Appendix – The Event Heap Wire Protocol

By Bradley Earl Johanson

This is a supplemental electronic appendix that is a companion to "Application Coordination Infrastructure for Ubiquitous Computing Rooms," a dissertation by the author which was written in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Electrical Engineering at Stanford University.

Preface

This appendix contains the Event Heap wire protocol version 1 as implemented by Event Heap versions 1.96 and higher. The protocol specification was compiled with assistance from Satyajeet Salgar. The most recent version of the protocol is always available as part of the iROS source tree on http://iros.sourceforge.net.

Overview

The protocol consists of two parts:

- How to send requests, tuples and associated other information to the server, and receive the replies back
- What are the valid requests to make to the server, and the response that can be expected

The first part is the wire bundle protocol, and is found in the first part of this protocol specification. The second part is the Event Heap server invocation specification, and can be found toward the end of this appendix. A proper Event Heap client needs to understand the wire bundle protocol, and speak a sub-set of the supported server invocation op-codes.

This specification is primarily intended to enable programmers to write their own Event Heap client in some language other than the original Java, but should contain enough information to write a new server as well.

The Wire Bundle Protocol

In order to access an Event Heap server, you need to speak to it over a standard socket using TCP/IP to the port on which the server is listening. After an initial handshake to agree on the protocol version, the underlying Event Heap wire protocol is a unidirectional bit-stream

transmitted using TCP/IP over a socket connection to some port. A stream of wire bundles is sent to the server from the client which specify requests, and a stream of response wire bundles from the server is sent back over the same socket. The semantics of the bundles are discussed in the Event Heap server invocation specification toward the end of the appendix. In this specification only sending of a valid stream is discussed, but receiving may be inferred since it is just the inverse operations.

Initial Hand-Shake and High-Level Protocol

Here is the high-level protocol, details of sending sub-objects follows:

1. Once the connection is established, send the version (as a UTF-8 String) of the wire protocol running on this entity (e.g. "EHWPv1" for this version of the wire protocol)
2. Read a UTF-8 String over the wire that is the protocol version running on the peer.
    - If the versions are different ---Exit
    - else ---continue
3. Send (or receive) WireBundles

Semantics for WireBundle Transfer

A WireBundle is a simple helper class which bundles up a set of tags, sequencing information and a tuple array for transmission or receipt by an entity using the Event Heap.

1. Construct and send a mask (one byte) that indicates if there are non-null values for each of the fields in WireBundle
    - 1s place set indicates a non-null destinationTag (i.e. the remote entity that should process this bundle)
    - 2s place set indicates a non-null returnTag (i.e. the entity that should be sent the results of the processing of this bundle)
    - 4s place set indicates a non-null sequenceInfo array
    - 8s place set indicates a non-null outTuples array
2. If the 1's place bit is set in the mask send (as a UTF-8 String) the destination Tag.

3. If the 2's place bit is set in the mask send (as a UTF-8 String) the return Tag.
4. If the 4's place bit is set in the mask send SequenceInfo object by:
    - Write to the socket the number of elements (as an Int) in the sequenceInfo array
    - Write each sequenceInfo object
5. If the 8's place bit is set in the mask send Tuple objects by:
    - Writing to the socket the number of elements (as an Int) in the outgoing Tuples array
    - Write each Tuple object

Semantics for SequenceInfo Transfer

This object is used to store sequencing information for one EventType. It contains an array with one SourceInfo object per known source that emits events with this EventType.

1. Write the EventType (as a UTF-8 String).
2. Write the number of SourceInfo objects that will be transferred (as an Int).
3. Write each SourceInfo object to the socket.

Semantics for SourceInfo Transfer

This class is used to indicate which sources and sessions are known to be generating specific event types by a given EventHeap client object.

1. Write the source (as a UTF-8 String).
2. Write the session ID (as an Int).
3. Write the maximum seen sequence number for the given source, in the given session (as an Int).

Semantics for Tuple Transfer

A tuple is the main unit of information transfer for the Event Heap. Conceptually each tuple is an unordered collection of fields, where each field is characterized primarily by its type, name, and post value and template value. Each tuple is transferred as a serialized byte array-- this allows each tuple to be read off the socket in one chunk, deferring parsing of tuple content until later.

Sending Serialized Tuple

1. Write the length of the serialized tuple (as an Int)
2. Write the serialized byte array

Serializing Tuple to a Byte Array

1. Write the tuple serial number (as an Int). Serial numbers are uniquely assigned to each new tuple written to an Event Heap server. The serial number zero indicates the tuple has never been on an Event Heap server.
2. Write the number of fields (as an Int)
3. Write each Field to the byte array

Note: Fields that have both post-values and template-values that are 'VIRTUAL' are known as pure virtual fields (i.e. the field is ignored in matching both when the tuple is posted and when it is used as a template). In this case, as an optimization, the field need not be transmitted, although it is not a violation of semantics if it is sent.

Semantics for Field Transfer

Fields hold the main content of tuples, which are a collection of fields. Like tuples, they are transferred as a serialized byte array allowing deferment of content parsing. This is useful since the server can avoid parsing fields that it never needs to access (namely those which never get used in any matching). Since the field name is a key value, it is transmitted before the serialized byte array. Full semantics of the various sub-fields of a field are discussed elsewhere (including in the Javadoc API documentation for the Java reference implementation).

Sending Serialized Field and Field Name

1. Write the name of the Field (as a UTF-8 String).
2. Write the length of the coming byte array (as an Int).

3. Write the serialized field as a byte array.

Serializing Field to a Byte Array

1. Write the Field type (as a UTF-8 String). This is used to indicate the data type of this field. Valid possibilities are:
    - "boolean", "int", "long", "float", "double", "string":
        - These are the standard data types that all implementations will be expected to understand.
    - "EHJava.{Fully qualified Java class name}":
        - "EHJava." concatenated to the beginning of a valid Java class name indicates the field values will be Java objects of the given class
    - "[other platform qualifier].{fully qualified class name}":
        - "[other platform qualifier]." concatenated to the beginning of a class identifier for that platform indicates that the field values will be objects of the given type in that language (e.g. "c_obj.transform_matrix"). Currently no other platforms besides Java are defined, but compliant implementations must be able to detect fields containing such data and move them around with the tuple undisturbed.
2. Write the post value of the field (see Semantics for Value Transfer)
3. Write the template value of the field (see Semantics for Value Transfer)

Semantics for Value Transfer

A value contains the actual data for a field. Values are used for both the 'post value' and 'template value' sub-fields. Again, full details of the semantics of possible values are discussed elsewhere, including in the Javadoc API documentation from the Java reference implementation.

1. Write a byte value which is one of the following:
    - ACTUAL=0: The field contains an actual object value which will follow.
    - FORMAL=1: The value is formal and matches any field with the same name and type regardless of value.

- VIRTUAL >= 2: The value is virtual, and the field containing it is never sent when it is being used (i.e. if a tuple is posted and one of its fields has a post value that is VIRTUAL, that field will not be sent). The only time VIRTUAL will be seen is in the non-used field value when the used value is non-virtual (i.e. if the template value is formal, a post value of virtual may be sent along with it). All values greater than 2 are considered virtual, but individual clients may use different values in this range to indicate subtle differences in types of virtual values (e.g. the Java version uses 3 to indicate the field value is automatically set, and 4 to indicate that it is automatically set but may be overridden by the application using the library).

2. Write the value if the value is ACTUAL:
    a. Write a boolean which is 'true' if the value to come is NULL (in which case no further bits will follow), or false otherwise.
    b. If the value is non-NULL
        i. If field type is one of the fundamental types, a value in the format for that type is written (see boolean , int , long , float , double , string)
        ii. If the field type is a non-fundamental type (platform specific):
            1. Write the length of the coming byte array (as an Int).
            2. Write the serialized value as a byte array.

Note: For non-fundamental types, the format of the serialized value is implementation dependent.

Transfer of Fundamental Types

- Int - four bytes, high byte first.
- Long - eight bytes, high byte first
- Float - the floating-point argument according to the IEEE 754 floating-point "single precision" bit layout, transmitted as four bytes, high byte first.
- Boolean - one byte. true is given by a non-zero value (normally 1), false is given by 0.
- Double - the floating-point value according to the IEEE 754 floating-point "double format" bit layout, transmitted as eight bytes, high byte first.

- Byte - 8-bits
- UTF-8 String:
    1. Write two bytes indicating the length of the coming string (high byte first).
    2. Write each character:
        a. If a character c is in the range \u0001 through \u007f, it is represented by one byte:
            - (byte)c
        b. If a character c is \u0000 or is in the range \u0080 through \u07ff, then it is represented by two bytes, to be written in the order shown:
            - (byte)(0xc0 | (0x1f & (c > > 6)))
            - (byte)(0x80 | (0x3f & c))
        c. If a character c is in the range \u0800 through \uffff, then it is represented by three bytes, to be written in the order shown:
            - (byte)(0xc0 | (0x0f & (c > > 12)))
            - (byte)(0x80 | (0x3f & (c > > 6)))
            - (byte)(0x80 | (0x3f & c))

Event Heap Server Invocation Specification

The Event Heap server invocation specification explains what information is sent in the wire bundles (defined in the Wire Bundle Protocol section) to actually execute commands on a running Event Heap server. Specifically it discusses how the return and destination tags, sequence info and tuple objects are used.

Tuple Matching

As can be inferred from the wire bundle protocol for tuples, a tuple is a set of fields, each of which has a name, a type, a post value and a template value. Tuples, when placed into the Event Heap as events (see Events versus Tuples), have the post values for their fields active. Tuples passed to the Event Heap as template events in a retrieval call have the template values for their fields active.

Field values can be one of several different things:

- **Formal:** This value matches any non-virtual value in a comparison tuple. This is equivalent to saying the field is a wild card field.
- **Virtual:** This field is inert and will be ignored in any matching, as if it doesn't exist. If both post and template values are virtual, Event Heap clients may omit the field in posting it to the Event Heap to save bandwidth. Its use is primarily for specifying optional fields without forcing them to be sent to the Event Heap when they are unused..
- **Value:** An actual value for the field that matches the type of the field.

When a retrieval call is made, the tuples passed with the call are used to match against tuples in the Event Heap, called comparison tuples. A template tuple can match a comparison tuple if and only if every field in the template tuple with a non-virtual template value is present in the comparison tuple. Beyond this, every field in the template tuple that has a non-virtual template value must have its template value match the post value in the equivalent field in the comparison tuple, where equivalency is determined by field name. Fields match if one or more of the following is true:

- The template tuple field's template value is formal.
- The comparison tuple field's post value is formal.
- The template tuple field's template value and the comparison tuple field's post value are both actual, and are equivalent.

Events versus Tuples

In the wire bundle protocol only tuples are referenced. This is because the wire bundle protocol was designed to be suitable for communication of any collection of name, type, value fields. Events are a specific type of tuple with required fields that obey certain semantics. This session details the required fields and the semantics.

The following tables detail the reserved fields:

Name	Type	Post Value	Template Value	Can be Overridden?	Description
EventType	String	INVALID_EVENT	Formal	Must be overridden*	In general, all events must have an event type. All events of the same type should contain the same minimum set of additional fields beyond the standard ones (but may contain additional fields beyond the minimum set).
TimeToLive	Integer	120000 ms	Formal	Post value only.	The amount of time after placement in the Event Heap until the event expires and becomes no longer retrievable.
Source	String	Set to Source of EventHeap object on posting	Formal	Template value only.	Indicates which EventHeap object placed this event into the Event Heap. Only one Source of a given name may be active in an Event Heap at one time.
Target	String	Formal	Set to Source of EventHeap object on template submission.	Yes.	The name of the of the Event Heap source for which this event is intended.
SourceApplication	String	Set to Application of EventHeap object on posting.	Formal	Template value only.	The name of the application which generated this event. Event Heap clients will try to generate this automatically if it is not specified.
TargetApplication	String	Formal	Set to Application of EventHeap object on template submission.	Yes.	The name of the application for which this event is intended.

(Table Continued on next page)

*Both template and post values must be overridden to be the same.

(Continued from previous page)

Name	Type	Post Value	Template Value	Can be Overridden?	Description
SourceDevice	String	Set to Device of EventHeap object on posting.	Formal	Template value only.	The name of the device which generated this event. Event Heap clients will generate this automatically from the DNS name or IP address of the device if it is not specified.
TargetDevice	String	Formal	Set to Device of EventHeap object on template submission.	Yes.	The name of the device for which this event is intended.

Table 1- Required Fields for Event Heap Events

Name	Type	Post Value	Template Value	Can be Overridden?	Description
SourcePerson	String	Virtual, or set to Person of EventHeap object, if non-null, on posting.	Virtual	Template value only.	The person most directly responsible for the generation of this event, if known. This may be specified using the appropriate method call to the EventHeap client object.
TargetPerson	String	Formal	Virtual, or set to Person of EventHeap object, if non-null, on posting.	Yes.	The person for whom this event is intended.
SourceGroup	String	Virtual, or set to Group of EventHeap object, if non-null, on posting.	Virtual	Template value only.	The group which generated this event. This may be specified using the appropriate method call to the EventHeap client object.
TargetGroup	String	Formal	Virtual, or set to Group of EventHeap object, if non-null, on template submission .	Yes.	The group for whom this event is intended.

Table 2- Optional Fields for Event Heap Events

Name	Type	Post Value	Template Value	Can be Overridden?	Description
SessionID	Integer	Set by EventHeap object on posting.	Virtual	No.	Used for event sequencing.
SequenceNum	Integer	Set by EventHeap object on posting.	Virtual	No.	Used for event sequencing.
EventHeapVersion	Integer	Set to Event Heap version of EventHeap object on posting.	Set to Event Heap version of EventHeap object on posting.	No.	Used to insure event compatibility and allow for event versioning.

Table 3- Internal Use Fields for Event Heap Events

Event Heap Client Responsibilities

An Event Heap client has several other responsibilities beyond just packaging up arguments of method calls and forwarding them to the server in wire bundles. Specifically, they are:

- For each EventType received at that client, the client must maintain a list of all sources (based on source field) from which events of the given type have been received. For each of these sources, the SessionID and SequenceNum values from the most recently received event must be stored. Whenever a sequenced event retrieval operation is executed, this sequencing information must be packaged up into the sequence info portion of the wire bundle. The sequence info needs to be sent only for event types represented in the template events being used in the sequenced retrieval call.
- The client should insure that all events which it submits to the Event Heap server, either for placement, or to be used as templates, must include all required fields, and those fields must be set in valid fashion (see Events versus Tuples).
- Each client instantiation within a single running application is strongly recommended to maintain a single connection to the Event Heap server. If multiple client instances are instantiated within the application, they should have unique Source names, but make their calls through the same socket.
- The client must maintain Source, Application, Device, Group and Person values for each client instance. These values should be used to fill in fields that are supposed to be automatically set as specified in 'Events versus Tuples'. Source should be a unique value, and it is suggested that a random integer be attached to either the application name or the user specified source name to insure this is so. Group and Person are optional, and if they are not specified by the application through the appropriate API calls, their associated fields may be omitted at the time the wire bundle is formed and sent.

- When sending an event in a wire bundle, clients may strip out any field that has both post value and template value set to virtual. Such fields are known as pure-virtual.
- Clients should automatically reconnect to the server anytime the connection goes down. This insures that both Event Heap client applications and the Event Heap server itself can be independently restarting without requiring the other to do so.

Event Heap Server Responsibilities

The server is responsible for handling all of the Event Heap server operations specified here by receiving appropriately formatted wire bundles and responding as required. The server must also assign each event placed into the Event Heap with the putEvent call an ID unique to the instantiation history of the server. This must be assigned even if the event being placed was formerly removed from the Event Heap and already has an ID. This ID can then be used by the server to uniquely determine which event to delete in the case of a deleteEvent call.

Event Heap Server Operations

General Usage

The general call response protocol is asynchronous. A client sends some request bundle to the server, and the response from the server will arrive at some later time. The client should in general not block until the response arrives, since especially in the case of blocking retrieval operations, the response may not come until the conditions required to meet the request are satisfied (e.g. a blocking retrieve won't return until the appropriate event to be retrieved arrives at the server). In general this is the case for multi-threaded clients who can handle multiple simultaneous outstanding requests to the server. In the case of a single threaded client, the client will need to block always until the return wire bundle for the request is received. This is not a problem as long as the client insures they only have one request out at a time.

In general, here are how the wire bundle fields are used:

Wire Bundle Field	Usage for Request from Client to Server	Usage in Response from Server to Client
Destination Tag	The server opcode for the desired Event Heap operation (see the rest of this section of the specification).	The return tag field from the request that caused this response to be generated. Note that this means that clients can use whatever routing scheme they wish by setting their return tag in the original request. When they receive the response they can look at the destination tag they themselves generated and use that information to get the actual request contents to the appropriate application thread.
Return Tag	The tag for the server to use in the destination tag field when responding to this request.	Empty, or may contain back channel information for some calls.
Sequence Information	Used only for sequenced retrieval operations. Contains information on what events have already been seen by the client that is used by the server to determine which events are valid to return to the client. See Event Heap Client Responsibilities for more information.	Empty
Tuple Objects	The set of events which are parameters to be used by the Event Heap server in executing the request. Note that while the wire bundle allows tuples here, events must be used when communicating with an Event Heap server. See Events versus Tuples for more information.	One of two things: A set of one or more events which satisfy a retrieval request operation to which this wire bundle is the response. Null if the retrieval request operation that caused this response found no matching events currently in the Event Heap, or the request operation that generated this response gets an ACK only.

Table 4- Meanings of Wire Bundle Fields

Single Returned Event

These operations are all retrieval calls that result in a wire bundle being returned with a single event that matches one or more of the events passed as template events in the request wire bundle.

getEvent

Operation Meaning:	Get an event currently stored on the server that satisfies template value fields of one or more of the events included in the request. Retrieved events will satisfy the sequencing constraints derived from the sequence information included in the wire bundle. The retrieved event will remain in the Event Heap for others to retrieve.
Expected Response from Server:	A wire bundle with exactly one event that matches one or more of the events passed as templates subject to the constraints of the passed sequencing information. Zero events will be returned if no matching event is in the Event Heap at the time the request is received.
Destination Tag:	getEvent
Sequence Info Needed?	Yes (see Event Heap Client Responsibilities)
Meaning of included Events:	The template value fields in each included event will be used to try and find a match among existing events (see Tuple Matching).

Table 5- getEvent Operation Description

removeEvent

Operation Meaning:	Remove an event currently stored on the server that satisfies template value fields of one or more of the events included in the request. Retrieved events will satisfy the sequencing constraints derived from the sequence information included in the wire bundle. The retrieved event will be removed from the Event Heap, so no others may retrieve it after this call returns.
Expected Response from Server:	A wire bundle with exactly one event that matches one or more of the events passed as templates subject to the constraints of the passed sequencing information. Zero tuples will be returned if no matching event is in the Event Heap at the time the request is received.
Destination Tag:	removeEvent
Sequence Info Needed?	Yes (see Event Heap Client Responsibilities)
Meaning of included Events:	The template value fields in each included event will be used to try and find a match among existing events (see Tuple Matching).

Table 6- removeEvent Operation Description

waitToRemoveEvent

Operation Meaning:	Remove an event currently stored on the server that satisfies template value fields of one or more of the events included in the request. Retrieved events will satisfy the sequencing constraints derived from the sequence information included in the wire bundle. The retrieved event will be removed from the Event Heap, so no others may retrieve it after this call returns. In the case when no event is found, the incoming event which eventually satisfies the request will not be placed into the Event Heap.
Expected Response from Server:	A wire bundle with exactly one event that matches one or more of the events passed as templates subject to the constraints of the passed sequencing information. If no event is found in the Event Heap at request submission time, the server will monitor events coming into the Event Heap, and return the response wire bundle when the first incoming event matching one or more of the template events in the request is found.
Destination Tag:	waitToRemoveEvent
Sequence Info Needed?	Yes (see Event Heap Client Responsibilities)
Meaning of included Events:	The template values of the fields in each included event will be used to try and find a match among existing events (see Tuple Matching). If no matching event is found at request time, all incoming events will be checked against the template values of the fields for each event, and after the first matching incoming event is found, it will be returned to the requester, having not been placed into the Event Heap.

Table 7- waitToRemoveEvent Operation Description

waitForEvent

Operation Meaning:	Get an event currently stored on the server that satisfies template value fields of one or more of the events included in the request. Retrieved events will satisfy the sequencing constraints derived from the sequence information included in the wire bundle. The retrieved event will remain in the Event Heap for others to retrieve, or in the case that no event is initially found, the incoming event which eventually satisfies the request will be both returned to the requester and placed into the Event Heap.
Expected Response from Server:	A wire bundle with exactly one event that matches one or more of the events passed as templates subject to the constraints of the passed sequencing information. If no event is found in the Event Heap at request submission time, the server will monitor events coming into the Event Heap, and return the response wire bundle when the first incoming event matching one or more of the template events in the request is found.
Destination Tag:	waitForEvent
Sequence Info Needed?	Yes (see Event Heap Client Responsibilities)
Meaning of included Events:	The template values of the fields in each included event will be used to try and find a match among existing events (see Tuple Matching). If no matching event is found at request time, all incoming events will be checked against the template values of the fields for each event, and after the first matching incoming event is found, it will be returned to the requester before being placed into the Event Heap.

Table 8- waitForEvent Operation Description

Multiple Returned Events

These operations return an array of events in the return wire bundle.

snoopEvents

Operation Meaning:	Retrieves all events currently stored on the server that satisfy the template value fields of one or more of the events included in the request. Sequencing is ignored, and copies of the retrieved events remain on the server.
Expected Response from Server:	A wire bundle with all events that match one or more of the events passed as templates. If no events are found, zero events will be returned
Destination Tag:	snoopEvents
Sequence Info Needed?	No
Meaning of included Events:	The template values of the fields in each included event will be used to try and find matches among existing events (see Tuple Matching).

Table 9- snoopEvents Operation Description

getAll

Operation Meaning:	Returns copies of all events currently in the Event Heap. Sequencing is ignored.
Expected Response from Server:	A wire bundle with copies of all events currently in the Event Heap, regardless of sequencing constraints.
Destination Tag:	getAll
Sequence Info Needed?	No
Meaning of included Events:	No events need be included, and any events sent are ignored.

Table 10- getAll Operation Description

Notification Streams

These operations register with the server to receive a stream of events. Instead of returning a single wire bundle, the server will continue to return wire bundles, each with a matching event, until a deregister operation (documented in the Void Calls section) is sent to the server with a Return Tag that is the same as the one used for the initial registration call. All returned events will use the same Return Tag. It is the responsibility of the client to insure that Return Tag is unique for that client.

registerForEvents

Operation Meaning:	Causes each event put on the server, from the point of wire bundle receipt onward, that matches the template value fields of one or more of the events included in the request to be returned in a wire bundle to the client making the call. Sequencing is ignored, and copies of the retrieved events remain on the server.
Expected Response from Server:	A stream of wire bundles each with an event that matched one or more of the events passed as templates. These bundles will continue to be sent by the server until deregister is called.
Destination Tag:	registerForEvents
Sequence Info Needed?	No
Meaning of included Events:	The template values of the fields in each included event will be used to try and find matches among existing events (see Tuple Matching).

Table 11- registerForEvents Operation Description

registerForAll

This call is intended primarily for logging and debugging applications.

Operation Meaning:	Causes each event put on the server, from the point of wire bundle receipt onward, to be returned in a wire bundle to the client making the call. Sequencing is ignored, and copies of the retrieved events remain on the server.
Expected Response from Server:	A stream of wire bundles each with a new event that was put onto the server. These bundles will continue to be sent by the server until deregister is called.
Destination Tag:	registerForAll
Sequence Info Needed?	No
Meaning of included Events:	No events need be included, and any events sent are ignored.

Table 12- registerForAll Operation Description

ACK Only

This call causes an ACK (a returned wire bundle containing no events) from the server, but no returned events.

putEvent

Operation Meaning:	Places the included event into the Event Heap.
Expected Response from Server:	A wire bundle with destination tag set to the return tag in the putEvent request wire bundle. All other fields will be empty. This bundle serves as an ACK from the server that the put was successful.
Destination Tag:	putEvent
Sequence Info Needed?	No
Meaning of included Events:	A single event to be added to the Event Heap. The post values for the fields in the event are used to match against the template values of fields in template events from future retrieval calls. The template values are ignored (although they may be used by a client application that retrieves this event).

Table 13- putEvent Operation Description

deleteEvent

Operation Meaning:	Deletes the specified event from the Event Heap, after which it will no longer be accessible by other Event Heap clients.
Expected Response from Server:	Nothing.
Destination Tag:	deleteEvent
Sequence Info Needed?	No
Meaning of included Events:	Must be a single event previously returned by the same Event Heap server. An internally set ID is used by the server to determine which event to delete.

Table 14- deleteEvent Operation Description

clear

Operation Meaning:	Deletes all events currently in the Event Heap, after which it will be empty.
Expected Response from Server:	Nothing.
Destination Tag:	clear
Sequence Info Needed?	No
Meaning of included Events:	No events need be included, and any events sent are ignored.

Table 15- clear Operation Description

deregister

Operation Meaning:	Stops future wire bundles form being sent for a Notification Stream.
Expected Response from Server:	Nothing.
Destination Tag:	deregister
Sequence Info Needed?	No
Meaning of included Events:	A single event should be included with type "EHS_DeregisterEvent" and a string type field named "StreamID" whose post value is set to the stream ID of the notification stream to be deregistered.

Table 16- deregister Operation Description

Unknown Operations

To allow for expansion of the protocol without breaking old servers, servers must gracefully handle unknown destination tags. If the return tag is set for a wire bundle request with unknown opcode in the destination field, an ACK with return tag set to 'UNKNOWN' is sent. If there is no return tag in the request, no response is given.

Expected Response from Server:	If return tag in the request is non-null, a simple ACK wire bundle is sent with destination tag taken from return tag and return tag set to 'UNKNOWN'. If return tag in the request is null, no response is sent.
Destination Tag:	Any not specified as a valid destination tag elsewhere in this document.