Network Working Group
INTERNET-DRAFT
Category: Informational
<draft-blount-acct-msix-00.txt>
28 July 1999

Alan Blount
Derek Young
MetraTech Corp.

Metered Service Information eXchange Protocol Specification Version 1.2

Status of this Memo

Abstract

This document defines MSIX, a protocol that enables transmission of service definitions and service usage information from client to server. Services can be defined dynamically such that a metering server can become capable of handling new metered service types during the normal course of business without need for recompilation or reconfiguration. Service usage description semantics support transactional data submission. Support for both simple and compound transactions is provided.

Table of Contents

1. Introduction
   1.1. Service Definition
   1.2. Session Description
   1.3. Unsupported

2. Terminology and Notation
3. Protocol Foundation
   3.1. Protocol Base
   3.2. XML
4. Common Element
   4.1. Message Wrappers
   4.2. Common Elements
   4.3. Status Responses
5. Message Sets
   5.1. Service Definition
   5.2. Session Submission
   5.3. Utilities
6. Security Considerations
7. References
8. Acknowledgments
9. Authors' Addresses
Appendix A. Document Type Definition
Appendix B. Timestamps
Appendix C. Example Protocol Exchanges

1. Introduction

This document defines the Metered Services Information eXchange (MSIX) specification. The acronym is pronounced "M6".

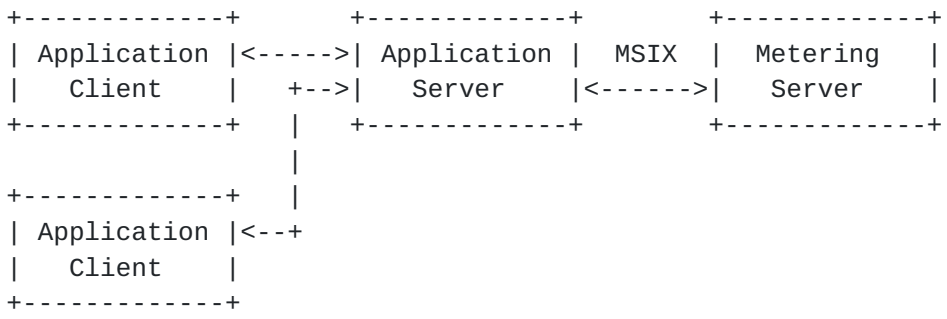

Fig. 1 -- MSIX Applicability

MSIX is used between application servers and metering servers. Application servers provide services to end users. Metering servers record descriptions of instances of service usage.

Any sort of metered service is applicable to description by MSIX: the transfer of information (a web page, an audio clip, a feature film, a phone call, a video or audio conference), processing of information (remote computation), storage of information (remote file systems). Service-usage information describes the aspects of the use of a server that a server records for later tracking, billing, and audit purposes.

MSIX enables two communication facilities: service definition and session submission. MSIX itself does not define services. That is, MSIX has no built-in notion of the measurable properties of a phone call, network connection, or any other service. MSIX provides definition of base data types (integers, strings, floats, and so on).

MSIX clients (Application Servers) are free to provide their own service definitions, which contain collections of these base types, to MSIX metering servers. MSIX metering server administrators are responsible for providing their own mapping from the service descriptions to any back-end processing (rating functions, logging policy, and so forth). MSIX does not address any billing or tracking issues except metering.

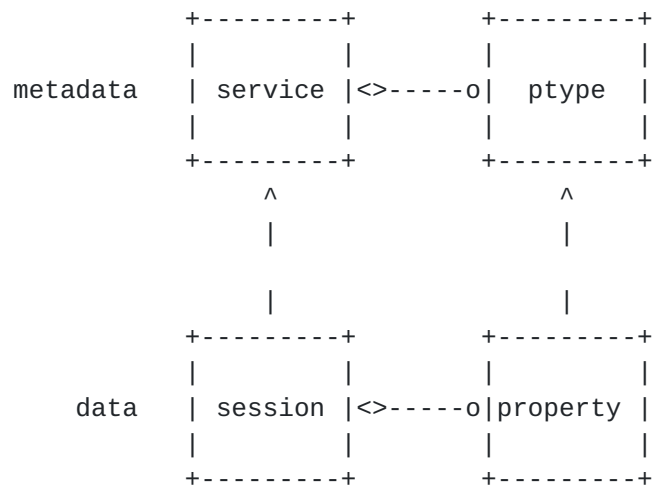

Fig. 2 -- Object Schema

1.1. Service Definition

Service definitions provide a schema for metered session description. Services are identified by a distinguished name (dn) and contain a set of property type, or "ptype" descriptions. Each ptype describes static type information for a particular property of a "session," which is an instantiation of a service.

A service relation facility enables the definition of "compound" services, which are services that refrerence other services as child or parent. These compounds services are useful for describing multifaceted billable events like telephone conference calls, which are events well represented as a collection of sub-events (the individual calls to or from the conference server). The explicit definition of a compound service enables an MSIX server to optionally verify that submitted sessions conform to a particular service hierarchy.

1.2. Session Submission

A Session is a description of an instance of service usage. Sessions are composed of a client-assigned unique ID (uid), a reference to a service, and a collection of properties. The schema of a particular session must match the schema defined by the service it references.

1.2.1. Bulk Data Transfer

Sessions can be transmitted in bulk during a single network connection, or piecewise over a number of subsequent network

connections. Each request from client to server carries a unique ID that can be used to correlate it to the server's response.

1.2.2. Transaction Support

Sessions can be described and committed in a single message, or they can be started in one message, added to or updated in another, and committed in yet another.

1.3. Unsupported

Facilities such as service and session deletion and query are out of the scope of this protocol. The authors expect that these facilities may become the subject of a related document that will address these and other administrative issues. This document deals with only those communication facilities necessary for an application server (MSIX client) to describe the service it performs to an MSIX server, and to submit descriptions of instances of service performance to an MSIX server.

2. Terminology and Notation

The following terms are used throughout the document.

Application Server
: An entity that provides application services to clients. Application servers are typically clients of MSIX servers.

Exchange
: A client request followed by a server response, each bearing the same unique identifier. Some exchanges are transactional (for example, session submission), some are not (for example, service definition).

Message
: An atomic, ordered collection of bytes that travels between client and server.

Property
: A (typically metered) component of a session. A session describing a phone call, for instance, might have the property "twenty seconds", that describes its duration

Ptype
: A component of a service definition that describes a property's metadata. A service definition for a phone call might have the ptype "duration", of type INT32.

Request
: A message that travels from client to server.

Response
   A message that travels from server to client.

Service
A service is a type of task that is performed by an application server for a client.

Session
Describes a particular instance of service-usage by an end-user.

3. Protocol Foundation

3.1 Protocol Base

MSIX requires a connection-oriented, reliable, byte stream protocol for transport. MSIX messages are transfered using the Hypertext Transfer Protocol (HTTP) [3], running over TCP/IP [8][7].

If an MSIX client or server is to be deployed in a hostile network, a security layer is required as well. In such cases, MSIX is used atop the Secure Sockets Layer [4] or Transport Layer Security [6].

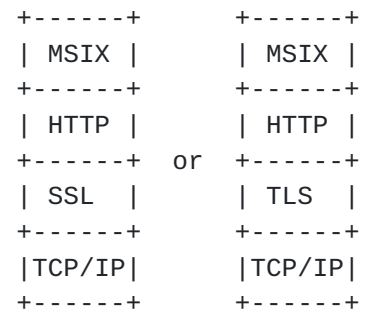

Fig. 3 -- Protocol Stacks

It is not strictly necessary that MSIX run atop a secure transport. In a secure network environment, entities could communicate by MSIX without the need for connection-level security. For the most part, MSIX is expected to be deployed in the open Internet and corporate intranets, where not all connected entities can be trusted.

3.2. XML

MSIX messages are formatted using the eXtensible Markup Language (XML) [2], which provides for tagged data. An XML Document Type Declaration for MSIX is located in Appendix A.

4. Common Elements

This section defines the MSIX root entity and details elements that appear throughout the protocol.

4.1. MSIX Root Entity

MSIX is a request-response protocol. An exchange is comprised of a request-response pairs that travel between client and server. An exchange is distinguished by the uid attribute within the msix root element.

```
<!DOCTYPE msix [
<!ELEMENT msix ( defineservice | beginsession | updatesession |
    commitsession | abortsession | status |
    defineservicers | beginsessionrs | updatesessionrs |
    commitsessionrs | abortsessionrs )
<!ATTLIST msix version   CDATA #FIXED '1.2'
               timestamp CDATA #REQUIRED
               uid       ID    #REQUIRED > ] >
```

version
MSIX Protocol version. The value of the version described by this document is fixed at '1.2'. MSIX versions are identified as follows:

```
version ::= major.minor
major ::= DIGIT+
minor ::= DIGIT+
```

Versions are ordered as floats, where the major number takes precedence over the minor.

timestamp
Denotes the time that the message was sent. Time is represented using a restricted form of the ISO 8601:1988 [9] timestamp. The definition is given in Appendix B.

uid
Identifies an MSIX client request/response pair. An MSIX server responds to a given request using the request's uid as its own. Also used as an attribute by messages that manipulate sessions.

Clients are responsible to provide a uid that is unique to the MSIX server. An MSIX server must reject client requests that have uids that conflict with uids of other messages (1) to which the server has not yet responded or (2) that contain a transactional request that is currently in the OPEN state. An MSIX server may reject a client message if it is unsatisfied with the uniqueness of its uid for any reason.

The value of the uid is to be treated by both client and server as an opaque string. Implementations must not extract information from this string for other uses.

Two forms of uids are suggested. The first, to be generally used, has the following production:

```
uid ::= "gen:/" hostname "/" unixtime "/" random "/" counter

hostname ::= ALPHANUM+ (.ALPHANUM+)*
```

Hostname of client

```
      unixtime ::= DIGIT+
         Time of message creation, in seconds since 12:00 a.m. January
         1, 1970 UTC.
```

random ::= DIGIT+
A large (8+ digit) random number.

counter ::= DIGIT+
A monotonically increasing counter, incremented for each message sent. Client should ensure that this counter does not reset to zero at system restarts.

Example: "gen:/foo.msix.org/929383942/6001338297/723"

The second form is used when the MSIX client is reading service usage information from a flat file that contains a plurality of transaction records. The client wishes to prevent accidental reruns, such that if the same client is run a second time against the same file, the MSIX server will reject the requests as duplicates. This is useful in instances where a flat file created by a legacy system lacks unique transaction identifiers.

uid ::= "hash:/" hostname "/" hash "/" recordno

hostname ::= ALPHANUM+ (.ALPHANUM+)*
Hostname of client

hash ::= 32ALPHANUM
The MD5 [5] hash of the source file.

recordno ::= DIGIT+
Zero-based index of record count.

Example: "hash:/foo.msix.org/8df6f439509bb7c0d430718a7c558321/15"

4.2. Common Elements

There are a number of elements used throughout the protocol.

<!ELEMENT dn (#PCDATA)>
Distinguished Name. Used to distinguish services and ptypes.

<!ELEMENT description (#PCDATA)>
<!ATTLIST description xml:lang NMTOKEN #IMPLIED>

Human-readable description. Generally more verbose than the distinguished name. Not intended to be programatically parsed.

4.3. Status Responses

A status response is composed of a status code, an optional status message, and an optional status detail. The status code element is the string "msix.org/" followed by a three digit integer result code of the attempt to understand and satisfy the request. The status

message is intended to give a short textual description of the status code. The status detail may be used to provide additional information about the results. The status code is intended for use

by automata and the status message and detail are intended for the human user. The client is not required to examine or display the status message or status detail.

The first digit of the status code defines the class of response. The last two digits do not have any categorization role. There are four values for the first digit:

msix.org/1xx: Informational - Request received, continuing process

msix.org/2xx: Success - The action was successfully received, understood, and accepted

msix.org/4xx: Client Error - The request contains bad syntax or cannot be fulfilled

msix.org/5xx: Server Error - The server failed to fulfill an apparently valid request

Each element of a status code is an alphanumeric string, without whitespace and optionally with dashes or underscores. Elements are separated by single slashes ("/"). The top-level vendor element is a domain name that belongs to the organization defining the status code.

The msix.org vendor prefix is specified so that user-specified status codes can be used in future versions of this protocol. This feature is not currently addressed or specified.

Status responses are represented as follows:

```
<!ELEMENT status ( code, message?, detail?) >

<!ELEMENT code (#PCDATA)>
   Short human-speakable identifier. A unique distinguished name.

<!ELEMENT message (#PCDATA)>
   Human-readable status message.

<!ELEMENT detail (#PCDATA)>
   Optional additional information about a particular instance of an
   error.
```

There are a number of status response messages that are common to several different requests. They are as follows:

code	message
msix.org/200	Request successful
msix.org/400	Bad request
msix.org/401	Request unauthorized
msix.org/408	Transaction timeout
msix.org/500	Internal Server Error
msix.org/501	Not implemented
msix.org/503	Service Unavailable
msix.org/505	MSIX version not supported

Status elements are always sent as internal components of defineservicers, relateservicers, beginsessionrs, updatesessionrs, commitsessionrs, abortsessionrs, and getversionrs elements. When an MSIX server fails to understand a request, it must respond with a status element within the top-level msix element, and omit all other response elements.

5. Message Sets

This section defines the messages that MSIX-compliant entities must support. Exchanges are composed of a request-response message pair. A request message definition is identified in this document by the REQUEST label at the top of the definition. The RESPONSE label identifies specific response messages. Each response includes the set of possible status codes that can result from the request. Note that the 5xx and 2xx status codes are not specifically identified in these blocks, as server errors are valid responses to any request, and success (one hopes) is always possible.

MSIX defines three message sets: service definition, session submission, and utilities.

5.1 Service Definition

This section describes how services are defined. Service definitions are performed by the client using a defineservice request. Service definitions are generally given names that are human-readable and pronounceable. Service names are represented as follows:

```
vendor/service/service...
```

Each element of a service name is an alphanumeric string, without whitespace and optionally with dashes or underscores. Elements are separated by single slashes ("/"). The top-level vendor element is a domain name that belongs to the organization defining the service. For example, if the organization that holds server.net were to define

voice and fax services, their service names would be of the form server.net/voicecall, server.net/fax, and so on. Vendors are responsible for maintaining their own namespace of services underneath their vendor element.

5.1.1. Define Service

5.1.1.1. defineservice REQUEST

This request initiates service definition. The combination of distinguished name and version must be unique.

Services are distinguished by two attributes: Their distinguished name (dn), and version. An MSIX server must reject a service definition request that carries a dn and version identical to a preexisting service definition.

A successful defineservice request results in the MSIX server sending a defineservicers response that contains the distinguished name and version of the the service that was submitted.

```
<!ELEMENT defineservice ( dn, version, description, ptype* ) >

<!ELEMENT dn (#PCDATA)>
   Unique name of service.  See section 5.1 for definition.

<!ELEMENT version (#PCDATA)>
   Version of service.

<!ELEMENT description (#PCDATA)>
<!ATTLIST description xml:lang NMTOKEN #IMPLIED>
   Human-readable description of service.

<!ELEMENT ptype ( dn, type, description?, defaultvalue? ) >
<!ATTLIST ptype required (Y | N) "N">

<!ELEMENT dn (#PCDATA)>
   Must be unique in containing defineservice element.

<!ELEMENT type (#PCDATA)>
   Type of data.  Valid types defined in table below.

required (y | n)
   If n, each session that is an instance of this service must
   contain a property of this ptype.
```

Value	Description
STRING	Arbitrary-length character string
UNISTRING	Arbitrary-length UTF8-encoded character string
INT32	String representation of 4-byte signed integer
FLOAT	String representation of IEEE 4-byte floating point number
DOUBLE	String representation of IEEE 8-byte floating point number
BOOLEAN	Single ASCII character: T (0x54) or F (0x46)
TIMESTAMP	ISO 8601:1988 date string, as defined in appendix

```
<!ELEMENT description (#PCDATA)>
<!ATTLIST description xml:lang NMTOKEN #IMPLIED>
   Optional description

<!ELEMENT defaultvalue (#PCDATA)>
   Optional default value
```

5.1.1.2. defineservicers RESPONSE

Service definition can fail for a number of reasons. The distinguished names of each each contained ptype must be unique. Conflicts cause an error to be returned. The dn and version combination must be unique. The ptype elements must be well-formed.

```
<!ELEMENT defineservicers ( status, dn, version ) >

<!ELEMENT status ( code, message?, detail? )>
```

code	message
msix.org/400	Bad request
msix.org/401	Request unauthorized
msix.org/defineservicers/450	dn name collision -- service of this version already defined
msix.org/defineservicers/451	Invalid ptype: multiple identical dn's
msix.org/defineservicers/452	Invalid ptype: specifies unsupported type

```
   <!ELEMENT dn (#PCDATA)>
      Distinguished name of service.

   <!ELEMENT version (#PCDATA)>
      Service version
```

5.1.2. Relate Services

Relationships can exist amongst services. Services can be parents or children of other services. Parent-child relationships serve to establish billing constraints among services and the sessions that are instantiations of them.

Sessions that have children are not considered complete until all their children have completed. This facility enables an MSIX client to instruct the server to wait on further processing of an ongoing transaction (rating, creating a bill, and so on) until it is complete. For instance, a conference call session can contain a collection of individual calls to and from a conference bridge. The conference bridge, an MSIX client, can report the individual calls by sending several messages over time, and then commit a parent "conference" session when the call has completed.

Services that are not related to other services are termed simple services. Services that are related to other services are called compound services.

A client may specify that a relationship is to be optional. If a relationship is not required, a session whose corresponding service has a parent may or may not identify a parent session. If a service has a required relationship, an MSIX server must reject sessions that identify parent sessions for which a corresponding service parent-child relationship has not been defined.

Relationships are not versioned. A relationship between parent and child applies to all versions of both parent and child.

5.1.2.1. relateservices REQUEST

```
<!ELEMENT relateservices ( parentdn, childdn ) > <!ATTLIST
relateservices required (y | n) "n">

<!ELEMENT parentdn (#PCDATA)>
   Distinguished name of parent service. See section 5.1 for
   definition.

<!ELEMENT childdn (#PCDATA)>
   Distinguished name of child service. See section 5.1 for
   definition.
```

5.1.2.2. relateservices RESPONSE

```
<!ELEMENT relateservicers ( status ) >

<!ELEMENT status ( code, message?, detail? )>
```

code	message
msix.org/400	Bad request
msix.org/401	Request unauthorized
msix.org/relateservicesrs/450	One or more unknown or invalid dn's
msix.org/relateservicesrs/451	Services already related

5.2. Session Submission

This section describes the messages that communicate service usage information between client and server.

As with services, sessions can be simple or compound. Simple sessions are those that are described by a single beginsession message. Compound sessions are submitted piecewise. A beginsession request may be followed by other beginsession requests, each of the latter referencing the former by use of its parentid element.

Sessions must match the schema of the service to which their service dn refers.

Each session, whether parent or child, must be committed before the MSIX server will further act on it (for example, make it available to a back end rating/billing system). If a parent session is committed or aborted, the commit or abort is recursively cascaded to all its children. Committed sessions may not be updated.

A client may committ a session immediately, without sending a separate commitsession message, by setting a commit attribute in a beginsession or updatesession message to "Y".

5.2.1. Transaction State

Session submission semantics are transactional so that events that happen to transpire across accounting/billing period boundaries won't be billed across two accounting/billing periods.

A beginsession request starts a transaction. Transactions are completed by one of three methods: commit by the client, abort by the client, or abort (timeout) by the server.

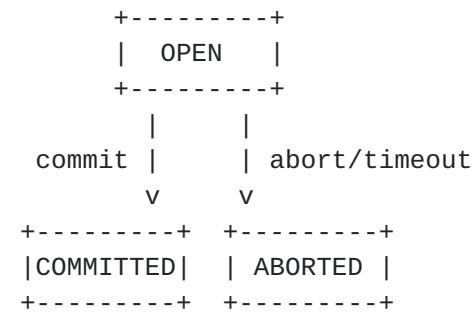

Fig. 4 -- State Transitions

This specification does not discuss the parameters that govern server timeout.

5.2.2. Begin Session

This request initiates a session transaction. The session transaction is maintained in the OPEN state until commit, abort, or timeout. The server returns a beginsessionrs response that identifies the uid of the session.

A beginsession request must not identify a committed session as its parent.

If the commit attribute is set to Y, the session is to be immediately committed.

Note that the uids of session submission are not the uids that are attributes of the msix root element. Separate uids are required for each beginsession request and are referenced in subsequent updatesession, commitsession, and abortsession requests.

5.2.2.1. beginsession REQUEST

```
<!ELEMENT beginsession ( dn, uid, parentid?, property* ) >
<!ATTLIST beginsession commit (y | n) "n">
   commit
      y or n.  If y, commit submission immediately

<!ELEMENT dn (#PCDATA)>
   Distinguished Name of service

<!ELEMENT uid (#PCDATA)>
   Unique identifier for this session submission

<!ELEMENT parentid (#PCDATA)>
   uid of parent session, if compound

<!ELEMENT property ( dn, value ) >
```

If the client wishes to make use of the default value defined by the referenced ptype, the client must refrain from sending a property element for that ptype in the session.

```
<!ELEMENT dn (#PCDATA)>
   Distinguished name -- used to identify <ptype>

<!ELEMENT value (#PCDATA)> Value of property.
```

5.2.2.2. beginsessionrs RESPONSE

```
<!ELEMENT beginsessionrs ( status, uid ) >

<!ELEMENT status ( code, message?, detail? )>
```

code	message
msix.org/400	Bad request
msix.org/401	Request unauthorized
msix.org/beginsessionrs/150	Undefined service dn
msix.org/beginsessionrs/400	Invalid parentid
msix.org/beginsessionrs/401	Invalid property aggregate: multiple identical dn's
msix.org/beginsessionrs/402	Invalid property: nonexistent ptype
msix.org/beginsessionrs/403	non-unique session uid
msix.org/beginsessionrs/404	required property not set

```
<!ELEMENT uid (#PCDATA)>
   uid of session
```

5.2.3. Update Session

This request asks an MSIX server to modify its record of an OPEN session. The server's existing knowledge of a particular session will be replaced by a new version. This facility enables a client to periodically refresh an MSIX server with new information about a service being performed.

As with beginsession, the commit attribute is available as a shortcut to committing a transaction. If the commit element is set to Y, the session is to be immediately committed. After commit, no further updates are allowed.

5.2.3.1. updatesession REQUEST

```
   <!ELEMENT updatesession ( uid, property* ) >
   <!ATTLIST updatesession commit (y | n) "n">
```

commit

y or n. If y, commit submission immediately

<!ELEMENT uid (#PCDATA)>
Unique identifier of the session being updated

The property element is defined in section 5.2.2.1. Note that all properties in a session need not be updated--only those that the client wishes to update. The "required" attribute is enforced only for beginsession and not for updatesession.

5.2.3.2. updatesessionrs RESPONSE

<!ELEMENT updatesessionrs (status, uid) >

<!ELEMENT status (code, message?, detail?)>

code	message
msix.org/400	Bad request
msix.org/401	Request unauthorized
msix.org/408	Transaction timeout
msix.org/updatesessionrs/400	Nonexistent session uid
msix.org/updatesessionrs/401	Invalid property aggregate: multiple identical dn's
msix.org/updatesessionrs/402	Invalid property: nonexistent ptype

5.2.4. Commit Session

This message requests that an MSIX server commit a session transaction. Commit of a parent session recursively cascades commit to all child sessions.

5.2.4.1. commitsession REQUEST

<!ELEMENT commitsession (uid) >

<!ELEMENT uid (#PCDATA)>
uid of beginsession request

5.2.4.2. commitsessionrs RESPONSE

<!ELEMENT commitsessionrs (status, uid) >

```
   <!ELEMENT status ( code, message?, detail? )>
```

code	message
msix.org/400	Bad request
msix.org/401	Request unauthorized
msix.org/408	Transaction timeout
msix.org/commitsessionrs/400	Nonexistent session uid
msix.org/commitsessionrs/401	Session transaction not OPEN

```
<!ELEMENT uid (#PCDATA)>
   uid of committed session
```

5.2.5. Abort Session

This message requests that an MSIX server abort a session transaction. Abort of a parent session recursively cascades abort to all child sessions.

An aborted transaction will not be provided as a billable record to a back-end rating/billing system.

5.2.5.1. abortsession REQUEST

```
<!ELEMENT abortsession ( uid ) >

<!ELEMENT uid (#PCDATA)>
   uid of beginsession request
```

5.2.5.2. abortsessionrs RESPONSE

```
<!ELEMENT abortsessionrs ( status, uid ) >

<!ELEMENT status ( code, message?, detail? )>
```

code	message
msix.org/400	Bad request
msix.org/401	Request unauthorized
msix.org/408	Transaction timeout
msix.org/commitsessionrs/400	Nonexistent session uid

```
   |msix.org/commitsessionrs/401 |Session transaction not OPEN        |
   +-----------------------------+-------------------------------------+

   <!ELEMENT uid (#PCDATA)>
```

uid of aborted session

5.3 Utilities

This section describes messages of general utility. They are not specific to service definition or session description.

5.3.1 Get Versions

Requests that the MSIX server identify all versions of the MSIX protocol that it supports. The MSIX server returns one or more version elements.

All future versions of MSIX must support the getversions request as defined in this document.

5.3.1.1 getversions

```
<!ELEMENT getversions EMPTY ) >
```

5.3.1.2 getversionsrs

```
<!ELEMENT getversionsrs ( status, version+ ) >

<!ELEMENT status ( code, message?, detail? )>
```

code	message
msix.org/400	Bad request

6. Security Considerations

MSIX depends on its underlying transport for security. In a non-secure environment, a number of attacks are possible. Services can be created or removed, false session records can be transmitted, and so on. No authorization facilities within MSIX are defined or supported.

7. References

[1] Bradner, S. "The Internet Standards Process -- Revision 3", RFC 2026, October 1996.

[2] Bray, T., J. Paoli, and C. Sperberg-McQueen, "Extensible Markup Language (XML) 1.0", W3C Recommendation, February 1998.

[3] Fielding, R., J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. "Hypertext Transfer Protocol--HTTP/1.1", RFC 2068,

January 1997.

[4] Freir, Alan O., P. Karlton, and P. Kocher, "The SSL Protocol Version 3.0, Netscape Communications Corporation, March 1996.

[5] Rivest, R. "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[6] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[7] Postel, J., editor, "INTERNET PROTOCOL", RFC 791, September 1981.

[8] Postel, J., editor, "TRANSMISSION CONTROL PROTOCOL", RFC 793, September 1981.

[9] "Data elements and interchange formats -- Information interchange -- Representation of dates and times", ISO 8601:1988.

8. Acknowledgments

The authors thank the MSIX partners for their guidance and support, and Eric Hughes for his early advice in the shaping of this protocol.

The Aurora team at NetCentric provided technical input and criticism. Team members included Gilbert Benghiat, Navdip Bhachech, Philip Kenny, Frank Kim, Bill O'Donnell, Jeff Rago, Scott Swartz, Yiwen Woo, and Derek Young. Michael Weintraub, Jianchao Wang, and Tsu-Junk Kung at GTE provided valuable criticism and advice.

The description of a timestamp specification that appears in the appendices was lifted from the HTML/4.0 specification, edited by Dave Raggett, Arnaud Le Hors, and Ian Jacobs. Text describing the status message set and a number of the error codes was poached from RFC 2068, by R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee.

9. Authors' Addresses

Alan Blount
MetraTech Corp.
411 Waverley Oaks Road
Waltham, MA 02452
Email: blount@alum.mit.edu

Derek Young
MetraTech Corp.
411 Waverley Oaks Road
Waltham, MA 02452
Phone: +1 781 398 2242
Fax: +1 781 398 2232
Email: dyoung@metratech.com

Appendices

Appendix A. Document Type Definition

```
<!--  Root Document  -->
<!DOCTYPE msix [
```

```
<!ELEMENT msix ( defineservice | beginsession | updatesession |
    commitsession | abortsession | status |
    defineservicers | beginsessionrs | updatesessionrs |
    commitsessionrs | abortsessionrs )
<!ATTLIST msix version   CDATA #FIXED '1.2'
               timestamp CDATA #REQUIRED
               uid       ID    #REQUIRED > ] >

<!--  Misc. elements and attributes -->
<!ELEMENT version (#PCDATA)>
<!ELEMENT timestamp (#PCDATA)>
<!ELEMENT uid (#PCDATA)>
<!ELEMENT dn (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ATTLIST description xml:lang NMTOKEN #IMPLIED>
<!ELEMENT parentdn (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT defaultvalue (#PCDATA)>
<!ELEMENT parentid (#PCDATA)>

<!--  ptype support  -->
<!ELEMENT ptype ( dn, type, description?, defaultvalue? ) >
<!ATTLIST ptype required (Y | N) "N">

<!--  property support  -->
<!ELEMENT property ( dn, value ) >
<!ELEMENT value (#PCDATA)>

<!--  status support  -->
<!ELEMENT status ( code, message?, detail? )>
<!ELEMENT code (#PCDATA)>
<!ELEMENT message (#PCDATA)>
<!ELEMENT detail (#PCDATA)>

<!--  Message sets  -->

<!--  Define Service  -->
<!ELEMENT defineservice ( dn, version, description, ptype* ) >
<!ELEMENT defineservicers ( status, dn, version ) >

<!--  Relate Service  -->
<!ELEMENT relateservices ( parentdn, childdn ) >
<!ATTLIST relateservices required (y | n) "n">
<!ELEMENT parentdn (#PCDATA)>
<!ELEMENT childdn (#PCDATA)>
<!ELEMENT relateservicers ( status ) >

<!--  Begin Session  -->
<!ELEMENT beginsession ( dn, uid, parentid?, property* ) >
```

```
   <!ATTLIST beginsession commit (Y | N) "N">
   <!ELEMENT beginsessionrs ( uid ) >

   <!--  Update Session  -->
   <!ELEMENT updatesession ( uid, property* ) >
```

```
   <!ATTLIST updatesession commit (y | n) "n">
   <!ELEMENT updatesessionrs ( status, uid ) >

   <!--  Commit Session  -->
   <!ELEMENT commitsession ( uid ) >
   <!ELEMENT commitsessionrs ( status, uid ) >

   <!--  Abort Session  -->
   <!ELEMENT abortsession ( uid ) >
   <!ELEMENT abortsessionrs ( status, uid ) >

   <!--  Get Versions  -->
   <!ELEMENT getversions EMPTY ) >
   <!ELEMENT getversionsrs ( version+ ) >

   ] >
```

Appendix B. Timestamps

MSIX supports a form of timestamp representation defined by ISO 8601:1988[1]. ISO 8601:1988 allows many options and variations in the representation of dates and times. This specification defines a specific format which is one of those allowed by ISO 8601:1988.

The format is:

```
   YYYY-MM-DDThh:mm:ssTZD
```

where:

```
   YYYY = four-digit year
   MM   = two-digit month (01=January, etc.)
   DD   = two-digit day of month (01 through 31)
   hh   = two-digits of hour (00 through 23) (am/pm NOT allowed)
   mm   = two digits of minute (00 through 59)
   ss   = two digits of second (00 through 59)
   TZD  = time zone designator
```

The time zone designator is one of:

```
Z
   indicates UTC (Coordinated Universal Time).
+hh:mm
   indicates that the time is a local time that is hh hours and mm
   minutes ahead of UTC.
-hh:mm
   indicates that the time is a local time that is hh hours and mm
   minutes behind UTC.
```

Exactly the components shown here must be present, with exactly this

punctuation. Note that the "T" appears literally in the string, to indicate the beginning of the time element, as defined in ISO8601.

If a generating application does not know the time to the second, it

may use the value "00" for the seconds (and minutes and hours if necessary). Both of the following examples correspond to November 5, 1994, 8:15:30 am, US Eastern Standard Time.

1994-11-05T13:15:30Z
1994-11-05T08:15:30-05:00

Appendix C. Example Message Exchanges

This section contains a number of exchanges that illustrate the use of MSIX. Material in braces "[]" is not literal. Literal replacements will be substituted in a later draft of this document.

C.1. Simple Service Definition

In this example a client defines a new service.

Client -> Server

Client sends a <defineservice> request.

```
POST cgi/msix HTTP/1.0
Content-Type: text/plain
Content-Length: [FIXME]

<?xml version=1.0?>
<msix version="1.2" timestamp="1997-07-01T15:25:01Z" uid="[uid0]">
    <defineservice>
        <dn>server.net/Fonecall</dn>
        <version>7.3</version>
        <description>Internet to PSTN telephone call</description>
        <ptype>
            <dn>AccountId</dn>
            <type>STRING</type>
        </ptype>
        <ptype>
            <dn>DialedNumber</dn>
            <type>STRING</type>
        </ptype>
        <ptype>
            <dn>Duration<dn>
            <type>INT32</type>
        </ptype>
        <ptype>
            <dn>StartTime</dn>
            <type>TIMESTAMP</type>
        </ptype>
    </defineservice>
</msix>
```

Server -> Client

Server responds with a defineservicers element.

```
HTTP/1.0 200 OK
Content-Type: text/plain
Content-Length: [FIXME]

<?xml version=1.0?>
<msix version="1.2" timestamp="1997-07-01T15:25:02Z" uid="[uid0]">
    <defineservicers>
        <status>
            <code>msix.org/200</code>
        </status>
        <dn>server.net/FoneCall</dn>
        <version>7.3</version>
    </defineservicers>
</msix>
```

C.2. Simple Session Submission

In this example a client submits a simple session to a server.

Client -> Server

```
POST cgi/msix HTTP/1.0
Content-Type: text/plain
Content-Length: [FIXME]

<?xml version=1.0?>
<msix version="1.2" timestamp="1997-07-01T15:25:03Z" uid="[uid1]">
    <beginsession commit="y">
        <uid>[uid2]</uid>
        <dn>server.net/FoneCall</dn>
        <property>
            <dn>AccountId</dn>
            <value>324955</value>
        </property>
        <property>
            <dn>DialedNumber</dn>
            <value>+16177205200</value>
        </property>
        <property>
            <dn>Duration</dn>
            <value>280</value>
        </property>
        <property>
            <dn>StartTime</dn>
            <value>1997-06-06T09:35:22Z</value>
        </property>
    </beginsession>
</msix>
```

Server -> Client

Server responds with a beginsessionrs message that includes the uid of the session.

```
HTTP/1.0 200 OK
Content-Type: text/plain
Content-Length: [FIXME]

<?xml version=1.0?>
<msix version="1.2" timestamp="1997-07-01T15:25:03Z" uid="[uid1]">
    <beginsessionrs>
        <status>
            <code>msix.org/200</code>
        </status>
        <uid>[uid2]</uid>
    </beginsessionrs>
</msix>
```

C.3. Compound Service Definition

In this example a client defines a new compound service.

Client -> Server

Client sends a defineservice request.

```
POST cgi/msix HTTP/1.0
Content-Type: text/plain
Content-Length: [FIXME]

<?xml version=1.0?>
<msix version="1.2" timestamp="1997-07-01T15:26:01Z" uid="[uid5]">
    <defineservice>
        <dn>server.net/FaxBroadcast</dn>
        <version>2.4</version>
        <description>Multiple Destination Fax</description>
        <ptype>
            <dn>AccountId</dn>
            <type>STRING</type>
        </ptype>
        <ptype>
            <dn>Priority</dn>
            <type>STRING</type>
        </ptype>
    </defineservice>
</msix>
```

Server -> Client

Server responds with a defineservicers element.

```
HTTP/1.0 200 OK
Content-Type: text/plain
Content-Length: [FIXME]
```

```
<?xml version=1.0?>
<msix version="1.2" timestamp="1997-07-01T15:26:02Z" uid="[uid5]">
    <defineservicers>
```

```
          <status>
              <code>msix.org/200</code>
          </status>
          <dn>server.net/FaxBroadcast</dn>
          <version>2.4</version>
      </defineservicers>
  </msix>

  Client -> Server

  Client sends another defineservice request.

  POST cgi/msix HTTP/1.0
  Content-Type: text/plain
  Content-Length: [FIXME]

  <?xml version=1.0?>
  <msix version="1.2" timestamp="1997-07-01T15:26:03Z" uid="[uid6]">
      <defineservice>
          <dn>server.net/FaxBroadcast/Fax</dn>
          <version>2.6</version>
          <description>Multiple Destination Fax</description>
          <ptype>
              <dn>DialedNumber</dn>
              <type>STRING</type>
          </ptype>
          <ptype>
              <dn>Duration</dn>
              <type>INT32</type>
          </ptype>
          <ptype>
              <dn>StartTime</dn>
              <type>TIMESTAMP</type>
          </ptype>
          <ptype>
              <dn>BitRate</dn>
              <type>INT32</type>
          </ptype>
      </defineservice>
  </msix>

  Server -> Client

  Server responds with a defineservicers element.

  HTTP/1.0 200 OK
  Content-Type: text/plain
  Content-Length: [FIXME]
```

```
   <?xml version=1.0?>
   <msix version="1.2" timestamp="1997-07-01T15:26:04Z" uid="[uid6]">
       <defineservicers>
           <status>
               <code>msix.org/200</code>
```

```
            </status>
            <dn>server.net/FaxBroadcast/Fax</dn>
            <version>2.6</version>
        </defineservicers>
    </msix>

    Client -> Server

    Client creates a parent-child relationship between the two services.

    POST cgi/msix HTTP/1.0
    Content-Type: text/plain
    Content-Length: [FIXME]

    <?xml version=1.0?>
    <msix version="1.2" timestamp="1997-07-01T15:26:05Z" uid="[uid7]">
        <relateservices required="y">
            <parentdn>server.net/FaxBroadcast</parentdn>
            <childdn>server.net/FaxBroadcast/Fax</childdn>
        </relateservices>
    </msix>

    Server -> Client

    Server responds whith a relateservicesrs element.

    HTTP/1.0 200 OK
    Content-Type: text/plain
    Content-Length: [FIXME]

    <?xml version=1.0?>
    <msix version="1.2" timestamp="1997-07-01T15:26:06Z" uid="[uid7]">
        <relateservicesrs>
            <status>
                <code>msix.org/200</code>
            </status>
        </relateservicesrs>
    </msix>
```

C.4. Compound Session Submission

```
    In this example a client submits a compound session.

    Client -> Server

    Client sends a "parent" session.

    POST cgi/msix HTTP/1.0
    Content-Type: text/plain
    Content-Length: [FIXME]
```

```
<?xml version=1.0?>
<msix version="1.2" timestamp="1997-07-01T15:27:03Z" uid="[uid10]">
    <beginsession commit="n">
```

```
            <dn>server.net/FaxBroadcast</dn>
            <uid>[uid11]</uid>
            <property>
                <dn>AccountId</dn>
                <value>bozo22</value>
            </property>
            <property>
                <dn>Priority</dn>
                <value>HIGH</value>
            </property>
        </beginsession>
    </msix>
```

Server -> Client

Server responds with a beginsessionrs messages.

```
    HTTP/1.0 200 OK
    Content-Type: text/plain
    Content-Length: [FIXME]

    <?xml version=1.0?>
    <msix version="1.2" timestamp="1997-07-01T15:27:03Z" uid="[uid10]">
        <beginsessionrs>
            <status>
                <code>msix.org/200</code>
            </status>
            <uid>[uid11]</uid>
        </beginsessionrs>
    </msix>
```

Client -> Server

Client sends a "child" session.

```
    POST cgi/msix HTTP/1.0
    Content-Type: text/plain
    Content-Length: [FIXME]

    <?xml version=1.0?>
    <msix version="1.2" timestamp="1997-07-01T15:27:04Z" uid="[uid12]">
        <beginsession commit="n">
            <dn>server.net/FaxBroadcast/Fax</dn>
            <uid>[uid13]</uid>
            <parentid>[uid11]</parentid>
            <property>
                <dn>DialedNumber</dn>
                <value>12815145802</value>
            </property>
```

```
            <property>
                <dn>Duration</dn>
                <value>229</value>
            </property>
            <property>
```

```
            <dn>StartTime</dn>
            <value>1997-07-01T15:23:57Z</value>
        </property>
        <property>
            <dn>BitRate</dn>
            <value>9600</value>
        </property>
    </beginsession>
</msix>
```

Server -> Client

Server responds with a beginsessionrs message.

```
HTTP/1.0 200 OK
Content-Type: text/plain
Content-Length: [FIXME]

<?xml version=1.0?>
<msix version="1.2" timestamp="1997-07-01T15:27:05Z" uid="[uid12]">
    <beginsessionrs>
        <status>
            <code>msix.org/200</code>
        </status>
        <uid>[uid13]</uid>
    </beginsessionrs>
</msix>
```

Client -> Server

Client commits parent session.

```
POST cgi/msix HTTP/1.0
Content-Type: text/plain
Content-Length: [FIXME]

<?xml version=1.0?>
<msix version="1.2" timestamp="1997-07-01T15:27:06Z" uid="[uid14]">
    <commitsession>
        <uid>[uid10]</uid>
    </commitsession>
</msix>
```

Server -> Client

Server responds with commitsessionrs elements.

```
HTTP/1.0 200 OK
Content-Type: text/plain
Content-Length: [FIXME]
```

```
<?xml version=1.0?>
<msix version="1.2" timestamp="1997-07-01T15:27:05Z" uid="[uid14]">
    <commitsessionrs>
```

```
            <status>
                <code>msix.org/200</code>
            </status>
            <uid>[uid10]</uid>
        </commitsessionrs>
    </msix>
```

C.5 Session Update and Abort

A client submits a simple session, makes an update, and then aborts.

Client -> Server

```
POST cgi/msix HTTP/1.0
Content-Type: text/plain
Content-Length: [FIXME]

<?xml version=1.0?>
<msix version="1.2" timestamp="1997-07-01T15:25:03Z" uid="[uid15]">
    <beginsession>
        <uid>[uid16]</uid>
        <dn>server.net/FoneCall</dn>
        <property>
            <dn>AccountId</dn>
            <value>324955</value>
        </property>
        <property>
            <dn>DialedNumber</dn>
            <value>+16177205200</value>
        </property>
        <property>
            <dn>Duration</dn>
            <value>723</value>
        </property>
        <property>
            <dn>StartTime</dn>
            <value>1997-06-06T11:32:15Z</value>
        </property>
    </beginsession>
</msix>
```

Server -> Client

Server responds with a beginsessionrs message that includes the uid of the session.

```
HTTP/1.0 200 OK
Content-Type: text/plain
Content-Length: [FIXME]
```

```
   <?xml version=1.0?>
   <msix version="1.2" timestamp="1997-07-01T15:25:04Z" uid="[uid15]">
       <beginsessionrs>
```

```
            <status>
                <code>msix.org/200</code>
            </status>
            <uid>[uid16]</uid>
        </beginsessionrs>
    </msix>
```

Client updates the session's Duration property.

Client -> Server

```
    POST cgi/msix HTTP/1.0
    Content-Type: text/plain
    Content-Length: [FIXME]

    <?xml version=1.0?>
    <msix version="1.2" timestamp="1997-07-01T15:25:05Z" uid="[uid17]">
        <updatesession>
            <uid>[uid16]</uid>
            <property>
                <dn>Duration</dn>
                <value>850</value>
            </property>
        </updatesession>
    </msix>
```

Server -> Client

Server responds with an updatesessionrs message.

```
    HTTP/1.0 200 OK
    Content-Type: text/plain
    Content-Length: [FIXME]

    <?xml version=1.0?>
    <msix version="1.2" timestamp="1997-07-01T15:25:06Z" uid="[uid17]">
        <updatesessionrs>
            <status>
                <code>msix.org/200</code>
            </status>
            <uid>[uid16]</uid>
        </updatesessionrs>
    </msix>
```

Client aborts session

Client -> Server

```
    POST cgi/msix HTTP/1.0
    Content-Type: text/plain
```

```
   Content-Length: [FIXME]

   <?xml version=1.0?>
   <msix version="1.2" timestamp="1997-07-01T15:25:07Z" uid="[uid18]">
```

```
       <abortsession>
           <uid>[uid16]</uid>
       </abortsession>
   </msix>
```

Server -> Client

Server responds with an abortsessionrs message.

```
   HTTP/1.0 200 OK
   Content-Type: text/plain
   Content-Length: [FIXME]

   <?xml version=1.0?>
   <msix version="1.2" timestamp="1997-07-01T15:25:08Z" uid="[uid18]">
       <abortsessionrs>
           <status>
               <code>msix.org/200</code>
           </status>
           <uid>[uid16]</uid>
       </abortsessionrs>
   </msix>
```