puzzel.

Digital Engagement

Visitor messaging – External API

Table of contents

1 Visitor Messaging - External

1.1 Introduction

This document describes how to implement visitor messaging to the Puzzel Digital Engagement (PDE).
Visitor messaging is a generalized communication channel for end user communication into the Puzzel Digital Engagement (PDE), which can be utilized by a 3'rd party to build clients into PDE, for example for letting their customers communicate with customer service agents in Puzzel Engage through SMS, email, Facebook or XMPP.

1.2 Description

The platform currently only supports plain text messages, but each conversation can be decorated by custom claims that are stored together with the conversation. The claims can be used for a multiple of tasks. For example, storing integration related data to ease integration implantation like storing telephone numbers or emails. The claims are a hash based stored facility for producers and consumers of the Visitor Messaging API, but the PDE platform itself does not actively use the claims. Future customization can be implemented using this storage facility.

Visitor messaging exposes an API on the PDE REST server using a HTTP and JSON content types. The API consists of a simple get and set interface where the consumer must poll (get), to retrieve new activities, and must publish (put) activities to send activities. These activities in turn then trigger conversations in the PDE Desktop application, for human agent consumption.

All endpoints handle JSON formatted information only, using the models described in the Activity Model Types section. The models are decorated with DataContract and DataMember attributes that describes the JSON formatting names.

The API date and time values are formatted using ISO8601, and PDE will always return UTC times.

All events in and out of the system are stored in an event store, and all always re-retrievable.

The API offers a very simple signature as defined below, see Activity Model Types for model description and details.

Get Activities

- **URL**
  - /api/v1/Channel/Visit/Events/Since/:conversationId
- **Method:**
  - GET
- **URL Params**
  - **Required:**
    - conversationId=[Guid]
    - Described the unique identifier of a conversation. The value always originates from an external party, and is used to uniquely reference a conversation

o **Optional:**
- id=[integer]
- Specifies from which activity id to return values from, to enable retrieving only new events since last polling.

• **Data Params**
o None
• **Success Response:**
o **Code:** 200
Content: [{ “reason”: “Agent closed conversation with reason: Success”, “type”: “conversationClosed”, “conversationId”: “20B3A503-5018-4A11-B892-702B0B9369C6”, “id”: 10, “createdAt”: “2017-07-02T22:46:46.8576016Z” }]
• **Error Response:**
o **Code:** 401 UNAUTHORIZED
Content: { error : “You are unauthorized to make this request.” }

• **Sample Call:**
curl -H “Content-type: application/json” -H “Authorization: SessionId eca08cd0-e609-43ba-a497-83ec3d8a9789+dsZEggznZPrHtgpwU42BTaY7GXU6yiiTpNWM2kKTcqM=” “http://localhost/api/v1/channel/visit/events/since/8F1641BB-922C-43EE-A991-625371C4E843”

Sample return
[{“group”: “Amelia”,”namedIdentity”: “Hanna”,”claims”: [{“type”: “PSN”,”values”:[“197010101234”],”-source”: “Amelia Integration”}],”type”: “start”,”conversationId”:“20B3A503-5018-4A11-B892-702B0B-9369C6”,”mediaType”: “SMS”,”id”: 0,”createdAt”: “2017-07-02T22:45:56.8576016Z”}, {“message”: “Hello”,”type”: “plainTextMessage”,”conversationId”: “20B3A503-5018-4A11-B892-702B0B-9369C6”,”id”: 1,”createdAt”: “2017-07-02T22:46:26.8576016Z”}, {“message”: “A note that will no be in the chat transscript”,”type”: “plainTextNote”,”conversationId”: “20B3A503-5018-4A11-B892-702B0B-9369C6”,”id”: 2,”createdAt”: “2017-07-02T22:46:31.8576016Z”}, {“agentName”: “Hanna”,”message”: “Hello back”,”type”: “plainTextMessage”,”conversationId”: “20B3A503-5018-4A11-B892-702B0B-9369C6”,”id”: 3,”createdAt”: “2017-07-02T22:46:42.8576016Z”}, {“reason”: “Agent closed conversation with reason: Success”,”type”:“conversationClosed”,”conversationId”: “20B3A503-5018-4A11-B892-702B0B9369C6”,”id”: 4,”createdAt”: “2017-07-02T22:46:46.8576016Z”}]

• **Code: 200**

• **Sample Call:**
Sample Call with timeStamp:curl -H “Content-type: application/json” -H “Authorization: SessionId eca08cd0-e609-43ba-a497-83ec3d8a9789+dsZEggznZPrHtgpwU42BTaY7GXU6yiiTpNWM2kKTcqM=” “http://localhost/api/v1/channel/visit/events/since/8F1641BB-922C-43EE-A991-625371C4E843?timeStamp=2017-07-02T22:46:46.0Z”

Sample return
[{“reason”: “Agent closed conversation with reason: Success”,”type”: “conversationClosed”,”conversationId”: “20B3A503-5018-4A11-B892-702B0B9369C6”,”mediaType”: “SMS”,”id”: 10,”createdAt”: “2017-07-02T22:46:46.8576016Z”}]

• **Code: 200**

1.3 Publish Activities

Posts one or more activities on one or more conversations. If posting to a conversationId that has not previously been used a new conversation is created. A new conversation must always start with a StartActivity or else activities will not be accepted.

- **URL**
  - o /api/v1/Channel/Visit/Events/Publish
- **Method:**
  - o POST
- **URL Params**
  - o None
- **Data Params**
  - o Array of JSON serialized Activities, see Activity Model Types for model details. Sample

[{“group”: “Amelia”,”namedIdentity”: “Hanna”,”claims”: [{“type”: “PSN”,”values”: [“197010101234”],”source”: “Amelia Integration”}],”type”: “start”,”conversationId”: “20B3A503-5018-4A11-B892-702B0B9369C6”,”mediaType”: “SMS”,”createdAt”: “2017-07-02T22:45:56.8576016Z”}, {“message”: “Hello”,”type”: “plainTextMessage”,”conversationId”: “20B3A503-5018-4A11-B892-702B0B9369C6”,”createdAt”: “2017-07-02T22:46:26.8576016Z”}]

- **Success Response:**
  - o **Code:** 200
- **Error Response:**
  - o **Code:** 401 UNAUTHORIZED
    Content: { error : “You are unauthorized to make this request.” }

OR

  - o **Code:** 500 Internal Server Error
    Reason-Phrase: Publish activity failed Content: {“error”:”For activity with index: 0 and id: 20b3a503-5018-4a11-b892-702b0b9369c6 an error occured Object reference not set to an instance of an object.;\r\nFor activity with index: 1 and id: 20b3a503-5018-4a11-b892-702b0b9369c6 an error occured Object reference not set to an instance of an object.;\r\nFor activity with index: 2 and id: 20b3a503-5018-4a11-b892-702b0b9369c6 an error occured Object reference not set to an instance of an object.;\r\nFor activity with index: 3 and id: 20b3a503-5018-4a11-b892-702b0b9369c6 an error occured Object reference not set to an instance of an object.;\r\nFor activity with index: 4 and id: 20b3a503-5018-4a11-b892-702b0b9369c6 an error occured Object reference not set to an instance of an object.;\r\nFor activity with index: 5 and id: 20b3a503-5018-4a11-b892-702b0b9369c6 an error occured Object reference not set to an instance of an object.”}

- **Sample Call:**

curl -H “Content-type: application/json” -H “Authorization: SessionId eca08cd0-e609-43ba-a497-83ec3d8a9789+dsZEggznZPrHtgpwU42BTaY7GXU6yiiTpNWM2kKTcqM=” --data ‘[{“group”:“Amelia”,”namedIdentity”: “Hanna”,”claims”: [{“type”: “PSN”,”values”: [“197010101234”],”source”: “Amelia Integration”}],”type”: “start”,”conversationId”: “20B3A503-5018-4A11-B892-702B0B9369C6”,”createdAt”: “2017-07-02T22:45:56.8576016Z”}, {“message”: “Hello”,”type”: “plainTextMessage”,”conversationId”: “20B3A503-5018-4A11-B892-702B0B9369C6”,”createdAt”: “2017-07-02T22:46:26.8576016Z”}]’

returns

- **Code:** 200

OR

• **Sample Call with invalid JSON:**
curl -H “Content-type: application/json” -H “Authorization: SessionId eca08cd0-e609-43ba-a497-83ec3d8a9789+dsZEggznZPrHtgpwU42BTaY7GXU6yiiTpNWM2kKTcqM=” --data ‘{“group”:
“Amelia”,”namedIdentity”: “Hanna”,”claims”: [{“type”: “PSN”,”values”: [“197010101234”],”source”: “Integration”}’
“http://localhost/api/v1/Channel/Visit/Events/Publish”

Returns

{“errorCode”:”modelValidationFailed”,”message”:”activity: Cannot deserialize the current JSON object (e.g. {\”name\”:\”value\”}) into type ‘vngage.WebApi.Model.Visit.Activity[]’ because the type requires a JSON array (e.g. [1,2,3]) to deserialize correctly.\r\nTo fix this error either change the JSON to a JSON array (e.g. [1,2,3]) or change the deserialized type so that it is a normal .NET type (e.g. not a primitive type like integer, not a collection type like an array or List<T>) that can be deserialized from a JSON object. JsonObjectAttribute can also be added to the type to force it to deserialize from a JSON object.\r\nPath ‘group’, line 3, position 8.”,”title”:”Model validation on input failed”,”referenceId”:”98b160e36c5c4e3183f597059a980c79”}

• **Code:** 400 Model validation on input failed

Activity Model Types

Activities are message types used to send and receive different activities with the Messaging API. See in code descriptions for details

```
/// <summary>
/// Base class for Visit Channel SDK using polymorphism
/// </summary>
[DataContract(Name = “activity”)]
public abstract class Activity
{
/// <summary>
/// Describes the activity type when serialized
/// This property is always automatically generated and never needs to be set manually
/// Externally set values are ignored
/// </summary>
[DataMember(Name = “type”)]
public string Type { get; set; }
/// <summary>
/// Described the unique identifier of a conversation
/// The value always originates from an external party, and is used to uniquely reference a conversation
/// The value must be set on every request it is used
/// </summary>
[DataMember(Name = “conversationId”)]
public Guid ConversationId { get; set; }
/// <summary>
/// Describes the ordering of activities in a conversation
/// The values is automatically set and never needs to be set manually
/// Externally set values are ignored
/// </summary>
```

```
[DataMember(Name = “id”)]
public int Id { get; set; }
/// <summary>
/// The time this activity was created
/// The values is automatically set and never needs to be set manually
/// Externally set values are ignored
/// </summary>
[DataMember(Name = “createdAt”)]
public DateTime CreatedAt { get; set; }
}

/// <summary>
/// The first activity used to start a conversation specifying conversation properties
/// </summary>
[DataContract(Name = “start”)]
public class StartActivity : Activity
{
/// <summary>
/// Specifies the group in which the conversation belongs to
/// </summary>

[DataMember(Name = “group”)]
public string Group { get; set; }
/// <summary>
/// Specified the identity of the visitor
/// Used as a display value in the conversation
/// </summary>
[DataMember(Name = “namedIdentity”)]
public string VisitNamedIdentity { get; set; }
/// <summary>
/// Specifies from what media the conversation belongs to, ie. SMS, email, XMPP
/// </summary>
[DataMember(Name = “mediaType”)]
public string MediaType { get; set; } = “Messaging”;
/// <summary>
/// Conversation meta data
/// </summary>
[DataMember(Name = “claims”)]
public Claim[] Claims { get; set; }
}

/// <summary>
/// Conversation metadata entry
/// Can contain any form of string data
/// </summary>
[DataContract(Name = “claim”)]
public class Claim

{
/// <summary>
/// Specifies the type of claim, this could for example be a phone number, email or personal
identification number
/// </summary>
[DataMember(Name = “type”)]
public string Type { get; set; }
```

```
/// <summary>
/// Actual values of the claim
/// </summary>
[DataMember(Name = “values”)]
public string[] Values { get; set; }
/// <summary>
/// Specifies from where the claim information originates, as there can be multiple sources of
claims
/// </summary>
[DataMember(Name = “source”)]
public string Source { get; set; }
}

/// <summary>
/// Class represents a text message in a chat conversation
/// </summary>
[DataContract(Name = “plainTextMessage”)]
public class PlainTextMessageActivity : Activity
{
/// <summary>
/// Contains the visual name of the entity from which the message originates
/// </summary>
[DataMember(Name = “speaker”)]
public string Speaker { get; set; }
/// <summary>
/// The actual plain text chat message
/// </summary>
[DataMember(Name = “message”)]
public string Message { get; set; }
}
/// <summary>
/// Represents a text message note in a chat conversation, but will not appear in transscripts
/// </summary>
[DataContract(Name = “plainTextNote”)]
public class PlainTextNoteActivity : PlainTextMessageActivity
{
}

/// <summary>
/// Sent by Puzzel Engage when a conversation has closed (Case closed)
/// Can only be created by Puzzel Engage, external instances willb e ignored.
/// </summary>
[DataContract(Name = “conversationClosed”)]
public class ConversationClosedActivity : Activity
{
[DataMember(Name = “reason”)]
public string Reason { get; set; }
}
```

1.4 Authorization

In order to communicate with the PDE REST service the Visitor Message consumer must use a pregenerated token for each of HTTP requests. This way authentication is skipped and only authorization for each request is done. The token is secret will never expire automatically and should only be used in API-to-API context. The token can be revoked/replaced at any time. The token will be delivered separately on request.

Here is an example of how to use the token with url parameter:

curl -X GET --header 'Accept: application/json' 'http://commsrva.
vergic.com/api/v1/Channel/Visit/Events/Since?sessionId=0387d170-7820-
4dd0-a36d-222a0589fc13%2BOg4CTkPl233pfsleXt4KCcZB5Xb045wxqVjq537C4Q%3D'

And here is an example of how to use the token with a HTTP header:
curl -X GET --header 'Accept: application/json' --header 'Authorization:
SessionId 0387d170-7820-4dd0-a36d-
222a0589fc13%2BOg4CTkPl233pfsleXt4KCcZB5Xb045wxqVjq537C4Q%3D'
'http://commsrv-a.vergic.com/api/v1/Channel/Visit/Events/Since'

For information about authentication and authorization see document: API authorization.docx

1.5 Messaging

PDE messaging is always open for processing, and new conversation messages are always excepted, even if there is no agent available inside the PDE group.

To start a conversation with an Agent in PDE start by publishing a StartActivity. All conversations must always start with a StartActivity. StartActivities can be published multiple times if Claims need to be updated. Consumer must explicitly specify for which group the conversation belong, in order for PDE to route the conversation to the correct agents. For all consumer published activities, a unique ConversationId (see Model declaration). The ConversationId is the primary key for the conversation. The ConversationId must be generated by the integrator, it is a random generated GUID. The PDE messaging will then reuse the ConversationId for all activities generated inside Visitor Messaging platform. This is also the key used when polling for changes.

If activities are published on conversation that do not have a StartActivity the PDE API will return with an error.

When a conversation start been started then text messages can be published and received. That is done via the PlainTextActivity. When a PlainTextActivity is published the conversation is enqueued in the PDE Desktop application. Now a human agent can respond with a message on the conversation. This is also done via the PlainTextActivity, where the Sender properties shows the name of the agent.

A Visitor Messaging API consumer must poll for changes on a specific conversation until a ConversationClosedActivity is retrieved. Polling is done per conversation by using the conversation primary key, the ConversationId as a parameter on the REST endpoint api/v1/channel/visit/events/since. There is one polling request per conversation. After receiving the ConversationClosedActivity PDE will no longer send any new activities on that specific conversation. But a Visitor Messager API consumer can still send messages on a conversation that have received the ConversationclosedActivity. If this occurs the conversation is reopened and the API consumer must restart polling until again the ConversationClosedActivity is received. This can be done indefinitely.

Every time a PlainTextActivity is published it will be forwarded to a human agent that will take appropriate action in due time.

The below swim lane diagram shows a typical flow a channel provider must do when polling for new activities in PDE on a specific conversation. The flow must exist for each concurrent active conversation.

Sample .net implementation of this exists, using the described method. This document can be provided upon request.