Telegram Road

Management instructions

Rel. 1.3

Introduction

Telegram Road is a platform for the development of **chatbots**, i.e. interactive applications in which the user chats with a server (a *bot*) through a messaging application, usually on a smartphone. The messaging application here is Telegram Messenger (from now on, **Telegram**), which is a wide-spread app available on many platforms.

The description of Telegram is outside the goals of this document, and the user should better refer to https://telegram.org/

Purpose of the document

This document describes how to write and manage a chatbot (from now on, *app*), through the platform Telegram Road.

It will be given for granted that the user has already created one app of his/her own. This is described in the document:

Telegram Road – description

The creation a new app will have been most likely performed on a smartphone, interacting with a couple of other bots :

- **BotFather**, the standard Telegram bot for creating and managing other Telegram bots
- **Telegram Road** bot, a bot which activates, registers and links the newborn app to the application itself. This application, which runs on the Telegram Road server, is what actually we are going to describe from now on.

Given the app, the standard user will interact through Telegram only with it. That's why Telegram Road is a powerful tool to develop applications without requiring the end-user to install anything new on his/her smartphone. The only required resident app is Telegram , nothing else.

Development environment

The suggested development tool is **not** a chatbot, but a traditional form-based application, run on a conventional **browser**. The reason for that is that a chatbot application would result too **slow**, from the standpoint of the user experience, due to the fact that a chatbot needs to split every request for information in several single questions. As the number of pieces of information to be dealt with is large, to overall process for a chatbot-based application would result boring.

Anyway, a browser is a tool that one finds not only on a desktop computer, but also on a smartphone, or any other mobile device, such as a tablet. The choice to use a smartphone-based browser, or a tablet browser, or a desktop browser, is up to the developer, based on different perspectives:

- the wider the screen, the richer the information available at once
- a desktop-based browser, such as Chrome, Firefox, Edge, Internet Explorer (Windows) or Chrome, Safari (Mac) is richer of features than the corrispondant on a mobile platform
- the use of a mouse (on desktop computer) can give further possibilities that a touch screen device can not

That said, it is possible to use either environment mixed together. In other words, one can start developing an app on a smartphone, perform the hard work on a desktop, and possibly to give the last cuts again on his mobile device. The same app will be always available on either platform.

App lifetime

Before entering the development passages , let's define two keypoints in an app lifetime:

- **Design time** is the time in which the manager of the app designs and configures it. Let's recall that the manager (*the person this document is aimed to),* is the one who defines the blocks (nodes, arcs) and their behavior, that the end user will use in run time
- **Run time** is when the end user opens, uses and interacts with the app. A good manager is supposed to bear in mind the end user's interest and ease of operation.

  These two terms will be referred to widely in this document .

Types of apps

Different types of apps are available in the Telegram Road environment. They scale up in power and complexity.

Static apps

Static apps are completely defined in design time: their behavior is completely defined by the app structure designed by the manager, both in the questions asked to the end user and in the answers he/she will be allowed to choose from.

The next chapter of this document will be devoted to the static app development only.

Static apps are the only available in the free version of Telegram Road. To develop a dynamic application, the manager will have to subscribe the Premium option of this platform.

Dynamic app, static database

While a static app has always the same behavior, as the data the end user can move in are fixed, a dynamic app lets the user enter, search and find proprietary data, in order to build his/her own database. With a dynamic app, the end user has a personal version of the app, allowing the storage of own data.

For example, an end user may want to build a Contacts database in which to store his buddies and their data. The data he will enter will be name, surname, etc (text) but also birthdate (date) or phone number (number). The structure of the data are defined by the manager in **design time**; the data themselves are chosen by the end user in **run time**.

Also the typology of the data are predefined : the end user will not be able to choose. Currently, the only families of available data are **Contacts** and **Clipboards**, short texts to remeber and retrieve (like a post-it).

A following section on Node Q will be dedicated to dynamic app development and usage.

Dynamic app, dynamic database

A dynamic app with a dynamic database lets the end-user choose not only his/her own data, but also the typology of data to deal with. One may wish to store a directory of books, or music, or maybe social events, and so on.

And since all those things are different, one will be allowed to define **in run time** the structure of the data, that is the attributes of the objects, i.e the features of the objects which will completely define them.

The operation is supposed to be much more articulated, and awareness of the structure of the data is required to the end-user as well (not only to the manager).

One single type of object will be allowed in any single app. In the case, a manager will deploy different apps on different topics.

A description of this sort of application will be given in the following chapter about Node X.

Application graph

The structure of a Telegram Road application is best described through a **graph**.

The nodes of the graph represent the steps the application is performing. In every time of its run, an application is on one and one only node of the graph.

NOTE: The best description of a Telegram Road graph is a **flowchart**, which is as a matter of fact a description of a flow of the commands being given to an application.

Some of the features of a full-blown IT flowchart are not used; the function of the nodes used within Telegram Road are described in the following paragraph.

Nodes

Nodes are the basic objects Telegram Road is based on. Any node bears a number of fields, which share the same name but can have different meanings, so they will be explained later on.

Node properties

Here are the fields of a generic node :

- Question name: a shorthand name for the question (node). Best to use concise and "speaking" names.
- Type : can be either “S”, “I”, “Q” “R” or “X” (see below)
- Presentation: a short introduction before the question itself.
- Image: an optional picture to be offered before the question. Supported formats are JPEG , PNG.
- Question: the actual text of the question offered to the end user
- Answers: a list of answers linked to the question.
- Field: This value can be used to store , under the name in it, a variable asked to the user (questions of type S, I or Q). If the value is “$CLR”, the effect is to clear any previously stored value.
- Score : a score for the question, to be added independently of the scores given for the answers
- NCol: the number of columns on which the button pad for answers will be lined up.
- Initial: check this checkbox to mark question as a Start. Only one question should be marked as Initial, otherwise un undetermined behavior may occur
- Final: a final question is a deadline for an application run. Differently from Initial question, more than one question can be marked as Final. A final question shouldn't have answers: it's supposed to be a "farewell" question.
- Next : a pointer to the question to be lead to on default, when no valid answer is given.
- Open : check this checkbox if the question has to be offered to the end-user as “open”, that is it will offer a “missing answer” option. In such case, the user will be able to expand the graph from that node, adding an answer and eventually one more following question. This option is described in the following chapter “Expanding an app”.
- Public : check this checkbox if the question has to be offered to anybody visiting the app. If not checked, it will be visible only to the owner (the creator) of the question. A question created through expansion of a graph (see previous point) is born with

```
Public=false, as it should be approved by an administrator before
being released to the public. This prevents  unappropriate material
to be offered to the public.
```

Nodes can be of either of the next different types.

Node S

A node of type “S”, where “S” stands for “selection”, is the primary kind of the Telegram Road nodes. **A static application is only formed of S nodes**.

As a static application implements a questions and answers structure, a node of type S acts always as a **question**, whereas the arc connecting the nodes are the **answers** between them.

That's why in this document the term “node” and the term “question” are used with the same meaning. Questions are asked for other types of nodes, too.

The graphical flowchart box representing an S node is the classical “Control” box :

with one input and as many outputs as the available answers for that question. The question name is written within or just aside the box.

Each answer text is aligned on the arc leading it.

Node I

A node of type “I”, which stands for “input”, allows the app manager to ask the end user to input a value, to be used in the following passages.

The given value is stored in a variable, whose name has to be provided in the “field” part of the node.

As the answer given by the end user may have no other function, a node of type “I” can have no answer associated; if any answer is provided, it will be discarded The node the user will be driven to after an I node is the one pointed by “Next” .

The grapical symbol used for a node of type I is the Input flowchart box :

with just one input and one output arc.

Node Q

A node of type Q (“query”) allows the app manager to perform a SQL query .

The structure (the”schema”) of the database is supposed to be well known to the App manager when using a Q node. **Actually, a Q node is the cornerstone of a dynamic application.**

The query text will have to be written in the “Question” part of a node. As a standard SQL query, it

will have to be finalized with a semicolon character ";"

The query may refer to one or more variables already inserted by preceding "I" or "S" nodes.

The query will also refer to a database which is pointed by the "datasource" field of the App property set. Since this value may be one only, it is possible to query one and one only datasource for each App.

The typical use for a Q node is to run a SELECT SQL statement, the purpose of which is to extract values from the database, which will be offered to the end user as the available answers.

The chosen value will be stored in a variable, whose name has to be provided (by the App manager) in the "field" part of the node.

Along with the extracted values, the manager may offer also static answers, such as "help" or "exit", to be used as escapes or default values (in the case of null set extracted by the query, for example). Such static answers will be evaluated in runtime before any other answer.

NOTES

1. one single field is allowed to be extracted
2. the actual SQL query has to be placed in the field "Question" of the node. That means that for a Q node only a presentation will be offered to the user , but NO question.
3. since in a generic database field names can be whatever, it is MANDATORY to alias extracted field names with the alias "**val**". That is, if the manager wants to present as possible answers the values of the "Jobname" field of a given table "Jobs", a proper query would be

   SELECT Jobname AS val FROM Jobs ORDER BY Jobname ;

Forgetting the alias AS would result an error such as

The column "Val" could not be found in table "Jobs".

A Q node may contain not only SELECT statements, but also INSERT, UPDATE or DELETE statements. As a matter of fact, the full set of SQL ANSI is provided. That gives the manager a great power to handle the database pointed to in the application.

The MySQL 5 database engine , for example, supports:

- multiple level JOINS
- Inner queries

The SQL query may refer values that have been inserted before through an "I" or "S" node, or even "Q" nodes. Such values are stored in a special purpose structure, called "CurrentQuery", which will be described later.

If a variable has been saved with a given name (e.g. var), it will be referenced by the query enclosing it with $, i.e var. For istance, if a value has been saved with a name "name", a valid query using it might be :

SELECT address AS val FROM friends WHERE friends.name = '$name$'

For a description of an App properties, see following chapter "APP Properties"

The graphical flowchart box used for a Q type node is

Node X

The combination of “S”, “I” and “Q” nodes gives the App manager a powerful toolset to build static and dynamic applications,

The limit of a Q node is the capability to run one single SQL statement. Every database operation which can be resolved in one single SQL statement can be handled by a Q node.

Not all situations, unfortunately, can be dealt with in such a way. If more database operations have to be performed, one can always chain more Q nodes, but the effect for the end user would be being asked more “questions” which could be boring and maybe not understandable, too.

The X node addresses this issue, giving the manager the capability to eXecute an external procedure. The name of the procedure has to be provided in the “field” part of the Node.

All input values to be provided to the procedure are to be set in the CurrentQuery structure (see below) through proper usage of S, I and Q nodes.

All ouput values will be made available by the procedure in the same CurrentQuery structure .

The side effect of this powerful tool is that for a X node the App manager is supposed to own programming skills, beyond the plain SQL skills required by the Q node.

That is why the X node use is currently limited to a small number of applications, for which a set of Coldfusion functions have been set up [1].

All of these applications belong to the the family previously named “Dynamic app, dynamic database”. The use of such functions and apps is covered under a separate document.

The graphical box used for a X type node is the following:

Node type summary

Here's a summary for each node input, output and behaviour.

Tipo	Output	Input	Next	Action
S	Provided answers	chosen (single) answer	Question pointed by answer	Can save selected in “field”
I	“Insert a value”	Inserted value	Next	save selected in “field”
Q	Selection from query+answers	Single selection	Next	Execute query, save selected in “field”
X	Selection from	Single	Question	Execute

1Coldfusion is the language Telegram Road is based on

	query+answers	selection	pointed by answer	external procedure
R	reserved	reserved	reserved	reserved

APP Properties

Every App, when generated and registered in Telegram Road, comes up with the following properties:

- App name : a generic name, which CAN be the same which has been registered in Telegram. Let's recall that the Telegram bot (created in Telegram through BotFather) and the Telegram Road App (running in the Telegram Road servers) are **different,** and they are only linked by the Token and the Template parameters (see below). The end user interacts with the Telegram Road App through the Telegram bot (the only thing he/she sees on the device)
- Owner : a number associated with the end user's device, loaded automatically in register time. NOT TO BE CHANGED
- Template : the name of the Webhook function associated to the app, created in register time. NOT TO BE CHANGED
- Token : an unique and secret string assigned by Botfather when creating the bot. NOT TO BE CHANGED
- Datasource: the name of the database the dynamic application will be running on. Not to be used for static applications

These fields can be accessed by the App manager through the web interface :

http://telegramroad.com/cfusion/interroga/index.cfm?chat_id=Owner

where "Owner" is the abovementioned number associated to the owner's device by Telegram.

The url is both accessible on a desktop platform (preferred) , or a mobile platform as well.

CurrentQuery

CurrentQuery is a data structure built-in within the Telegram Road application. It is dedicated to the storage of input and output parameters used in the application lifetime, which must be maintained during the various steps (the "questions") the App is composed of.

The nature and number of such parameters are up to the App manager. Their name is defined in the "Field" part of an "S" or "I" node (see "Node Properties")

Parameters stored in CurrentQuery can be used in the following ways:

1. they can be recalled in questions, so that the end-user will be asked questions referring to them. The way the App manager can reference such parameters is to enclose their name within a pair of dollar signs "$". For example, if some "I" nodes have asked for paramerers named "Name" and "Surname", and stored them in CurrentQuery , a further question in an "S" node may contain :
   - Hello, $Name$ $Surname$! How can we go on?

   thus offering the and-user a personalized question.

NOTE : only the “Question” part of a node can translate parameter names in such a way. “Presentation” part can not.

2. They can be used in queries, which is way more useful in building queries (*Dynamic Apps only*). We have seen before that queries can be built through “Q” nodes, writing the query text in the “Question” field. Thus, it is possible to reference a stored variable, again using the $variable$ notation.

For example, if a former “S” node did let the end-user to choose a table to work on, and that “S” node stored that value in the variable “myTable” , then a following “Q” node might reference it building in the Question field a SQL statement like this:

```
SELECT names AS val FROM $MyTable$  LIMIT 10;
```

to let the end user extract data from the table previously chosen.

****Please note the use of aliasing on the val name, as described in Q node usage rules. ****

Beyond explicit values, loaded in CurrentQuery through “S” or “I” nodes, the following variables are loaded and made available by default:

- **chat_id** : the identifier which is passed by Telegram , unique for any user device
- **ds**: the datasource assigned to the App, as configured in design time by the App manager.

Geo-location

It is possible to build apps aware of the location of the terminal they are run from. Coupling the info about the position of the terminal with info where some services are available allows an application builder/manager great power.

Note 1: The position (location) of a terminal is supposed to be of an actual terminal. Location services are NOT provided by Telegram on emulation platforms such as Telegram Web.

Note 2: In order to be localized, a terminal should have enabled its own location facilities, such as Wi-Fi or (better) GPS facilities. Such services are offered by the smartphone operating system (either iOS or Android) and will be checked for activation if necessary.

Geo-location is performed in Telegram Road through two different passages, both of which must be foreseen by the app manager:

Location detection

The retrieval of location information in the app is performed placing an answer in which the offered text is one of a few reserved words . At current time such reserved words are the following:

- geo
- location
- whereami
- localize

If the answer is any of these texts, and if the user chooses this answer, the application will :

- ask the user confirmation about the will to provide localization information
- save it in the CurrentQuery structure, under **location.longitude** and **location.latitude,** the coordinates (numeric) of the terminal;

- proceed with the default next node. BEWARE: only the default next node (the one pointed by the Next field of a question) will be followed, not the one pointed by the answer itself. For such an answer, the provided answer text is discarded

Location usage

After localization, the application can proceed retrieving the info stored in **location.longitude** and **location.latitude** within CurrentQuery**,** and comparing it with information to be extracted by the database specific to the application. As this may vary with the application, it is not possible to describe it in this document.

For sure, it will be necessary to operate on some Q node (query) or X node (Execute) to perform some high level control, based on extraction and transformation of the available location info.

Scores

Score is a cheap and useful way to build applications that sum up a ranking depending of the history of the path the end-user makes in the App run.

Examples could be tests, culture assessments, and so on.

Scores can be assigned either to :

- nodes : visiting a node its default score (if any) is added
- answers : choosing an answer a further score may be added

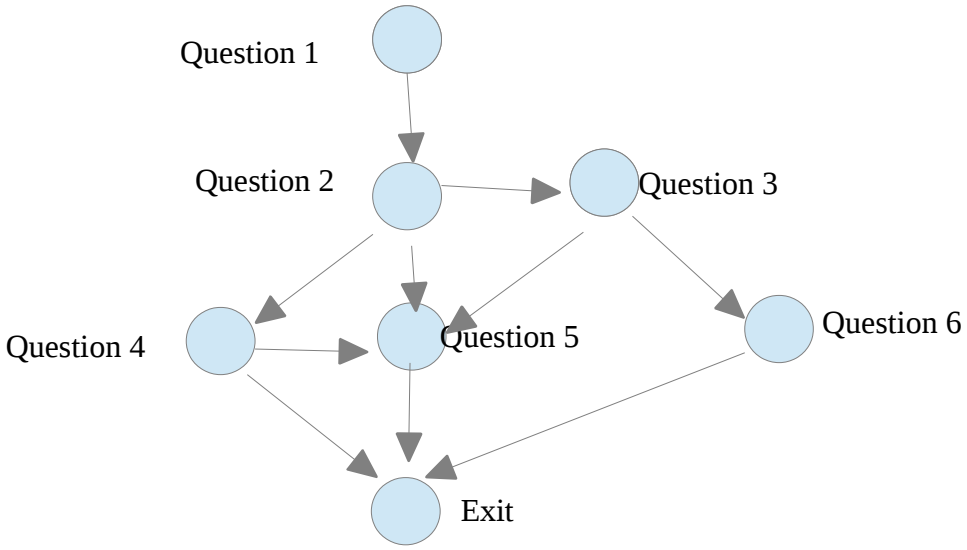
When starting the App, the initial score is set to zero.

The final sum of the accrued scores can be presented, possibly in a final node, just including the term “score” (verbatim) in the Question .

It's that easy.

Application creation and registration

1 – First of all you need paper and pencil. On a sheet, you can sketch the questions, drawn as boxes (or “balls”) which are connected through arcs , representing answers, and leading to other boxes. You can draw a graph. representing your application, like this:



This graph will undergo modifications and updates, as long as the App will grow. So, Don't worry if the correct form is not found at once.

2 – The next step is translating it on a bot on Telegram. You need to have Telegram on your own mobile device, or on your PC (*desktop*). In any case, the address to visit is :

http://telegram.org/dl

Note: since the installation is tied to a cell phone number, if you give the same number you are granted to have on any platform perfect replicas of your profile (chat, bot, preferences, …), always aligned in real time. The cell phone number is only required in this passage , and by no means will it be necessary or released to whatsoever.

3 – After Telegram installation, you will search for a particular bot, called **BotFather**, which acts as a "host" for creation of new bots. The creation of a new bot, as well as all other Telegram functions, is **free of charge**.

4 – With the command **/newbot**, after a handful of easy passages, we can give our bot a name and get an unique key (the *token*) giving access to the newborn bot .

5 – The next step , always within Telegram, is looking for another bot, **@Troadbot**. This bot, peculiar for the platform Telegram Road, will ask us the name to be given to our new bot (better the same one of Telegram) and the token which BotFather assigned to it. (Best idea is to copy/paste).

6 – After registering the new bot, we are given the possibility to run a form-based application where we will be able to input the graph drawn in step 1 , working in the so called *backstage* of Telegram Road.

All these activities are up to the administrator of the chatbot (the "App manager"), who own credentials (password) granting him control on the Apps.

7 – At this point the App is ready to be released on a desktop or tablet platform (via web), and possibly undergo a debug phase .For instance, this is how our chatbot questions set might be available to the manager:

8 – We will insert here the boxes , corresponding to the questions, and the arcs, corresponding to the answers , our App is composed of.

9 - At any time, the App can be as well tested on our mobile device, running it in Telegram environment

An alternative way of entering the graph is shown in the following chapter.

Expanding an app

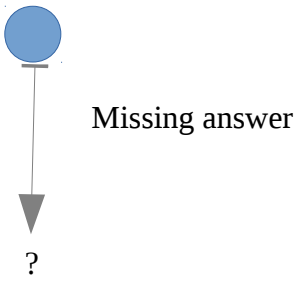
As from release 1.3, not only a manager is given the possibility to build an app, but also the end user can. That corresponds to a collaborative way to create an app, as anybody can add nodes to a new or an existing app, bringing new options (nodes, arcs) to a graph representing the app.

This can easily bring, for instance, to :

- troubleshooting or “how-to” manuals, in which anyone knowing the right way to solve an issue is given the possibility to share his/her knowledge (a *wiki*)
- tourism guides, where new hints, news and material are add on the way by tourists themselves
- games in which a story is actually bred by the very same gamers in a collaborative fashion

This list can naturally grow with the fantasy of users and developers.

When a new app is born, the very first question carries a “**missing answer**” option.



This is a keystone of a Telegram Road application, as the user choosing this option:

- will be asked the answer (text) he feels missing (this text will replace the “missing answer” string
- will be asked if he knows a follow-up for such answer. This can be thought of as asking a way to solve a problem , which is pointed by the new answer being added
- if the user does not know the way, answering “no” will let him close this option, waiting for someone else to follow. Adding an answer will just let other users know that an issue is still “open” and waiting for a solution.
- When another user (or, maybe, the user himself) will be visiting the app and following the new answer, he will be asked if it feels ready to submit a solution.
- Accepting, he will be asked a new question, that is a node in the graph, along with all its features, such as an introduction, the question itself and possibly a picture to be shown. (see node features in 4)
- The newborn question is given the features “open” = true and “public”=false. This will allow further possibility of growth of the graph , but inhibit the publication of the node until an administrator has authorized it. Only the creator will be allowed to see at the moment.
- New arcs (answers) will be possibly added to the new questions, repeating the steps above. Each new node will carry along a “missing answer” option to let it grow .

This possibility of feeding contents from the end user side gives Telegram Road a great power to build innovative and collaborative applications with no developing tools.

The app manager will be still free to edit each single node and arc of the graph, the same way he would enter them with the standard editing tools (1)

NOTE: nodes entered this way will only be of type “S” (see 5) . It is not possible to enter other types of nodes but with manual editing. That means that apps built this way are always **static** ones (see 3).

How can I get more info?

E-mail : telegramroad@gmail.com

Web : http://telegramroad.com