Localize Direct Socket API Version 1.0

Localize Direct Socket API Version 1.0

This document describes what's required and how to use the LocalizeDirect Socket API. It's assumed that the reader is fairly familiar with socket-programming and communication. The basic idea of the Socket API is that text messages are sent via a TCP-socket to the server and then the server responds by returning a result message to the client.

The text messages are sent to and returned from the server as XML-formatted text (UTF8), using a protocol called the MapleProtocol (described below).

Create a Socket Connection

First of all a Socket Connection (TCP) needs to be created, specifying the LocalizeDirect Server host IP and port.

Sending Data

The next step is to send some data to the LocalizeDirect Message Application Server (MAPLE). The server uses its own protocol called the MapleProtocol. This protocol resembles the HTTP-protocol in many ways. Every time a message is sent to the server or received from the server, the first part of the message contains a simple ASCII-header, that describes the data that follows the header. Just like the HTTP-protocol, the header of the MapleProtocol always ends with a double newline, each in the form of a carriage return followed by a line feed (e.g. \r\n\r\n).

Right after the header follows the message data. The MapleProtocol can transport any type of data, but the LocalizeDirect API uses XML-structured messages.

The MapleProtocol Header

Below is an example of what a typical MapleProtocol header might look like. The header consists of a number of header fields that describes the message to be sent or received. Each field must end with a newline, in the form of a carriage return followed by a line feed (\r\n)

```
MAPLE/1.1
Client: console-client
Destination: com.localizedirect.server.Connector
Date: 2010-06-01 16:25:39 +0200
Id:
Message-Name:
Message-Length: 579
Message-Type: text
Message-Charset: UTF8
Message-Encrypted: false
Message-Compressed: false
```

The first part "MAPLE/1.1" just informs that this is a MapleProtocol and that it is of version 1.1. The other header fields are described below.

Client field

Lets the server know what type of client is sending the message. This can for example be “localize-direct-client”, “web-client” or “console-client”.

Destination field

This field lets the message application server know exactly where the message is supposed to go – the receiver of the message. For the Socket API, this should always be:

```
com.localizedirect.server.Connector
```

Date field

The Date-field specifies when the message was sent. This is merely information that can be used for logging, tracing and debugging purposes. The client can opt to leave this field blank, but it’s still a good idea fill it in. The date-format can be decided by the client, but the server will always respond with a date using the format: “yyyy-MM-dd HH:mm:ss Z”.

Id field

The Id-field can be set by the caller to identify that the message sent, really is the message which later is received (the message in response from the server). This is an optional field and it can be left blank.

Message-Name field

This message can be used to let the message application server know the name of the message that is sent. Using the Socket API, this field can be left blank, as the message application server only will pass on the message to the specified destination (Destination-field).

Message-Length field

This field describes the length of the message in bytes - header excluded. This field MUST be set correctly by the client. The server will only read the number of bytes specified and then stop reading, therefore it’s important that this length is correct and that it’s the length in bytes of the message data.

Message-Type field

This field describes the type of data sent or received. Using the Socket API, this will always be “text”, as the XML messages are classified as text.

Message-Charset field

This field describes the charset used in the data and only applies if the Message-Type is “text”. Using the Socket API, this should always be “UTF8” to ensure that Unicode characters will not be lost in translation.

Message-Encrypted field

This field specifies whether or not the data is encrypted.

Message-Compressed field

This field specifies whether or not the data is compressed.

XML-Messages

The messages sent to and received from the server follows a simple XML-structure. The xml-messages have some key-elements that will be described below.

EXECUTION-element

A message sent from the client is called an "Execution". An execution-message can consist of one or many statements or so called "tasks". A task describes a specific action for the server to perform. Tasks can have one or many arguments. Below is an example of what an execution message may look like (this is only an example message meant to explain the message structure and it is not a message that is part of the LocalizeDirect Socket API):

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION secId="1212" client="Console" version="1.0">
  <TASK name="GetUser">
    <OBJECT name="User">
         <userName/>
         <userEmailAddress/>
    </OBJECT>
    <WHERE>
         <userId>123</userId>
    </WHERE>
  </TASK>
</EXECUTION>
```

The first part "`<?xml version="1.0" encoding="UTF-8"?>`" is important as it describes the XML-format. This should always be set for each XML-message sent to the server.

The EXECUTION-element in the example has three attributes. The first attribute called "secId", declares the session under which this message will be executed. This attribute is described later in this document (under Login). The client-attribute describes what type of client is requesting the execution. This should always be set to "Console". The version attribute, should be set to the current API-version (currently 1.0).

The TASK-element describes the task to be performed. This element got one attribute – the name-attribute. This attribute simply specifies the name of the task. In the example above, this happens to be a task called "GetUser". The TASK-element must have two child elements called OBJECT and WHERE that can be used to specify arguments for the task. Note that not all tasks take arguments, but the elements are still required as they are part of the structure.

The OBJECT-element let the server know what type of object is affected by the execution and specifically which object fields that are affected. The name of the object is specified using the name-attribute and in the example above it happens to be a User-object. The server will read this attribute and pass on the task to the so called “server data object manager” that manages this kind of object. As previously described, by specifying child elements under the OBJECT element, the client can let the server know which fields to be affected. In the example, the client lets the server know that it is interested in getting the field called “userName” and the field called “userEmailAddress”. If this was an update-task, then this would instead mean that these two fields where to be updated. If that was the case, then also values would have been required for those fields (e.g. `<userName>John Smith</userName>`).

The WHERE-element specifies specific arguments for the task. In the example above, it’s simply used to let the server know that the user with id “123” should be fetched from the server.

Each task will have its own specification of valid child elements for the OBJECT and WHERE field.

RESULTSET-element

When a task have been sent to the server and executed, the server will return a responding XML-message. The server will by default return the execution-message sent to the server and add a so called RESULTSET-element that contains the returning data and/or result-message (error-, warning- or information-messages). The first part returned from the server will of course be a MapleProtocol-header, following the same structure described before. The header will let the client know how many bytes the returning message is. When the message has been read, it may look something like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION committed="true" client="Console" version="1.0">
  <TASK name="GetUser">
    <OBJECT name="User">
         <userName/>
         <userEmailAddress/>
    </OBJECT>
    <WHERE>
         <userId>123</userId>
    </WHERE>
    <RESULTSET>
      <DATASETS>
        <DATASET>
         <userName>John Smith</userName>
         <userEmailAddress>john@smith.com</userEmailAddress>
        </DATASET>
      </DATASETS>
    </RESULTSET>
  </TASK>
</EXECUTION>
```

As you can see, the message sent to the server will be returned, but a new element called RESULTSET has been attached. Also, on the EXECUTION-element a new attribute has been

created named committed. This is the primary way for the client to know whether a task has been successfully executed on the server side. The LocalizeDirect server is a fully transactional server (using the ACID rules), which means that a task can either only be fully performed (committed) or not at all (roll-backed). Specifically this applies to messages that modify (messages that only read data will not risk data inconsistency by failing).

A task can result in one or many DATASET-elements that will be found under the DATASETS-element. The datasets contains the data requested by the client as shown in the example above.

The server might also return task-specific error-, warning- or information-messages. These messages will be put under the RESULTSET-element in a element called RESULT, under which one or many MESSAGE-elements can exist. For example:

```
<RESULTSET>
   <DATASETS/>
   <RESULT>
      <MESSAGE id="1" type="Warning">No such user!</MESSAGE>
      <MESSAGE id="2" type="Warning">Other message!</MESSAGE>
   </RESULT>
</RESULTSET>
```

A returning message will always have a RESULTSET-element unless something has gone wrong, but it is not required to have the DATASETS- or RESULT-element. What can be returned by a task is described in the tasks API-description.

Logging In to the LocalizeDirect Server

In order to be able to use the functionality of LocalizeDirect Socket API, a user account needs to be created on the server. This needs to be done by an administrator that can access the server using a LocalizeDirect Client.

Sessions

When a user exist, the properties of that user can be used to login to the server. When a user has logged into the system, a session will be created. This session will remain until either the user has sent a logout-message or the session has timed-out.

Session Id (SecId)

On a successful login, the server will return a session id called “secId”. This session id, must after the login be set as an attribute on the EXECUTION-element each time a message sent to the server. For example:

```
<EXECUTION secId="123" client="Console" version="1.0">
</EXECUTION>
```

This way the server can verify that the user session is valid.

The Login-Task

The client must first log in to the server using a login-task:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION client="Console" version="1.0">
  <TASK name="Login">
    <OBJECT name="Security" />
    <WHERE>
      <userName>John</userName>
      <password>htimS</password>
    </WHERE>
  </TASK>
</EXECUTION>
```

In the example above the user with the name “John” will try to login using the password “htimS”. Assuming that the MapleProtocol header and the message have been correctly set then the server will respond with:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION committed="true" client="Console" version="1.0">
  <TASK name="Login">
    <OBJECT name="Security" />
    <WHERE>
      <userName>John</userName>
      <password>htimS</password>
    </WHERE>
    <RESULTSET>
      <DATASETS>
        <DATASET datatype="result">
          <secId>418EEBF8-D068-2137-8AB7-6443471BE836</secId>
          <moduleName>localizedirect-server</moduleName>
          <timeZone>Europe/Berlin</timeZone>
```

```
          <moduleVersion>2.6.0</moduleVersion>
        </DATASET>
      </DATASETS>
      <RESULT>
        <MESSAGE id="0" type="Information">The user 'John' has successfully
logged in!</MESSAGE>
      </RESULT>
    </RESULTSET>
  </TASK>
</EXECUTION>
```

The message in return confirms that the login was successful, by returning a RESULTSET-element with the session id (secId) and some additional fields. A result message has also been created that simply informs that the login was successful. The committed-attribute on the EXECUTION-element also indicates that the task was successfully committed on the server side (when the session was created).

If the user name or password would have been wrong, a result message would have been returned:

```
<RESULTSET>
  <DATASETS />
  <RESULT>
   <MESSAGE id="1" type="Error">Failed to login! User name or password is
invalid.</MESSAGE>
   </RESULT>
</RESULTSET>
```

After the user has logged in, the client can start to send other task-messages to the server. As previously described those task-messages will need to set the EXECUTION-element attribute “secId” (that was returned by the login-task).

If the user for some reason has been logged out or never has logged in, the server will respond with the following RESULTSET message:

```
<RESULTSET>
  <DATASETS />
  <RESULT>
   <MESSAGE id="10615" type="Error">You need to login to
continue!</MESSAGE>
  </RESULT>
</RESULTSET>
```

The Logout-Task

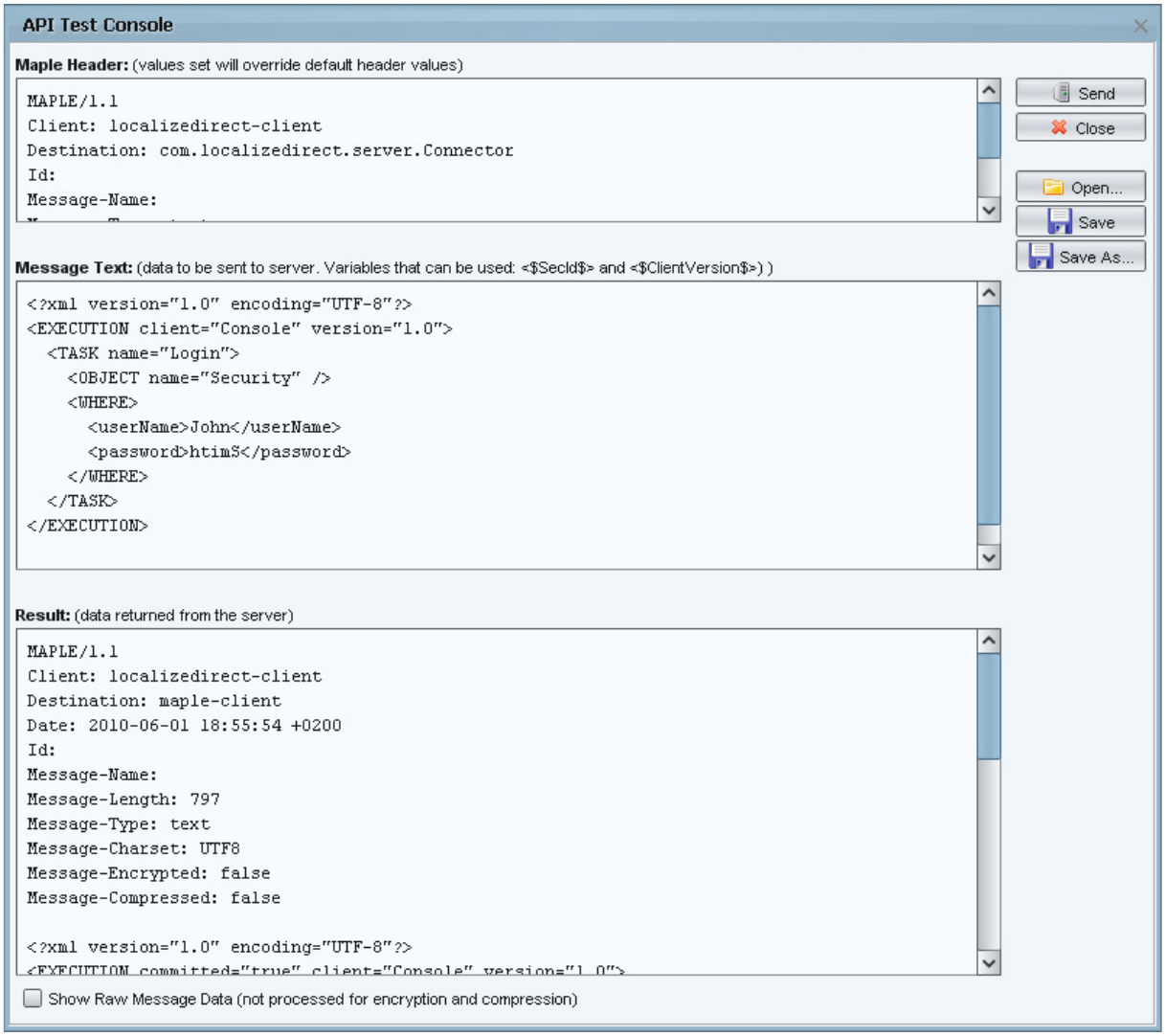
The client can log out from the server using the logout-task. This is not required as sending a login-task automatically will logout existing sessions for that user account, but if possible it’s a good idea to do this.

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION secId="418EEBF8-D068-2137-8AB7-6443471BE836" client="Console"
version="1.0">
  <TASK name="Logout">
    <OBJECT name="Security" />
    <WHERE>
      <userId>418EEBF8-D068-2137-8AB7-6443471BE836</userId>
    </WHERE>
  </TASK>
</EXECUTION>
```

The session id is specified as a userId under the WHERE-element. This can then be sent to the server and the session will end in a proper way.

Tools

The LocalizeDirect client has a built in tool to test the Socket API. This tool can be accessed from the Administration menu. From this tool it's possible to test the result of messages and change the attributes of the MapleProtocol:



It's also possible to load and save template messages, view the message data as raw data and of course change the message elements to whatever is required.