KonaKart
Enterprise Java eCommerce

KonaKart Messenger Bot

12th December 2017

DS Data Systems (UK) Ltd.,
9 Little Meadow
Loughton, Milton Keynes
Bucks
MK5 8EH
UK

Table of Contents

What is the KonaKart Messenger Bot?

The purpose of the KonaKart Messenger Bot is to help customers find products managed by a KonaKart store, using the Messenger interface. The bot goes beyond simple keyword searches and uses the wit.ai (https://wit.ai/) artificial intelligence engine to attempt to understand certain message types (intents) and to respond accordingly.

More technically, it’s a servlet that receives messages from Messenger (https://www.messenger.com) and uses wit.ai artificial intelligence to attempt to understand them. Wit.ai allows you to build a Siri-like speech or message interface by turning natural language (speech or messages) into actionable data.

Based on the interpretation by the artificial intelligence engine, the bot servlet communicates with a KonaKart engine to find products and sends the result back to the Messenger user as a reply message.

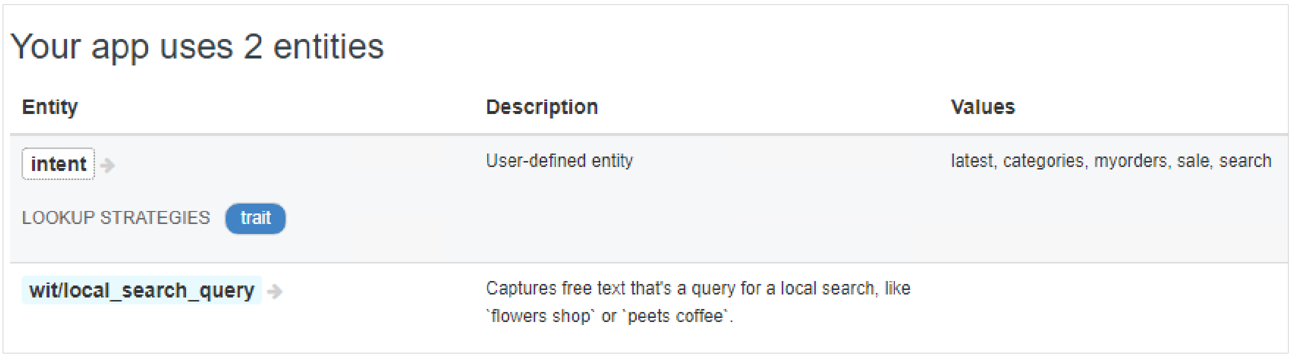
The full source code is available, and it’s architected in order to allow you to add functionality in order to process more message types. It can be considered as a framework allowing you to create your own bot with bespoke functionality. KonaKart also provides a consultancy service to customise the Messenger Bot to match your business requirements.

You can create intents (user defined entities) and train your Wit app to recognize these intents so that the bot can act on them. You should enable the *wit/local_search_query* entity which captures free text that's a query for a local search.

Default Implementation

The standard implementation of the Messenger Bot will act upon a number of intents that have been defined in our Wit app. These are:

- categories
- search
- sale
- latest
- myorders



Your app uses 2 entities

Entity	Description	Values
intent LOOKUP STRATEGIES trait	User-defined entity	latest, categories, myorders, sale, search
wit/local_search_query	Captures free text that's a query for a local search, like `flowers shop` or `peets coffee`.	

Screen capture of wit.ai control panel

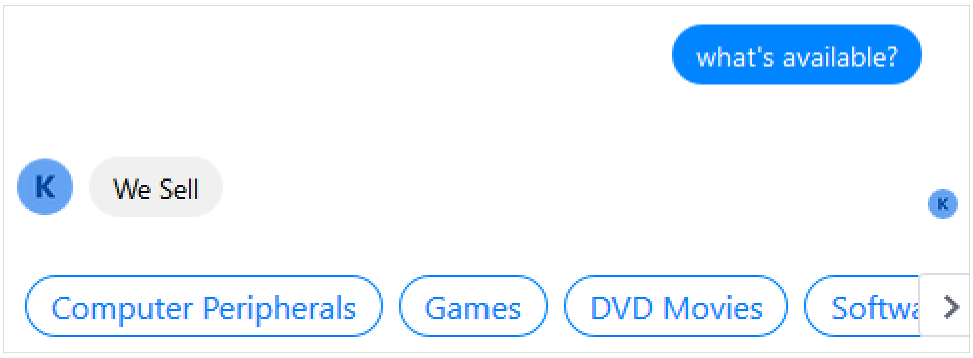
Note that you will need to create your own Wit app and train it in order to recognise commands and return the intents, The Wit app you create has a wit token which you need to configure your Facebook app with, in order to connect the two. You do this when you enable NLP in your Facebook app by selecting a custom model.

categories intent

The type of text it recognises is:

- What's available?
- What products are available?
- What products do you sell?
- What's your range of products?
- What do you sell?

When KonaKart detects the *categories* intent it sends back a list of top level category names that can be selected.



search intent

In order to get good results, you should enable the SOLR search engine. The servlet uses the Suggested Search functionality of KonaKart for fetching search results.

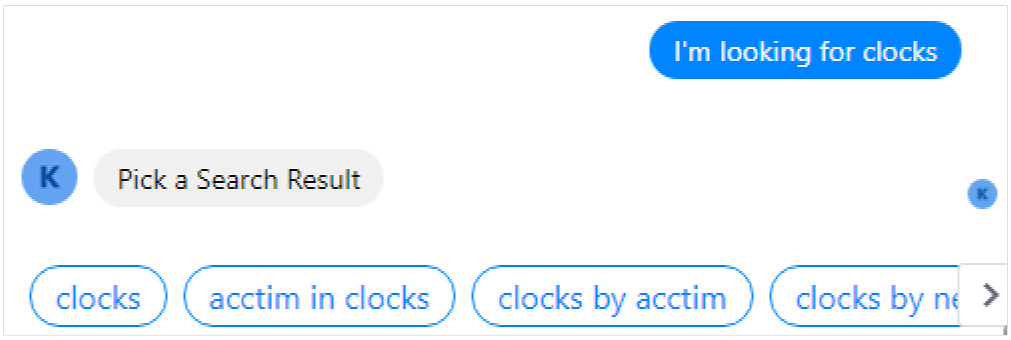
The *search* intent is detected if you type in one or two keywords such as:

- Phones
- Clocks
- Lawnmowers
- Bosch lawnmowers

It also recognises phrases such as:

- I'm looking for clocks
- Do you sell lawnmowers
- Find ipad

The *wit/local_search_query* entity detects the keywords that are used for searching. The servlet returns a list of results that allow the user to refine the search.


I'm looking for clocks

Pick a Search Result

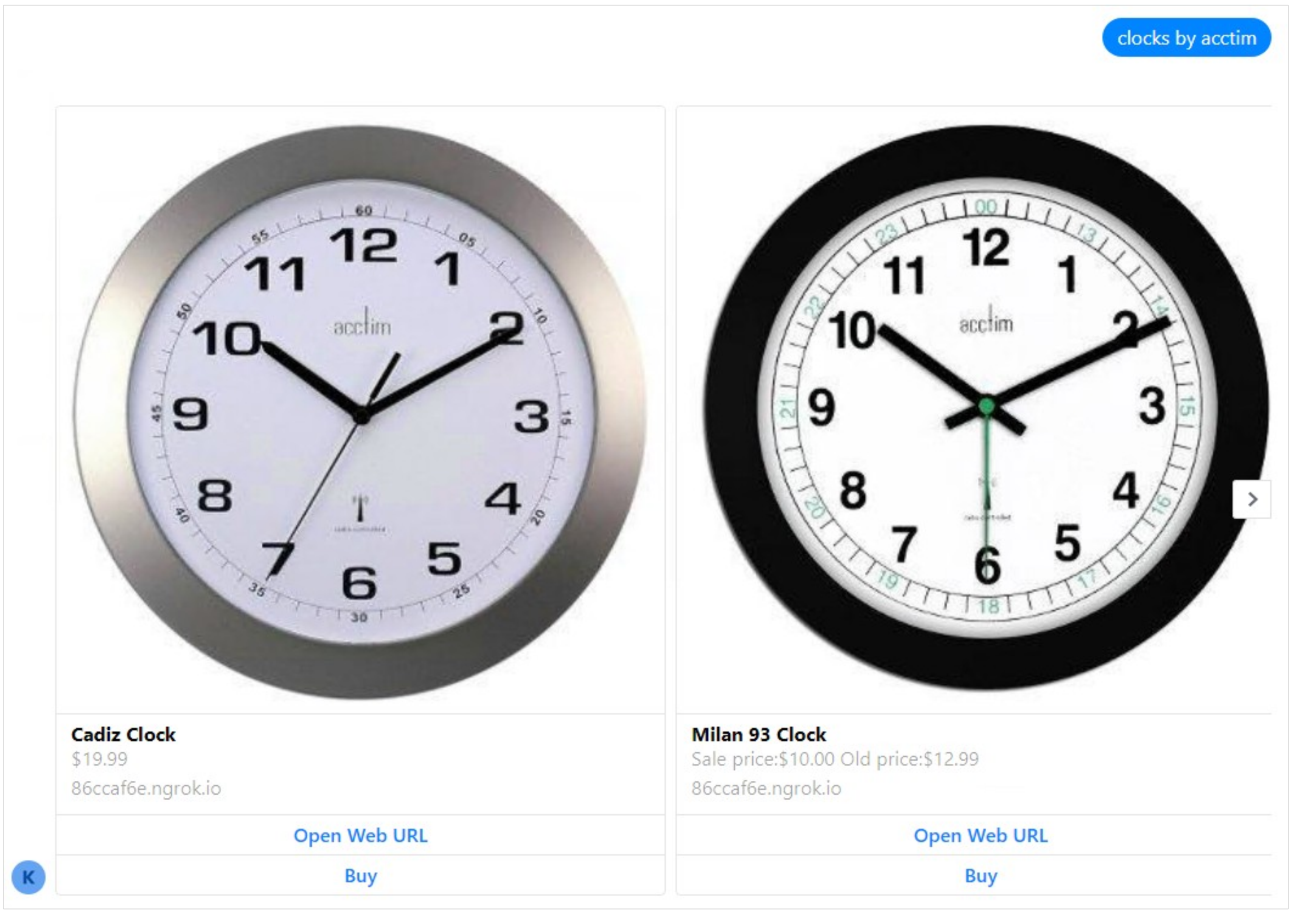
If the user selects “clocks by acctim” the result is a list of products which can be scrolled through (see below). Each product has a couple of buttons:

Open Web URL

By clicking on this button the customer is taken to the product detail page of the online store.

Buy

When this button is clicked, the product is added to the cart and the customer can then check out using the online store.


clocks by acctim

Cadiz Clock
$19.99
86ccaf6e.ngrok.io

Milan 93 Clock
Sale price:$10.00 Old price:$12.99
86ccaf6e.ngrok.io

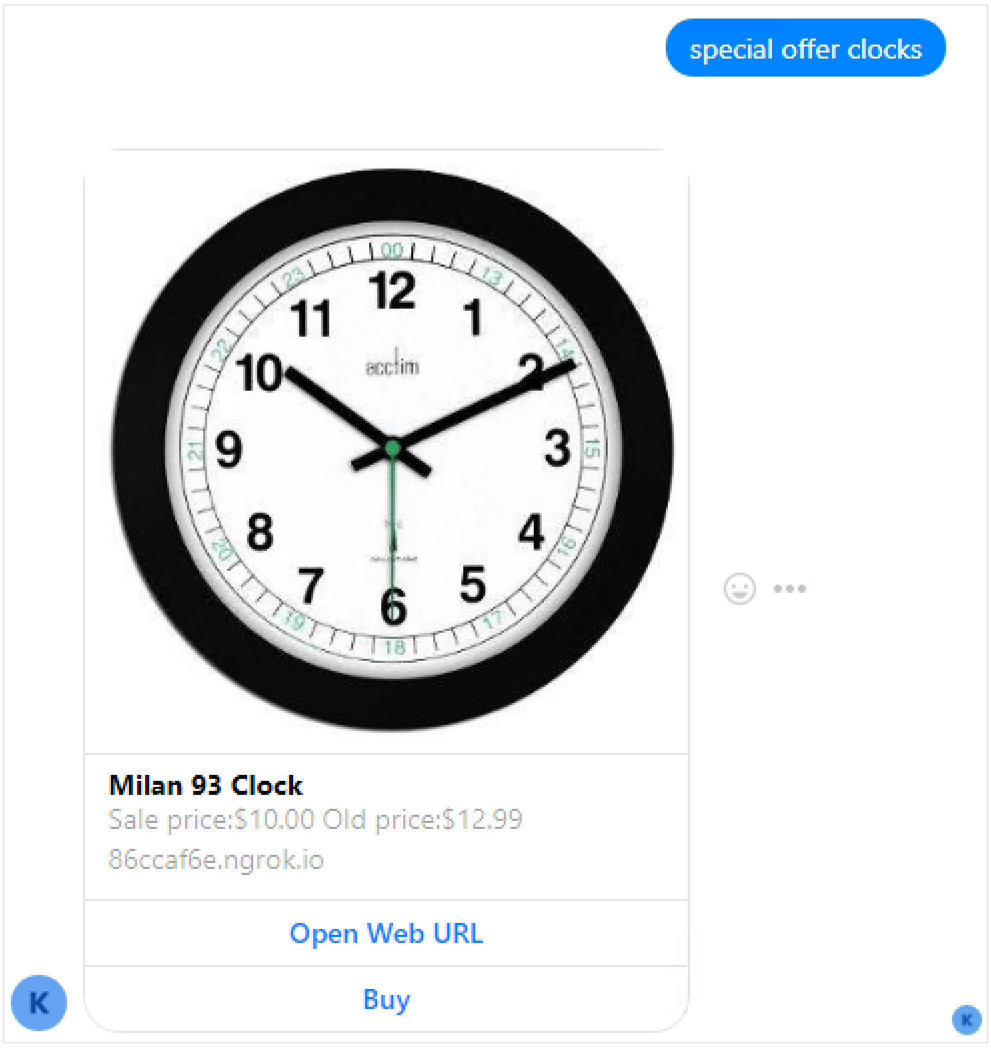
sale intent

The *sale* intent is triggered when the Wit app recognises text such as

- Show me clocks on sale
- Reduced clocks

- Clocks on sale
- Special offer clocks
- What clocks are reduced
- What’s on offer

If the customer asks for a products such as clocks or lawnmowers, then the *wit/local_search_query* entity detects the keyword so that the API call to KonaKart can narrow down the search. For example text such as “Special offer clocks” would return any clocks on sale:



special offer clocks

Milan 93 Clock

Sale price:$10.00 Old price:$12.99

86ccaf6e.ngrok.io

latest intent

The *latest* intent is triggered when the Wit app recognises text such as

- Latest clocks
- What’s new?
- Most recent clocks
- New clocks
- Show me new clocks
- Show me what’s new

If the customer asks for a particular product such as clocks or lawnmowers then the *wit/local_search_query* entity detects the keyword so that the API call to KonaKart can narrow down the search. The products are displayed, ordered by the date in which they were inserted into the catalog, showing the most recent ones first.

myorders intent

This intent has no direct implementation in code. The servlet looks up a message from the message catalog called "intent.myorders" and returns it if it exists.

Other intents

If an intent is received that KonaKart doesn't recognise, it attempts to look up a reply message from the message catalog in the format intent. {INTENT-NAME}. If an entry is found then the message is sent back to the customer. For example if the intent received is called *myorders* then we try to find a message where the key is *intent.myorders*.

Message Formats

Developer information regarding the Messenger platform is available here https://developers.facebook.com/docs/messenger-platform .All messages to and from Messenger are in JSON format. Some example messages have been added to Appendix A at the end of this document and in files after an installation under the custom/konakart_messenger/messages directory.

The messages received from Messenger have a *messaging* section as shown below.

```
"messaging": [{
  "sender": {
   "id": "1643303712406390"
  },
  "recipient": {
   "id": "1328384363951576"
  },
  "timestamp": 1510067979202,
  "message": {
   "mid": "mid.$cAAS4KLYvTbRlyFYLwlflxDLJKwgD",
   "seq": 1892,
   "text": "find lawnmowers",
   "nlp": {
    "entities": {
     "local_search_query": [{
       "suggested": true,
       "confidence": 0.93809,
       "value": "lawnmowers",
       "type": "value"
      }
     ],
     "intent": [{
       "confidence": 0.99957876346761,
       "value": "search"
      }
     ]
    }
   }
  }
```

The *text* attribute of the message contains the words typed by the user. In this case they are “find lawnmowers”. The nlp section contains the results from the Wit app . In this case it tells us that it has recognised the *search* intent with a high confidence score and the *wit/local_search_query* entity has returned the value “lawnmowers” with a high confidence score.

In the current implementation, the Messenger Bot servlet makes use of the following message types for replies:

- Quick Replies - https://developers.facebook.com/docs/messenger-platform/send-messages/quick-replies
- Generic Template - https://developers.facebook.com/docs/messenger-platform/send-messages/template/generic
- Button template - https://developers.facebook.com/docs/messenger-platform/send-messages/template/button

Examples can be seen in Appendix A.

Getting Started

There are a few steps independent of KonaKart that need to be taken in order to create your own bot accessible through Messenger. A guide is available online at https://developers.facebook.com/docs/messenger-platform/getting-started/quick-start . You must:

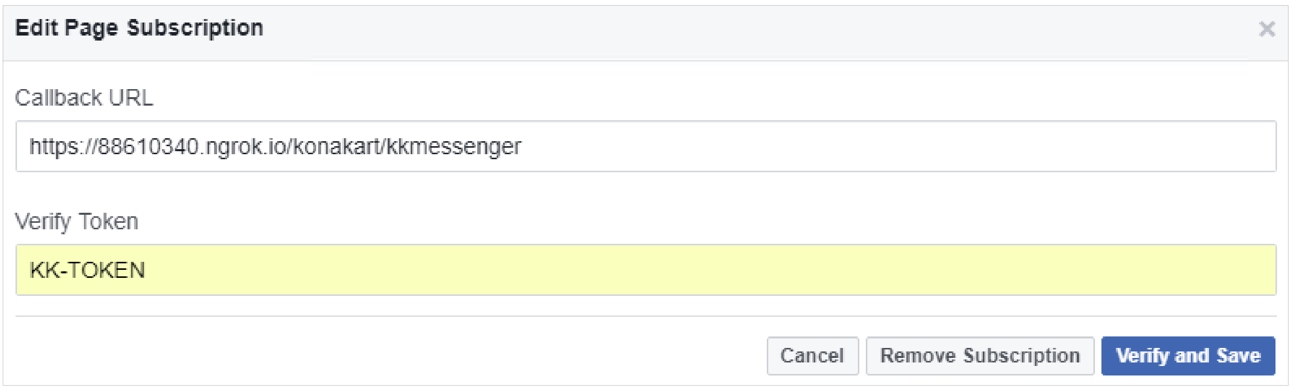
1. Create a Facebook app and ensure that the Messenger platform is added to the app
2. Configure the webhook for your app (see below for more details)
3. Create a Facebook page and subscribe your app to the page. Generate a page access token which must be added to the web.xml for the KonaKart servlet since it is used to make API requests.
4. Enable built in NLP for the Facebook App so that the messages received by the Messenger Bot include NLP information. Select a custom model and give it the Wit token of your Wit app where you've defined your intents.

Webhooks

To receive messages and other events sent by Messenger users, the app should enable webhooks integration. The selected events are:

- messages
- messaging_postbacks

In order to setup webhook subscriptions, you need to specify a Callback URL and a Verify Token.


Edit Page Subscription

Callback URL

https://88610340.ngrok.io/konakart/kkmessenger

Verify Token

KK-TOKEN

Facebook attempts to contact the KonaKart servlet through the callback URL and the servlet must respond with the verify token. The verify token must be added to the web.xml so that it can be read by the servlet and returned to Facebook.

The callback URL must be https and self signed certificates are not accepted. During development this hurdle can be overcome by using ngrok (https://ngrok.com/) which can supply a tunnel to your localhost.

Servlet Configuration

The Messenger Bot consists of a servlet that must be defined in web.xml for the webapp where it is installed. A typical configuration looks like:

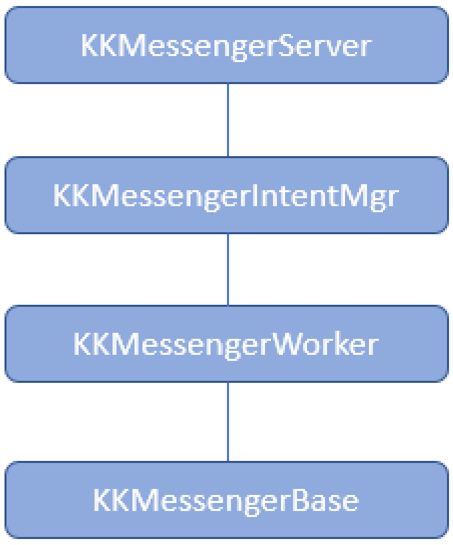
```
<servlet>
      <servlet-name>KonaKart_Messenger_Servlet</servlet-name>
      <servlet-class>
            com.konakart.messenger.servlet.KKMessengerServer
      </servlet-class>
      <init-param>
            <param-name>pageAccessToken</param-name>
            <param-value>ACCESS-TOKEN</param-value>
      </init-param>
      <init-param>
            <param-name>verifyToken</param-name>
            <param-value>VERIFY-TOKEN</param-value>
      </init-param>
      <init-param>
            <param-name>locale</param-name>
            <param-value>en_GB</param-value>
      </init-param>
      <init-param>
            <param-name>storeId</param-name>
            <param-value>store1</param-value>
      </init-param>
      <init-param>
            <param-name>catalogId</param-name>
            <param-value></param-value>
      </init-param>
      <init-param>
            <param-name>minConfidence</param-name>
            <param-value>0.8</param-value>
      </init-param>
      <init-param>
            <param-name>maxQuickReplyButtons</param-name>
            <param-value>11</param-value>
      </init-param>
      <init-param>
            <param-name>maxNumGenericTemplateProducts</param-name>
            <param-value>10</param-value>
      </init-param>
      <load-on-startup>30</load-on-startup>
</servlet>

<servlet-mapping>
      <servlet-name>KonaKart_Messenger_Servlet</servlet-name>
      <url-pattern>/kkmessenger</url-pattern>
</servlet-mapping>
```

The configuration parameters are:

pageAccessToken	It is generated by Facebook. In the Messenger section of the App Administration interface you can associate your app with a Facebook page and then generate the access token.
verifyToken	The verify token is the token that Messenger expects to receive when it contacts the Messenger Bot through the callback URL with doGet() rather than doPost(). The value written in web.xml must be the same value that was configured when setting up the page subscription.
locale	The locale for the servlet which determines the message catalog used and the language used when retrieving products and categories.
storeId	The store Id of the KonaKart store that the servlet communicates with
catalogId	The catalog Id used when getting products. It may determine the product price and whether the product is available.
minConfidence	This is the minimum confidence level used by the servlet in order for it to accept NLP information.
maxQuickReplyButtons	Currently only 11 quick reply buttons can be sent. It is configurable in case this changes in the future.
maxNumGenericTemplateProducts	Currently only 10 products can be sent using the generic template. It is configurable in case this changes in the future.
storeSecureBaseUrl	The store secure base URL used for checking out, is normally created from the KonaKart engine base URL by substituting http with https. However, sometimes this may not function correctly if the port numbers are also specified in the URL.

Source Code

The servlet consists of four main objects inheriting from each other:



KKMessengerServer

This class is the entry point for messages. It processes all received messages and implements doGet() and doPost(). Depending on the contents of the received message it decides what to call next.

KKMessengerIntentMg

This manager is called to manage messages when an intent is recognised by your Wit app. The class should be customised to add your own new intents. When an intent isn't recognised, the manager attempts to see whether a reply message exists in the message catalog.

KKMessengerWorker

This is the layer where calls are made to the KonaKart engine to search for products.

KKMessengerBase

The base layer contains initialisation code that creates a KonaKart engine instance and loads reusable objects such as configuration variables, the category tree and the language object. It also contains code to manage the message catalog and to create the JSON messages.

Object Model

The *com.konakart.messenger.om.** classes are used to implement an object model for the data sent to and received from Messenger. Gson from Google is used to serialize and deserialize data to create an object structure from incoming JSON messages and to create outgoing JSON messages from a populated object model.

Message Catalog

All messages are looked up in message catalogs that can be found in the custom/konakart_messenger/resources directory. During installation these catalogs are also copied to webapps/konakart/WEB-INF/classes in order to make them available on the classpath at runtime. The default catalog is called KKMessengerMessages_en.properties. If you wish to add a new catalog for a different language (e.g. German) it should be called KKMessengerMessages_de.properties and the locale in the web.xml should be set to de_DE.

Installation and Customisation

From v8.8.0.0 of KonaKart the Messenger Bot source code will be installed automatically in the custom/konakart_messenger directory. The messages directory contains some example messages that are a useful guide to understanding the Messenger Bot code.

In the KonaKart webapp's web.xml (at webapps/konakart/WEB-INF/web.xml) you will see the Messenger Bot servlet (and associated servlet mapping) commented out. In order to run the Messenger Bot you will need to enable these by uncommenting these servlet sections (or use the custom ANT build target "enable_KonaKart_Messenger" to uncomment these two sections automatically):

Example on Windows:

```
C:\KonaKart\custom>bin\kkant enable_KonaKart_Messenger
Buildfile: C:\KonaKart\custom\build.xml
Trying to override old definition of task javac

enable_KonaKart_Messenger:
     [echo] Uncomment KonaKart Messenger in
C:\KonaKart\custom/../webapps/konakart//WEB-INF/web.xml

BUILD SUCCESSFUL
Total time: 0 seconds
```

Once the servlet sections have been uncommented you need to configure it as explained previously. The *pageAccessToken* and *verifyToken* need to be given correct values so that the servlet can connect to Messenger.

Eclipse Project

An Eclipse project is provided (in the custom/konakart_messenger directory of all Enterprise installations (from v8.8.0.0) so that the Messenger can be customised and extended. An ANT build file is also provided for rebuilding and re-deploying the konakart_messenger-N.N.N.N.jar and all of the resources. The ANT build file is also provided in the custom/konakart_messenger directory of all Enterprise installations (from v8.8.0.0).

```
Main ANT build targets:

 build                 Compiles all the messenger code and creates the jar

 clean                 Clears away everything that's created during a build

 compile               Compile the messenger code

 copy_jars             Copy the jar into the main custom jar directory

 copy_resources        Copy the resources into the konakart webapp directory

 debugenv              Debug the environment

 make_jars             Create the messenger jar
```

Example on Windows:

```
C:\KonaKart\custom\konakart_messenger>..\bin\kkant build copy_resources
copy_jars

Buildfile: C:\KonaKart\custom\konakart_messenger\build.xml


clean:

     [echo] Cleanup...
```

```
compile:
     [echo] Compile the messenger code
    [mkdir] Created dir: C:\KonaKart\custom\konakart_messenger\classes
    [javac] Compiling 29 source files to
C:\KonaKart\custom\konakart_messenger\classes

make_jars:
     [echo] Create the messenger jar
    [mkdir] Created dir: C:\KonaKart\custom\konakart_messenger\jar
      [jar] Building jar:
C:\KonaKart\custom\konakart_messenger\jar\konakart_messenger-8.8.0.0.jar

build:

copy_resources:
     [echo] Copy the resources into the konakart webapp directory

copy_jars:
     [echo] Copy the jars into the main custom jar directory
     [copy] Copying 1 file to C:\KonaKart\custom\jar
     [echo] Execute copy_jars in main custom build

copy_jars:
     [echo] Copy the custom jars to their respective lib directories
     [copy] Copying 1 file to C:\KonaKart\webapps\konakartadmin\WEB-INF\lib
     [echo] Copy the custom jars to their respective lib directories
     [copy] Copying 1 file to C:\KonaKart\webapps\konakart\WEB-INF\lib

copy_jars_EE:
     [echo] Copy konakartadmin_app-8.8.0.0.jar to the konakartadmin_app lib
directory

BUILD SUCCESSFUL
Total time: 2 seconds
```

Debug

In order to see debug messages you must add a section to webapps\konakart\WEB-INF\classes\konakart-logging.xml.

```
      <Logger name="com.konakart.messenger" level="DEBUG" additivity="false">
```

```
            <AppenderRef ref="STDOUT"/>
            <!-- KonaKart File Logger
            <AppenderRef ref="FILE"/>
            End of KonaKart File Logger -->
        </Logger>
```

Logging at DEBUG level allows you to see the messages sent and received from Messenger as well as other interesting information to help you grasp a better understanding of how the messages are being processed.

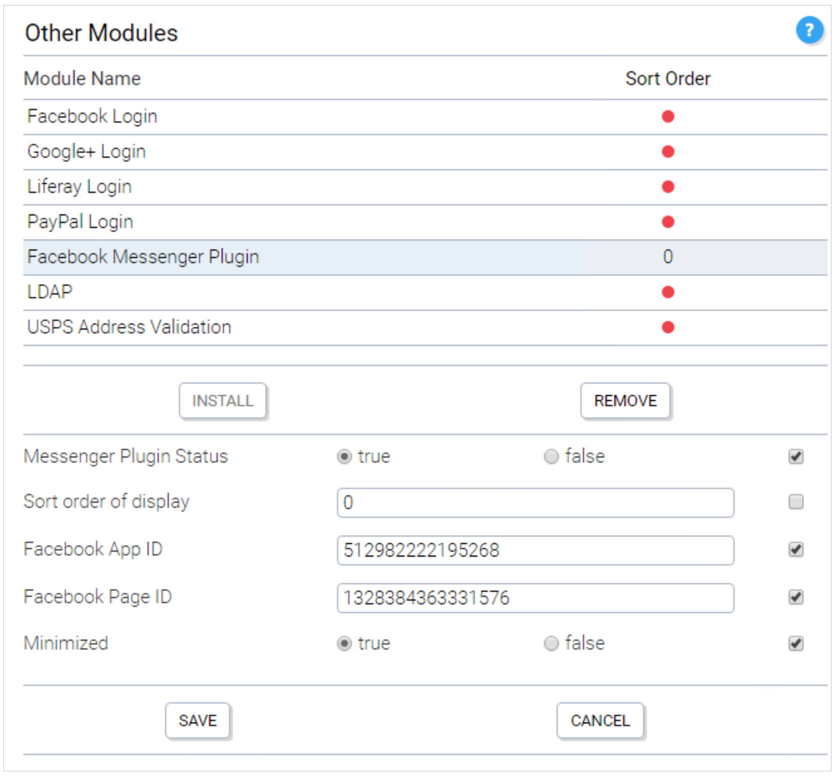
Help from KonaKart Professional Services

One of the services offered by KonaKart professional services is to help you get started with your Messenger Bot and to implement a new set of Wit app intents that satisfy your business needs.

If you'd like to consider this approach, please contact sales@konakart.com in order to discuss your requirements and see how we can help.

Messenger customer chat plugin

The Messenger customer chat plugin is a seamless way for businesses and people to communicate with one another anytime, anywhere. Unlike traditional web-based chat, people can continue a conversation from website to Messenger and vice versa, allowing for a better overall experience at a pace that works for everyone. Customer chat also allows businesses to utilize many of the benefits of the Messenger Platform, so businesses can incorporate automation, NLP, rich media and more into their website experience.


Other Modules

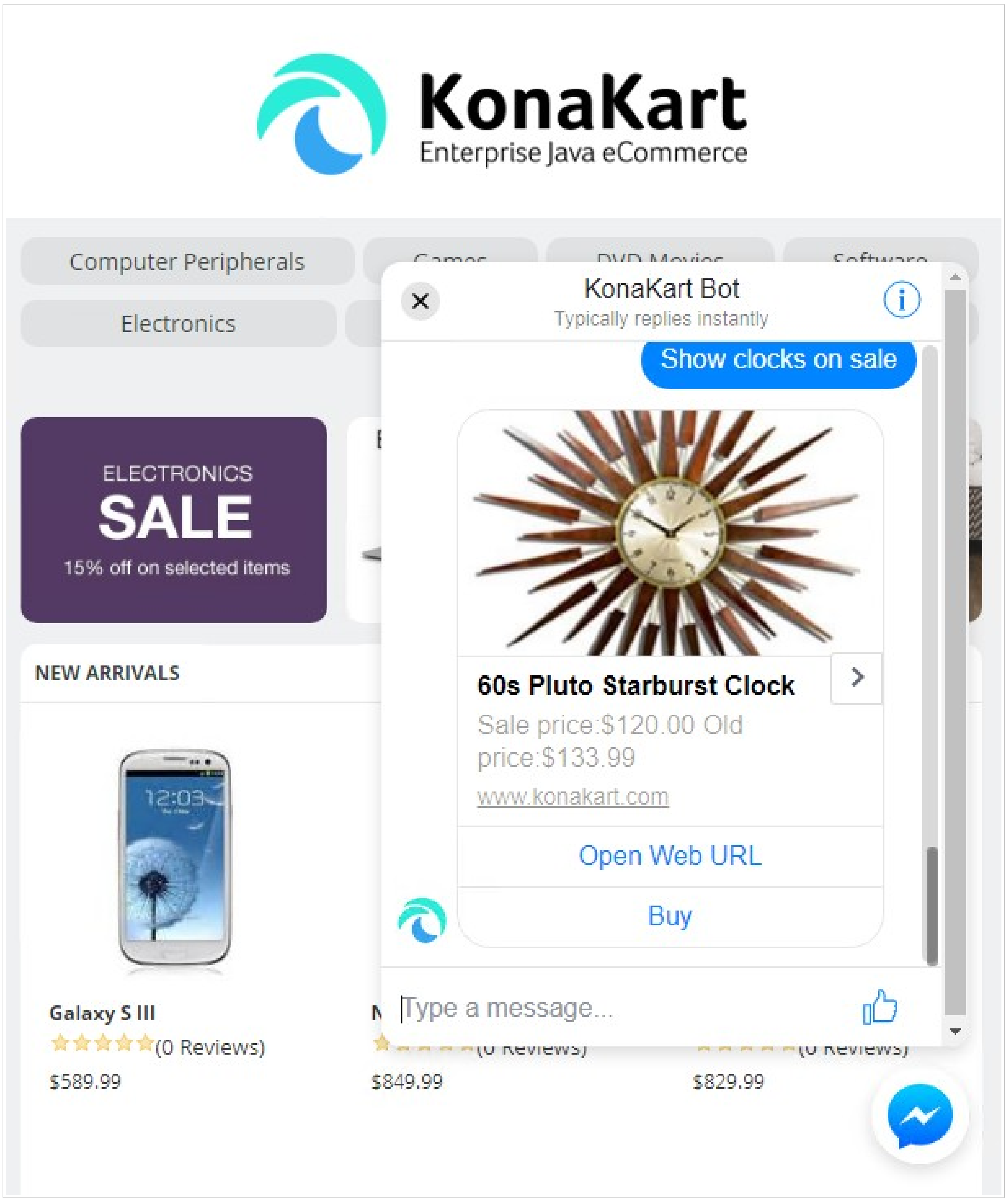
Module Name	Sort Order
Facebook Login	●
Google+ Login	●
Liferay Login	●
PayPal Login	●
Facebook Messenger Plugin	0
LDAP	●
USPS Address Validation	●

Messenger Plugin Status	[✓ true]	false	✓
Sort order of display	0		
Facebook App ID	512982222195268		✓
Facebook Page ID	1328384363331576		✓
Minimized	[✓ true]	false	✓

The Messenger customer chat plugin is available in the standard storefront KonaKart application. In order to activate it, you must install the Facebook Messenger Plugin module as shown above. The Facebook Page ID can be retrieved from the About section of your page.

In order for the plugin to function correctly you must white list the URL of the storefront. This can be configured under Settings >> Messenger Platform of your Facebook page.

Once configured, the plugin will show in the storefront as can be seen below.



If your intention is to make the Messenger Bot available through the chat plugin, it probably makes sense to define a number of different intents in the Wit app since the customer is on the storefront and can easily search for products without using the bot.

Appendix A

Message received from Messenger when user types “find lawnmowers”.

```
{
 "object": "page",
 "entry": [{
   "id": "1328384363951576",
   "time": 1510067979681,
   "messaging": [{
     "sender": {
      "id": "1643303712406390"
     },
     "recipient": {
      "id": "1328384363951576"
     },
     "timestamp": 1510067979202,
     "message": {
      "mid": "mid.$cAAS4KLYvTbRlyFYLwlflxDLJKwgD",
      "seq": 1892,
      "text": "find lawnmowers",
      "nlp": {
       "entities": {
        "local_search_query": [{
          "suggested": true,
          "confidence": 0.93809,
          "value": "lawnmowers",
          "type": "value"
         }
        ],
        "intent": [{
          "confidence": 0.99957876346761,
          "value": "search"
         }
        ]
       }
      }
     }
    }
   ]
  }
 ]
}
```

Message sent back to Messenger with a list of Quick Replies that the user can select from in order to determine what products to see. The payload contains encoded information that is sent back to the bot when the user selects a reply.

```
{
 "recipient": {
  "id": "1643303712406390"
 },
 "message": {
  "text": "Pick a Search Result",
  "quick_replies": [{
    "content_type": "text",
    "title": "lawnmowers",
    "payload":
"eyJhY3Rpb24iOiJTZWxlY3RQcm9kdWN0cyIsImlkIjotMSwiaWQxIjotMSwiaWQyIjozMH0\u003d"
   }, {
    "content_type": "text",
    "title": "bosch in lawnmowers",
    "payload":
"eyJhY3Rpb24iOiJTZWxlY3RQcm9kdWN0cyIsImlkIjotMSwiaWQxIjoxNywiaWQyIjozMH0\u003d"
   }, {
    "content_type": "text",
    "title": "lawnmowers by bosch",
    "payload":
"eyJhY3Rpb24iOiJTZWxlY3RQcm9kdWN0cyIsImlkIjotMSwiaWQxIjoxNywiaWQyIjozMH0\u003d"
   }, {
    "content_type": "text",
    "title": "flymo in lawnmowers",
    "payload":
"eyJhY3Rpb24iOiJTZWxlY3RQcm9kdWN0cyIsImlkIjotMSwiaWQxIjoxOCwiaWQyIjozMH0\u003d"
   }, {
    "content_type": "text",
    "title": "lawnmowers by flymo",
    "payload":
"eyJhY3Rpb24iOiJTZWxlY3RQcm9kdWN0cyIsImlkIjotMSwiaWQxIjoxOCwiaWQyIjozMH0\u003d"
   }
  ]
 }
}
```

Message received from Messenger after a user has selected a quick reply. The encoded payload information instructs the bot on what action to take and which manufacturer and category has been selected. When a quick reply is received, the nlp information is ignored.

```
{
 "object": "page",
 "entry": [{
   "id": "1328384363951576",
   "time": 1510068824760,
   "messaging": [{
     "sender": {
      "id": "1643303712406390"
     },
     "recipient": {
      "id": "1328384363951576"
     },
     "timestamp": 1510068824328,
     "message": {
      "quick_reply": {
       "payload":
"eyJhY3Rpb24iOiJTZWxlY3RQcm9kdWN0cyIsImlkIjotMSwiaWQxIjoxNywiaWQyIjozMH0="
      },
      "mid": "mid.$cAAS4KLYvTbRlyGLxCFflx2wdhM26",
      "seq": 1895,
      "text": "bosch in lawnmowers",
      "nlp": {
       "entities": {
        "local_search_query": [{
          "suggested": true,
          "confidence": 0.93625,
          "value": "bosch",
          "type": "value"
         }, {
          "suggested": true,
          "confidence": 0.93629,
          "value": "lawnmowers",
          "type": "value"
         }
        ],
        "intent": [{
          "confidence": 0.99733388355917,
          "value": "search"
         }
        ]
       }
      }
     }
    }
   ]
  }
 ]
}
```

Message sent to Messenger using the generic template. It consists of a couple of Bosch lawnmower products.

```
{
 "recipient": {
  "id": "1643303712406390"
 },
 "message": {
  "attachment": {
   "type": "template",
   "payload": {
    "template_type": "generic",
    "elements": [{
      "title": "Rotak 40 Ergoflex",
      "subtitle": "$159.99",
      "item_url": "https://784da321.ngrok.io/konakart/SelectProd.action?
prodId\u003d39",
      "image_url":
"http://www.konakart.com/konakart/images/prod/1/5/F/8/15F8FBB1-13DA-4A47-B3D2-
0F1E7BECAE7C_1_big.jpg",
      "buttons": [{
        "type": "web_url",
        "url": "https://784da321.ngrok.io/konakart/SelectProd.action?
prodId\u003d39",
        "title": "Open Web URL"
       }, {
        "type": "postback",
        "title": "Buy",
        "payload": "eyJhY3Rpb24iOiJCdXkiLCJpZCI6MzksImlkMSI6LTEsImlkMiI6LTF9"
       }
      ]
     }, {
      "title": "Bosch SHM 38G",
      "subtitle": "$55.95",
      "item_url": "https://784da321.ngrok.io/konakart/SelectProd.action?
prodId\u003d42",
      "image_url":
"http://www.konakart.com/konakart/images/prod/1/1/B/C/11BC0D25-1B08-4141-BABB-
1B1E633CB382_1_big.jpg",
      "buttons": [{
        "type": "web_url",
        "url": "https://784da321.ngrok.io/konakart/SelectProd.action?
prodId\u003d42",
        "title": "Open Web URL"
       }, {
        "type": "postback",
        "title": "Buy",
        "payload": "eyJhY3Rpb24iOiJCdXkiLCJpZCI6NDIsImlkMSI6LTEsImlkMiI6LTF9"
       }
      ]
     }
    ],
    "image_aspect_ratio": "square"
   }
  }
 }
}
```

Message received from Messenger when the user clicks on the Buy button. The message contains a *postback* section that contains the encoded payload data.

```
{
 "object": "page",
 "entry": [{
   "id": "1328384363951576",
   "time": 1510069136256,
   "messaging": [{
     "recipient": {
      "id": "1328384363951576"
     },
     "timestamp": 1510069136256,
     "sender": {
      "id": "1643303712406390"
     },
     "postback": {
      "payload": "eyJhY3Rpb24iOiJCdXkiLCJpZCI6MzksImlkMSI6LTEsImlkMiI6LTF9",
      "title": "Buy"
     }
    }
   ]
  }
 ]
}
```

Message sent to Messenger containing a button of type *web-url* which when clicked on takes the user to the url defined by the *url* attribute. In this case it takes the user to the KonaKart storefront after the selected product has been added to the cart.

```
{
 "recipient": {
  "id": "1643303712406390"
 },
 "message": {
  "attachment": {
   "type": "template",
   "payload": {
    "template_type": "button",
    "text": "The product has been added to the cart. Click below to
checkout..",
    "buttons": [{
      "type": "web_url",
      "url": "https://784da321.ngrok.io/konakart/InitFromToken.action?
key\u003d70aa3af4-b53b-48bf-913e-45f71719e068",
      "title": "Checkout"
     }
    ]
   }
  }
 }
}
```