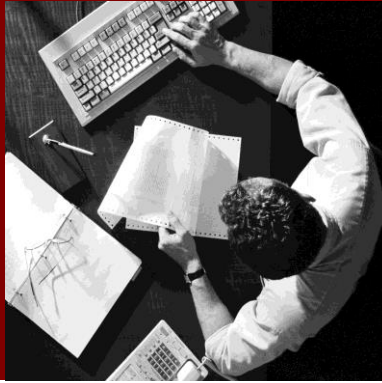




SAP Manufacturing Execution 15.2

Notification Messages

Document Version 2.0 – November 2018



SAP SE

Dietmar-Hopp-Allee 16
69190 Walldorf
Germany
T +49/18 05/34 34 34
F +49/18 05/34 34 20
www.sap.com

© 2018 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies. Please see [http://www.sap.com/corporate-](http://www.sap.com/corporate-en/legal/copyright/index.epx#trademark)

[en/legal/copyright/index.epx#trademark](http://www.sap.com/corporate-en/legal/copyright/index.epx#trademark)  for additional trademark information and notices.

Documentation in the SAP Service Marketplace

You can find this documentation at the following Internet address: <http://service.sap.com/instguides>

Typographic Conventions

Type Style	Represents
<i>Example Text</i>	Words or characters that appear on the screen. These include field names, screen titles, pushbuttons as well as menu names, paths and options. Cross-references to other documentation
Example text	Emphasized words or phrases in body text, titles of graphics and tables
EXAMPLE TEXT	Names of elements in the system. These include report names, program names, transaction codes, table names, and individual key words of a programming language, when surrounded by body text, for example, SELECT and INCLUDE.
Example text	Screen output. This includes file and directory names and their paths, messages, names of variables and parameters, source code as well as names of installation, upgrade and database tools.
Example text	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
<Example text>	Variable user entry. Pointed brackets indicate that you replace these words and characters with appropriate entries.
EXAMPLE TEXT	Keys on the keyboard, for example, function keys (such as F2) or the ENTER key.

Icons

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax

History of Changes

The following table provides an overview of the most important changes that were made in the latest versions.

Version	Important Changes
1.0 (October 26, 2015)	First version
2.0 (November 19, 2018)	The following sections were added: <i>Using the Notification Framework for Custom Development</i> and <i>POD Supported Notification Messages</i> .

Table of Contents

History of Changes	4
Table of Contents.....	5
Introduction	6
References	6
Notification Message Framework	7
Server-Sent Events (SSE).....	7
MessagingIO.js.....	7
Notification Message	8
Notification Message Consumer	8
Notification Message Queue	8
Notification Message Servlet	8
Notification Message Service	9
SAP ME POD Implementation	10
Notification Message Manager	10
Adding Plug-In as Consumer.....	10
POD Connection Notification Messages	15
Using the Notification Framework for Custom Development	17
POD Supported Notification Messages	18

Introduction

This design document provides an overview of the implementation of Server-Sent Event technology for pushing Notification Messages from the server to client browsers.

The target audience for this document is technical designers, developers, architects and system integrators that will be producing and consuming Notification Messages.

References

Document Title	Document URL
What is Server-Sent Events?	Server-Sent Events
What is EventSource class?	EventSource API

Notification Message Framework

Notification Messages is a framework developed to support real-time event messages from the server to a client browser. Utilizing Server-Sent Event (SSE) technology and classes developed in SAP ME, the framework can be implemented for any browser-based technology using HTML and JavaScript. The initial implementation of the Notification Message framework is utilized by the POD, however it could just as easily be applied to Activities built with Coral or UI5.

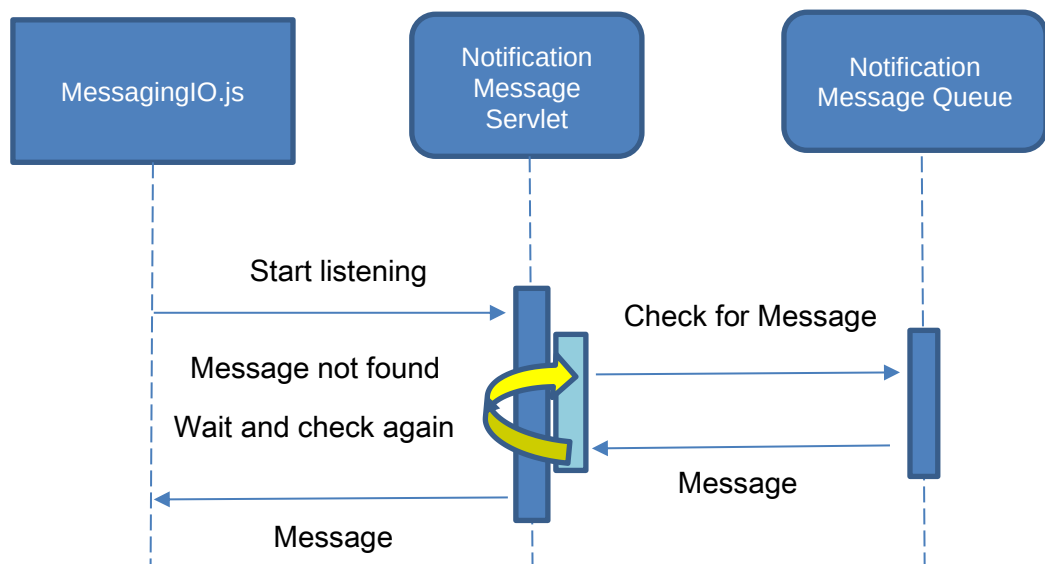
Server-Sent Events (SSE)

Server-Sent Events (SSE) technology is supported by most modern browsers except Internet Explorer (IE) through the `EventSource` JavaScript class. SAP ME implementation of Server-Sent Events does support IE using an internally developed version of the `EventSource` class.

Server-Sent Events are real-time events sent from the server to a browser. These are similar to WebSockets except they are much easier to implement using standard http and only support one way communication from the server to the browser. WebSockets are bidirectional and require an installed infrastructure on the server and client to work. Events sent from the server basically consist of a name and data associated with the event.

MessagingIO.js

SAP ME implementation of Server-Sent Events is through the `MessagingIO` JavaScript class. This class is responsible for opening an HTTP connection with the server and waiting for a response in the form of a Notification Message event. A `MessagingIO` JavaScript function opens a connection to a servlet which in turn listens for Notification Messages. When a message is received, the servlet sends the message back to `MessagingIO`. `MessagingIO` checks the message type and if it is a Notification Message, it fires the message to a defined listener. `MessagingIO` then posts a request back to the servlet to restart the listening process.



Notification Message

Notification Messages are identified by a “topic” name and can contain an optional message and a list of attributes (name/value pairs). A Notification Message is added to the Notification Message Queue using the Notification Message Service API

Module: com.sap.me.messaging

Service name: NotificationMessageService service

Service method: addNotificationMessage

PAPI web service request for method **addNotificationMessage**:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:mep="mepapi:com:sap:me:messaging" xmlns:mes="http://www.sap.com/me/messaging">
  <soapenv:Header/>
  <soapenv:Body>
    <mep:addNotificationMessage>
      <mep:Site>SITE</mep:Site>
      <mep:Request>
        <mes:topicName>POD_INPUT_MESSAGE@TEST</mes:topicName>
        <mes:comments>Update POD Plugin</mes:comments>
        <mes:messageAttributeList>
          <attribute>INPUT</attribute>
          <value>SFC1</value>
        </mes:messageAttributeList>
        <mes:messageAttributeList>
          <attribute>PARAM</attribute>
          <value>Param1</value>
        </mes:messageAttributeList>
      </mep:Request>
    </mep:addNotificationMessage>
  </soapenv:Body>
</soapenv:Envelope>
```

Notification Message Consumer

A Notification Message Consumer is identified by the “topic” name of the Notification Message it is listening for and a unique client ID generated when the client opens a connection to the servlet. This allows multiple “consumers” (that is, clients) to be listening for a single message identified by the topic name.

Notification Message Queue

The Notification Message Queue is used to hold Notification Messages temporarily until all Notification Consumers currently listening have been sent the message. Notification Messages are added to the Notification Message Queue using the Notification Message Service API.

Module: com.sap.me.messaging

Service name: NotificationMessageService service

Service method: addNotificationMessage

Notification Message Servlet

The Notification Message Servlet polls the Notification Message Queue for messages assigned to a specific client consumer based on a unique client ID generated when the client opens a

connection to the servlet. When a request is made to listen for messages, the servlet enters into a long polling process. It first looks for all consumers that have the same “client” identifier as was passed in the servlet request. For each consumer found the servlet will retrieve all Notification Messages that have a matching Notification Topic from the queue. Each message found is then converted to a JSON object and added to a JSON array. Once all messages have been processed for this request, the resulting JSON array is passed back to `MessagingIO` in the servlet response and forwarded to a defined listener(s). When all possible active consumers have obtained the Notification Message, the message is removed from the message queue.

Notification Message Service

The `NotificationMessageService` is the public API (PAPI) used to add Notification Messages and Consumers to the Notification Message Queue. Additional methods are provided for finding and removing messages and consumers from the queue.

Server-side code uses the following method in the `NotificationMessageService` PAPI to send a message to “Consumers” that are defined as listeners for messages with a specific “topic”:

```
/**
 * Adds a Notification Message based on the information provided in the
 * {@link com.sap.me.messaging.AddNotificationMessageRequest}.
 *
 * @param request {@link com.sap.me.messaging.AddNotificationMessageRequest}
 * @throws RequiredValueMissingException if required values are missing
 * @throws BusinessException thrown if any other business exception occurs.
 */
public void addNotificationMessage(AddNotificationMessageRequest request)
    throws RequiredValueMissingException, BusinessException;
```

The following Example code shows how to add a message to be sent to “Consumers” of messages with the topic of “`EXAMPLE_MESSAGE_TOPIC`”:

```
import com.sap.me.messaging.NotificationMessageServiceInterface;
import com.sap.me.messaging.AddNotificationMessageRequest;

NotificationMessageServiceInterface service = (NotificationMessageServiceInterface)
    ServiceLocator.getService("com.sap.me.messaging", "NotificationMessageService");

AddNotificationMessageRequest request = new
    AddNotificationMessageRequest("EXAMPLE_MESSAGE_TOPIC");
request.setComments("This is an example message");

service.addNotificationMessage(request);
```

SAP ME POD Implementation

SAP ME POD has implemented Notification Messages to allow plug-ins to listen for messages from the server. This makes it possible to update the POD on the client without the need for polling the server. To accomplish this, the `MessagingIO` JavaScript was implemented into the POD framework and works from a separate form than that of the POD. This allows notification message communication to the server to occur without interrupting user interaction with the POD.

Notification Message Manager

The Notification Message Manager manages the interaction between messages sent to the Notification Message Queue and plug-ins. It provides the ability for plug-ins to assign themselves as “consumers” of Notification Messages. It is also responsible for starting the listening for Notification Messages. When new messages are found on the server that applies to the defined “consumers”, the Notification Message Manager will receive the messages and send them to applicable “consumer” plug-ins such as Notification Message Events. The “consumer” plug-ins must implement the interface “`com.sap.me.wpmf.NotificationMessageListenerInterface`” to receive Notification Message Events.

Adding Plug-In as Consumer

A plug-in is defined as a “consumer” of Notification Messages using methods defined in the “`com.sap.me.wpmf.NotificationMessageManagerHelper`” class:

```
/**
 * Helper method to add a new Consumer of a Notification Message topic event
 * @param topic Topic of Notification Message to listen for
 * @param listener NotificationMessageListenerInterface to add
 */
public static void addConsumer(String topic, NotificationMessageListenerInterface
listener);
```

The plug-in must implement the “`NotificationMessageListenerInterface`” interface and add the following method, which will be called when a message with the defined “*topic*” is received. The “`com.sap.me.wpmf.NotificationMessageEvent`” class contains the message information passed from the server, and it is up to the plug-in to perform any control updates to the UI if necessary.

```
/**
 * Method for processing a Messaging notification event
 * @param event the MessageNotificationEvent to process
 */
public void processMessage(NotificationMessageEvent event);
```

The following plugin example shows how to add (and remove) a plug-in as a consumer of messages with the topic Name of “`POD_TEST_MESSAGE@`” followed by the connection identifier. The connection identifier (`CONNECTION_ID`) value is passed as a URL parameter. The connection identifier is determined by the customer. The connection identifier is appended to the topic name of the Notification Message to create a message that can be sent to a limited number of POD Plugins.

Example of POD URL:

http://<host>:<port>/manufacturing/com/sap/me/wpmf/client/template.jsf?WORKSTATION=OPERATION_DEF&ACTIVITY_ID=DEF_OPER_POD&sap-lsf-PreferredRendering=standards&CONNECTION_ID=TEST

```

package com.sap.me.production.podclient.impl;

import java.util.List;
import java.util.Map;

import com.sap.me.frame.Utills;
import com.sap.me.production.podclient.BasePodPlugin;
import com.sap.me.production.podclient.NotificationMessageEvent;
import com.sap.me.production.podclient.NotificationMessageListenerInterface;
import com.sap.me.production.podclient.NotificationMessageManagerHelper;
import com.sap.me.production.podclient.PodOptionType;
import com.sap.me.wpmf.util.ExceptionHandler;

public class NotificationPluginExample extends BasePodPlugin implements
NotificationMessageListenerInterface {

    private String notificationTopicName = null;

    public NotificationPluginExample() {
        super();
        getPluginEventManager().addPluginListeners(this.getClass());
    }

    @Override
    public void dispose() {
        // remove when POD is being unloaded or session timeout.
        removeConsumer();
    }

    @Override
    public void processPluginLoaded() {

        // if connection id is defined, make a consumer of
        POD_TEST_MESSAGE topic
        String value =
        getPodConfigurationOption(PodOptionType.CONNECTION_ID, "");
        if (value != null && value.length() > 0) {
            addConsumer(value);
        }
    }

    @Override
    public void processMessage(NotificationMessageEvent event) {
        if (!notificationTopicName.equals(event.getTopic())) {
            return;
        }
        try {
            List<Map<String, String>> messageData = event.getMessageData();
            if (messageData == null || messageData.isEmpty()) {
                return;
            }
            for (Map<String, String> map : messageData) {
                // process notification messages here, update UI elements with
                new data content
            }
        }
    }
}

```

```

    }
} catch (Exception ex) {
    ExceptionHandler.handle(ex);
}
}

private void addConsumer(String connectionId) {
    try {

        // build new topic name
        StringBuffer buf = new StringBuffer();
        buf.append("POD_TEST_MESSAGE@");
        buf.append(connectionId);

        // add as consumer of topic for resource
        if (Utils.isBlank(notificationTopicName)) {
            notificationTopicName = buf.toString();

            NotificationMessageManagerHelper.addConsumer(notificationTopicName,
this);
        }

    } catch (Exception ex) {
        ExceptionHandler.handle(ex);
    }
}

private void removeConsumer() {
    if (!Utils.isBlank(notificationTopicName)) {
        try {

            NotificationMessageManagerHelper.removeConsumer(notificationTopicName
, this);
        } catch (Exception ex) {
            ExceptionHandler.handle(ex);
        } finally {
            notificationTopicName = null;
        }
    }
}

public void closePlugin() {
    removeConsumer();
    closeCurrentPlugin(false, false);
}
}

```

Example of a notification message request for a POD Plugin that listens for events with the topicName of "POD_TEST_MESSAGE@TEST".

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:mep="mepapi.com:sap:me:messaging" xmlns:mes="http://www.sap.com/me/messaging">
  <soapenv:Header/>
  <soapenv:Body>
    <mep:addNotificationMessage>
      <mep:Site>KSU</mep:Site>
      <mep:Request>
        <mes:topicName>POD_TEST_MESSAGE@TEST</mes:topicName>

```

```
<mes:comments>Update POD Plugin</mes:comments>
<mes:messageAttributeList>
  <attribute>INPUT</attribute>
  <value>SFC2</value>
</mes:messageAttributeList>
<mes:messageAttributeList>
  <attribute>PARAM</attribute>
  <value>Param1</value>
</mes:messageAttributeList>
</mep:Request>
</mep:addNotificationMessage>
</soapenv:Body>
</soapenv:Envelope>
```


POD Connection Notification Messages

The Notification Messaging framework was designed to send a single message to multiple consumers. In a production environment there are business cases where it is desirable to limit the consumer of the message to a limited number of active PODs. For example, you might have a device or machine identified by a resource that will be sending messages, and it is desirable to have only the PODs of the operators next to the machine receive the message and not all of the PODs currently active in the system. This use case was recently implemented in the core products Operation and Work Center PODs to provide the ability to update the SFC field automatically based on a Notification Message that was sent from a machine. To support this requirement, we have added the ability to define a connection identifier (that is, `CONNECTION_ID`) to the URL that is used to launch the POD. The connection identifier is appended to the topic name of the Notification Message to create a message that can be sent to a limited number of PODs.

For the Operation and Work Center PODs, the *POD Selection* plug-in defines itself as a listener for messages using the topic Name of "POD_INPUT_MESSAGE@" followed by the connection identifier. The connection identifier `CONNECTION_ID` is determined by the customer.

For example, the customer might want to use a combination of the work center and resource for the connection identifier. In this case, the URL defined to start the operators POD might be something like this:

```
http://&host:&port/manufacturing/com/sap/me/wpmf/client/template.jsf?WOKSTATION=WORK_CENTER_TOUCH&ACTIVITY_ID=TOUCH_WC_POD&WORKCENTER=SHEET-METAL&RESOURCE=BENCH&CONNECTION_ID=SHEET-METAL_BENCH
```

The partner/customer defines and sends a Notification Message using the OData service, Web Service or Notification Message PAPI directly using the topic name of "POD_INPUT_MESSAGE@SHEET-METAL_BENCH" and an attribute list with the mandatory name/value pair of "INPUT" / value (for example, "INPUT" / SFC number).

Can use WSDL URL to PAPI web service: `http://<host>:<port>/manufacturing-papiservices/NotificationMessageServiceWSService?wsdl`

Send PAPI request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:mep="mepapi:com:sap:me:messaging" xmlns:mes="http://www.sap.com/me/messaging">
  <soapenv:Header/>
  <soapenv:Body>
    <mep:addNotificationMessage>
      <mep:Site>SITE</mep:Site>
      <mep:Request>

<mes:topicName>POD_INPUT_MESSAGE@SHEETMETAL_BENCH</mes:topicName>
      <mes:messageAttributeList>
        <attribute>INPUT</attribute>
        <value>SFC_NAME</value>
      </mes:messageAttributeList>
    </mep:Request>
  </mep:addNotificationMessage>
</soapenv:Body>
</soapenv:Envelope>
```

POD Selection Areas is subscribed to listen for the topic name = `POD_INPUT_MESSAGE@SHEETMETAL_BENCH` and will process INPUT attribute to populate the SFC field. The Connection ID is specified after @ symbol in topicName field:

```
<mes:topicName>POD_INPUT_MESSAGE@SHEET-METAL_BENCH</mes:topicName>
```

The Notification Message Manager receives the message and forwards it to the POD Selection plug-in, which in turn updates the SFC field. Only specific PODs (with CONNECTION_ID = SHEET-METAL_BENCH) will be updated with the notification message.

Using the Notification Framework for Custom Development

Please read the following recommendations before starting custom development that uses the Notification Framework:

- See SAP Note [2520082](#).
- Do not trigger complex business logic from the Notifications. Only use for refreshing the User Interface.
- Limit the number of messages sent concurrently to a single Production Operator Dashboard using the Notifications.
- Carefully design the Production Operator Dashboard order of events when considering the Notification Messages due to potential blockers from SAP ME plug-in behavior.
- Avoid designing transaction flow in such a way that delays can create discrepancies if not executed due to blockers.
- Alternatively, allow tolerance and contingency methods if these situations occur.

POD Supported Notification Messages

Topic Name	Attribute Name	Attribute Value	Description
POD_INPUT_MESSAGE@CONNECTION_ID	INPUT	SFC/Shop Order/Process Lot name	Update SFC/SO/PL input field at POD Selection Area. In Operation and Work Center PODs, the POD Selection Area plug-in defines itself as a listener for messages using the topic Name of "POD_INPUT_MESSAGE@" followed by the connection identifier CONNECTION_ID. The connection identifier is determined by the customer.
OPERLIST_REFRESH@CONNECTION_ID			Refresh Operation List. Operation List plug-in defines itself as a listener for messages using the topic Name of "OPERLIST_REFRESH@" followed by the connection identifier CONNECTION_ID.
SUBSTEPLIST_REFRESH@CONNECTION_ID			Refresh Substep List. Substep List plug-in defines itself as a listener for messages using the topic Name of "SUBSTEPLIST_REFRESH@" followed by the connection identifier CONNECTION_ID.
WORKLIST_REFRESH@CONNECTION_ID			Refresh POD Work List. Work List plug-in defines itself as a listener for messages using the topic Name of "WORKLIST_REFRESH@" followed by the connection identifier CONNECTION_ID.