SmoothSailing Interface Guidelines

Contents

1 Introduction

This API reference document provides detailed information on the SmoothSailing Java Framework. SmoothSailing is a technique to follow calls through your application to troubleshoot problems.

These public interfaces are intended for designers and deployment engineers and include:

- SmoothSailing Framework – Java APIs for the developer. See Section 2.
- SmoothSailing configuration – for deployment staff. See Section 4.
- SmoothSailing formatting guidelines – for the developer. See Section 3.

2 SmoothSailing Framework

The SmoothSailing Framework is a Java interface to enable applications to register to receive notification of an event sailing on the application interface. Refer to the Javadoc for the interface description. An overview of the APIs is given below.

```
public interface SailingProvider
```

- `SailingReturnCode registerListener(final SailingListener listener, final InterfaceName interfaceName);`
  Registers a listener on the specified interface.
- `public Set<Sailors> getSailors(InterfaceName interfaceName);`
  Gets all Sailors currently defined on the specified interface.

Notifications are provided for each sailor added/deleted/modified. The intended receiver of the event needs to supply a concrete implementation of this interface.

```
public interface SailingListener
```

- `public void sailorCreated(final Sailors sailor);`
  Invoked when a sailor is created on the registered Interface Name.
- `public void sailorDeleted(final Sailors sailor);`
  Invoked when a sailor on the registered Interface Name is deleted.
- `Public void sailorUpdated(final Sailors sailor);`
  Invoked when a sailor is updated on the registered Interface Name.

3 Usage

To correlate sailing events by time and criteria, each event must begin with a one line header that starts with a fully qualified timestamp, formatted as

`yyyy-mm-ddThh:mm:ss.nnn+|-hh:mm`

followed by the header, defined in JSON format. Following this header, the application can add any information required by its interface.

Format:

```
<FullyQualifiedTimestamp> – Sailor=<header>/=Sailor <unparsed/unindexed
payload>
```

3.1 Header Format

The header will contain the following information. Fields marked mandatory are required to be included; optional information can be added as needed.

Field	Data Type	Description	Optional / Mandatory
SailorID	String	The unique identifier of the sailor. The unique identifier is an UUID as described in the bss.cmn.uuid.1.json common schema.	M
TransactionId	String	A unique identifier that makes it possible to correlate the same transaction.	M
Interface	String	The context where the sailor is defined. This parameter shall include the application name and interface over which sailor is sailing. Format: *application name/interface /service/serviceoperation* where service and serviceoperation are optional. Example: *myApp/REST/currentWeather/waveHeight* camel case (starts with a lower-case letter) is used for each part of the string separated with /.	M

Field	Data Type	Description	Optional / Mandatory
Module	String	The name of application function handling the sailor.	M
Level	Integer	The granularity of the information printed: • 0 – External, on enter/exit of the interface • 1 – Internal, external plus more information Other levels may be defined by the application.	M
SessionId	String	An identifier that enables correlation of transactions belonging to the same session.	O
Exception	Boolean	An exception flag to indicate that something went wrong in the module.	O
ContentType	String	The type of payload. Valid values: TEXT or JSON. If not given, default is TEXT.	O

3.2 Example Header

```
2017-05-01T12:090:41.26+02:00 Sailor = {“SailorID”: “asbd21334”,
“TransactionID”: “dakjda-101ks-23jlk”, “Interface”: “secureApp/REST”,
“Module”: “wind”} /Sailor
```

The payload may be formatted as plain text or as JSON. When content type is JSON, the payload should be prefixed with Payload={ } and given as a JSON blob. No new line characters should be included. When content type is TEXT, each payload must be 500 lines or less. Much less is better for readability.

4 Interface Registration

Application interfaces need to be registered with the SmoothSailing framework to trigger sailing over an interface. The application interface definition and supported filtering criteria are given in JSON file(s) and included in the application package. The definition should be loaded as part of application deploy/upgrade. Each invocation will overwrite the previous definition.

```
sailor load --file <JSON file>
```

Command Options	Description	Optional (O) Mandatory (M)
JSON file	Defines the path and name of the JSON file to load	M

The JSON file will have the following format:

```
 {
  “version”:1.0,
  "interfaces":
    [ {
       "interface":"<interface name>",
       “displayName”:<interface name used in CLI and GUI>”
       “description”:”<description for help text>”,
       "criteriaList": [
        {
          "criteria":”<criteriaType>”,
          "criteria":”<criteriaType >”, …,
          "criteria":”<criteriaType >”
        } ],
       "level":<integer>,
       “coverage”:”<type>”
       },
      …
    ]
   }
```

Valid values:

- interface name - defined as <application name>/<interface>. This name isn’t expected to change.
- display name – application interface name used in the CLI and GUI. Must be unique.
- description – description of the interface. Used in CLI help text.
- criteriaList – list of any criteria types (not validated).
- level – Highest level supported.

- 0 = External
- 1 = External and internal
- …

• coverage – LOCAL, GLOBAL, BOTH (local and global).

Example:

```
{
  "version": 1.0,
  "interfaces": [{
     "interface": "myAPP/REST",
     “displayName”: “APP-REST-BOAT”,
     "description": "Secure REST Interface for sailing",
     "criteriaList": [
        {"criteria": "OCEAN"},
        {"criteria": "BAY"}
     ],
     "level": 1,
     "coverage": "BOTH"
     }]
}
```