Service Delivery Broker

Tracing

Version 0.2 – 2014-03-26

Table of Contents

1. Legal Disclaimer

2. Abstract

In this document it is described the Service Delivery Broker (SDB) traces and the integration with client applications and service enablers.

3. Introduction

When a request is handled by the SDB Runtime, information about the processing of that request is collected. This information can be later consulted through the SDB Backoffice or by calling the corresponding support service. The information that is collected helps in finding the root cause of problems by providing information that is usually not available by looking at the response of the service. For example, if a service enabler responds with HTTP status code 500, and exception shielding is enabled on the strategy that processed the request, the response that will reach the client will be "Could not route request". By consulting the corresponding trace, the exact message that was returned by the service enabler can be viewed.

SDB uses the concept of activities to correlate various requests. An activity is simply a group of requests that are logically related. This allows to view entire user interactions that may involve calling various APIs.

In order to achieve the best possible visibility, both client applications and service enablers must cooperate with the SDB Runtime. Client applications should use the activity identifier to correlate various API calls during the same user interaction. Service enablers can publish additional trace information to complement the traces that are collected automatically.

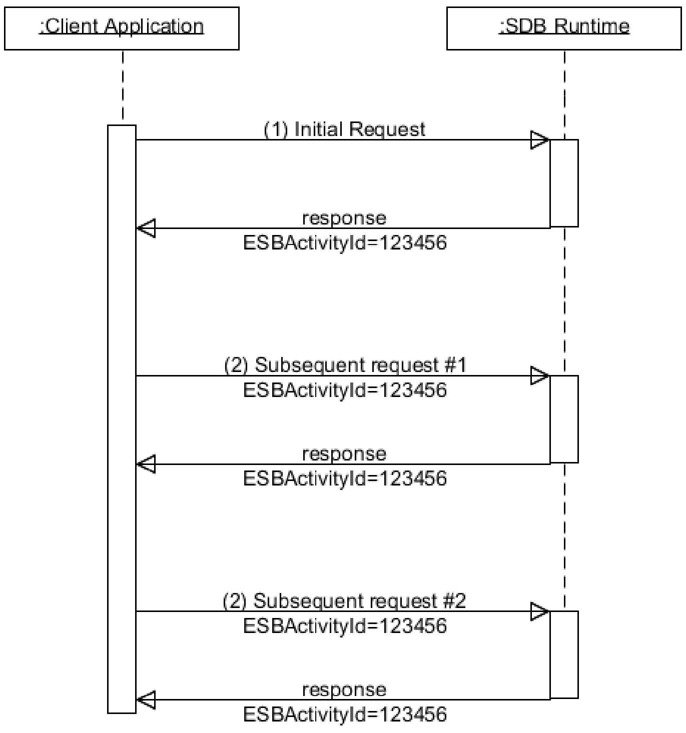
4. Tracing Integration with Client Applications

When a client application is consuming an API on behalf of a user, it is useful to be able to correlate the various calls that the application performed during a given user interaction. This allows to validate that the client application is using the API in the correct way, and also helps when a failed requests is due to a previous request that was performed. Consider an application to manage pictures. If the application requests a pictures and receives a 404 error, being able to know that the previous request was a delete of that picture allows to identify that there is a bug in the application because it is trying to retrieve a photo that it just deleted. Without that information, it would be difficult to reach this conclusion.

Request Correlation

In order to indicate that various requests belong to the same activity, the client application must use an HTTP header according to the following protocol:

1. On the first request of a given activity, the client application should not include the ESBActivityId header. The response will include an HTTP header named ESBActivityId. The value of that header should be stored.
2. On subsequent requests for that activity, the client application should send an HTTP header named ESBActivityId with the value that was previously received. The response will include the same ESBActivityId header.



Additional HTTP Headers

In addition to the ESBActivityId header, the SDB Runtime includes one additional headers that can be used by the client application determine the identifier of the trace that has been produced by a request.

HTTP Headers	
ESBTraceId	The identifier of the trace occurrence. Uniquely identifies the request.

5. Tracing Integration With Service Enablers

Service enablers can contribute to the diagnostics with their own traces. Additionally, if they are themselves clients of other services, they should forward contextual information that enables correlating the various calls.

Trace Publication

On every request made by the SDB Runtime to a service enabler, an HTTP header named ESBTraceContext is included. The value of that header is a token that can be used to publish a child trace by calling the following endpoint.

```
POST /SDBSS/Diagnostics/Trace/DetailsByTraceId?traceContext={traceContext}
Content-Type: application/json

{occurrence}
```

Parameters	
traceContext	Opaque string received in the ESBTraceContext HTTP header.
occurrence	An Occurrence object in JSON format, as described in section Trace Format Description.

Calling Other Services

When calling other services from a service enabler, the ESBTraceContext header should be added to the request headers. This enables the SDB Runtime to link the various traces together and associate them to the same activity.

Additional HTTP Headers

In addition to the ESBTraceContext header, the SDB Runtime includes two additional headers that can be used by the service enabler to correlate a trace or activity with additional information that the service enabler might be producing, such as application logs.

HTTP Headers	
ESBTraceId	The identifier of the trace occurrence. Uniquely identifies the request.
ESBActivityId	The identifier of the current activity.

6. Trace Format Description

This section describes the data types that are used in trace creation, as described in Trace Publication.

Occurrence {

start (Date): The trace start date, in UTC,

end (Date): The trace end date, in UTC,

server (Server, *optional*): Information about the server that hosts the service enabler,

summary (Summary, *optional*): Summary information,

entries (array[Entry]): Trace entries,

messages (array[Message], *optional*): HTTP messages referenced by the trace entries

}

Server {

name (string, *optional*): Name of the server,

address (string, *optional*): IP address of the server,

deployDate (Date, *optional*): Deployment date of the service enabler, in UTC,

version (string, *optional*): Version number of the service enabler code

}

Summary {

result (string): Textual representation of the result of the operation, e.g. "The operation completed successfully",

success (boolean): True if the request succeeded; otherwise false

}

Entry {

severity (string, *optional*) = ['Debug', 'Info', 'Warn', 'Error', 'Fatal']: Severity class,

description (string, *optional*): Textual description of the entry,

offset (number, *optional*): Offset of this entry, relative to the start of the trace, in millisecond,

duration (number, *optional*): Duration of this entry, in millisecond,

messageNumber (integer, *optional*): Array index of the associated message,

exception (Exception, *optional*): Exception that caused this entry,

properties (NameValueCollection, *optional*): Additional properties of this entry,

entries (array[Entry], *optional*): Child entries,

linkedTraceId (string, *optional*): Identifier of another trace to be linked from this entry

}

Message {
  request (Request, *optional*): Additional information when the message is a request,
  response (Response, *optional*): Additional information when the message is a response,
  headers (object, *optional*): HTTP headers of the message,
  body (string, *optional*): Body of the message, in base 64
}

Request {
  verb (string): HTTP Verb, e.g. 'GET',
  secure (boolean): True if the request was over HTTPS; otherwise false,
  pathAndQuery (string): Path and query string
}

Response {
  statusCode (integer): Status code of the response, e.g. 404,
  reasonPhrase (string): Reason phrase for the status code, e.g. 'Not found'
}

Exception {
  type (string): Name of the type of the exception,
  stackTrace (string): Stack trace of the exception
}

NameValueCollection – An object whose values are either string or NameValueCollection. E.g.:

```
{
  "key-A": "scalar value A",
  "key-B": {
    "sub-key-B1": "scalar B1",
    "sub-key-B2": "scalar B2",
  }
}
```

7. Glossary

Activity

A set of requests performed by an application that are logically related.

8. Document Versions

Version	Date	Change Log
0.2	2014-03-26	Removed SMI description.
0.1	2013-12-02	Initial version.