EMS Specification
COMPONENTS OF THE SAVANT

DATA REQUEST

PML RESPONSE

SOAP INTERFACE

STORE LEVEL SAVANT

Task Management System (TMS)

Real-time In-memory Database (RIED)

Relational Database

Event Management System (EMS)

Device

Device

Device
EVENT MANAGEMENT SYSTEM (EMS)

- Event Management System captures, filters, broadcasts & logs EPC data
- Event Management System is implemented on Edge Savants (ES) since the EPC data enters the Savant network only through the Edge Nodes
**Requirements of EMS**

- EMS should be a high-performance system
- Support multiple EPC readers or devices that communicate using different Auto-ID center protocols
- Support event filters with one input stream and multiple output streams for:
  - **Smoothing, Coordination, Forwarding**
- Support for multiple loggers
  - Database, Memory model, remote server (HTTP, SOAP, JMS)
- Ability to handle spikes in the event volume
**Components of EMS**

- **Adapter**
  Communicates with the readers to capture EPC events

- **Event Queues**
  Asynchronous/Synchronous queuing system that captures EPC data from various reader adapters and broadcasts the data to multiple event loggers

- **Event Filters**
  Gets data from one or more input event streams and posts the data to multiple output streams, after filtering the data.

- **Event Loggers**
  Logs data to a database, memory model or a remote server (HTTP, SOAP, JMS)
EMS Adapter

- EMSAdapter is a producer of events
- Implements the interface `org.autoidcenter.ems.EMSAdapter`
  - EMSAdapter just implements a shutdown() method
- Implements a constructor taking a `String` argument, followed by an `EMSHandle` argument. The first argument is the initialization string, and will be supplied by the EMS Manager. The second argument is the object to which events have to be logged.
EMS ADAPTER LIFE CYCLE

Event Manager creates an instance of the EMS Adapter and handles events.

1. create() method is called.
2. Creates an instance of the EMS Adapter with an initialization string and handle.
3. Logs an event.
4. Returns control.
5. Shuts down the adapter.
6. Returns control to Event Manager.

Event Manager removes the reference to the adapter. The JVM garbage collects the logger later.
EMS Logger

- EMSLogger is a consumer of events

- Implements the interface `org.autoidcenter.ems.EMSReader`

- Implements a constructor taking a `String` argument. This argument, referred to as the initialization string will be supplied by the Event Manager.
EMS Logger Life Cycle
EMS Filters

- EMSFilters is both a producer and consumer of events

- Implements the interface
  `org.autoidcenter.ems.EMSFilter` which extends `EMSReader`

- Implements a constructor taking a String argument. This argument is the initialization string, and will be supplied by the Event Manager.

- A filter can receive events from multiple event producers and can log events in multiple `EMSHandle` objects. The output handles are configured in the filter using the `setListeners` method, which accepts an array of `EMSHandle` objects wrapping `EMS units`
EMS Filter Life Cycle
EMS Unit Interfaces

- EMSAdapter
- EMSQueue
- EMSHandle
- Service Interface
- SourcedEMSHandle
- EMSFilter
- EMSReader
Queues

- Event producers and event consumers can register with event queues
- An event queue broadcasts every event it receives from a producer to all its consumers.
- Custom queues can be written by implementing `org.autoidcenter.ems.EMSQueue`
COMMON TYPES OF QUEUES

- **Synchronous queues.** Synchronous event queues are non-threaded queue implementations. When a producer logs an event to a synchronous event queue, it iterates over each consumer and logs the event to that consumer.

- **Asynchronous queues.** Asynchronous event queues are threaded queue implementations. When a producer logs an event to an asynchronous event queue, it stores the event in a buffer. A worker thread is associated with each consumer. This worker thread submits unprocessed events from the queue to the consumer.

- **Categorized queues.** Categorized event queues are non-threaded queue implementations that allow producers and consumers to register for events belonging to a certain *event category.*
Asynchronous queues are typically used whenever there is a need for multi-processing among consumers and its producers, or to prevent spikes in the arrival of events from affecting performance.
A categorized queue is a synchronous implementation of EMSQueue.

This queue implementation provides a generalized event type-based routing of events from producers to consumers. Each consumer registers with the queue using a specific category. Only events matching that category will be sent to that consumer.
# Event Categories

<table>
<thead>
<tr>
<th>Event prefix</th>
<th>Event type</th>
<th>Successful match?</th>
</tr>
</thead>
<tbody>
<tr>
<td>event_queue</td>
<td>event_queue</td>
<td>Yes</td>
</tr>
<tr>
<td>event_queue</td>
<td>event_queue.buffer_overflow</td>
<td>Yes</td>
</tr>
<tr>
<td>event_queue</td>
<td>event_queue.buffer_overflow.foo</td>
<td>Yes</td>
</tr>
<tr>
<td>event_queue</td>
<td>buffer_overflow</td>
<td>No</td>
</tr>
<tr>
<td>status.reader.1234</td>
<td>status.reader.1234.online</td>
<td>Yes</td>
</tr>
<tr>
<td>status.reader.1234</td>
<td>status.reader.1235.offline</td>
<td>No</td>
</tr>
</tbody>
</table>
**Event Manager**

- The Event Manager is responsible for managing the Event Processing Network
- The Event Manager is defined by the interface

```java
org.autoidcenter.ems.EventManagerProvider
```

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>deregisterPublicListener</td>
<td><code>listenerName : String</code> : void</td>
</tr>
<tr>
<td>getALIMSPQueueNames</td>
<td><code>[] : String</code></td>
</tr>
<tr>
<td>getDefaultQueueMapByType</td>
<td><code>queueType : int</code> : EMSQueue</td>
</tr>
<tr>
<td>getEMSPQueueByName</td>
<td><code>EMSSequence : int</code> : EMSQueue</td>
</tr>
<tr>
<td>registerPublicListener</td>
<td><code>listenerName : String, queueName : String, loggerClassName : String, loggerArgs : String</code> : void</td>
</tr>
<tr>
<td>registerPublicListener</td>
<td><code>String, queueName : String, loggerClassName : String, loggerArgs : String</code> : void</td>
</tr>
<tr>
<td>shutdown</td>
<td>: void</td>
</tr>
<tr>
<td>startup</td>
<td>: void</td>
</tr>
</tbody>
</table>
Configuration

- EMS configuration
Sample EMS

- Reader Adapter
  - Event Queue
    - Event Logger
      - To Remote Machine
    - Event Filter
      - Event Queue
        - Event Logger
          - Database
        - Memory Event Logger
      - Memory Event Logger
        - RIED Memory Model
EMS SOAP Interface

- public static String startup()
- public static void shutdown()
- public static Vector listPublicListeners()
- public static String addPublicListener(String listenerName, String queueName, String loggerClass, String loggerArgs)
- public static String addPublicListener(String listenerName, String queueName, String loggerClass, String loggerArgs, String filterClass, String filterArgs)
- public static void removePublicListener(String listenerName)
Spec Roadmap

• By March 13th SAG we will be releasing
  Spec for TMS
  Spec for REID
  Savant V2 with the new EMS components
• Prof Sanjay Sarma (sesarma@mit.edu)
• Laxmiprasad Putta (prasad@oatsystems.com, 617-335-0693)
• Sridhar Ramachandran (sridhar@oatsystems.com)