: A Web server technology for dynamically rendering HTML pages using a combination of HTML, Javascript, CSS, and server-side logic.

ASP.NET State Server: A Windows service that provides a default server implementation of the ASP.NET State Server Protocol. When the service is enabled on a computer, that computer can act as a state server. The state server accepts requests to load, store, delete, and temporarily lock **session state** items.

Session State: A feature for temporarily storing data associated with a browser session. **Session state** can be stored outside the process space of a **session state** client. The **ASP.NET State Server** is the default implementation for storing **session state** out of process.

URL: A uniform resource locator that identifies network-addressable endpoints. In the context of the ASP.NET State Server Protocol, a **URL** is used to identify a running instance of a state server implementation. The format of a state server **URL** is as specified in [RFC1738].

User Session Identifier: A unique identifier used as part of the identifying key when storing and retrieving session data.

Web Application Identifier: Each **ASP.NET** application running on a Web server is uniquely identified with a **Web application identifier**. The **Web application identifier** is the virtual path of the Web application on the Web server. A **Web application identifier** is used as part of the identifying key on a state server when storing and retrieving session data for a specific browser session.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as specified in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[RFC1738] Berners-Lee, T., Masinter, L., and McCahill, M., "Uniform Resource Locators (URL)", RFC 1738, December 1994, http://www.ietf.org/rfc/rfc1738.txt

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt

[RFC2396] Berners-Lee, T., Fielding, R., and Masinter, L., "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998, http://www.ietf.org/rfc/rfc2396.txt

[RFC2616] Fielding, R., et al., "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999, http://www.ietf.org/rfc/rfc2616.txt

1.2.2 Informative References

[ASPNET] Microsoft Corporation, "The Official Microsoft ASP.NET 2.0 Site", http://www.asp.net/

[ASPNETDC] Microsoft Corporation, "ASP.NET Developer Center", http://msdn2.microsoft.com/en-us/asp.net/default.aspx

[MSDN-WSWN] Microsoft Corporation, "System.Web.SessionState Namespace", http://msdn2.microsoft.com/en-us/library/system.web.sessionstate(vs.71).aspx

1.3 Protocol Overview (Synopsis)

Web applications need to store state information that is associated with a specific user session. Earlier Web technologies, such as Active Server Pages (ASP), included a **session state** feature that stored state information in memory. **ASP.NET** also has an implementation of in-memory session state.

However, in-memory session state solutions are not suitable for Web farms (clusters of two or more Web servers). In Web farms there is no guarantee that a user session will reconnect to the same Web server across multiple requests. As a result, if an in-memory session state solution is used, session data will appear to be lost when the user session connects to different servers.

The ASP.NET State Server Protocol was developed to address the use of session state in a Web farm. A session state server can host an out of process session state store. Client applications such as Web applications can store and retrieve session data across a Web farm as long as each instance of the application is pointed at the same state server instance. The ASP.NET State Server Protocol specifies the rules for communicating session state data between a client application and a state server.

When using the ASP.NET State Server Protocol, there is a client and a server component to each network conversation. The general sequence is as follows:

1. A client application runs code that requires session state. For example, a Web application could process a browser request, and as part of the processing, the application needs to access the session state associated with the browser session.

2. The client sends an **HTTP** request to the state server to retrieve session state. The request includes an identifier that correlates to the browser's user session, as well as an identifier for the specific Web application making the request.

3. The state server receives the request, and based upon the **user session identifier**, **application domain** identifier, and **Web application identifier**, retrieves the requested session data. The session data is returned in an HTTP response to the client.

4. The application code accesses session state, getting and setting values as necessary.

5. When the client finishes processing a request, it makes an HTTP request to the state server to save any changes application code has made to session state. This request contains the updated session state data, as well user session, application domain, and Web application identifiers.

6. The state server accepts the HTTP request to update state data. It stores the information keyed by the user session identifier, application domain identifier, and application identifier.

1.4 Relationship to Other Protocols

The ASP.NET state server relies on HTTP (as specified in [RFC2616]).

The **URL** for the ASP.NET state server follows the definitions in "Uniform Resource Locators (URL)," as specified in [RFC1738].

The allowable characters for strings that are defined in section 2.2.1.6 come from the definition of a relative URI that is defined in "Uniform Resource Identifiers (URI): Generic Syntax," as specified in [RFC2396].

1.5 Prerequisites/Preconditions

The ASP.NET State Server Protocol is used when a client application communicates with a state server. Client and server implementations of the ASP.NET State Server Protocol should agree on how a client discovers a state server. Implementations should also agree on how a state server starts up and becomes available to process requests.

1.6 Applicability Statement

The ASP.NET State Server Protocol is intended for use by clients that need to store session state data outside the process space of the client application.

1.7 Versioning and Capability Negotiation

The ASP.NET State Server Protocol includes version information in server response messages that a client can use to implement versioning behavior. This protocol enforces strict version compatibility between client and server.

1.8 Vendor-Extensible Fields

There are no extensible fields in the ASP.NET State Server Protocol.

1.9 Standards Assignments

There are no standards specific to the ASP.NET State Server Protocol.

2 Messages

The following sections specify transport for the ASP.NET State Server Protocol and details of message syntax, including common structures, certificate requirements, and common error codes.

2.1 Transport

The ASP.NET State Server Protocol uses HTTP, as specified in [RFC2616], as the transport layer. The client indicates the requested data as part of an HTTP request header, and packages request data, as specified in section 3.2.

State server implementations MUST accept the request data and provide responses according to the specifications in section 3.1.

2.2 Message Syntax

2.2.1 Common Definitions

The following common constructions are used throughout this document. The constructions that are defined in the following sections are used only for convenience in constructing other messages; they have no other semantics.

2.2.1.1 Digit

The Digit is defined in the following code sample.

```
DIGIT = any US-ASCII digit "0".."9"
```

2.2.1.2 Octet

The Octet is defined in the following code sample.

```
OCTET = any 8-bit sequence of data
```

2.2.1.3 Carriage Return Line Feed

The Carriage Return Line Feed is defined in the following code sample.

```
CR = "\r" | ascii carriage return
LF = "\n" | ascii linefeed
CRLF = CRLF
```

2.2.1.4 Space

The Space is defined in the following code sample.

```
SP = " " | ASCII space character
```

2.2.1.5 Delimiter

The Delimiter is defined in the following code sample.

```
delimiter = "%2f" | "/"
```

2.2.1.6 Stringtext

Stringtext consists of the characters that are specified in [RFC2396] section 2.

```
stringtext = *(a-z|A-Z|0-9|+|/|=|relativeURI)
```

2.2.2 Common HTTP Headers and Fields

2.2.2.1 HTTP Version

The http-version field is as specified in [RFC2616] section 3.1.

2.2.2.2 Host Header

The host-information header field is as specified in [RFC2616] section 14.23.

2.2.2.3 Content Length

The content-length entity-header field is as specified in [RFC2616] section 14.13.

Example:

```
Content-Length: 134\r\n
```

2.2.2.4 Content

When a client or server sends session data, the session data is contained in a message body. The message body contains one or more 8-bit sequences representing the session data.

A state server implementation MUST treat content as opaque and MUST round-trip the value back to the client when requested.

Both the client and server MUST send and receive content as byte arrays (char* or byte[]).

```
content = *OCTET
```

2.2.3 State Server Headers and Fields

2.2.3.1 Application Identifier

State server implementations MUST treat the application identifier as an opaque field.<1>

```
application-identifier = stringtext
```

2.2.3.2 Application Domain Identifier

State server implementations MUST treat the application domain identifier as an opaque field.<2>

```
appdomain-identifier = "(" stringtext ")"
```

2.2.3.3 Session Identifier

State server implementations MUST treat the user session identifier as an opaque field.<3>

```
session-identifier = stringtext
```

2.2.3.4 ASP.NET Version

A state server implementation indicates which version of the state server is using this response header.<4>

```
aspnet-version =
"X-AspNet-Version:" SP ("1.x.yyyyy" | "2.x.yyyyy") CRLF
```

Example:

```
X-AspNet-Version: 2.0.50727\r\n
```

2.2.3.5 Timeout

Timeout is an integer that defines the expiry time in minutes for session state data. If this field is not set, the default time-out is 20 minutes.

```
timeout = "Timeout:" SP 1*DIGIT CRLF
```

Example:

```
Timeout: 120\r\n
```

2.2.3.6 Exclusive Lock Acquire

An ASP.NET Web server indicates that it wants to acquire an exclusive lock on session state data by including this header in the request.

```
exclusive-acquire = "Exclusive: acquire" CRLF
```

2.2.3.7 Exclusive Lock Release

A client indicates that it wants to release an exclusive lock on session state data by including this header in the request.

```
exclusive-release = "Exclusive: release" CRLF
```

2.2.3.8 Lock Date

Lock date is a 64-bit integer value that indicates the date a session state lock was created. A Lock date is measured in 100-nanosecond ticks since midnight, January 1, 0001 C.E., in the local time zone of the session state server.

```
lock-date = "LockDate:" SP 1*DIGIT CRLF
```

Example:

```
LockDate: 127916792367495010\r\n
```

2.2.3.9 Lock Cookie

Lock cookie is a 32-bit positive signed integer value indicating the lock identifier that MUST be associated for a piece of locked session state data. This value can be any positive signed integer value.

```
lock-cookie = "LockCookie:" SP 1*DIGIT CRLF
```

Example:

```
LockCookie: 12\r\n
```

2.2.3.10 Lock Age

Lock age is a 64-bit integer value indicating the current age of the lock cookie measured in 100 nanosecond click ticks.

```
lock-age = "LockAge:" SP 1*DIGIT CRLF
```

Example:

```
LockAge: 27819380594\r\n
```

2.2.3.11 Extra Flags

The extra flags represent optional information about session state that a client requires a state server to store. A client can send this header when sending session data to a state server for the first time (that is, a Set Request call). A value of "0" means no special action will be necessary for the session state data. A value of "1" means the client is creating an uninitialized session state item in the session state store.

```
extra-flags = "ExtraFlags:" SP ("0" | "1") CRLF
```

Example:

```
ExtraFlags: 1\r\n
```

2.2.3.12 Action Flags

The action flags represent optional data that a state server implementation returns to a client. A value of "0" means no special action by the client is necessary for the session state data. A value of "1" means the client MUST perform extra initialization work for the session.

```
action-flags = "ActionFlags:" SP ("0" | "1") CRLF
```

Example:

```
ActionFlags: 1\r\n
```

2.2.3.13 Unique identifier

Unique identifier is a concatenation of the application, application domain, and session identifier fields. The combined value is used as a unique identifier for session state.

```
unique-identifier =
application-identifier appdomain-identifier delimiter
session-identifier
```

Example:

```
/w3svc/root
 /fxstatebvt(NDbkwGi0191wFdDv0yOUOobtHns%3d)
%2f15hgq1uszp2tjt45lkwxmb55
```

2.2.4 Response Status Codes

2.2.4.1 Response Status Code - OK

This status code indicates that the requested operation succeeded.

```
status-code-ok = "HTTP/1.1 200 OK" CRLF
```

2.2.4.2 Response Status Code - Bad Request

This status code indicates that the state server could not understand the client request.

```
status-code-badrequest = "HTTP/1.1 400 Bad Request" CRLF
```

2.2.4.3 Response Status Code - Not Found

This status code indicates that the state server could not find the requested data.

```
status-code-notfound = "HTTP/1.1 404 Not Found" CRLF
```

2.2.4.4 Response Status Code - Locked

This status code indicates that the requested data cannot be retrieved because another instance of the user session locked the session state data for exclusive access.

```
status-code-locked = "HTTP/1.1 423 Locked" CRLF
```

2.2.5 Messages

2.2.5.1 Get_Request

The Get_Request message is sent by a client to request session state data in a non-exclusive manner. If multiple instances of the same user session are active, the request for session state data will not block session state requests from other instances of the same user session.

Sections 3.1.5.1 and 3.2.4.1 specify using this message.

```
Get_Request =
"GET" SP unique-identifier SP http-version host-information
```

Example:

```
GET /w3svc/root/fxstatebvt(NDbkwGi0191wFdDv0yOUOobtHns%3d)%
2f15hgq1uszp2tjt45lkwxmb55 HTTP/1.1
Host: 10.0.0.100:42424
```

2.2.5.2 Get_Response

The Get_Response message is sent by a state server implementation to a client in response to a Get_Request message. Sections 3.1.5.1 and 3.2.4.1 specify using this message.

```
Get_Response =
response-ok | response-bad-request | response-not-found |
response-locked
response-ok = status-code-ok content-length
aspnet-version timeout [action-flags] CRLF content
response-bad-request = status-code-badrequest
content-length aspnet-version
response-not-found = status-code-notfound
content-length aspnet-version
```

```
response-locked = status-code-locked
content-length aspnet-version lock-cookie lock-age lock-date
```

Examples:

Response-ok:

```
HTTP/1.1 200 OK
Content-Length: 2589
X-AspNet-Version: 2.0.50727
Timeout: 1200
ActionFlags: 1

...session state content here...
```

Response-bad-request (this response format is the same for all server responses):

```
HTTP/1.1 400 Bad Request
Content-Length: 589
X-AspNet-Version: 2.0.50727
```

Response-not-found (this response format is the same for all server responses):

```
HTTP/1.1 404 Not Found
Content-Length: 205
X-AspNet-Version: 2.0.50727
```

Response-locked (this response format is the same for all server responses):

```
HTTP/1.1 423 Locked
Content-Length: 589
X-AspNet-Version: 2.0.50727
LockCookie: 1
LockAge: 1275008970
LockDate: 1337890127
```

2.2.5.3 GetExclusive_Request

The GetExclusive_Request message is sent by a client to request session state data in an exclusive manner. This means the current requestor wants an exclusive lock to be maintained on the session state data until the current requestor releases the lock. If other instances of the user session request the same session state data, they will not receive any session data in a response because the current requestor has already locked it for exclusive access.

Sections 3.1.5.1 and 3.2.4.2 specify using this message.

```
GetExclusive_Request =
"GET" SP unique-identifier SP http-version host-information
```

```
exclusive-acquire
```

Example:

```
GET /w3svc/root/fxstatebvt(NDbkwGi0191wFdDv0yOUOobtHns%3d)%
2f15hgq1uszp2tjt45lkwxmb55 HTTP/1.1
Host: 10.0.0.100:42424
Exclusive: Acquire
```

2.2.5.4 GetExclusive_Response

The GetExclusive_Response message is sent by a state server implementation to a client in response to a GetExclusive_Request message.

Sections 3.1.5.2 and 3.2.4.2 specify using this message.

```
GetExclusive_Response = response-ok |
response-bad-request | response-not-found | response-locked
response-ok = status-code-ok content-length
aspnet-version timeout [action-flags] lock-cookie CRLF content
response-bad-request = status-code-badrequest
content-length aspnet-version
response-not-found = status-code-notfound
content-length aspnet-version
response-locked = status-code-locked
content-length aspnet-version lock-cookie lock-age lock-date
```

Example of response-ok:

```
HTTP/1.1 200 OK
Content-Length: 3340
X-AspNet-Version: 2.0.50727
Timeout: 30
LockCookie: 19

...session state content here...
```

2.2.5.5 Set_Request

The Set_Request message is sent by a client to a state server to store session state data for the current requestor.

Sections 3.1.5.3 and 3.2.4.3 specify using this message.

```
Set_Request =
"PUT" SP unique-identifier SP http-version host-information
content-length timeout lock-cookie [extra-flags] CRLF content
```

Example:

```
PUT /w3svc/root/fxstatebvt(NDbkwGi0191wFdDv0yOUOobtHns%3d)%
2f15hgq1uszp2tjt45lkwxmb55 HTTP/1.1
Host: 10.0.0.100:42424
Content-Length: 134
Timeout: 1200
Lock-Cookie: 12345
ExtraFlags: 1
...session state content here...
```

2.2.5.6 Set_Response

The Set_Response message is sent by a state server implementation to a client in response to a Set_Request message.

Sections 3.1.5.3 and 3.2.4.3 specify using this message.

```
Set_Response = response-ok | response-bad-request
response-ok = status-code-ok content-length aspnet-version
response-bad-request = status-code-badrequest
aspnet-version content-length
```

Example: response-ok:

```
HTTP/1.1 200 OK
Content-Length: 0
X-AspNet-Version: 2.0.50727
```

2.2.5.7 ReleaseExclusive_Request

The ReleaseExclusive_Request message is sent by a client to indicate to a state server that the current requestor is releasing its exclusive lock on a piece of session state data.

Sections 3.1.5.4 and 3.2.4.4 specify using this message.

```
ReleaseExclusive_Request =
"GET" SP unique-identifier SP http-version host-information
exclusive-release lock-cookie
```

Example:

```
GET /w3svc/root/fxstatebvt(NDbkwGi0191wFdDv0yOUOobtHns%3d)%
2f15hgq1uszp2tjt45lkwxmb55 HTTP/1.1
Host: 10.0.0.100:42424
Exclusive: release
Lock-Cookie: 12345
```

2.2.5.8 ReleaseExclusive_Response

The ReleaseExclusive_Response message is sent by a state server implementation to a client in response to a ReleaseExclusive_Request message.

Sections 3.1.5.4 and 3.2.4.4 specify using this message.

```
ReleaseExclusive_Response =
response-ok | response-bad-request | response-not-found
response-ok = status-code-ok content-length aspnet-version
response-bad-request =
status-code-badrequest content-length aspnet-version
response-not-found =
status-code-notfound content-length aspnet-version
```

Example:

```
HTTP/1.1 200 OK
Content-Length: 0
X-AspNet-Version: 2.0.50727
```

2.2.5.9 Remove_Request

The Remove_Request message is sent by a client to delete session state information associated with the current requestor on a state server.

Sections 3.1.5.5 and 3.2.4.5 specify using this message.

```
Remove_Request =
"DELETE" SP unique-identifier SP http-version
host-information lock-cookie
```

Example:

```
DELETE /w3svc/root/fxstatebvt(NDbkwGi0191wFdDv0yOUOobtHns%3d)%
2f15hgq1uszp2tjt45lkwxmb55 HTTP/1.1
Host: 10.0.0.100:42424
Lock-Cookie: 12345
```

2.2.5.10 Remove_Response

The Remove_Response message is sent by a state server implementation to a client in response to a Remove_Request message.

Sections 3.1.5.5 and 3.2.4.5 specify using this message.

```
Remove Response = response-ok | response-not-found | response-bad-request
response-ok = status-code-ok content-length aspnet-version
response-bad-request =
status-code-badrequest content-length aspnet-version
response-not-found = status-code-notfound
```

```
content-length aspnet-version
```

Example:

```
HTTP/1.1 200 OK
Content-Length: 0
X-AspNet-Version: 2.0.50727
```

2.2.5.11 ResetTimeout_Request

The ResetTimeout_Request message is sent by a client to reset the time-out counter of session state data that is associated with the current requestor so that the data is not automatically released by the state server.

Sections 3.1.5.6 and 3.2.4.6 specify use of this message.

```
ResetTimeout_Request =
"HEAD" SP unique-identifier SP http-version host-information
```

Example:

```
HEAD /w3svc/root/fxstatebvt(NDbkwGi0191wFdDv0yOUOobtHns%3d)%
2f15hgq1uszp2tjt45lkwxmb55 HTTP/1.1
Host: 10.0.0.100:42424
```

2.2.5.12 ResetTimeout_Response

The ResetTimeout_Response message is sent by a state server implementation to a client in response to a ResetTimeout_Request message.

Sections 3.1.5.6 and 3.2.4.6 specify using this message.

```
ResetTimeout_Response =
response-ok | response-not-found
response-ok =
status-code-ok content-length aspnet-version
response-not-found =
status-code-notfound content-length aspnet-version
```

Example:

```
HTTP/1.1 200 OK
Content-Length: 0
X-AspNet-Version: 2.0.50727
```

3 Protocol Details

The following sections specify protocol details that include client role, server details, and algorithms.

3.1 Server Details

3.1.1 Abstract Data Model

An application that operates in a stateless manner may need to store data associated with certain transient operations. If a unique identifier is flowed from one transient operation to another, it is possible for an application to store data externally keyed to this unique identifier. Client applications can load this data by providing a unique key to a state server implementation.

A state server provides the service for externally storing such data. It operates against a virtual storage model, where individual pieces of data are indexed via a unique identifier. Each unique identifier points to a table of name-value pairs containing session state information.

When a client requests session state information, the server finds the unique identifier and returns the name-value pairs associated with that key. When a client updates session state information, it provides the state server with the unique key and a set of name-value pairs. The state server stores the name-value pairs and associates them with the unique key for subsequent retrieval.

The state server supports basic locking semantics to ensure that concurrent read and write attempts do not corrupt session state.

3.1.2 Timers

None.

3.1.3 Initialization

The initialization requirements for a state server are implementation-dependent.<5>

3.1.4 Higher-Layer Triggered Events

None.

3.1.5 Message Processing Events and Sequencing Rules

3.1.5.1 Processing Non-Exclusive Get Requests

A client that uses a state server makes either an exclusive or a non-exclusive request to a state server implementation for session state data.<6>

For a non-exclusive request, the client sends an HTTP request that uses the message format that is specified in section 2.2.5.1. A state server implementation MUST attempt to retrieve the session state data that corresponds to the unique identifier that is contained in the combination of application-identifier, appdomain-identifier, and session-identifier. A state server MUST not interpret these values or assign any specific relevance to them. Rather, a state server implementation MUST simply use the combination of those values as the unique identifier for retrieving any previously stored session state that is associated with the combination of those identifiers.

The state server MUST send a response back to the client by using one of the message formats that are specified in section 2.2.5.2.

If the state server finds session data that is associated with the requested identifier, and the data is not locked by another request, it MUST reply to the client Web server by using the response-ok message, as specified in section 2.2.5.2. As part of this message, the state server MUST include the action-flags information if during a previous set operation, as specified in section 2.2.5.5, the client Web server sent extra-flags along with session state data.

If the state server finds session data that is associated with the requested identifier, but the session data is locked by another request (for example, two or more clients are simultaneously running and each client is using the same identifier), the state server MUST respond with a response-locked message, as specified in section 2.2.5.2. The response-locked message contains a lock-age (section 2.2.3.10) and lock-date (section 2.2.3.8) in addition to the value of the current lock-cookie (section 2.2.3.9). The lock-cookie is an integer representation of the current lock. The lock-date value MUST contain the date and time that the existing lock was placed on the session state data. The lock-age header MUST contain a representation for the age of the current lock.

If the state server cannot find any session data that is associated with the requested identifier, the state server MUST respond with a response-not-found message, as specified in section 2.2.5.2.

The response-bad-request message, as specified in section 2.2.5.2, is conceptually equivalent to throwing an exception. The session state server MUST send this message if something goes wrong and the server is unable to process the request.

3.1.5.2 Processing Exclusive Get Requests

A client that uses a state server makes either an exclusive or a non-exclusive request to a state server implementation for session state data.<7>

For an exclusive request, the client uses the message format that is specified in section 2.2.5.3. A state server implementation MUST attempt to retrieve the session state data that corresponds to the unique identifier that is contained in the combination of application-identifier, appdomain-identifier, and session-identifier. A state server MUST not interpret these values or assign any specific relevance to them. Rather, a state server implementation MUST simply use the combination of those values as the unique identifier for retrieving any previously stored session state that is associated with the combination of those identifiers.

The state server MUST send a response back to the client by using one of the message formats that are specified in section 2.2.5.4.

If the state server finds session data that is associated with the requested identifier and the data is not locked by another request, it MUST reply to the client with the response-ok message. As part of this message, the state server MUST include the action-flags information, if during a previous set operation, as specified in section 2.2.5.5, the client sent extra-flags along with session state data.

The state server MUST also internally mark the session data in a way that indicates the session data is now considered locked, and should not be made available to other requesters. As part of this logical operation, the state server MUST return an integer representation of the lock to the client. The lock-cookie (section 2.2.3.9) portion of the response-ok message is where the state server MUST include this lock information in its response. Internally, the state server MUST also note the date and time when the lock is established.

If the state server finds session data that is associated with the requested identifier but the session data is locked by another request (that is, two or more clients are simultaneously running, and each client is using the same identifier), the state server MUST respond by using a response-locked message, as specified in section 2.2.5.4. The response-locked message contains a lock-age (section 2.2.3.10) and lock-date (section 2.2.3.8) in addition to the value of the current lock-cookie. The lock-cookie is an integer representation of the current lock. The lock-date value MUST contain the

date and time the existing lock was placed on the session state data. The lock-age header MUST contain a representation for the age of the current lock.

If the state server cannot find any session data that is associated with the requested identifier, the state server MUST respond with a response-not-found message, as specified in section 2.2.5.4.

The response-bad-request message, as specified in section 2.2.5.4, is conceptually equivalent to throwing an exception. The session state server MUST send this message if something goes wrong and the server is unable to process the request.

3.1.5.3 Saving Session Data with a Set Request

When a client needs to store session data in an out-of-process state server, it makes a request to the state server by using the message format that is specified in section 2.2.5.5.

A state server MUST store the session data that is contained in this message and associate it with the unique identifier that is contained in the combination of application-identifier, appdomain-identifier, and session-identifier. A state server MUST not interpret these values or assign any specific relevance to them. Rather, a state server implementation MUST simply use the combination of those values as the unique identifier for storing session state associated with the combination of those identifiers.

If the client sent the optional extra-flags value, the state server MUST also store this information along with the session state data. The state server MUST be able to return this value in the action-flags value of the response messages, as specified in sections 2.2.5.2 and 2.2.5.4.

Because a client sends a lock-cookie value along with the session state data, the state server MUST store the lock-cookie value. Internally, the state server MUST also store the date and time when the state server received the lock-cookie value. This information is necessary if the state server ever has to send response-locked messages, as specified in sections 2.2.5.2 and 2.2.5.4.

The state server MUST also store the time-out value that is sent from the client. This time-out value is returned as part of the response-ok messages that are specified in sections 2.2.5.2 and 2.2.5.4. This value is also used when refreshing session state time-outs by using the message as specified in section 2.2.5.11.

Internally, the state server MUST also store the date and time of the current request to save session state. This date and time information is necessary for the state server to remove out-of-date session data. In order to prevent memory or storage exhaustion from storing data for an infinite time period, a state server implementation MUST implement some type of cleanup or scavenging mechanism that can detect expired sessions. A session is considered expired if the current date and time is greater than the session time-out value that is added to the date and time of either the last Set_Request message or the last ResetTimeout_Request message.

The state server MUST send a response back to the client by using one of the message formats that are specified in section 2.2.5.6.

If the state server successfully stored the session state, it MUST return a response-ok message.

The response-bad-request message, as specified in section 2.2.5.6, is conceptually equivalent to throwing an exception. The session state server MUST send this message if something goes wrong and the server is unable to process the request.

3.1.5.4 Releasing an Exclusive Session State Lock

A client can acquire an exclusive lock on a session state by using either a successful GetExclusive_Request or Set_Request message. In either case, the client has the lock-cookie value that is associated with a piece of locked session state. Alternatively, a client can acquire the lock-cookie value of a locked piece of session state from failed calls to GetExclusive_Request or Get_Request, where a response-locked message was returned.

A client sends a ReleaseExclusive_Request message to request that the lock on a piece of session state data be released. A state server implementation MUST construct a unique identifier based on the values contained in the combination of application-identifier, appdomain-identifier, and session-identifier. A state server does not need to interpret these values or assign any specific relevance to them. Rather, a state server implementation MUST simply use the combination of those values as the unique identifier for referencing previously stored session state that is associated with the combination of those identifiers.

The state server MUST compare the lock-cookie value that is associated with the unique identifier to the lock-cookie value that is sent by the client. If the values match, the state server MUST release the lock on the session state data and respond to the Web server with a response-ok message, as specified in section 2.2.5.8.

If the state server cannot find any session data that is associated with the unique identifier and lock-cookie, the state server MUST respond with a response-not-found message, as specified in section 2.2.5.8.

The response-bad-request message, as specified in section 2.2.5.8, is conceptually equivalent to throwing an exception. The session state server MUST send this message if something goes wrong and the server is unable to process the request.

3.1.5.5 Removing Session State

A client can acquire an exclusive lock on session state by using either a successful GetExclusive_Request or Set_Request message. In either case, the client has a lock-cookie value that is associated with a piece of locked session state. Although a client can obtain a lock-cookie value from failed attempts to get session state, a client only sends a Remove_Request message if the client was able to successfully obtain an exclusive lock through a previous GetExclusive_Request or Set_Request operation.

A client sends a Remove_Request message to request that a specific set of session data be removed from the state server. A state server implementation MUST construct a unique identifier that is based on the values that are contained in the combination of application-identifier, appdomain-identifier, and session-identifier. A state server does not need to interpret these values or assign any specific relevance to them. Rather a state server implementation MUST simply use the combination of those values as the unique identifier for referencing the previously stored session state that is associated with the combination of those identifiers.

The state server MUST compare the lock-cookie value that is associated with the unique identifier, to the lock-cookie value that is sent by the client. If the values match, the state server MUST remove the corresponding session state data and respond to the Web server by using a response-ok message, as specified in section 2.2.5.10.

If the state server cannot find any session data that is associated with the requested identifier and lock-cookie, the state server MUST respond to the Web server by using a response-not-found message, as specified in section 2.2.5.10.

The response-bad-request message, as specified in section 2.2.5.10, is conceptually equivalent to throwing an exception. The session state server MUST send this message if something goes wrong and the server is unable to process the request.

3.1.5.6 Resetting Session State Time-out

A client can send a ResetTimeout Request message to request that a state server refresh the time-out for a specific piece of session data.

A state server implementation MUST construct a unique identifier that is based on the values that are contained in the combination of application-identifier, appdomain-identifier, and session-identifier. A state server does not need to interpret these values or assign any specific relevance to them. Rather, a state server implementation MUST simply use the combination of those values as the unique identifier for referencing the previously stored session state that is associated with the combination of those identifiers.

If the state server finds session state data that is associated with the unique identifier, it MUST increase the expiration date of the session state. The new expiration date for the session state data MUST be set to the time-out value. This value was previously supplied as part of a Set Request plus the current date and time on the state server. After the expiration date of the session state has been successfully updated, the state server MUST send a response-ok message to the client, as specified in section 2.2.5.12.

If the state server cannot find any session data that is associated with the requested identifier, the state server MUST respond with a response-not-found message, as specified in section 2.2.5.12.

3.1.6 Timer Events

None.

3.1.7 Other Local Events

None.

3.2 Client Details

3.2.1 Abstract Data Model

An application that operates in a stateless manner may need to store data that is associated with certain transient operations. If a unique identifier is flowed from one transient operation to another, it is possible for an application to externally store data that is keyed to this unique identifier. Client applications can load this data by providing a unique key to a state server implementation.<8>

A state server provides the service for externally storing such data. It operates against a virtual storage model, where individual pieces of data are indexed by using a unique identifier. Each unique identifier points at a table of name-value pairs that contain session state information.

When a client requests session state information, the server finds the unique identifier and returns the name-value pairs that are associated with that key. When a client updates session state information, it provides the state server with the unique key and a set of name-value pairs. The state server stores the name-value pairs and associates them with the unique key for subsequent retrieval.

The state server supports basic locking semantics to ensure that concurrent read and write attempts do not corrupt session state.

3.2.2 Timers

None.

3.2.3 Initialization

The initialization requirements for client startup are implementation-dependent.<9>

There is one per-request initialization requirement for every client message. All requests to a state server require information that uniquely identifies session state information. The unique identifier is a combination of application identifier, application domain identifier, and session identifier. The specific values that are used for these fields are implementation-dependent.<10> However, client implementations MUST ensure that the combined values for these fields are unique. In other words, at least one of the three identifiers has to be unique to ensure that a state server can differentiate between different pieces of session state information.

3.2.4 Message Processing Events and Sequencing Rules

3.2.4.1 Non-Exclusive Get Requests

A client that uses a state server makes either an exclusive or a non-exclusive request to a state server implementation for session state data.<11>

For a non-exclusive request, the client sends an HTTP request by using the message format that is specified in section 2.2.5.1.

If the server responds with a response-ok message, as specified in section 2.2.5.2, the client MUST perform implementation-specific initialization tasks if the server returns an action-flags value of "1". However, because this is a non-exclusive get request, a client MUST not attempt to send session state updates back to a state server. As a result, any side effects from initialization tasks that change session state information MUST not be sent back to the state server.

If the server responds with a response-locked message, as specified in section 2.2.5.2, the client MAY retain the lock-cookie, lock-age, and lock-date values for use with custom concurrency handling.<12>

If the server responds with either a response-bad-request or a response-not-found message, as specified in section 2.2.5.2, the client MAY surface some type of error back to the client's caller.<13>

3.2.4.2 Exclusive Get Requests

A client that uses a state server makes either an exclusive or a non-exclusive request to a state server implementation for session state data.<14>

For an exclusive request, the client sends an HTTP request by using the message format that is specified in section 2.2.5.3.

If the server responds with a response-ok message, as specified in section 2.2.5.4, the client MUST perform implementation-specific initialization tasks if the server returns an action-flags value of "1". The client MUST retain the value of the lock-cookie field. If the client needs to update the session state data, it MUST send the same lock-cookie value back to the state server as part of a Set_Request message. When the client no longer requires an exclusive lock on the session state data, it MUST send the same lock-cookie value back to the state server. This occurs when the client releases its lock with a ReleaseExclusive_Request message.

If the server responds with a response-locked message, as specified in section 2.2.5.4, the client MAY retain the lock-cookie, lock-age, and lock-date values for use with custom concurrency handling.<15>

If the server responds with either a response-bad-request or a response-not-found message, as specified in section 2.2.5.4, the client MAY surface some type of error back to the client's caller.<16>

3.2.4.3 Saving Session Data with a Set Request

When a client needs to store session data in a state server, it makes a request to the state server by using the message format that is specified in section 2.2.5.5.

If the client needs to perform custom initialization tasks on the session state data during a subsequent Get_Request or GetExclusive_Request, the client MUST set the extra-flags field to "1".

If this is the first time that the session state data is being stored for the unique identifier that is contained in the combination of application-identifier, appdomain-identifier, and session-identifier, the client MUST generate a lock-cookie that conforms to the format that is specified in section 2.2.3.9. It is always the responsibility of the client to create the lock-cookie value that is initially associated with a piece of session state data.

The value of the **timeout** field in the Set_Request message is implementation-dependent.<17>

If the server responds with a response-bad-request, as specified in section 2.2.5.6, the client MAY surface some type of error back to the client's caller.<18>

3.2.4.4 Releasing an Exclusive Session State Lock

A client can acquire an exclusive lock on session state with either a successful GetExclusive_Request or Set_Request message. In either case, the client has the lock-cookie value associated with a piece of locked session state. Alternatively, a client can acquire the lock-cookie value of a locked piece of session state from failed calls to GetExclusive_Request or Get_Request where a response-locked message was returned.

A client requests that the lock on a piece of session state data should be released using a ReleaseExclusive_Request message.

If the server responds with either a response-bad-request or a response-not-found message, as specified in section 2.2.5.8, the client MAY surface some type of error back to the client's caller.<19>

3.2.4.5 Removing Session State

A client can acquire an exclusive lock on session state with either a successful GetExclusive_Request or Set_Request message. In either case, the client has a lock-cookie value that is associated with a piece of locked session state. Although a client can obtain a lock-cookie value from failed attempts to get session state, a client MUST only send a Remove_Request message if the client was able to successfully obtain an exclusive lock through a previous GetExclusive_Request or Set_Request operation.

A client sends a Remove_Request message to request that a specific set of session data be removed from the state server.

If the server responds with a response-bad-request, as specified in section 2.2.5.10, the client MAY surface some type of error back to the client's caller.<20>

3.2.4.6 Resetting Session State Time-out

A client can send a ResetTimeout_Request message to request that a state server refresh the time-out for a specific piece of session data.

If the server responds with a response-not-found message, as specified in section 2.2.5.12, the client MAY surface some type of error back to the client's caller.<21>

3.2.5 Timer Events

None.

3.2.6 Other Local Events

None.

4 Protocol Examples

A client sends a Set Request to the state server in order to create new session data:

```
PUT /w3svc/root/fxstatebvt(NDbkwGi0191wFdDv0yOUOobtHns%3d)%
2f15hgq1uszp2tjt45lkwxmb55 HTTP/1.1
Host: 172.30.189.147:42424
Content-Length: 2381
Timeout: 10
Lock-Cookie: 1
ExtraFlags: 0
<actual session data>
```

The server responds:

```
HTTP/1.1 200 OK
Content-Length: 0
X-AspNet-Version: 2.0.50727
```

The client sends a GetExclusive Request to the state server:

```
GET /w3svc/root/fxstatebvt(NDbkwGi0191wFdDv0yOUOobtHns%3d)%
2f15hgq1uszp2tjt45lkwxmb55 HTTP/1.1
Host: 172.30.189.147:42424
Exclusive: Acquire
Content-Length: 184
```

The server responds:

```
HTTP/1.1 200 OK
Content-Length: 2561
X-AspNet-Version: 2.0.50727
Timeout: 10
LockCookie: 1

...session state content here...
```

A different client tries to get the same item that was locked by using a Get Request:

```
GET /w3svc/root/fxstatebvt(NDbkwGi0191wFdDv0yOUOobtHns%3d)%
2f15hgq1uszp2tjt45lkwxmb55 HTTP/1.1
Host: 172.30.189.147:42424
Content-Length: 163
```

The server responds:

```
HTTP/1.1 423 Locked
Content-Length: 378
X-AspNet-Version: 2.0.50727
```

```
LockCookie: 1
LockAge: 1275008970
LockDate: 1337890127
```

A client with lock updates the session by using Set_Request:

```
PUT /w3svc/root/fxstatebvt(NDbkwGi0191wFdDv0yOUOobtHns%3d)%
2f15hgq1uszp2tjt45lkwxmb55 HTTP/1.1
Host: 172.30.189.147:42424
Content-Length: 2981
Timeout: 10
Lock-Cookie: 1
ExtraFlags: 0
<updated session data>
```

The server responds:

```
HTTP/1.1 200 OK
Content-Length: 0
X-AspNet-Version: 2.0.50727
```

A client with lock then releases it by using ReleaseExclusive_Request:

```
GET /w3svc/root/fxstatebvt(NDbkwGi0191wFdDv0yOUOobtHns%3d)%
2f15hgq1uszp2tjt45lkwxmb55 HTTP/1.1
Host: 172.30.189.147:42424
Exclusive: release
Lock-Cookie: 1
```

The server responds:

```
HTTP/1.1 200 OK
Content-Length: 0
X-AspNet-Version: 2.0.50727
```

A second client tries to get the session again by using the non-exclusive Get_Request:

```
GET /w3svc/root/fxstatebvt(NDbkwGi0191wFdDv0yOUOobtHns%3d)%
2f15hgq1uszp2tjt45lkwxmb55 HTTP/1.1
Host: 172.30.189.147:42424
Content-Length: 163
```

The server responds:

```
HTTP/1.1 200 OK
Content-Length: 2982
X-AspNet-Version: 2.0.50727
Timeout: 20
```

```
<updated session data>
```

5 Security

The following sections specify security considerations for administrators.

5.1 Security Considerations for Implementers

There are no security considerations for implementers of this protocol.

5.2 Index of Security Parameters

There are no security parameters for this protocol.

6 Appendix A: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows Server 2008
- Windows Server 2003
- Windows Vista
- Windows XP
- Windows 2000
- Windows NT

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

<1> Section 2.2.3.1: An ASP.NET Web server uses the virtual path of the current application as an application identifier.

<2> Section 2.2.3.2: An ASP.NET Web server obtains the application domain identifier from HttpRuntime.AppDomainAppIdInternal and then hashes the value by using the ASP.NET machine validation key. The result is then encoded by using base64. It is the base64-encoded representation that an ASP.NET Web server uses as the application domain identifier of a Web application.

<3> Section 2.2.3.3: ASP.NET Web servers acting as session state clients use a specific value for this field.

<4> Section 2.2.3.4: A state server implementation MUST follow certain conventions for this field when it is used with an ASP.NET Web server as the client.

<5> Section 3.1.3: The default Microsoft state server implementation requires that the state server is started and running prior to its use by a client.

<6> Section 3.1.5.1: The ASP.NET Web server allows developers to specify whether Web pages require exclusive or non-exclusive access to session state.

<7> Section 3.1.5.2: The ASP.NET Web server allows developers to specify whether Web pages require exclusive or non-exclusive access to session state.

<8> Section 3.2.1: ASP.NET stores a unique session identifier in an HTTP cookie that the browser passes back to an ASP.NET Web server on each request.

<9> Section 3.2.3: The default Microsoft state server implementation requires that the state server is started and running prior to its use by a client.

<10> Section 3.2.3: The default Microsoft client implementation uses specific values for these fields, as described in Appendix A.

<11> Section 3.2.4.1: The ASP.NET Web server allows developers to specify whether Web pages require exclusive or non-exclusive access to session state.

<12> Section 3.2.4.1: The default Microsoft client retains these values and uses them to attempt to asynchronously unlock session state when the lock time has expired.

<13> Section 3.2.4.1: The default Microsoft client raises an exception if a bad request occurred.

<14> Section 3.2.4.2: The ASP.NET Web server allows developers to specify whether Web pages require exclusive or non-exclusive access to session state.

<15> Section 3.2.4.2: The default Microsoft client retains these values and uses them to attempt to asynchronously unlock session state when the lock time has expired.

<16> Section 3.2.4.2: The default Microsoft client raises an exception if a bad request occurred.

<17> Section 3.2.4.3: The default Microsoft client obtains this value from the configuration.

<18> Section 3.2.4.3: The default Microsoft client raises an exception if a bad request occurred.

<19> Section 3.2.4.4: The default Microsoft client raises an exception if a bad request occurred.

<20> Section 3.2.4.5: The default Microsoft client raises an exception if a bad request occurred.

<21> Section 3.2.4.6: The default Microsoft client raises an exception.

7 Index