ERICSSON

# Dynamic Journey Service Programmers Guide

## Programmers Guide

## User Guide

14/1553-CSH 109 698/4 Uen F

## Disclaimer

The contents of this document are subject to revision without notice due to continued progress in methodology, design and manufacturing. Ericsson shall have no liability for any error or damage of any kind resulting from the use of this document.

# Contents

# 1 Dynamic Journey Service Programmers Guide Introduction

This section is an introduction to this document. It contains information about the purpose, scope, and target group for the document.

## Purpose and Scope

The purpose of this document is to provide a guideline for:

— Development of dynamic journeys.

— Development of front-end client applications that interact with dynamic journeys.

## Target Group

The following are included in the target group for this document:

— System Integrator

— Software Developer

# 2 Dynamic Journey Service

## Overview

The Dynamic Journey Service (DJS) offers functionality to manage and execute dynamic journeys on Ericsson Wallet Platform (EWP).

It is possible for service providers (partners) to develop and manage journeys used to provide customized offerings to account holders.

The journey defines an execution plan that involves interaction with account holders for information retrieval and callbacks to the service provider for information processing and retrieval of dynamic instructions.

The journey is defined as XML structure following a pre-defined syntax. It is written and tested by the service provider and uploaded to the EWP. Journey Definition Overview on page 3 describes the XML syntax and its semantics in detail.

Once a journey is made available on EWP, client applications can use the provided front-end API to select journeys, start a session for execution and interact with EWP in order to provide user input and get responses to be displayed to the user.

Dynamic Journey Client Application on page 23 describe how clients can make use of the front-end API.

# 3 Journey Definition Overview

The dynamic journey is described by its journey definition.

The journey designer must compose this journey definition in XML format which can be interpreted by EWP.

### Journey Definition Overview

The journey definition is a structured group of instructions that form an acyclic graph and is described in XML syntax.

During execution of the journey, this graph is traversed based on chosen options and input data retrieved from user interaction. The journey execution ends when a response instruction is reached. This will give the result of the journey.

The chapters below describe the different entities of the XML syntax used to specify a journey for execution on EWP.

**Note:** XSD does not reflect all restrictions that are applicable for specific type definitions. Such restrictions are nevertheless enforced by EWP.

## 3.1 Journey Definition Instructions

### 3.1.1 Journey Definition Root Element

#### 3.1.1.1 Journey

Journey is the root element of a journey definition.

Table 1 Elements

| Name | Type | Description |
|---|---|---|
| journey | list<instruction> | The list of top-level instructions for a journey. Execution starts with the first instruction of this list. |

#### 3.1.1.2 Instruction

The instruction is a choice of one of the available instruction types. These types are the building blocks for a journey definition that specify the behavior during journey execution.

Some instructions may refer to other instructions, creating a parent-child relationship.

Once an instruction is executed, its children instructions will be executed recursively in a depth first strategy, and then another instruction in the same level as the parent instruction will execute.

The following instruction types are defined:

— argument

— dynamicargument

— question

— options

— dynamicoptions

— exists

— matches

— switch

— response

— responsematching

The available set of instructions is below grouped into instructions for user input, session data, control flow and session finalization. The following chapters describe each of the listed instruction types in detail.

### 3.1.2 Session Data Instructions

The instructions in this section can be used to store data for the ongoing session.

Any data that is captured during journey execution is stored as key value pairs called arguments.

These arguments can be referred to from all instructions, the syntax $ `{argument-key}` is during execution resolved to the argument-value stored for the given argument-key.

#### 3.1.2.1 Argument

**Purpose**

The argument instruction is used to add data to the user session so it can be retrieved later by other instructions, see page 5.

The data to be stored is a `key` and `avalue` pair, see Table 2.

Any previous value that was associated to the same key will be overwritten.

Table 2 Elements

| Name | Type | Description |
|---|---|---|
| key | argumentkey | The key of the argument. |
| value | textformat | The value to be set for the given argument. |

```
<argument>
  <key>yearofbirth</key>
  <value>1990</value>
</argument>

<argument>
  <key>yearssincebirth</key>
  <value>${age}</value>
</argument>
```

### Example Explanation

The argument with key `yearofbirth` will be added to the user session, with the value of `1990`.

The argument with key `yearssincebirth` will be added to the user session, with the value contained in the argument with key `${age}`.

If no argument is found an error will be thrown.

## 3.1.2.2 dynamicarguments

### Purpose

To get arguments from a service provider to be added to the user session.

This instruction causes a request with a set of arguments being sent to a given URL. In response a set with arguments is received that will be added to the user session.

The request is synchronous and journey execution will halt until response is received or time-out.

With this construction, it is possible to pass user input onto the service provider for further processing.

The arguments sent in response could be used to control the execution path of the journey or they could be used in response to the end user.

Table 3 Elements

| Name | Type | Description |
|---|---|---|
| url | uniformResourceLocator | The URL to which the request should be sent. |
| arguments | list<argument> | Arguments (key, value) that are to be sent as payload to the specified URL. |

```
<dynamicarguments>
  <url>https://provider-gateway.com/djs/dynamicarguments?get</url →
>
    <arguments>
       <argument>
         <key>username</key>
         <value>Walden</value>
       </argument>

       <argument>
         <key>age</key>
         <value>${age}</value>
       </argument>
    <arguments>
</dynamicarguments>
```

#### Example Explanation

A request with the arguments `username` and `age` will be sent to https://provider-gateway.com/djs/dynamicarguments?get.

The `Username` has value `Walden`, and `age` has the value of age argument from the user session.

All data received in the response from the service provider will be added to the session. If no argument with the key age is found, an error will be thrown.

### 3.1.3 User Input Instructions

This set of instructions allows for gathering input data from the user.

The journey execution proceeds until a user input instruction is reached, then it halts and returns a response to be used by the client application to prompt the user for input data.

The client should enter the required user input to continue the execution of the journey.

#### 3.1.3.1 Question

To present a question to the user and to retrieve a corresponding answer.

When a question instruction is encountered, the journey execution is paused, and the question is prompted to the user.

The client should then resume the journey execution by sending an answer to the question in a new request.

**Note:** It is important to flag questions as confidential if the expected answer may contain confidential data, otherwise the received answer could be exposed in log files.

Table 4 Elements

| Name | Type | M/O | Description |
|---|---|---|---|
| key | argumentkey | M | Key of the argument that will hold the answer. |
| retries | long | O | The maximum number of retry attempts in case an invalid answer was given.<br>Maximum value: 5 |
| confidential | boolean | M | A flag indicating whether the answer contains confidential data (confidential data will not be exposed in logs). |
| display | display | M | The question to be presented to the user (display, consisting of text and language code). |
| validation | validation | O | Pattern given as regular expression to be applied on the answer (optional) answers not following the pattern will be rejected with error message. |
| transform | transform | O | Transformation pattern to be applied on the answer (optional), for example, can be used to apply a prefix to the answer. |

```
<question>
    <key>username</key>
    <retries>3</retries>
    <confidential>false</confidential>
    <display>
         <texts>
             <text>
                 <languagecode>en</languagecode>
                 <textmessage>Please enter your name</textmessage →
>
             </text>
          </texts>
    </display>
    <validation>
         <pattern>[a-zA-Z]+</pattern>
        <errormessage>
             <texts>
                 <text>
                     <languagecode>en</languagecode>
                     <textmessage>Only letters allowed.</textmess →
age>
                 </text>
              </texts>
          </errormessage>
    </validation>
    <transform><format>Name: ${username}</format></transform>
</question>
```

## Example Explanation

A question will be prompted to the user, its answer will be stored in an argument with the key `username`. The question is not confidential, so the answer will appear in the audit logs.

The text `Please enter your name` will be displayed to the user case its preferred language is English, or DJS will prompt an error stating that there is no text for the user's language, and the session will be terminated.

The answer will be validated against the `[a-zA-Z]+` regular expression.

If the answer is valid, its value will be transformed into the string: `Name: ${username}`, and `${username}` is the placeholder that will be replaced with the value which has been just entered by the user.

After the transformation, which is optional, the value will be stored as an argument in the session with the key `username`.

If the answer is invalid, the user will have three others retry attempts. If all the retry attempts fail, the session will be terminated, and an error will be prompted to the user.

In every retry attempt, an error message with the text `Only letters allowed.` is displayed to the user if it has preferred language as English, or DJS will prompt an error stating that there is no text for the user's language, and the session will be terminated.

#### 3.1.3.2 Options

A list of options to be presented to the user, who may select only one choice. This is kind of a (sub)-menu presented to the user.

When a choice is selected, the set of instructions associated to it will be executed.

This set of instructions can include any instruction, including other options instruction.

When an options instruction is encountered, the journey execution is paused, and the available options are prompted to the user.

The client should then resume the execution by sending an answer containing the chosen option in a new request.

This is explained in detail in the API section of this document.

Table 5 Elements

| Name | Type | M/O | Description |
|---|---|---|---|
| header | display | O | A header to be displayed on top of the list of options. |
| footer | display | O | A footer to be displayed at the bottom of the list of options. |
| option | list<option> | M | The list of options to choose from. |

Option Definition

Table 6 Elements

| Name | Type | M/O | Description |
|---|---|---|---|
| display | display | M | Text presented to the user to describe an option |
| instructions | list<instruction> | M | The instructions to be executed if this choice is made. Mandatory, but the list can be empty. |

```
<options>
    <header>
         <texts>
              <text>
                   <languagecode>en</languagecode>
                   <textmessage>Final Confirmation</textmessage>
              </text>
          </texts>
    </header>
    <optionslist>
    <option>
        <display>
             <texts>
                  <text>
                      <languagecode>en</languagecode>
                      <textmessage>Purchase product</textmessage>
                  </text>
              </texts>
        </display>
        <instructions>
             <response>
                   <texts>
                        <text>
                             <languagecode>en</languagecode>
                             <textmessage>Your order has been pla →
ced</textmessage>
                        </text>
                    </texts>
               </response>
          </instructions>
    </option>
    <option>
        <display>
             <texts>
                  <text>
                       <languagecode>en</languagecode>
                       <textmessage>Cancel</textmessage>
                  </text>
              </texts>
        </display>
        <instructions>
             <response>
                    <texts>
                          <text>
                                <languagecode>en</languagecode>
```

```
                                        <textmessage>Order cancelled</tex →
tmessage>
                                </text>
                            </texts>
                        </response>
                    </instructions>
                </option>
                </optionlist>
</options>
```

### Example Explanation

A set of options will be prompted to the user.

There is a header to explain what the options are, and if the user has English as preferred language, the text will be `Final Confirmation`.

Two options will be displayed, one called **Purchase product**, the other one called **Cancel**.

If the first option is chosen, a response instruction will run and show the text `Your order has been placed` to the user.

If the second option is chosen, a response instruction will run and show the text `Order cancelled` to the user.

## 3.1.3.3 dynamicoptions

Dynamic options work in an analogous way as options, but instead of having pre-defined options written in the journey definition, the options can be dynamically fetched during the journey execution. This makes it possible to customize the options for each user based on either provided user input or other user preferences.

When this instruction is encountered, a list of set of arguments is retrieved from the specified URL. Each option is comprised by a set of arguments, and you receive a list of options, thus a list of set of arguments. An option in the dynamic options can only contain the argument instruction. Except for the default option, which can contain any instruction and it is pre-defined in the journey definition. The default option will be chosen automatically if no options are given by the service provider. If no options are received and there is no default option, an error will be thrown.

Analogous to dynamic arguments, you can specify arguments in the request, enabling the service provider to customize the response.

There is only one display text in dynamic options, and this will be used to render every option. This display text should contain placeholders that will be substituted by the set of arguments that exists for each option.

When a dynamic options instruction is encountered, the journey execution is paused, and the available options are prompted to the user. The client should

then resume the execution by sending an answer containing the chosen option in a new request. This is explained in detail in the API section of this document.

When the option is chosen, each argument in the option's set of arguments is stored in the session.

Table 7 Elements

| Name | Type | M/O | Description |
|---|---|---|---|
| header | display | O | A header displayed on top of the list of options. |
| footer | display | O | A footer displayed at the bottom of the list of options. |
| url | uniformResourceLocator | M | The URL to which the request should be sent. |
| display | display | M | Text presented to the user to describe an option this serves as a template for each option and will be rendered with the arguments retrieved from URL. |
| arguments | list<argument> | M | A list of arguments (key, value) that are to be sent to the specified URL. |
| defaultoption | option | O | A default option to be applied in case no options were received |

```
<dynamicoptions>
    <header>
         <texts>
             <text>
                   <languagecode>en</languagecode>
                   <textmessage>Special Offers</textmessage>
             </text>
          </texts>
    </header>
    <footer>
          <texts>
               <text>
                    <languagecode>en</language>
                    <textmessage>Please select one of the options →
 above</textmessage>
               </text>
          </texts>
    </footer>
    <url>https://provider-gateway.com/djs/dynamicoptions?get</url →
>
    <display>
         <texts>
              <text>
                  <languagecode>en</languagecode>
                  <textmessage>Your bonus of ${amount} with valid →
ity of ${validity}</textmessage>
              </text>
          </texts>
    </display>
    <arguments>
```

```
        <argument>
             <key>campaign</key>
             <value>summerspecial</value>
        </argument>
        <argument>
              <key>code</key>
              <value>${customkey}</value>
        </argument>
    </arguments>
</dynamicoptions>
```

**Example Explanation**

A list of options will be retrieved from https://provider-gateway.com/djs/dynamicoptions?get

Two arguments will be sent to the provider: `campaign`, with the value `summerspecial`, and code, with the value contained in the argument with key `customkey`.

If no options are received, an error will be thrown, since there is no default option.

A header will be displayed with the text **Special Offers**, and a footer with the text **Please select one of the options above**.

A list of options will be displayed, substituting the argument templates for the ones received in the response. One example of the list could be:

— `Your bonus of 10USD with validity of 1 month`

— `Your bonus of 50USD with validity of 3 months`

## 3.1.4 Control Flow Instructions

This set of instructions allows to specify different execution paths to be chosen based on the session data (stored arguments).

### 3.1.4.1 exists

This is a branch condition that verifies the existence of a specific argument in the session data.

In case the argument exists, the `yes`instruction will be executed. Otherwise, the `no`instruction will be executed.

Table 8 Elements

| Name | Type | M/O | Description |
|---|---|---|---|
| key | argumentkey | M | The key of the argument to be verified |
| yes | list<instruction> | M | Instructions to execute when the argument with given key exists |

| Name | Type | M/O | Description |
|---|---|---|---|
| no | list<instruction> | M | Instructions to execute when the argument with given key does not exist |

```
<exists>
    <key>username</key>
    <yes>
        <instructions>
             <response>
                    <texts>
                         <text>
                             <languagecode>en</languagecode>
                             <textmessage>Registered name: ${user →
name}</textmessage>
                         </text>
                     </texts>
                </response>
          </instructions>
     </yes>
    <no>
          <instructions>
                 <response>
                       <texts>
                           <text>
                                <languagecode>en</languagecode>
                                <textmessage>The username is unkn →
own</textmessage>
                           </text>
                      </texts>
                 </response>
            </instructions>
    </no>
</exists>
```

### Example Explanation

The instruction will check if an argument with the `key` `username` exists in the session.

If `yes`, it will execute the instruction in `Yes` section, returning a response with the text: `Registered username: John`, for example.

If `no`, it will execute the instruction in the `No` section, returning a response with the text `The username is unknown`.

## 3.1.4.2 matches

This is a branch condition that verifies whether the value of a given argument matches a predefined pattern.

The pattern is described as regular expression. In case the value matches the pattern, the `yes`instructions will be executed, otherwise the `no`instructions will be executed.

If the argument does not exist, the `no` instructions will be executed.

Table 9 Elements

| Name | Type | M/O | Description |
|---|---|---|---|
| key | argumentkey | M | The key of the argument to be verified. |
| pattern | regularexpression | M | The pattern to be applied on the argument value. |
| yes | list<instruction> | M | Instructions to execute when the argument value matches the given pattern. |
| no | list<instruction> | M | Instructions to execute when the argument value does not match the given pattern. |

```
<matches>
    <key>username</key>
    <pattern>[a-zA-Z]{8}</pattern>
    <yes>
         <instructions>
              <response>
                   <texts>
                        <text>
                             <languagecode>en</languagecode>
                             <textmessage>The format of username →
 is accepted</textmessage>
                         </text>
                    </texts>
                </response>
         </instructions>
    </yes>
    <no>
         <instructions>
               <response>
                     <texts>
                          <text>
                               <languagecode>en</languagecode>
                               <textmessage>The username is not  →
valid</textmessage>
                           </text>
                     </texts>
               </response>
         </instructions>
    </no>
</matches>
```

### Example Explanation

This instruction will check if there is an argument value with `key username` that matches the pattern `[a-zA-Z]{8}`.

If yes, it will return a response with the text `The format of username is accepted`, otherwise it will return a response with the text `The username is not valid`.

### 3.1.4.3 switch

This is a branch condition that allows to switch on the value of a given argument.

Depending on which of the foreseen values of an argument is given, a different instruction can be executed.

The first match is chosen. When no match is found, the default case is chosen. If the given argument is not found in the session, the default case is chosen.

If no match is found and there is no default case, nothing will be executed.

Table 10 Elements

| Name | Type | M/O | Description |
|---|---|---|---|
| key | argumentkey | M | The key of the argument to switch on. |
| case | list<case> | O | Instructions to execute based on specific argument values (several cases can be specified for different argument values). |
| defaultcase | list<instruction> | O | Instructions to be executed when the argument values do not match any of the given alternatives. |

Case Definition

Table 11 Elements

| Name | Type | M/O | Description |
|---|---|---|---|
| value | textformat | M | The value to be matched with argument value. |
| instructions | list<instruction> | M | Instructions to be executed when argument value the value defined for this case. |

```
<switch>
    <key>color</key>
    <cases>
        <case>
            <value>red</value>
            <instructions>
                  <response>
                        <texts>
                              <text>
                                    <languagecode>en</languagecode →
>
                                    <textmessage>Item will be deli →
vered in red color.</textmessage>
                              </text>
                        </texts>
                  </response>
```

```
                </instructions>
            </case>
            <case>
                <value>green</value>
                <instructions>
                      <response>
                           <texts>
                                <text>
                                      <languagecode>en</languagecode>
                                      <textmessage>Item will be deliv →
ered in green color.</textmessage>
                                </text>
                           </texts>
                      </response>
                 </instructions>
             </case>
        </cases>
        <defaultcase>
            <instructions>
                  <response>
                        <texts>
                            <text>
                                 <languagecode>en</languagecode>
                                 <textmessage>Item will be delivered  →
in default black color.</textmessage>
                             </text>
                        </texts>
                    </response>
                </instructions>
        </default-case>
</switch>
```

### Example Explanation

The instruction will check user session for an argument with the `key color`.

It will try to match its value with the string red, returning a response with the text: `Item will be delivered in red color.`

It will try to match its value with the string green, returning a response with the text: `Item will be delivered in green color.`

If the argument does not exist in the session or the value is different than red or green, the default case will be executed and it will return a response with the text: `Item will be delivered in default black color.`

## 3.1.5 Session Finalization Instructions

The journey execution terminates when one of the instructions in this section is reached. The specified response of these instructions will be sent in answer to the

pending client request along with an indication that the journey has been terminated.

Each journey definition must have one of these instructions as the last instruction to be executed. In case there are subsequent instructions in a list of instructions, those would never be executed and if the last instruction reached is not for session finalization, no final answer could be sent to the client and this is considered to be an error.

### 3.1.5.1 response

#### Purpose

This instruction terminates the journey and specifies a resulting text to be presented to the user.

Table 12 Elements

| Name | Type | M/O | Description |
|---|---|---|---|
| texts | list<translation> | M | The resulting text to be presented to the user. |

```
<response>
       <texts>
            <text>
                 <languagecode>en</languagecode>
                 <textmessage>final answer</textmessage>
            </text>
         </texts>
</response>
```

#### Example Explanation

The instruction returns the text `final answer` and ends the journey.

### 3.1.5.2 responsematching

#### Purpose

This is a convenient instruction to choose between different response messages to be rendered based on the arguments available in session data.

Arguments are matched with a list of different alternative responses to be given. The first response in the list for which all required arguments are available will be chosen.

Table 13 Elements

| Name | Type | M/O | Description |
|---|---|---|---|
| responses | list<response> | O | An alternative response, typically involving arguments. |
| defaultresponse | response | M | The response to be given if none of the alternatives in list could be chosen. |

Provided that the following arguments are defined: product="balloon", quantity="20", then the second response below would be chosen because argument 'color', which would be necessary to render the first response alternative, is not available, see page 18.

```
<responsematching>
    <responses>
        <response>
             <texts>
                 <text>
                      <languagecode>en</languagecode>
                      <textmessage>You ordered ${quantity} ${prod →
uct} of color ${color}.</textmessage>
                 </text>
             </texts>
        </response>
        <response>
              <texts>
                   <text>
                        <languagecode>en</languagecode>
                        <textmessage>You ordered ${quantity} ${pr →
oduct} of any color.</textmessage>
                   </text>
              </texts>
        </response>
    </responses>
    <defaultresponse>
        <texts>
             <text>
                  <languagecode>en</languagecode>
                  <textmessage>Your order could not be completed. →
</textmessage>
             </text>
         </texts>
    </defaultresponse>
</responsematching>
```

### Example Explanation

The instruction will return the first answer if the arguments quantity, product and color are available. It will return the second answer if the arguments quantity and product are available. Otherwise, it will return the third answer.

## 3.1.6 Journey Definition Instructions Data Type Definitions

The type definitions in this chapter are used to form more complex data types and are listed for reason of completeness. All types are referred from the various instructions described above therefore only basic information is given in this chapter.

### argumentkey

A key identifying a value in the user's session Restricted string: `[a-zA-Z0-9]`, length: `1 – 64`.

### display

Defines texts to be displayed.

Table 14 Elements

| Name | Type | Description |
|---|---|---|
| text | list<translation> | A text to be displayed. Each element is a translation of the same text for different languages identified by language code. |

### errormessage

Error messages to be displayed.

Table 15 Elements

| Name | Type | Description |
|---|---|---|
| text | list<translation> | The error message to be given, can be specified in different languages. |

### languagecode

Restricted string: two- or three-letter language codes based on ISO 639-1 or ISO 639-3 standard with the priority of the two-letter notation (ISO 639-1), see *639-2 Language Code List*, accessible at http://www.loc.gov/standards/iso639-2/php/code_list.php.

### regularexpression

A regular expression.

Restricted string, length 1 – 512, must form a regular expression. Example: `^[0-9]{3}.+`

### textformat

A text that can contain place holders. Each place holder is replaced by the value in the user's session matching the key.

Restricted string, length: 1 – 1024.

### transform

A definition for transforming a value before storing it in the user's session.

Table 16 Elements

| Name | Type | M/O | Description |
|---|---|---|---|
| format | textformat | M | The format (template) to be applied when transforming the value. |

### translation

A definition of a translation entry.

Table 17 Elements

| Name | Type | M/O | Description |
|---|---|---|---|
| textmessage | textformat | M | The text to be displayed. |
| languagecode | languagecode | M | The language code specifying in which language the text is written. |

### uniformResourceLocator

Restricted string must form a valid URL.

### validation

A definition that will validate a value before it is added to a user's session.

Table 18 Elements

| Name | Type | M/O | Description |
|---|---|---|---|
| pattern | regularexpression | M | The pattern that should be used for validation. |
| errormessage | errormessage | M | The error message to be given when the validation fails. |

```
<userargument>
  <key>input</key>
  <usertext>myanswer</usertext>
</userargument>
```

### usertext

A text to be used as input or output by the end user.

Restricted string, length: 1 – 1024.

## 3.2 Example Journey

The example below illustrates a small, complete journey definition.

Example 1

```
<?xml version="1.0" encoding="UTF-8"?>
<op:journeydefinition xmlns:op="http://www.ericsson.com/em/djs/journey/v1_0/common→
/" xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```
<instructions>
    <options>
        <header>
            <texts>
               <text>
                  <languagecode>en</languagecode>
                  <textmessage>Select internet offer</textmessage>
               </text>
            </texts>
        </header>
        <optionslist>
        <option>
            <display>
                <texts>
                   <text>
                      <languagecode>en</languagecode>
                      <textmessage>10 EUR (1 Month)</textmessage>
                   </text>
                </texts>
            </display>
            <instructions>
             <argument>
                 <key>amount</key>
                 <value>10</value>
              </argument>
           </instructions>
        </option>
        <option>
            <display>
                <texts>
                  <text>
                      <languagecode>en</languagecode>
                      <textmessage>50 EUR (6 Months)</textmessage>
                   </text>
                </texts>
            </display>
            <instructions>
              <argument>
                 <key>amount</key>
                 <value>50</value>
              </argument>
            </instructions>
        </option>
        </optionslist>
    </options>
    <question>
        <key>pin</key>
        <confidential>true</confidential>
        <display>
           <texts>
              <text>
                 <languagecode>en</languagecode>
                 <textmessage>Enter PIN to confirm refill of Internet for ${amount→
} EUR</textmessage>
              </text>
           </texts>
        </display>
    </question>
    <dynamicarguments>
        <url>https://provider.com/buyinternet</url>
        <arguments>
            <argument>
               <key>amount</key>
               <value>${amount}</value>
            </argument>
            <argument>
               <key>pin</key>
               <value>${pin}</value>
            </argument>
        </arguments>
    </dynamicarguments>
    <exists>
        <key>bonusCombo</key>
        <yes>
        <instructions>
            <dynamicoptions>
                <url>https://provider.com/buybonus</url>
                <header>
                    <texts>
                       <text>
```

```
                            <languagecode>en</languagecode>
                            <textmessage>Select which ${bonusCombo} you would like:<→
/textmessage>
                        </text>
                    </texts>
                </header>
                <display>
                    <texts>
                        <text>
                            <languagecode>en</language>
                            <textmessage>${bonusValue}</textmessage>
                        </text>
                    </texts>
                </display>
                <arguments>
                    <argument>
                        <key>bonusCombo</key>
                        <value>${bonusCombo}</value>
                    </argument>
                </arguments>
            </dynamicoptions>
        </instructions>
        </yes>
        <no>
        <instructions>
            <argument>
               <key>noBonus</key>
               <value>no bonus</value>
            </argument>
        </instructions>
        </no>
    </exists>
    <responsematching>
        <responses>
        <response>
            <texts>
                <text>
                   <languagecode>en</languagecode>
                   <textmessage>Transaction successful. You have bought ${amount} →
of Internet. You will also receive ${bonusValue}.</textmessage>
                </text>
            </texts>
        </response>
        <response>
            <texts>
                <text>
                   <languagecode>en</languagecode>
                   <textmessage>Transaction successful. You have bought ${amount} →
of Internet. There was ${noBonus} available today.</textmessage>
                </text>
            </texts>
        </response>
        <response>
            <texts>
                <text>
                   <languagecode>en</languagecode>
                   <textmessage>Transaction successful. You have bought ${amount} →
of Internet.</textmessage>
                </text>
            </texts>
        </response>
        </responses>
        <defaultresponse>
            <texts>
                <text>
                   <languagecode>en</languagecode>
                   <textmessage>Transaction successful.</textmessage>
                </text>
            </texts>
        </defaultresponse>
    </responsematching>
</instructions>
</op:journeydefinition>
```

# 4 Dynamic Journey Client Application

## 4.1 Front-end API

The Dynamic Journey Service offers an HTTP interface that client applications can use to invoke different operations. These operations allow the retrieval of details about available journeys, interaction with the service to start and continue journey execution on behalf of an account holder and exchange of user input and results of the journey execution.

The operations of the front-end API are described in the following sections.

### 4.1.1 Get Available Journeys Operation

This operation is used to retrieve a list of journeys that can be executed by an account holder. The returned response contains details of published journeys only.

If the `includeunpublishedonly` flag is set, journeys with `INITIAL` status will be returned, and this will be used to test non-published journeys.

If `includeunpublishedonly` flag is `true` then journeys other than published status for dedicated testers are returned.

Request: `getavailablejourneysrequest`

Table 19 getavailablejourneysrequest

| Name | Type | Mandatory | Description |
|---|---|---|---|
| category | JourneyCategory | no | Journey category |
| provideridentity | Identity | no | Provider identifier |
| includeunpublishedonly | boolean | no | Include only unpublished journeys. Default `false`. |
| listPageSize | ListPageSize | no | Maximum number of entries to be listed (default: 50) |
| listOffset | ListOffset | no | Offset at which entry to start the list (default: 1) |

```
<?xml version="1.0" encoding="UTF-8"?>
<ns0:getavailablejourneysrequest xmlns:ns0="http://www.ericsson.c →
om/em/djs/journey/v1_1/frontend/client/">
  <category>Buy</category>
  <includeunpublishedonly>true</includeunpublishedonly>
  <listpagesize>50</listpagesize>
  <listoffset>1</listoffset>
</ns0:getavailablejourneysrequest>
```

Response: `getavailablejourneysresponse`

Table 20 getavailablejourneysresponse

| Name | Type | Mandatory | Description |
|---|---|---|---|
| journeys | List<JourneyDetails> | yes | List of journeys |

Example 2

```
<?xml version="1.0" encoding="UTF-8"?><ns0:getavailablejourneysresponse xmlns:ns0=→
"http://www.ericsson.com/em/djs/journey/v1_1/frontend/client/">
  <journeys>
    <journey>
      <journeyidentifier>95a06df6-6400-487e-a7e3-8f30ff117c1e
      </journeyidentifier>
      <providername>ID:88811256/SP</providername>
      <provideridentity>ID:88811256/SP</provideridentity>
      <category>Buy</category>
      <status>INITIAL</status>
      <description>Under Test</description>
    </journey>
    <journey>
      <journeyidentifier>527397ac-18b0-4d18-999d-6b8ada18d8ab
      </journeyidentifier><providername>ID:123987654/SP</providername>
      <provideridentity>ID:123987654/SP</provideridentity>
      <category>Buy</category>
      <status>INITIAL</status>
      <description>Under Test</description>
    </journey>
  </journeys>
</ns0:getavailablejourneysresponse>
```

### 4.1.2 Initiate Journey Operation

This operation is used to start the execution of a journey. A new session that keeps track of the execution status will be created and a unique ID will be assigned to it. The new session ID is returned in response as well as the journey output. The journey output is used to request user input that is required to continue the journey.

The channel manager tester and provider tester can initiate journey in any state. Whereas regular user can only initiate journeys in published state.

Request: `initiatejourneyrequest`

Table 21 initiatejourneyrequest

| Name | Type | Mandatory | Description |
|---|---|---|---|
| journeyidentifier | JourneyIdentifier | yes | Journey identifier |
| languagecode | LanguageCode | yes | Language code |

```
<?xml version="1.0" encoding="UTF-8"?>
<ns0:initiatejourneyrequest xmlns:ns0="http://www.ericsson.com/em →
/djs/journey/v1_0/frontend/client/">
  <journeyidentifier>e307c767-4e2e-4796-829b-9088e320561a
  </journeyidentifier>
  <languagecode>en</languagecode>
</ns0:initiatejourneyrequest>
```

Request: `initiatejourneyresponse`

Table 22 initiatejourneyresponse

| Name | Type | Mandatory | Description |
|---|---|---|---|
| sessionid | SessionId | yes | Session id |
| sessionstate | SessionStatus | yes | Session state during the journey. |
| output | JourneyOutput | yes | List of options, a question or a final message that ends the journey |

Example 3

```
<?xml version="1.0" encoding="UTF-8"?>
<ns0:initiatejourneyresponse xmlns:ns0="http://www.ericsson.com/em/djs/journey/v1_→
0/frontend/client/">
  <sessionidentifier>7392fa00-6e0c-497c-8286-878a645709e0
  </sessionidentifier>
  <sessionstatus>INPUT</sessionstatus>
  <journeyoutput>
    <question>
      <key>amount</key>
      <question>Please enter an amount:</question>
      <validationpattern>\d+</validationpattern>
<isconfidential>true</isconfidential>
    </question>
  </journeyoutput>
</ns0:initiatejourneyresponse>
```

### 4.1.3 Continue Journey Operation

This operation is used to continue the execution of a journey that had been initiated before. The request for this operation contains the user input that had been requested in the response of the previous initiate or continue operation.

The output is then used to request additional user input which in turn shall be sent in proceeding continue journey operations until the session ends.

Request: `continuejourneyrequest`

Table 23 continuejourneyrequest

| Name | Type | Mandatory | Description |
|---|---|---|---|
| sessionid | SessionId | yes | Session id |
| input | JourneyInput | yes | A key-value pair with the user input |

```
<?xml version="1.0" encoding="UTF-8"?>
<ns0:continuejourneyrequest xmlns:ns0="http://www.ericsson.com/em →
/djs/journey/v1_0/frontend/client/">
  <sessionidentifier>7392fa00-6e0c-497c-8286-878a645709e0
  </sessionidentifier>
<journeyinput>
<argumentkey>amount</argumentkey><confidentialargumentvalue>10</c →
onfidentialargumentvalue></journeyinput>
</ns0:continuejourneyrequest>
```

Response: `continuejourneyresponse`

Table 24 continuejourneyresponse

| Name | Type | Mandatory | Description |
|---|---|---|---|
| sessionid | SessionId | yes | Session id |
| state | SessionStatus | yes | Session state during the journey. |
| output | JourneyOutput | yes | List of options, a question or a final message that ends the journey |

Example 4

```
<?xml version="1.0" encoding="UTF-8"?>
<ns0:continuejourneyresponse xmlns:ns0="http://www.ericsson.com/em/djs/journey/v1_0/frontend/client/">
  <sessionstatus>DONE</sessionstatus>
  <journeyoutput>
    <response>You have entered the amount of 10.</response>
  </journeyoutput>
</ns0:continuejourneyresponse>
```

### 4.1.4 Journey Output

The journey output contains either a response to be presented to the account holder as result of the journey execution or request for user input.

Table 25 Elements

| Name | Type | M/O | Description |
|---|---|---|---|
| response(1) | usertext | O | A response to be presented to the account holder. |
| question(1) | questionoutput | O | A question to be presented to the account holder and for which the answer needs to be collected. |
| options(1) | optionsoutput | O | A set of options to be presented to the account holder. The chosen option needs to be collected. |

(1) response, question and options are mutually exclusive

#### 4.1.4.1 questionoutput

A question to be presented to the account holder.

Table 26 questionoutput

| Name | Type | M/O | Description |
|---|---|---|---|
| question | usertext | M | The question to be presented to the account holder. |
| errormessage | usertext | O | An error message to be presented to the account holder. The error message is given if this is a retry because validation of the previous answer failed. |
| validationpattern | regularexpression | O | A validation pattern to be applied on the answer. Client applications with the corresponding capabilities can use this pattern for client-side validation of the given answer. |

| Name | Type | M/O | Description |
|---|---|---|---|
| key | argumentkey | M | The key of the argument that should carry the answer to the question when sent in `JourneyInput`. |
| isconfidential | boolean | M | Tells if the journey input is confidential or not.<br>If true, the `confidentialargumentvalue` should be used in the input for the next `Continuejourney` request.<br>If false, `argumentvalue` should be used.<br>Note: It is important to use the correct attribute in the input so confidential data is not logged. |

#### 4.1.4.2 optionoutput

A set of options to be presented to the account holder.

Table 27 optionoutput

| Name | Type | M/O | Description |
|---|---|---|---|
| header | usertext | O | An optional header to be displayed on top of the list of options. |
| footer | usertext | O | An optional footer to be displayed at the bottom of the list of options. |
| options | list<argument> | M | The list of available options. The option is identified by the argument key. Argument value contains a description of the option. It is depending on the client application whether both key and value are presented to the account holder or only the value. For the chosen option, both argument key and value must be sent in `JourneyInput`. |

### 4.1.5 Journey Input

The journey input contains the requested user input collected from the account holder in form of answers to questions or the option that was chosen from a list of available options.

Table 28 Elements

| Name | Type | Description |
|---|---|---|
| argumentkey | argumentkey | An argument key.<br>In case the argument contains the answer to a question, the argument key is the same key that was received along with the question in the `JourneyOutput`.<br>In case the argument contains a chosen option, then argument key and value reflect those of the chosen option as received in `JourneyOutput`. |
| argumentvalue(1) | usertext | A text format.<br>In case the argument contains the answer to a question, the argument value will then contain the answer to that question as given by the account holder.<br>In case the argument contains a chosen option, then argument key and value reflect those of the chosen option as received in `JourneyOutput`. |

| Name | Type | Description |
|---|---|---|
| | | Note: When answering questions, this should be used when the question is not confidential, otherwise data will not be logged. |
| confidentialargumentvalue(1) | confidentialargumentvalue | A text format.<br>In case the argument contains the answer to a question, the argument value will then contain the answer to that question as given by the account holder.<br>In case the argument contains a chosen option, then argument key and value reflect those of the chosen option as received in `JourneyOutput`.<br>Note: When answering questions, this should be used when the question is confidential, otherwise confidential data will be logged. |

(1) argumentvalue and confidentialargumentvalue are mutually exclusive.

### 4.1.6 Front-end API Data Type Definitions

ConfidentialArgumentValue

A text that contains a confidential input value.

#### Identity

The identity of a provider or account holder.

Required format: `ID:<identity>/<identity type>`

Example: `ID:99887766/MSISDN`

#### JourneyCategory

The category of journey. Categories are assigned by the channel manager.

Example: Car Rental

#### JourneyDescription

A description of the journey.

Maximum length: 256 characters.

#### JourneyDetails

Detailed information about an available journey.

Table 29 JourneyDetails

| Name | Type | Description |
|---|---|---|
| journeyidentifier | JourneyIdentifier | The unique identifier of the journey. |
| providername | ProviderName | The name of the provider that offers the journey. |

| Name | Type | Description |
|---|---|---|
| provideridentity | Identity | The identifier of the provider that offers the journey. |
| category | JourneyCategory | The category to which the journey belongs. |
| status | JourneyStatus | The status of the journey. |
| description | JourneyDescription | A description of the journey. |

### JourneyIdentifier

A generated unique identifier for a journey.

Required format: `UUID`.

Example: `3fcd6dce-2bc9-4bc6-ac38-188e48d267d1`

### JourneyStatus

The status of the journey.

Table 30 JourneyStatus

| Status | Description |
|---|---|
| INITIAL | The journey definition has been uploaded to EWP, but not been verified yet. It is only available for tests. |
| VERIFIED | The uploaded journey definition is verified. It is only available for tests. |
| ASSIGNED | The journey definition is assigned. It is only available for tests. |
| REJECTED | The journey definition is rejected. It is only available for tests. |
| REPLACED | The journey definition is replaced. It is available for the provider to be published. |
| APPROVED | The journey definition is approved. It is available for the provider to be published. |
| PUBLISHED | The journey definition has been published by the provider and is available for execution. |

### LanguageCode

Two- or three-letter language codes based on ISO 639-1 or ISO 639-3 standard with the priority of the two-letter notation (ISO 639-1). see reference *639-2 Language Code List*, accessible at http://www.loc.gov/standards/iso639-2/php/code_list.php.

Example: `EN`

### ListOffset

Offset at which entry to start the list.

Minimum value: 1. Default value: 1

### ListPageSize

Maximum number of entries to be returned by the list operation.

Value range: 1 – 50. Default value: 50.

### ProviderName

A name of a journey provider displayed to end-users.

Maximum length: 64 characters.

### SessionId

A generated unique identifier for a session.

Required format: `UUID`.

Example: `3fcd6dce-2bc9-4bc6-ac38-188e48d267d1`

### SessionStatus

The status of a journey session.

Must be one of the pre-defined values:

Table 31 SessionStatus

| Status | Description |
|---|---|
| CONTINUE | Last user input to the session had been accepted. Session can be continued. |
| DONE | Last user input to the session had been accepted. The journey execution has been finished and the session is terminated. |
| INPUT | User input is required to continue journey execution. |
| ERROR | Last user input to the session had been rejected. An error has occurred, and the session has been terminated. |
| EXPIRED | The session has been expired due to inactivity. |

## 4.2 Client Application Workflow

The picture below depicts the interaction between account holder, client application and Dynamic Journey Service during journey execution.

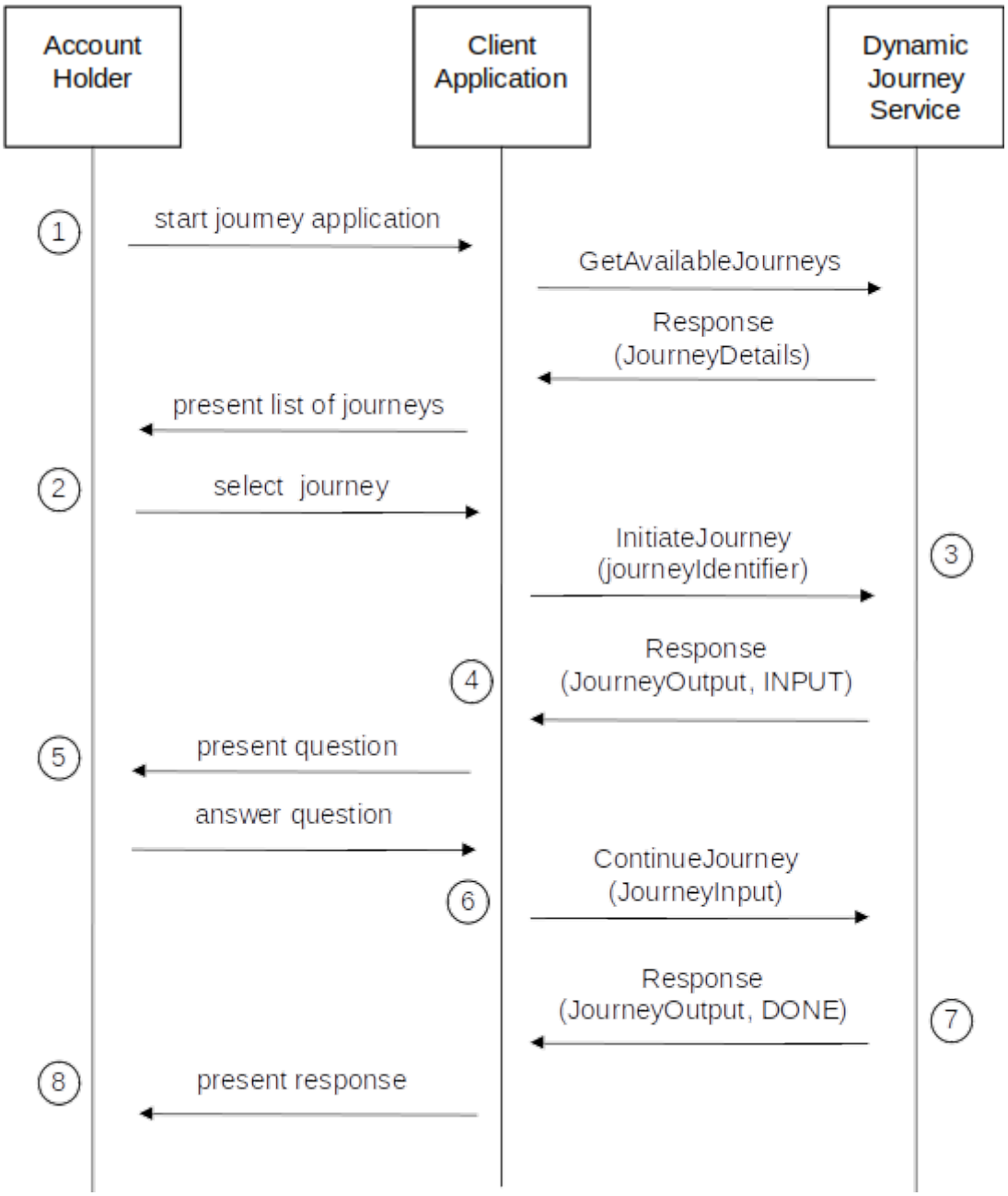


Figure 1 Workflow

1. A client application should be able to present the journeys that are available for execution to the account holder. The `GetAvailableJourneys` operation is used by the client to retrieve information about the journeys. Not all details received are relevant for the account holder, but the following data should be presented to the account holder in a suitable format: provider name, journey category and journey description.

2. When the account holder selects a journey for execution, the client application starts execution of the journey with the `InitiateJourney` operation. The journey to execute is identified by the journey identifier that had been received as part of the journey details from a previous `GetAvailableJourneys` operation. The client must also provide the account holders identity and the required language when initiating the journey execution.

3. While the client application is waiting for an answer to its request, the Dynamic Journey Service (DJS) will create a new session and start execution of the requested journey. When the journey execution comes to a point where user input is required, the DJS will compose a response that is sent to the client. The response contains a session identifier, the status of the session and a journey output that describes the requested user input.

4. The client application needs to keep track of the session identifier because it is used in all subsequent requests to identify the ongoing session for the journey execution. The session status instructs the client application about how to proceed with the journey execution. If no error occurred and the journey execution neither finished, the journey status will be set to `INPUT`.

5. The client application continues by gathering the required user input from the account holder. The information received in `JourneyOutput` should be rendered in a suitable format and presented to the account holder.

6. Once the user input is collected, the client application should continue journey execution with the `ContinueJourney` operation. The previously received session identifier is used to identify the journey and the user input is sent as `JourneyInput` to the Dynamic Journey Service.

7. The DJS resumes the journey execution with the received user input and composes a new response when either additional user input is required, the journey execution finished, or an error occurred.

8. Steps 5 to 7 are repeated until session status `DONE` or `ERROR` are received in response. In this case the session cannot be continued, any attempt to do this would result in an error. When the session reaches status `DONE` or `ERROR`, the client application should render the corresponding result or error message and present it to the account holder.

# 5 Dynamic Journey Support in Partner System

The Dynamic Journey Service (DJS) offers the possibility to interact with a partner system through the Partner Gateway. During journey execution, the DJS can send requests to the partner gateway as specified in the journey definition. The partner gateway will proxy these requests and corresponding responses to/ from partner system.

This enables partners to apply dynamic behavior to the journey execution for example based on user preferences.

The partner system needs to support two operations that could be invoked through Partner Gateway. These operations are described in this chapter.

## 5.1 Default Journey Session Arguments

In order to help the provider to identify if the journey is being tested, the following data is added when creating a journey session.

The following parameters will be added to the arguments cache in the session:

- `ACCOUNT_HOLDER_MSISDN` - The MSISDN of the account holder who initiated the journey
- `ACCOUNT_HOLDER_EMAIL` – The email of the account holder who initiated the journey
- `JOURNEY_CURRENT_STATUS`- The current journey status, being it `INITIAL`, `VERIFIED`, `ASSIGNED`, `APPROVED`, `PUBLISHED`.

## 5.2 Dynamic Journey Support in Partner System Operations

The `GetDynamicArguments` and `GetDynamicOptions` operation both have the same request parameters.

**Note:** Both the request and response for the `GetDynamicArguments` and `GetDynamicOptions` operations are using **JSON** as their data format when used in the integration to the Partner System. This is similar to all operations in the Partner Gateway, but not similar to other operations in EWP which are using XML.

In the respective request, the partner system may receive a set of arguments as input. It is specified in the journey definition which arguments are to be sent, if any. These arguments can for example carry answers to previous questions and their use depends on the partner's business case.

The session identifier can be used by the partner system to keep track on different sessions and to correlate several requests that might be sent during the same session.

### 5.2.1 GetDynamicArguments

This operation allows the partner to enrich the account holder's session data with a set of arguments (key value pairs). These arguments can then in the journey be used to branch on execution flows or to relay information to the account holder.

```
HTTP Method: POST

Data format: JSON

URI: /{baseContext}/getdynamicarguments
```

#### Parameters

**getdynamicargumentsrequest:**

Table 32 getdynamicargumentsrequest

| Name | Type | Mandatory | Description |
|---|---|---|---|
| arguments | list<argument> | no | A list of arguments (key value pairs) that can be used for evaluation |
| languageCode | LanguageCode | yes | Specifies the language used for the journey execution |
| sessionIdentifier | SessionIdentifier | yes | Identifies the session for journey execution on behalf on an account holder |
| journeyIdentifier | JourneyIdentifier | yes | Identifies the journey definition. |

**getdynamicargumentsresponse:**

Table 33 getdynamicargumentsresponse

| Name | Type | Mandatory | Description |
|---|---|---|---|
| arguments | list<argument> | yes | A list of arguments (key value pairs) that should be added to the session data |

### 5.2.2 GetDynamicOptions

This operation allows the partner to customize options that should be presented individually for each account holder.

The response contains a list of lists of arguments (key value pairs).

The overall list describes the entire set of options that should be presented to the account holder.

Each list of arguments within the overall list represents one option. The different arguments of such list are used to render the option that should be presented to the account holder. The pattern for option rendering is specified in the journey definition, therefore the returned arguments must match the expected arguments used in the journey definition.

```
HTTP Method: POST

Data format: JSON

URI: /{baseContext}/getdynamicoptions
```

### Parameters

#### getdynamicoptionsrequest:

Table 34 getdynamicoptionsrequest

| Name | Type | Mandatory | Description |
|---|---|---|---|
| arguments | list<argument> | no | A list of arguments (key value pairs) that can be used for evaluation |
| languageCode | LanguageCode | yes | Specifies the language used for the journey execution |
| sessionIdentifier | SessionIdentifier | yes | Identifies the session for journey execution on behalf on an account holder |
| journeyIdentifier | JourneyIdentifier | yes | Identifies the journey definition. |

#### getdynamicoptionsresponse:

Table 35 getdynamicoptionsresponse

| Name | Type | Mandatory | Description |
|---|---|---|---|
| argumentsList | list<list<argument>> | yes | A two-dimensional list of arguments (key value pairs) that should be used to render a list of options to the account holder |

## 5.2.3 Data Type Definitions

See data type definitions in Dynamic Journey Client Application on page 23.

# 6 Example Use Case

This chapter describes the use case of internet service provider (service provider) which offers its services to the customers (account holders) with help of the Dynamic Journey Service.

## 6.1 Business Scenario Description

The following business scenario is realized by the journey definition that the service provider has been uploaded to EWP.

Customers can choose to purchase different data plans. Once they made a choice, they are asked for their PIN code. The PIN code and chosen amount to buy for are transmitted to the service provider. Some customers may be eligible for different bonus and in case they are, they may choose which bonus they want to use for their purchase. The purchase is then concluded with a confirmation message to the customer. The message will look different based on the amount that had been spent and based on the bonus granted.

## 6.2 Journey Execution Sequence

The different steps of the journey execution are described below. They involve all possible interfaces of the Dynamic Journey Service and the respective actors.

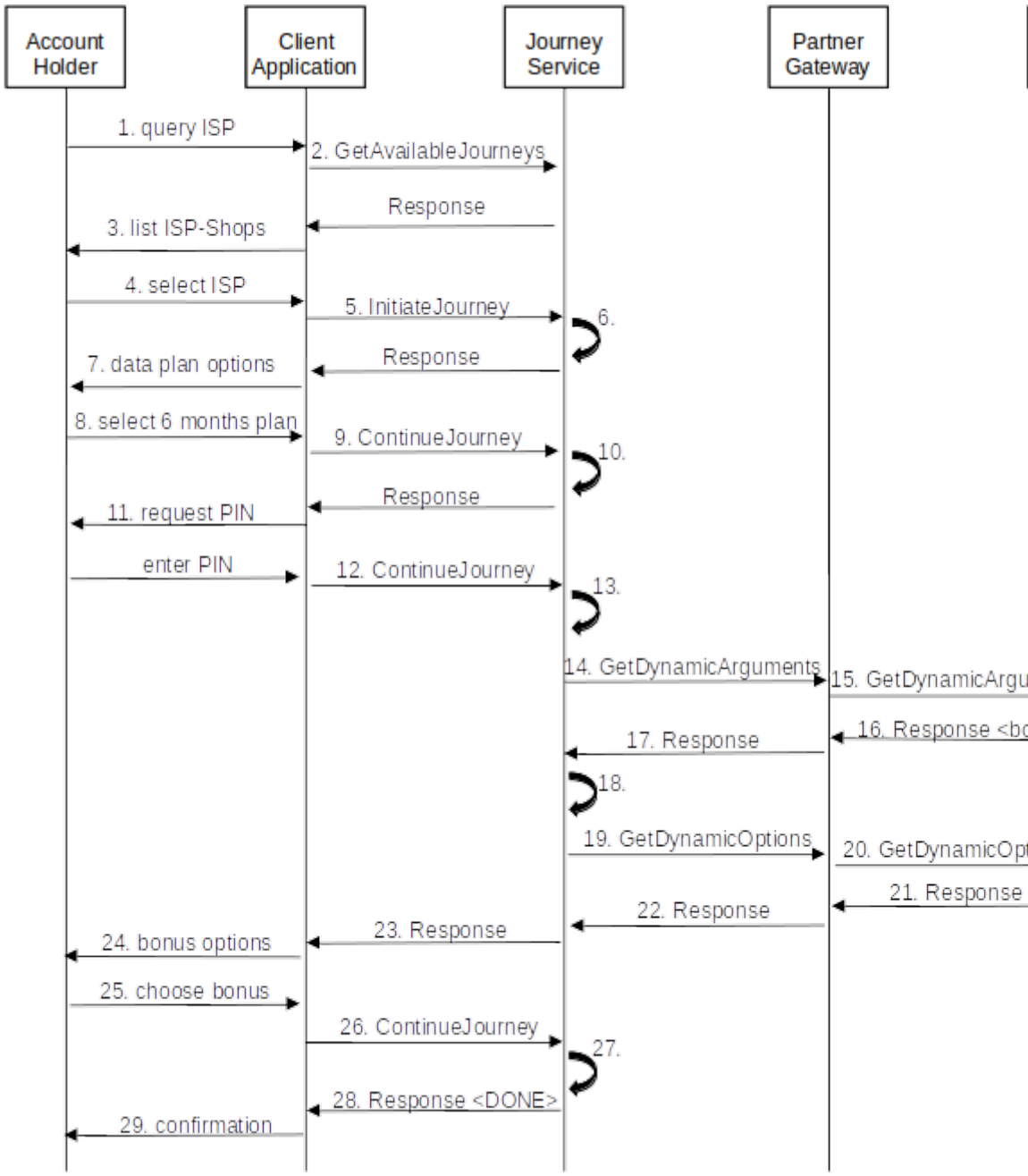


Figure 2

1. An account holder wants to buy a data plan and queries the application for available internet service providers.

2. The client application sends a `GetAvailableJourneys` request to the Dynamic Journey Service (DJS) and sets the category field in the request to `ISP Shop`to filter the journeys for shops of internet service providers only.

3. The DJS returns a list of all journeys matching the criteria category `ISP Shop`. This list is presented to the account holder as available ISP-shops.

4. The account holder selects one of the listed ISP-shops that he wants to visit.

5. The client application sends an `InitiateJourneyRequest` to DJS. The selected journey will be identified with the journey identifier that was given in the list of journeys.

6. The DJS will create a new session for execution of the journey on behalf of the account holder. Execution of the journey is started immediately with the first instruction which is of type options.

7. These options are sent in `InitiateJourneyResponse` to the client application which presents them as available data plans to the account holder.

8. The account holder selects the 6 months data plan.

9. The client application sends a `ContinueJourneyRequest` to DJS and gives the made choice with `key=amount, value=50` as input to the journey execution.

10. DJS resumes execution of the journey, adds the received input to the session data and continues with the next instruction of type question.

11. The question is sent in `ContinueJourneyResponse` to the client application which presents it to the account holder asking for a PIN code.

12. The account holder enters his PIN code and the client application sends a new `ContinueJourneyRequest` to DJS this time with the PIN code in input data.

13. DJS adds the PIN code to its session data. Since the question is marked as confidential in the journey definition, DJS does not write the given answer, the PIN code, to any log file where it might become visible to third parties.

14. DJS executes the next instruction of type dynamic arguments. A `GetDynamicArgumentsRequest` is sent to the Partner Gateway (PG). The request contains the collected user input and URL as specified in the instruction along with session information.

15. PG proxies the request to the partner system of the ISP.

16. The partner system does perform the purchase and determines that the account holder is eligible for bonus. An argument with key `bonusCombo` is sent to PG in `GetDynamicArgumentsResponse`.

17. PG proxies the response to DJS.

18. DJS updates the session data with argument received in response and executes the next instruction which is of type exists. Exists checks whether an argument with key `bonusCombo` exists in the session data. Since

bonusCombo has been received in the previous response, the argument exists and therefore DJS executes the instruction of the yesbranch of the exists instruction. This is a dynamic options instruction.

19. DJS sends a GetDynamicOptionsRequest to the PG. The request contains the bonusCombo argument and URL as specified in the instruction along with session information.

20. PG proxies the request to the partner system of the ISP.

21. The partner system evaluates the available bonus and send a list of available bonus values in GetDynamicOptionsResponse to PG.

22. PG proxies the response to DJS.

23. DJS uses the received bonus values to render a list of options according to the pattern given in display of the dynamic options instruction. The list of options is sent in ContinueJourneyResponse to the client applications.

24. The client application presents the available bonus options to the account holder.

25. The account holder chooses the bonus he wants to apply.

26. The client application sends a ContinueJourneyRequest to DJS with the chosen bonus as input data.

27. DJS adds the received bonus value to the session data and resumes execution of the journey with the next instruction of type responsematching. This instruction does specify different response options which each require different arguments to be present in the session data. DJS evaluates the list of possible responses top to bottom and concludes that all arguments of the first response can be matched.

28. The matched response is sent in ContinueJourneyResponse to the client application. Since this is a final response, DJS se the status of the session to DONE, indicating to the client application that journey execution has been terminated and cannot be continued.

29. The client application presents the response to the account holder.

## 6.3 Journey Definition

The use case is realized with the following journey definition.

Example 5

```
<?xml version="1.0" encoding="UTF-8"?>
<op:journeydefinition xmlns:op="http://www.ericsson.com/em/djs/journey/v1_0/common →
/" xmlns:xs="http://www.w3.org/2001/XMLSchema">
<instructions>
    <options>
        <header>
            <texts>
               <text>
```

```
                    <languagecode>en</languagecode>
                    <textmessage>Select internet offer</textmessage>
                 </text>
              </texts>
          </header>
          <optionslist>
          <option>
              <display>
                  <texts>
                     <text>
                        <languagecode>en</languagecode>
                        <textmessage>10 EUR (1 Month)</textmessage>
                     </text>
                  </texts>
              </display>
              <instructions>
               <argument>
                  <key>amount</key>
                  <value>10</value>
               </argument>
             </instructions>
          </option>
          <option>
              <display>
                  <texts>
                     <text>
                        <languagecode>en</languagecode>
                        <textmessage>50 EUR (6 Months)</textmessage>
                     </text>
                  </texts>
              </display>
              <instructions>
                <argument>
                    <key>amount</key>
                    <value>50</value>
                </argument>
              </instructions>
          </option>
          </optionslist>
      </options>
      <question>
           <key>pin</key>
          <confidential>true</confidential>
          <display>
              <texts>
                 <text>
                    <languagecode>en</languagecode>
                    <textmessage>Enter PIN to confirm refill of Internet for ${amoun→
t} EUR</textmessage>
                 </text>
              </texts>
          </display>
      </question>
      <dynamicarguments>
          <url>https://provider.com/buyinternet</url>
          <arguments>
              <argument>
                  <key>amount</key>
                  <value>${amount}</value>
              </argument>
              <argument>
                  <key>pin</key>
                  <value>${pin}</value>
              </argument>
          </arguments>
      </dynamicarguments>
      <exists>
          <key>bonusCombo</key>
          <yes>
          <instructions>
              <dynamicoptions>
                  <url>https://provider.com/buybonus</url>
                  <header>
                      <texts>
                         <text>
                           <languagecode>en</language>
                           <textmessage>Select which ${bonusCombo} you would like:</→
textmessage>
                         </text>
                     </texts>
```

```
                </header>
                <display>
                    <texts>
                        <text>
                           <languagecode>en</languagecode>
                           <textmessage>${bonusValue}</textmessage>
                        </text>
                    </texts>
                </display>
                <arguments>
                    <argument>
                       <key>bonusCombo</key>
                       <value>${bonusCombo}</value>
                    </argument>
                </arguments>
            </dynamicoptions>
        </instructions>
        </yes>
        <no>
        <instructions>
            <argument>
               <key>noBonus</key>
               <value>no bonus</value>
            </argument>
        </instructions>
        </no>
    </exists>
    <responsematching>
        <responses>
        <response>
            <texts>
                <text>
                   <languagecode>en</languagecode>
                   <textmessage>Transaction successful. You have bought ${amount} →
of Internet. You will also receive ${bonusValue}.</textmessage>
                </text>
            </texts>
        </response>
        <response>
            <texts>
                <text>
                   <languagecode>en</languagecode>
                   <textmessage>Transaction successful. You have bought ${amount} →
of Internet. There was ${noBonus} available today.</textmessage>
                </text>
            </texts>
        </response>
        <response>
            <texts>
                <text>
                   <languagecode>en</languagecode>
                   <textmessage>Transaction successful. You have bought ${amount} →
of Internet.</textmessage>
                </text>
            </texts>
        </response>
        </responses>
        <defaultresponse>
            <texts>
                <text>
                   <languagecode>en</languagecode>
                   <textmessage>Transaction successful.</textmessage>
                </text>
            </texts>
        </defaultresponse>
    </responsematching>
</instructions>
</op:journeydefinition>
```

# 7 Dynamic Journey Service Command Line Interface

The Dynamic Journey Service offers a command line interface (CLI) that can be used for maintenance when no front-end application is available.

The CLI is intended for system administrator and requires (in difference to the front-end API) admin privileges for all commands.

## General

Application name: `djs`

Command group: `dynamicjourney`

## Delete

**Note:** It is not possible to delete published journeys. Revoke the published journey first, and then delete it.

Command: `delete`

Description: Deleting a journey.

Table 36 Delete Parameters

| Name | Constraints | Mandatory | Description |
|---|---|---|---|
| journeyidentifier | IsUUID | yes | The identifier of the journey |

Example 6

```
lwac-cli -n djs dynamicjourney delete --journeyidentifier 3fd4ec7f-ade4-4125-bf5 →
8-b62905590905
```

# 8 Dynamic Journey Service Error Codes

This is the list of error codes thrown by the dynamic journey service and their descriptions.

Table 37 Error Codes

| Error Code | Description |
|---|---|
| ACCOUNT_HOLDER_NOT_FOUND | Account holder identity not found |
| ARGUMENT_NOT_FOUND | Argument not found in session |
| INPUT_NOT_VALID | Journey input is not valid. |
| JOURNEY_ALREADY_EXISTS | Journey for requested provider and category already exists. |
| JOURNEY_ALREADY_PUBLISHED | Journey has already been published. |
| JOURNEY_CATEGORY_ALREADY_EXISTS | Journey category with requested name already exists. |
| JOURNEY_CATEGORY_NOT_FOUND | Journey category not found |
| JOURNEY_FOR_CATEGORY_EXISTS | Journey cannot be deleted because journey for given category exists |
| JOURNEY_MAX_RETRIES_EXCEEDED | The maximum amount of allowed retries has been exceeded |
| JOURNEY_NOT_FOUND | Journey with requested identifier was not found. |
| JOURNEY_SESSION_ALREADY_FINALIZED | Journey session has been terminated or is in a final state. |
| JOURNEY_SESSION_EXPIRED | Journey session has expired. |
| JOURNEY_SESSION_NOT_FOUND | Journey session with requested session identifier was not found. |
| JOURNEY_STATUS_TRANSITION_NOT_ALLOWED | Journey status transition not allowed |
| LANGUAGE_NOT_FOUND | Journey is not available in the requested language |
| NO_AVAILABLE_OPTIONS | No options available. |
| PROVIDER_JOURNEY_TESTER_ALREADY_EXISTS | Tester already registered to a provider |
| PROVIDER_JOURNEY_TESTER_NOT_FOUND | Provider journey tester not found |
| PROVIDER_TESTER_IDENTITY_MATCH | Provider identity and account holder identity are the same |
| PUBLISHED_JOURNEY_DELETION_NOT_ALLOWED | PUBLISHED Journey can not be deleted |