Clickworker Marketplace API

Application Programming Interface Description

Version: 2.0.43; 2017-03-13

Table Of Contents

Welcome

Clickworker's REST-based Web services allow customers to access the company's crowdsourcing services through an API and integrate them seamlessly into their applications.

The API has been designed to meet the needs of two distinct customer groups:
- Customers buying predefined products from the Clickworker Marketplace
- Customers uploading data for projects managed by the Clickworker solutions team.

Customers may use the API to order services offered in the Clickworker Marketplace. Clickworker offers a wide range of services, such as content creation, text translation and Internet research. Customers may use these services through the Web based marketplace portal on www.clickworker.com or by integrating them through the Clickworker Marketplace API. Most of our marketplace products can be customized to meet the specific data format requirements of customers, such as the number of fields to research. All products are based on Clickworker's leading edge quality control and quality assurance technology.

Customers who order custom projects managed by the Clickworker solutions department may use the API to integrate their IT systems with the Clickworker crowdsourcing platform in order to submit new tasks and review results on an ongoing basis.

Who Should Read This Guide

This document's primary audience is our clients' software developers. It aims to provide a brief description of each API entity and its available operations. Practical examples will be included to assist with implementation.

Required Knowledge and Skills

Familiarity with the following is recommended before using this guide:
- Basic understanding of the Extensible Markup Language (XML) or JavaScript Object Notation (JSON)
- Basic understanding of the Hypertext Transfer Protocol (HTTP)

Typing

The following conventions will be used throughout this guide:

- Arguments, in either the form of request parameters or response variables, are put inside curly braces and prefixed by a dollar sign ($). They use a unique name that corresponds to the associated explanation (Example: "${id}").
- References to API entities use square brackets, prefixed by a dollar sign (Example: $[Request Status]). If the reference points to a list containing any amount of instances of the referenced entity (or none at all, for that matter), an asterisk (*) is appended to the entity's name.
- Values for variables and parameters are put inside chevrons and prefixed by a dollar sign. If the value can be chosen from a predefined list of values, the pipe character is used. Examples: "$<GET|POST>".
- Patterns that need to be replaced by a meaningful substitute are put inside square brackets, like [context].

Typing

Change Log

Version 2.0.43 (March 2017)
- Added calls for iframe integration

Version 2.0.42 (October 2016)
- Added clickworker notifications and team mamangement

Version 2.0.41 (September 2016)
- Added bonus payments for clickworker

Version 2.0.40 (January 2016)
- Various document fixes

Version 2.0.30 (24th September 2013)
- Added search tasks by customer reference, state and creation date.
- Added ability to update of task template attributes.

Version 2.0.29 (7th August 2013)
- _page_size parameter was limited to 50 records by default. Handling exception when _page_number or _page_size were specified as blank.

Version 2.0.28 (13th February 2013)
- Correction to version number and title page.

Version 2.0.27 (7th February 2013)
- Describe the changes for supporting cloned task.

Version 2.0.26 (7th January 2013)
- Fixing error in example for task create operation (JSON version)

Version 2.0.25 (3rd January 2013)
- Fixing postal code, CEO and year of copyright

Version 2.0.24 (12th September 2012)
- Add author in notification

Version 2.0.23 (12th January 2012)
- Fixing of API-Documentation

Version 2.0.22 (24th October 2011)
- Fixing name of CEO

Version 2.0.20 (11th October 2011)
- Notifications is now optional in create task request.

- Changed update job json request to wrap the input body by a items hash.
- Renamed top node of index view xml representation from task to tasks.
- Changed representations of link from hash to array of hashes in task and job JSON responses.
- Updated XML and JSON create task template request and task template details response: added obligatory attribute textcreatewithkeywords_proof_read and changed value of attribute textcreatewithkeywords_text_length, changed is_output of result element to true and added a new element topic with is_output false. Set keyword element to is_output false. Added keyword element reference_code. Wrapped form body by new elements hash in JSON request.
- Renamed node task_template to task_templates in index view of task templates.
- Updated JSON representation of product attribute key "helps".
- Updated product link href and product code.

Version 2.0.3 (30th August 2011)
- Fixing Form Element reference

Version 2.0.3 (16th August 2011)
- Rewritten for API Version 2.0

Version 1.0.6 (3rd December 2010)
- Corrected spec for geospatial API calls

Version 1.0.5 (27th October 2010)
- Added geospatial support for customer projects

Version 1.0.4 (27th July 2010)
- Added support for additional POST parameters in notifications

Version 1.0.3 (14th July 2010)
- Call "create task" extended to allow specifying delivery call backs
- Notifications now include payload data (optional).
- Support for JSON and custom response format
- Corrected documentation to correctly show .xml extension in REST calls

Version 1.0.2 (1st April 2010)
- Added error codes
- Added task state descriptions

Version 1.0.1 (23rd March 2010)
- Extended call "task_info" to include the task progress history.
- Renamed the request parameter "url" to "customer_url" for call "add_notification"
- Call "product_details" returns a list of attributes required for task creation, as well as the list of job types that may require customer input.

Version 1.0 (2010)
- Initial Version

Key Concepts

Business Objects

Clickworker offers a wide range of services that enable data that is unsuitable for automatic processing to be handled by human workers. We call this "crowdsourcing." The data provided by the customer is converted into what we call "tasks" (page 45ff). Using a customizable workflow, jobs (page 68ff) are created for every task, representing the workflow's steps. Examples of jobs include "create text" and "review text." Clickworker then offers these jobs to registered users worldwide. These users are called "clickworkers".

Once all jobs are finished, the task is completed. The resulting data is then returned to the customer, who will be given the opportunity to verify and sign off on the result.[1]

Server/Client Communication

The Clickworker Marketplace API provides access to business objects (resources) in a "RESTful" manner:

- Resources are identified using a Uniform Resource Identifier (URI). They can be represented in various formats, like XML or JSON.
- HTTP "verbs" (e.g. GET or POST) are used to call available operations on these resources. The outcome of an operation is reported using HTTP status codes (e.g. "200 OK"). Additional information may be available in the response's body.
- Clients must authenticate themselves using HTTP authentication ("Basic Auth"). To ensure confidentiality, Transport Layer Security (TLS/SSL) is used.

A more in-depth description of the API principles is available on page 11ff.

Runtime Environments

To support testing of new clients and gather feedback for new API versions as early as possible, various environments host installations of the Marketplace API:

- **Production:** This is the production environment. It uses the latest released version of the Marketplace API, works with production data, and fully interacts with external systems, like email servers or partner gateways.

  The current **production** environment is available at
  https://api.clickworker.com/api/marketplace/v2/

- **Sandbox:** This is the testing environment. It uses the latest released version of the Marketplace API, works with dedicated test data, and has limited interaction with external systems, like email servers or partner gateways. Customers can use this

---

[1] Depending on the product and the customer's requirements.

environment to test their client implementations.

The current **sandbox** environment is available at
https://sandbox.clickworker.com/api/marketplace/v2/

- **Sandbox beta:** This is the staging environment. It uses the upcoming version of the Marketplace API. Because this is a beta environment, it may contain bugs. The sandbox beta environment works with dedicated test data and has limited interaction with external systems. Customers can use this environment to update their clients to the next version of the API.

The current **sandbox beta** environment is available at
https://sandbox-beta.clickworker.com/api/marketplace/v2/

Localization Support

The Marketplace API supports localization. Clients can send information about the preferred locale using standard HTTP headers (Accept-Language). The server will provide information about the content language using the Content-Language HTTP header.

Entities that support multiple languages during creation (as TaskTemplate does) make use of embedded XML elements or JSON value pairs, using the language code as the key.[2]

There will always be an English version available; all other languages are optional. For simplicity, English is the only language that will be used in examples demonstrating API entities.

Example:
- XML Representation

```
<titles>
 <de>Dies ist eine deutsche Nachricht</de>
 <en>This is an English message</en>
</titles>
```

- JSON Representation

```
titles: {
 de: "Die ist die deutsche Nachricht",
 en: "This is the English message"
}
```

---

[2] See http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt for details.

API Concepts

This chapter describes the API's key concepts. All available entities are listed in the following sections, along with the available operations to manipulate them.

Almost every entity supports at least some of the typical CRUD operations (Create, Read, Update, Delete). For every operation, the following details are given:

1. Request format and content, built from:
    a. The request line, according to the HTTP 1.1 protocol
    b. The request body description (if any)
2. Response format and content, built from:
    a. The main HTTP status codes (see section "Reporting status using HTTP codes" for all status codes)
    b. The response body content (if any)
3. Arguments that are used in the request (parameter) or the response (variables)

Context

All HTTP request lines used in this section make use of a special [context] marker. The context is the base part of every operation's URI. It is built from the host's name, the API version, and the API base path.

The [context] marker must be replaced by the following content:
https://api.clickworker.com/api/marketplace/v2/

Common Aspects

In general, a request that creates or modifies (updates) an entity must contain a valid representation of that entity (in XML or JSON format) in the request body. It must contain the writable attributes of the entity.

When one or more entities are part of a response body, they will be enclosed in a container element (XML element or anonymous JSON object), prefixed by a request status.

Attributes

Entities support a variety of attributes. The specific attributes are listed inside a table for each entity. Attributes that can be set using the API at creation time are marked as "r/c". Those that can be set at any time are marked as "r/w". All others are read-only ("r/-").

Attribute names are re-used wherever they serve the same purpose. The following table lists the most common attributes:

Name	Type	Access	Description
id	Integer	r/-	Auto-generated, environment-local resource identifier that cannot be changed. An id is specific to a given environment (production, sandbox, etc.). Example: A Job's id
href	URI	r/-	A fully qualified uniform resource identifier, pointing to a resource collection or a resource instance. This attribute is used to address or link to other resources.
code	String	r/c	A unique resource identifier that stays the same when porting an entity from one environment to another. The code must be URL compliant and can only be set at creation time. Example: A Product's code
name	String	r/w	A symbolic entity name that can be altered by an authorised user at any time. Unlike the "code" attribute, the name must not be URL compliant. The attribute's value is a single string that does not support localisation.
titles	Localized String	r/w	A descriptive entity title that can be altered at any time by an authorised user. Unlike the "name" attribute, titles do support localisation.
descriptions	Localized Text	r/w	An extensive, localizable description of the entity. Can be changed by an authorised user at any time.
helps	Localized Text	r/w	A context-aware description, providing further information on how to manage the current resource

Table 1: Common Attributes

Common Request Parameters

There is a set of URL parameters that control common aspects of all API operations or result representations.

The following table lists and explains them:

Name	Value(s)	Synopsis
_method	"PUT" or "DELETE"	Can be used to call update (PUT) or delete operations while using HTTP POST requests. This parameter is designed to be used by clients that are not capable of using HTTP verbs other than GET or POST
_suppress_http_status	"1"	Always report status code 200 ("OK") via the HTTP Status header. The real status is only available via a dedicated element in the response (see "Request Status" below). The default is "0"
_page_size	Integer value between 1 and 50	Results of index requests use pagination to prevent excessive transfer volume. This URL parameter can control the page size. The default

		is 50. More size will be adjusted to 50 as well.
_page_number	Positive, integer value	Defines the current page of a paginated result. E.g., a page size of 25 and a page number of 4 will display records from 76 to 100. The default is 1
api_method	Predefined string value	This parameter is used to call a specific API method that cannot be addressed solely by HTTP verbs.
filter_attribute	Predefined string value	Defines the entity attribute to use for filtering. See details of entity indexing requests to see a list of supported filter attributes.
filter_value	String	Value required for filter attribute to make the filter match

Table 2: Common Entity Attributes

Common Response Parameters

Every response contains a set of common parameters. Depending on whether a single resource (entity) or a collection was requested, either a single entity or a list of entities is returned. Every response will contain the request status, indicating whether the request was successful or resulted in an error.

If a single entity was created, updated, or requested for reading, this entity will be part of the response, in addition to the request status. For technical reasons, a wrapper element (XML element or JSON object) will be used for response bodies.

The naming conventions for the global container elements are
- [entity type] + "response" for single instance responses, e.g. "customer_response"
- [entity type] + "s_response" for multiple instance responses, e.g. "customers_response"

If the response contains multiple instances, no additional container elements will be used. Instead, every element's name will follow the convention [entity_type] + "s". For JSON, the actual entities in the collection will be anonymous JSON objects.

Examples
- XML Representation (Single Entity Response)
```
<task_response>
 <request_status>…</request_status>
 <task> … </task>
</task_response>
```
- XML Representation (Multiple Entities Response)
```
<tasks_response>
 <request_status>…</request_status>
 <tasks>…</tasks>
 <tasks>…</tasks>
```

```
</tasks_response>
```

- JSON Representation (Single Entity Response)

```
task_response: {
 request_status: { … },
 task: { … }
}
```

- JSON Representation (Multiple Entities Response)

```
tasks_response: {
 request_status: { … },
 tasks: [{
 ]}
}
```

Request Status

The Request Status provides the success or error information of the request. In general, it repeats the HTTP status code and contains additional, human-readable error messages. It also contains information about the pagination of the results.

Attributes

Name	Type	Access	Description
id	ID	r/-	A unique request id. Clients are encouraged to store this id for support and debugging purposes.
valid	Boolean	r/-	Request validity indicator. <true> if the request could be processed successfully
status_code	numeric	r/-	The HTTP status code
status_text	String	r/-	The equivalent HTTP status text
messages	List of text	r/-	A list of additional messages, to provide more in-depth information
total_count	Positive integer or 0	r/-	Reports the total number of entities matching the request. Will be zero for empty results.
page_num	Positive integer	r/-	The current page in a paginated result. Page numbering starts at page 1, even if the result is empty (total_count is zero)
page_size	Positive integer	r/	The current page size in a paginated result. The default value is 50, unless overridden by request parameter -_page_size.

Table 3: Request Status Attributes

<u>Representation</u>

- XML Representation

```
<request_status>
 <id>${id}</id>
 <valid>${valid}</valid>
 <status_code>${status_code}</status_code>
 <status_text>${status_text}</status_text>
 <messages>${message}</messages>
```

```
<total_count>${total_count}</total_count>
<page_num>${page_num}</page_num>
<page_size>${page_size}</page_size>
</request_status>
```

- JSON Representation

```
request_status: {
 id: "${id}",
 valid: ${valid},
 status_code: ${status_code},
 status_text: ${status_text},
 messages: [
  "${message}"
 ],
 total_count: ${total_count},
 page_size: ${page_size},
 page_num: ${page_num}
}
```

<u>Examples</u>

- XML Representation

```
<request_status>
 <id>az3n12j3h5b453j</id>
 <valid>false</valid>
 <status_code>409</status_code>
 <status_text>Conflict</status_text>
 <messages>The Task Template cannot be deleted. It is used by
the following Tasks: 1313, 1213, 1234!</messages>
 <total_count>0</total_count>
 <page_size>0</page_size>
 <page_num>1</page_num>
</request_status>
```

- JSON Representation

```
request_status: {
 id: "az3n12j3h5b453j",
 valid: true,
 status_code: 204,
 status_text: No Content,
 messages: [
  "Resource has been deleted."
 ],
 total_count: 0,
 page_size: 0,
 page_num: 1
}
```

API Usage

The following chapter lists all available entities and their operations. It is structured to match the typical development process, beginning with customer account handling (see below) and moving on to selection and configuration of products (page 16ff), creation of task templates (page 23ff) and tasks (page 53ff), and progress observation using jobs (page 69ff) and notifications (page 63ff).

Before performing specific API operations, developers should verify the general communication setup by using the two dedicated "echo" services, described on page 88ff.

For every entity, a short description, an attribute list (which excludes the common attributes on page 13) and a description of the available operations are given.

Customer

The Customer entity contains information related to the customer's account at clickworker.com, such as the current account balance. Customers order products using Task Templates and provide data to create Tasks from it.

All of the attributes exposed to the Customer entity are read-only (r/-). Their primary purpose is to allow account balance monitoring.

Attributes

Name	Type	Access	Description
balance_amount	Numeric	r/-	The money available on the customer's account
credit_limit	Numeric	r/-	The credit limit granted to the customer. It's always a numeric value greater than or equal to zero. Customers can only submit new orders and tasks as long as the cost for the task <= balance amount + credit limit.
reserved_amount	Numeric	r/-	The amount of money reserved for running tasks
currency_code	String	r/-	The customer's currency, based on ISO codes. This will be applied to all clickworker services requested by the customer.

Table 4: Customer Entity Attributes

Representation

XML Representation

```xml
<customer>
 <link href="${href}" rel="self" type="application/xml" />
 <balance>
  <balance_amount>${balance_amount}</balance_amount>
  <reserved_amount>${reserved_amount}</reserved_amount>
  <credit_limit>${credit_limit}</credit_limit>
  <currency_code>${currency_code}</currency_code>
 </balance>
</customer>
```

JSON Representation

```json
customer: {
 link: [{
  href: "${href}",
  rel: "self",
  type: "application/json"
 }],
 balance: {
  balance_amount: ${balance_amount},
  reserved_amount: ${reserved_amount},
  credit_limit: ${credit_limit},
  currency_code: "${currency_code}"
 }
}
```

Operations

Account information is returned for the customer account, which authenticates using the request's credentials. As a result, there is no customer id or name present in the URI.

Calling /customer without further information will print the customer's account data.

View Customer Account

Request
Request line:
        GET [context]/customer
Request parameter:
        (Common only)
Request body:
        (Empty)

<u>Response</u>

Response status:

- 200, if the resource was found
- 401, if the incoming request does not contain useable credentials
- 403, if the given credentials do not match a registered customer

Response body:

- XML Example

```
<customer_response>
 <request_status> […] </request_status>
 <customer>
    <link href="/api/marketplace/v2/customer" rel="self"
type="application/xml" />
  <balance>
    <balance_amount>20000.00</balance_amount>
    <reserved_amount>5000.00</reserved_amount>
    <credit_limit>3000.00</credit_limit>
    <currency_code>EUR</currency_code>
  </balance>
 </customer>
</customer_response>
```

- JSON Example

```
customer_response: {
 request_status: […],
 customer: {
  link: [{
   href: "/api/marketplace/v2/customer",
   rel: "self",
   type: "application/json"
  }],
  balance: {
   balance_amount: 20000.00,
   reserved_amount: 5000.00,
   credit_limit: 3000.00,
   currency_code: "EUR"
  }
 }
}
```

Product

A Product relates to a specific crowdsourcing service offered by Clickworker, such as text creation, translation, etc. There are two types of products:

1. Standard products: These are off-the-shelf, preconfigured products for direct use as offered via the Marketplace
2. Solutions products: These are custom services tailored for specific customer projects.

Using the Marketplace API, customers can order and use preconfigured Marketplace products or submit new tasks for custom products. In order to use a standard product, the customer must select the product using the appropriate resource URI and provide the additional parameters required for the chosen product, such as the target language for text creation.

Attributes

Name	Type	Access	Description
available	Boolean	r/-	<true>, if the product is currently available for ordering
attributes	List	r/-	A list of product attributes that are supported (and sometimes required) to configure the product. See the "Product Attribute" section for a formal description.

Table 5: Product Entity Attributes

Representation

The Product entity supports indexed and detailed views (see "Representing Resources Using XML or JSON" on page 85 for an explanation).

XML Representation
- Detail View

```
<product>
 <link href="[context]/products/${code}" rel="self"
  type="application/xml" />
 <code>${code}</code>
 <title>${title}</title>
 <description>${description}</description>
 <available>${available}</available>
 <attributes>
  <code>${code}</code>
  <type>${type}</type>
  <is_mandatory>${mandatory}</is_mandatory>
  <titles>
   <$<en|de|fr>>${title}</$<en|de|fr>
  </titles>
  <helps>
   <$<en|de|fr>>${help}</$<en|de|fr>
  </helps>
  <options>
   <titles>
    <$<en|de|fr>>${title}</$<en|de|fr>
   </titles>
   <value>${opt_value}</value>
  </options>
```

```
 </attributes>
</product>
```

- Index View

```
<product>
<link href="[context]/products/${code}" rel="self"
 type="application/xml" />
<code>${code}</code>
<titles>
 <$<en|de|fr>>${title}</$<en|de|fr>
</titles>
<available>${available}</available>
</product>
```

JSON Representation

- Detail View

```
product: {
 link: [{
  href: "[context]/products/${code}",
  rel: "self",
  type: "application/json"
 }],
 code: "${code}",
 titles: {
  "$<en|de|fr>": "${title}"
 },
 descriptions: {
  "$<en|de|fr>": "${description}"
 },
 available: ${available},
 attributes: [{
  "code": "${code}",
  "type": "${type}",
  "is_mandatory": ${is_mandatory},
  titles: {
   "$<en|de|fr>": "${title}"
  },
  help: "${title}",
  options: [{
   titles: {
    "$<en|de|fr>": "${title}"
   },
   value": "${opt_value}"
  }]
 }]
}
```

- Index View

```
{
 link: [{
```

```
 href: "[context]/products/${code}",
 rel: "self",
 type: "application/json"
}],
code: "${code}",
titles: {
 "$<en|de|fr>": "${title}"
},
available: ${available}
}
```

Operations

List Available Products

Displays a brief summary of each product. Note: This operation only lists products that are currently available. Deprecated products may still be accessed using their URI for reference in existing orders.

<u>Request</u>
Request line:
        GET [context]/products/
Request parameter:
        (Common only)
Request body:
        (Empty)

<u>Response</u>
Response status:
   •   200, if the request could be properly handled (even if no products were found)
Response body:
   •   XML Example

```
<products_response>
 <request_status>...</request_status>
 <products>
  <link href="/api/marketplace/v2/products/TextCreate"
rel="product"
   type="application/xml" />
  <code>TextCreate</code>
  <titles>
   <en>Text creation</en>
  </titles>
  <available>true</available>
 </products>
 <products>
  <link
href="/api/marketplace/v2/products/TextCreateWithKeywords"
rel="product" type="application/xml" />
```

```
 <code>TextCreateWithKeywords</code>
 <titles>
  <en>Text creation with keywords</en>
 </titles>
 <available>true</available>
</products>
</products_response>
```

- JSON Example

```
{
 request_status: ... ,
 products: [{
  link: [{
   href: "/api/marketplace/v2/products/TextCreate",
   rel: "product",
   type: "application/json"
  }],
  code: "TextCreate",
  titles: {
   en: "Text creation"
  },
  available: true
 },{
  link: [{
   href: "/api/marketplace/v2/products/TextCreateWithKeywords",
   rel: "product",
   type: "application/json"
  }],
  code: "TextCreateWithKeywords",
  titles: {
   en: "Text creation with keywords"
  },
  available: true
 }]
}
```

View Product Details

Request
Request line:
       GET [context]/products/${code}
Request parameter:
       (Common only)
Request body:
       (Empty)

Response
Response status:
- 200, if the addressed resource was found

- 404, if the addressed resource was not found

Response body:

- XML Example

```xml
<product_response>
 <request_status>...</request_status>
 <product>
  <link href="/api/marketplace/v2/products/product_research"
rel="self"
   type="application/xml" />
  <code>product_research</code>
  <titles>
   <en>Product Research</en>
  </titles>
  <descriptions>
   <en>Identify product details using a Internet search.</en>
  </descriptions>
  <available>true</available>
  <attributes>
   <code>research_language</code>
   <type>option</type>
   <is_mandatory>true</is_mandatory>
   <titles>
    <en>Research Language</en>
   </titles>
   <helps>
    <en>
    Defines the language to use for creating the research result.
    </en>
   </helps>
   <options>
    <titles>
     <en>German</en>
    </titles>
    <value>de</value>
   </options>
   <options>
    <titles>
     <en>English</en>
    </titles>
    <value>en</value>
   </options>
  </attributes>
 </product>
</product_response>
```

- JSON Example

```json
product_response: {
 request_status: ... ,
 product: {
```

```
  link: [{
   href: "/api/marketplace/v2/products/product_research",
   rel: "self",
   type: "application/json"
  }],
  code: "product_research",
  titles: {
   en: "Product Research"
  },
  descriptions: {
   en: "Identify product details using a Internet search."
  },
  available: true,
  attributes: [{
   code: "research_language",
   type: "option",
   is_mandatory: true,
   titles: {
    en: "Research Language"
   },
   helps: {
    en: "Defines the language to use for creating the research
result."
   },
   options: [{
    titles: {
     en: "German"
    },
    value: "de"
   },{
    titles: {
     en: "English"
    },
    value: "en"
   }]
  }]
 }
}
```

Product Attribute

Product Attributes provide meta information about parameters required for task submission.
Each product may require one or more additional parameters to be supplied when submitting
a task, such as the requested target language for text creation. Task Templates are used to
store sets of Product Attribute values along with other information such as the work
instructions.

Although every Product may have a dedicated set of required and optional Product Attributes, the Product Attribute itself is a standardized element.

Attributes

Name	Type	Access	Description
code	String	r/-	A symbolic name for the Product Attribute, URI compatible
type	String	r/-	Product Attribute type indicator. One of the following: • string: for text attributes • option: for predefined attribute values • integer: for numeric attributes • date: for time-based attributes
titles	Localized String	r/-	The localized title of this Product Attribute
helps	Localized Text	r/-	The localized help text for this Product Attribute
is_mandatory	Boolean	r/-	Mandatory indicator
min_length	numeric	r/-	Minimum text value length for type=string
max_length	numeric	r/-	Maximum text value length for type=string
options	List	r/-	A list of predefined attribute values, only present if type is "option." An option has two attributes: 1. The options value (opt_value) 2. The options title (opt_title), localized
value	String	r/w	The Product Attribute's value

Table 6: Product Attribute Entity Attributes

Representation

XML Representation

```
<attribute>
 <code>${code}</code>
 <type>${type}</type>
 <is_mandatory>${mandatory}</is_mandatory>
 <titles>
  <$<en|de|fr>>${title}</$<en|de|fr>>
 </titles>
 <helps>
  <$<en|de|fr>>${help}</$<en|de|fr>>
 </helps>
 <options>
  <titles>
   <$<en|de|fr>>${opt_title}</$<en|de|fr>>
  </titles>
  <value>${opt_value}</value>
 </options>
```

```
</attribute>
```

JSON Representation

```json
{
 "code": "${code}",
 "type": "${type}",
 "is_mandatory": ${is_mandatory},
 titles: {
  "$<en|de|fr>": "${title}"
 },
 helps: {
   "$<en|de|fr>": "${title}"
 },
 options: [{
  titles: {
   "$<en|de|fr>": "${title}"
  },
  value": "${opt_value}"
 }]
}
```

Clickworkers

Clickworkers are the people working and fulfilling the customer's tasks. In order to work on a task, a clickworker must meet specific prerequisites, such as being proficient in a specific language. In addition, clickworkers must pass a set of assessments and have gained a reasonable level of experience ("score"). Once they have a sufficient score, they may work on available jobs. For each product, clickworker.com controls which entry criteria must be met by a clickworker.

Due to privacy concerns, data about individual clickworkers cannot be made public via the Marketplace API.

Operations

Unconditional access to clickworkers is not permitted due to privacy concerns. However, filters may be applied to select certain subsets of the clickworker user base. Methods are provided to make sure there are clickworkers available to handle certain tasks. Calling these methods requires additional request parameters to be provided.

Count Clickworkers In Radius

Counts all clickworkers within a given radius. Its geographical center and radius describe the range. The minimum radius allowed is 10 kilometers (6.2 miles); smaller radiuses will be adjusted accordingly.

Request
Request line:

GET [context]/clickworkers?api_method=count_in_radius

&latitude=${latitude}&longitude=${longitude}&r=${radius}

Request body:

(empty)

Request parameters:

Name	Type	Synopsis	Mandatory
latitude	Numeric	The latitude of the geographical center	Yes
longitude	Numeric	The longitude of the geographical center	Yes
r	Integer	The radius in kilometers	Yes

Response

Response status:

- 200, if the request could be handled properly (even if no clickworkers were found)

Response body

- XML Example

```
<clickworkers_response>
 <request_status>...</request_status>
 <clickworker_count>9200</clickworker_count>
</clickworkers_response>
```

- JSON Example

```
clickworkers_response: {
 request_status: { … },
 clickworker_count: 9200
}
```

Count Clickworker in Bounding Box

Counts all clickworkers located within a given area. Its geographical boundaries (southwest and northeast corners) describe a square area. The minimum latitude / longitude difference of the two corner coordinates must be equivalent to an area of at least 10km^2 (3.9 square miles). If a smaller area is defined, it will maintain the southwest origin and expand.

Request

Request line:

GET [context]/clickworkers?api_method=count_in_bounding_box
&sw_latitude=${sw-lat}&sw_longitude=${sw-long}
&ne_latitude=${ne-lat}&ne_longitude=${ne-long}

Request parameters:

Name	Value(s)	Synopsis	Mandatory
sw_latitude	Numeric	The latitude of the box's southwest coordinate	Yes
sw_longitude	Numeric	The longitude of the box's southwest coordinate	Yes
ne_latitude	Numeric	The latitude of the box's northeast coordinate	Yes
ne_longitude	Numeric	The longitude of the box's northeast coordinate	Yes

Request body:

(Empty)

<u>Response</u>

Response status:

- 200, if the request could be handled properly (even if no clickworkers were found)

Response body

     (Identical to the response body of operation "Count Clickworkers In Radius")

Form Element

Presenting the customer's data to the clickworker requires a visual representation. Customers can control the output and apply rules to the input using a "Form." A Form is a container element living inside a Task Template or a Job (see pages 31 and 68). It defines Form Elements, which are used for rendering data or input fields in various formats, like HTML.

Every Form Element has a type. Currently, the following types are available:

- Single Line Text (type marker "text_field")
- Multi-Line Text (type marker "text_area")
- Numeric (type marker "number")
- Date (type marker "date")
- Multi media (type marker "media")
- Keyword (type marker "keyword")
- URL (type marker "url")
- E-Mail Address (type marker "email")
- Selection from a definable list of options in several visual representations
    - Drop down box (type marker "drop_box")
    - List box with multiple selection (type marker "multi_select")
    - Group of check boxes (type marker "check_box")
    - Group of radio buttons (type marker "radio_button")

Please note: Forms are presented in the clickworker's own language, if available. At the very least, customers must provide an English version of attributes like title and help.

Attributes

Name	Type	Access	Description
type	String	r/c	An indicator for the elements type. Supported values are: - text_field - text_area - number - keyword - url - email - select
code	String	r/-	The element's symbolic name. Used to identify the element in certain situations.
titles	Localized	r/w	The Form Element's title, as presented to the

	String		clickworker
descriptions	Localized Text	r/-	The description is only used by Job entities (see page 68), which reuse the concept of Form Elements.
is_output	Boolean	r/w	If set to <true>, the element will be used to output task data, if set to <false> it will store the received data to the Task.
is_mandatory	Boolean	r/w	The Form Element's "mandatory" indicator. This attribute must be set to <true> if the clickworker is required to enter a value for this element.
item_code	String	r/w	Allows the Task's data to be referenced by this element. If the element has set the "is_output" attribute to <true>, the data to display is fetched from the given column. If it's set to <false>, the column is populated by the element's value
options	Object	r/w	Options to configure the Form Element, depending on its type. Every option has two attributes: • The option's code (opt_code) • The option's value (opt_value)

Table 8: Form Element Entity Attributes

As stated above, every Form Element supports a set of options based on its type. The following sections list the available options for each type.

Representation

- XML Representation (Single Instance)

```xml
<elements>

<type>$<text_field|text_area|number|keyword|email|url|select></type>
 <titles>
  <$<en|fr|de>>${title}</$<en|fr|de>>
 </titles>
 <descriptions>
  <$<en|fr|de>>${description}</$<en|fr|de>>
 </descriptions>
 <item_code>${item_code}</item_code>
 <is_output>${is_output}</is_output>
 <is_mandatory>${is_mandatory}</is_mandatory>
 <options>
  <code>${opt_code}"></code>
  <value><![CDATA[${opt_value}]]></value>
 </options>
</elements>
```

- JSON Representation (Single Instance)

```
{
 type:
“$<text_field|text_area|number|keyword|email|url|select>”,
 titles: {
  “$<en|fr|de>”: “${title}”
 },
 descriptions: {
  “$<en|fr|de>”: “${description}”
 },
 item_code: “${item_code}”,
 is_output: ${is_output},
 is_mandatory: ${is_mandatory},
 options: [{
  code: “${opt_code}”,
  value: ${opt_value}
 }]
}
```

Options

Single Line Text ("text_field") Options

Code	Value	Description	Required
is_mandatory	Boolean	Defines whether this field must be filled out by the clickworker. The default is "false"	Yes
min_length	Integer	Minimum input length in characters	No
max_length	Integer	Maximum input length in characters	No

Table 9: Element "text_field" Options

Multi Line Text ("text_area") Options

Code	Value	Description	Required
is_mandatory	Boolean	Defines whether this field must be filled out by the clickworker. The default is "false"	Yes
min_length	Integer	Minimum input length in characters	No
max_length	Integer	Maximum input length in characters	No
is_richtext	Boolean	Defines, whether this field supports HTML rich text. The default is "false"	No

Table 10: Element "text_area" Options

Numeric ("number") Options

Code	Value	Description	Required
is_mandatory	Boolean	Defines whether this field must be filled out by the clickworker. The default is "false"	Yes
min_value	Integer	The minimum value allowed	No
max_value	Integer	The maximum value allowed	No

Table 11: Element "number" Options

Date ("date") Options

Code	Value	Description	Required
is_mandatory	Boolean	Defines whether this field must be filled out by the clickworker. The default is "false"	Yes
min_date	Date	Defines the earliest date that can be entered by the clickworker	No
max_date	Date	Defines the latest date that can be entered by the clickworker	No

Table 12: Element "date" Options

Multi media ("media") Options

Code	Value	Description	Required
is_mandatory	Boolean	Defines whether this field must be filled out by the clickworker. The default is "false"	Yes
media_player	String	Defines which media player should be used (at this time only "flowplayer" is supported)	No

Table 13: Element "media" Options

Keyword ("keyword") Options

Code	Value	Description	Required
is_mandatory	Boolean	Defines whether this field must be filled out by the clickworker. The default is "false"	Yes
min_occurrence_ref	String	The minimum keyword occurrence count is not set directly, but is instead read from the Task's data. This option defines the Task's data item code to read from.	No
max_occurrence_ref	String	The maximum keyword occurrence count is not set directly, but is instead read from the Task's data. This option defines the Task's data item code to read from.	No
reference_code	String	Defines the Task's data item where keywords are expected and will be counted. Enter the code of a data item here to check for keywords only in this item. Otherwise, leave blank or use the special value ":all" to scan for keywords in all suitable fields	No
density_unit	"percent" or "absolute"	Defines how the min/max occurrence values will be interpreted: • Using "percent" will interpret the value as a percent value. To use "5" as an example, the given keywords must make up 5% of the entire input • Using "absolute" will interpret the value as a fixed quantity. With an example of "5," the given keywords must occur 5 times in all scanned input fields.	No

Table 14: Element "keyword" Options

E-Mail ("email") Options

Code	Value	Description	Required
is_mandatory	Boolean	Defines whether this field must be filled out by the clickworker. The default is "false"	Yes

Table 15: Element "email" Options

URL ("url") Options

Code	Value	Description	Required
is_mandatory	Boolean	Defines whether this field must be filled out by the clickworker. The default is "false"	Yes
with_protocol	Boolean	Defines whether the user has to enter the protocol prefix (e.g. http://)	No

Table 16: Element "url" Options

Select ("drop_box", "multi_select", "check_box" and "radio_button") Options

Code	Value	Description	Required
is_mandatory	Boolean	Defines whether this field must be filled out by the clickworker. The default is "false"	Yes
alternatives	List	List of alternatives to pick from. Every alternative has two attributes: • The localized title • The alternatives value that will be used when setting the Form Element's value	Yes, at least two alternatives must be given

Table 17: Element "drop_box", "multi_select", "check_box", "radio_button" Options

XML Example

```xml
<elements>
 <type>drop_box</type>
 <titles>
   <en>Please choose</en>
   <de>Bitte wählen Sie</de>
 </titles>
 <item_code>selected_language</item_code>
 <options>
  <code>alternatives</code>
  <value>
    <titles>
      <en>German</en>
      <de>Deutsch</de>
    </titles>
    <value>de</value>
  </value>
  <value>
    <titles>
      <en>French</en>
      <de>Französisch</de>
    </titles>
    <value>fr</value>
  </value>
 </options>
</elements>
```

JSON Example

```json
{
 type: "drop_box",
 title: "Please select a language",
 item_code: "selected_language",
 options: [{
  code: "alternatives",
  value: [{
   titles: {
     en: "German",
```

```
    de: "Deutsch"
   },
   value: "de"
  },{
  titles: {
   en: "French",
   de: "Französisch"
   },
   value: "fr"
  }]
 }]
}
```

Task Template

In order to submit work to Clickworker through the API, the customer must provide information on how his request is structured and in which form results are expected back. For example, one customer might want to order a text using our TextCreate product, which should be structured using a title section and a content section, while another customer might need the text to be structured using title, abstract, and content sections.

Similarly, one customer might provide a list of URLs of company websites when using our product AddressResearch, while another customer might supply the company name and address for verification using the AddressResearch product.

TaskTemplates are used to define input and result structures, as well as to specify additional parameters required to fulfill a task, such as the target language for an article.
For each input and result field, additional options can be supplied to control the presentation of each field. Constraints, such as whether a field is optional or mandatory, can also be supplied.

To summarize, a Task Template serves multiple purposes:
- Select and configure an existing product by using its defined Product Attributes.
- Define how Tasks are presented to the clickworker using Form Elements
- Select or exclude clickworkers from certain Tasks using Clickworker List references.

Attributes

Name	Type	Access	Description
href	URI	r/-	The unique resource identifier
code	String	r/-	The symbolic Task Template code
name	String	r/w	A short text helping the customer identify the purpose of the Template
titles	Localized String	r/w	A short text used when presenting tasks to clickworkers (either as a list or in detail)
descriptions	Localized Text	r/w	A more extensive text containing a task summary that is presented to the clickworker.
product	Product	r/w	The referenced product and its configuration
form	Form	r/w	The embedded form and its configuration. See Form description (page ff) for details.
is_draft	Boolean	r/-	An indicator as to whether any tasks already use this Task Template.

Table 18: Task Template Entity Attributes

Representation

- XML Representation (Index view)

```
<task_templates>
 <link href="[context]/customer/task_templates/${code}"
rel="self"
   type="application/xml" />
 <name>${name}</name>
 <product>
  <link href="/api/marketplace/v2/products/${product_code}"
rel="product"
    type="application/xml" />
 </product>
 <is_draft>${is_draft}</is_draft>
</task_template>
```

- XML Representation (Detail View)

```
<task_templates>
 <link href="[context]/customer/task_templates/${code}"
rel="self"
  type="application/xml" />
 <code>${code}</code>
 <name>${name}</name>
 <titles>
  <$<en|fr|de>>${title}<$<en|de|fr>>
 </titles>
 <descriptions>
  <$<en|fr|de>>${description}</$<en|fr|de>>
 </descriptions>
 <product>
```

```
  <link href="[context]/products/${product_code}" rel="product"
    type="application/xml" />
  <attributes> … </attributes>
 </product>
 <form> … </form>
 <is_draft>${is_draft}</is_draft>
</task_template>
```

- JSON Representation (Index View)

```
task_templates: [{
 link: [{
  href: "[context]/customer/task_templates/${code}",
  rel: "self",
  type: "application/json"
 }],
 name: "${name}",
 product: {
  link: [{
   href: "[context]/products/${product_code}",
   rel: "product",
   type: "application/json"
  }],
 },
 is_draft: ${is_draft}
}]
```

- JSON Representation (Detail View)

```
task_template: {
 link: [{
  href: "[context]/customer/task_templates/${code}",
  rel: "self",
  type: "application/json"
 }],
 code: "${code}",
 name: "${name}",
 titles: {
  "$<de|en|fr|…>": "${title}"
 },
 descriptions: {
  "$<de|en|fr|…>": "${description}"
 },
 product: {
  link: [{
   href: "[context]/products/${product_code}",
   rel: "product",
   type: "application/json"
  }],
  attributes: [ … ]
 },
 form: [ … ],
```

```
  is_draft: ${is_draft}
}
```

Operations

Index Registered Task Templates

Indexing registered Task Templates will produce a brief overview.

Request

Request line:

> GET [context]/customer/task_templates/

Request parameter:

> (none)

Request body:

> (empty)

Response

Response status:

- 200, if the request was successfully processed, even if no Task Templates were found

Response body:

- XML Example

```
<task_templates_response>
 <request_status>...</request_status>
 <task_templates>
  <link
href="/api/marketplace/v2/customer/task_templates/translate"
rel="task_template" type="application/xml" />
  <name>Übersetzungstemplate</name>
  <product>
   <link href="/api/marketplace/v2/products/TextCreate"
rel="product" type="application/xml"/>
  <is_draft>false</is_draft>
 </task_templates>
</task_templates_response>
```

- JSON Representation

```
task_templates_response: {
 request_status: ... ,
 task_templates: [{
  link: [{
   href:
"/api/marketplace/v2/customer/task_templates/translate",
   rel: "task_template",
   type: "application/json"
  }],
  name: "Übersetzungstemplate",
  product: {
```

```
  link: [{
   href: "/api/marketplace/v2/products/TextCreate",
   rel: "product",
   type: "application/json"
  }]
 },
 is_draft: false
}]
}
```

Create a new Task Template

<u>Request</u>
Request line:
    POST [context]/customer/task_templates/
Request parameter:
    (none)
Request body:
*   JSON Example

```
task_template: {
 code: "tpl_text_create_keywords_en",
 name: "Standard Text Creation Template (English)",
 titles: {
  en: "Please write a message with keywords"
 },
 descriptions: {
  en: "You are required to write a short English message,
containing at least 50 words, with a keyword in it."
 },
 product: {
  link: [{
   href: "/api/marketplace/v2/products/TextCreateWithKeywords"
   rel: "product
   type: "application/json"
  }],
  attributes: [{
   code: "textcreatewithkeywords_language",
   value: "en"
  },{
   code: "textcreatewithkeywords_text_length",
   value: "10#55"
  },{
   code: "textcreatewithkeywords_proof_read",
   value: "textcreate_proof_no"
  }]
 },
 form: {
```

```
"elements": [
  {
     type: "text_field",
     titles: {
       en: "Topic"
     },
     item_code: "topic",
     is_output: false
  },{
     type: "text_area",
     titles: {
      en: "Your Message"
     },
     item_code: "result",
     is_output: true,
     is_mandatory: true,
     options: [{
      code: "min_length",
      value: "50"
     },{
      code: "max_length",
      value: "150"
     }],
  },{
     type: "keyword",
     item_code: "keyword_a",
     is_output: false,
     is_mandatory: true,
     options: [{
      code: "min_occurrence_ref",
      value: "keyword_a_min"
     },{
      code: "max_occurrence_ref",
      value: "keyword_a_max"
     },{
      code: "reference_code",
      value: "all"
     }]
  }
 ]
 }
}
```

Response

Response status:

- 200, if the Task Template was successfully created
- 409, if a Task Template of the given code already exists

Response body:

(See "View Task Template Details" on page 40 for examples)

View Task Template Details

Request

Request line:

GET [context]/customer/task_templates/${code}

Request parameter:

Name	Type	Synopsis	Mandatory
code	String	The Task Templates code as defined by the customer at the time of creation	Yes

Request body:

(empty)

Response

Response status:

- 200, if the request was successfully handled
- 404, if the addressed Task Template does not exist

Response body:

- XML Example

```xml
<task_template_response>
 <request_status>...</request_status>
 <task_template>
  <link href="/api/marketplace/v2/customer/task_templates/
tpl_text_create_keywords_en" rel="self" type="application/xml"
/>
 <code>tpl_text_create_keywords_en</code>
 <name>Standard Text Create Template (English)</name>
 <titles>
  <en>Please write a SEO message</en>
 </titles>
 <descriptions>
  <en>
  You are required to write a short English message, containing
at least
  50 words, with a keyword in it.
  </en>
 </descriptions>
 <product>
  <link
href="/api/marketplace/v2/products/TextCreateWithKeywords"
rel="product" type="application/xml"/>
  <attributes>
   <code>textcreatewithkeywords_language</code>
   <value>en</value>
  </attributes>
```

```xml
    <attributes>
     <code>textcreatewithkeywords_text_length</code>
     <value>10#55</value>
    </attributes>
    <attributes>
     <code>textcreatewithkeywords_proof_read</code>
     <value>textcreate_proof_no</value>
    </attributes>
 </product>
 <form>
  <elements>
     <type>text_field</type>
     <titles>
       <en>Topic</en>
     </titles>
     <item_code>topic</item_code>
     <is_output>false</is_output>
  </elements>
  </elements>
  <elements>
   <type>text_area</type>
   <titles>
    <en>Your Message</en>
   </titles>
   <item_code>result</item_code>
   <is_output>true</is_output>
   <is_mandatory>true</is_mandatory>
   <options>
    <code>min_length</code>
    <value>50</value>
   </options>
   <options>
    <code>max_length</code>
    <value>150</value>
   </options>
  </elements>
  <elements>
   <type>keyword</type>
   <titles>
    <en>Keyword A</en>
   </titles>
   <item_code>keyword_a</item_code>
   <is_output>false</is_output>
   <is_mandatory>true</is_mandatory>
   <options>
    <code>min_occurrence_ref</code>
    <value>keyword_a_min</value>
   </options>
```

```xml
   <options>
    <code>max_occurrence_ref</code>
    <value>keyword_a_max</value>
   </options>
   <options>
    <code>reference_code</code>
    <value>all</value>
   </options>
  </elements>
 </form>
 </task_template>
</task_template_response>
```

- JSON Example

```json
task_template_response: {
 request_status: { … },
 task_template: {
 link: [{
   href: "/api/marketplace/v2/customer/task_templates/
tpl_text_create_keywords_en",
   rel: "self",
   type: "application/json"
 }],
product: {
  link: [{
   href: "/api/marketplace/v2/products/TextCreateWithKeywords"
   rel: "product
   type: "application/json"
  }],
  attributes: [{
   code: "textcreatewithkeywords_language",
   value: "en"
  },{
   code: "textcreatewithkeywords_text_length",
   value: "10#55"
  },,{
   code: "textcreatewithkeywords_proof_read",
   value: "textcreate_proof_no"
  }]
 },
 form: {
   "elements": [
     {
       type: "text_field",
       titles: {
         en: "Topic"
       },
       item_code: "topic",
       is_output: false
```

```
      },{
        type: "text_area",
        titles: {
         en: "Your Message"
        },
        item_code: "result",
        is_output: true,
        is_mandatory: true,
        options: [{
         code: "min_length",
         value: "50"
        },{
         code: "max_length",
         value: "150"
        }],
      },{
        type: "keyword",
        item_code: "keyword_a",
        is_output: false,
        is_mandatory: true,
        options: [{
         code: "min_occurrence_ref",
         value: "keyword_a_min"
        },{
         code: "max_occurrence_ref",
         value: "keyword_a_max"
        },{
         code: "reference_code",
         value: "all"
        }]
      }
    ]
  }
}
```

Table 18

Update Task Template

Request

Request line:

PUT [context]/customer/task_templates/${code}

Request parameter:

Name	Type	Synopsis	Mandatory
code	String	The Task Templates code as defined by the customer at the time of creation	Yes

Request body:
• JSON Example

```
task_template: {
 name: "Standard Text Creation Template (English)",
 titles: {
  en: "Please write a message with keywords"
 },
 descriptions: {
  en: "You are required to write a short English message,
containing at least 50 words, with a keyword in it."
 }
}
```

Response

Response status:

- 200, if the Task Template was successfully updated
- 409, if a Task Template has invalid attributes

Response body:

(See "View Task Template Details" on page 40 for examples)

Delete a Task Template

When deleting a Task Template, it must not be in use by any Task (regardless of the Task's state).

Request

Request line:

DELETE [context]/customer/task_templates/${code}

or

POST [context]/customer/task_templates/${code}?_method=DELETE

Request parameter:

Name	Type	Synopsis	Mandatory
code	String	The Task Template's code, as defined by the customer at the time of creation	Yes

Request body:

(Empty)

Response

Response status:

- 204, if the request could be successfully handled
- 404, if the given resource was not found
- 409, if the Task Template is in use by Tasks.

Response body:

(Empty)

Add Clickworker to Task Template blacklist
Clickworkers on the blacklist are not allowed to work on this task. When removing the valid_until parameter the Clickworker will stay on the blacklist indefinitely.

<u>Request</u>
Request line:

PUT [context]/customer/task_templates/${code}/blacklist

Request parameter:

Name	Type	Synopsis	Mandatory
code	String	The Task Templates code as defined by the customer at the time of creation	Yes

Request body:
- JSON Example

```
blacklist: {
  clickworker_ids: [36, 40],
  valid_until: "2017-06-13"
}
```

<u>Response</u>
Response status:
- 200, if the Clickworkers were successfully added to the blacklist

Response body:

(Empty)

Task

A Task represents a specific work item (payload of work) to be processed and delivered by clickworkers, such as a test to be written, or a single address to be researched. Before a task can be submitted, a Task Template must be defined, specifying the details of how the task should be fulfilled. While the Task Template defines aspects that are common to all Tasks (like selected product, input form, etc.) a Task carries the actual payload. The Task's data can roughly be separated into two sections:

1. Instructional data is the input presented to the clickworker (like text to translate, keywords, instructions, etc.)
2. Result fields contain the final results of the task resolution (like translated text, created text, research results)

Attributes

Name	Type	Access	Description
id	Integer	r/-	The Task identifier
href	URI	r/-	The Task's unique resource identifier
Cloned_task	URI	r/-	Show the cloned task if presend
customer_ref	String	r/w	The customer may use this attribute to transport internal references.

template	URI	r/-	The Task Template's unique resource identifier. Contains a reference to the selected product.
amount	numeric	r/-	The amount after tax in the customer's currency for this task. Price may vary depending on the input parameters (e.g. length of text, quality assurance, etc.)
currency	String	r/-	The ISO currency symbol for all amount data
net_amount	numeric	r/-	The amount before tax in the customer's currency for this task. Price may vary depending on the input parameters (e.g. length of text, quality assurance, etc.)
tax_amount	numeric	r/-	The tax amount for this Task
state	String	r/w	The current Task state. See section "Task States" for details.
input	List	r/c	A list of items to be used as input data. Every item has two attributes: 1. The item's code (item_code) 2. The item's content (item_content)
result	List	r/-	A list of items to be used as result data. Every item has two attributes: 1. The item's code (item_code) 2. The item's content (item_content)
Notifications (optional)	List	r/c	A list of notification instances to automatically register after Task creation (see page 64).
progress_logs	List	r/-	A record of the Task's progress. Every entry provides a timestamp (log_timestamp) and a code (log_code), suitable for automatic processing.

Table 19: Task Entity Attributes

Task Result Additions

The result attribute contains input that was created by clickworkers using the Form Elements defined by the associated Task Template. In addition, service and statistical information **may** be added by the system.

Task States

The Task State indicates the current status of the Task. It is derived from a more extensive set of internal states but mapped to the following "public" states:

- Unconfirmed
  The task has been created but needs to be confirmed by the customer.
- Confirmed
  The task has been confirmed by the customer and is ready for queuing.
- Queued
  The task is waiting for clickworkers to apply for the associated jobs
- Running
  Clickworkers are currently working on the task's jobs.
- Cancelled
  The task has been cancelled by the customer

- Feedback
  The task requires feedback – such as a buyoff by the customer.
- Finished
  The task is finished and contains the result data.
- Deleted
  The task has been marked as deleted and will not be listed in index requests any more. However, it is still available as long as it is referenced by other entities, likes Jobs or Notifications.

Representation

- XML Representation (detail view)

```
<task>
  <link href="[context]/customer/tasks/${id}" rel="self"
    type="application/xml" />
 <id>${id}</id>
 <customer_ref>${customer_ref}</customer_ref>
 <template>
  <link href="[context]/customer/task_templates/${template}"
rel="template"
    type="application/xml" />
  <product>
   <link href="[context]/products/${product_code}" rel="product"
     type="application/xml" />
  </product>
 </template>
 <net_amount>${net_amount}</net_amount>
 <tax_amount>${tax_amount}</tax_amount>
 <amount>${amount}</amount>
 <currency>${currency}</currency>
 <state>${state}</state>
 <input>
  <items>
   <code>${item_code}</code>
   <content>${item_content}</content>
  </items>
 </input>
 <result>
  <items>
   <code>${item_code}</code>
   <content>${item_content}</content>
  </items>
 </result>
 <notifications>
  <link
href="[context]/customer/tasks/${task_id}/notifications/${id}"
    rel="notifications" type="application/xml" />
  <events>${event}</events>
```

```
  <events>TASK_COMPLETED</events>
  <callback_url>http://notification.example.com/</callback_url>
  <callback_method>POST</callback_method>
  <payload_format>XML</payload_format>
 </notifications>
 <progress_logs>
  <timestamp>${log_timestamp}</timestamp>
  <code>${log_code}</code>
 </progress_logs>
</task>
```

- XML Representation (index view)

```
<tasks>
 <link href="[context]/customer/tasks/${id}" rel="self"
   type="application/xml" />
 <customer_ref>${customer_ref}</customer_ref}
 <template>
  <link href="[context]/customer/task_templates/${template}"
rel="template"
    type="application/xml" />
  <product>
   <link href="[context]/products/${product_code}" rel="product"
     type="application/xml" />
  </product>
 </template>
 <state>${state}</state>
</tasks>
```

- JSON Representation (detail view)

```
{
 link: [{
  href: "[context]/customer/tasks/${id}",
  rel: "self",
  type: "application/json"
 }],
 id: ${id},
 customer_ref: "${customer_ref}",
 template: {
  link: [{
   href: "[context]/customer/task_templates/${template}",
    rel: "template",
    type: "application/json"
  }],
  product: {
   link: [{
    href: "[context]/products/${product_code}",
    rel: "product",
    type: "application/json"
   }]
```

```
  }
},
net_amount: ${net_amount},
tax_amount: ${tax_amount},
amount: ${amount},
currency: "${currency}",
state: "${state}",
input: {
 items: [{
  "code": "${item_code}",
  "content": "${item_content}"
 }]
},
result: {
 items: [{
  "code": "${item_code}",
  "content": "${item_content}"
 }]
},
notifications: [{
 link: [{
  href:
"[context]/customer/tasks/${task_id}/notifications/${id}",
  rel: "self",
  type: "application/json"
 }],
 event: "${event}",
 callback_url: "${callback_url}",
 callback_method: "${callback_method}",
 payload_format: "${payload_format}"
}],
progress_logs: [{
 timestamp: "${log_timestamp}",
 code: "${log_code}"
}]
}
```

- JSON Representation (index view)

```
{
link: [{
 href: "[context]/customer/tasks/${id}",
 rel: "self",
 type: "application/json"
}],
customer_ref: "${customer_ref}",
template: {
 link: [{
```

```
   href: "[context]/customer/task_templates/${template}",
   rel: "template",
   type: "application/json"
  }],
  product: {
   link: [{
    href: "[context]/products/${product_code}",
    rel: "product",
    type: "application/json"
  }]
  }
 },
 state: "${state}"
}
```

Operations

List Tasks

Request

Request line:

GET [context]/customer/tasks/

Request parameter:

Name	Type	Synopsis	Mandatory
state	String	Filters tasks by specified state. List of possible values, see above.	No
from	Date	Filters tasks by created_at attribute from specified date. Expected format is *dd.mm.yyyy*	No
to	Date	Filters task by created_at attribute to specified date. Expected format is *dd.mm.yyyy*	No
finished_from	Date	Filters finished tasks by finished_at attribute from specified date. Expected format is dd.mm.yyyy	No
finished_to	Date	Filters finished task by created_at attribute to specified date. Expected format is dd.mm.yyyy	No
order_id	Date	If you are using different projects under one account, these are separated by an order_id. On task creation the order id is returned.	No

Request body:

(empty)

Response

Response status

- 200, if the request could be successfully handled (even if no tasks were found)

Response body
- XML Example

```xml
<tasks_response>
 <request_status>...</request_status>
 <tasks>
  <link href="/api/marketplace/v2/customer/tasks/123" rel="task"
     type="application/xml" />
  <customer_ref>translate_run_0</customer_ref>
  <task_template>
   <link
href="/api/marketplace/v2/customer/task_templates/tt_default"
   rel="task_template" type="application/xml"/>
   <product>
    <link href="/api/marketplace/v2/products/TextTranslate"
rel="product" type="application/xml"/>
   </product>
  </task_template>
  <state>Running</state>
 </tasks>
 <tasks>
  <link href="/api/marketplace/v2/customer/tasks/456" rel="task"
     type="application/xml" />
  <customer_ref>translate_run_0</customer_ref>
  <task_template>
   <link
href="/api/marketplace/v2/customer/task_templates/tt_default"
   rel="task_template" type="application/xml"/>
   <product>
    <link href="/api/marketplace/v2/products/TextTranslate"
rel="product" type="application/xml"/>
   </product>
  </task_template>
  <state>Queued</state>
 </tasks>
</tasks_response>
```

- JSON Representation

```json
tasks_response: {
 request_status: ... ,
 tasks: [{
  link: [{
   href: "[context]/customer/tasks/123",
   rel: "task",
   type: "application/json"
  }],
  customer_ref: "translate_run_0",
  template: {
```

```
    link: [{
     href:
"/api/marketplace/v2/customer/task_templates/translate_en_de",
      rel: "task_template",
      type: "application/json"
     }],
    product: {
     link: [{
     href: "/api/marketplace/v2/products/TextTranslate",
     rel: "product",
     type: "application/json"
     }]
   },
   state: "Running"
  },{
   link: [{
    href: "[context]/customer/tasks/456",
    rel: "task",
    type: "application/json"
   }],
   customer_ref: "translate_run 0",
   template: {
    link: [{
     href:
"/api/marketplace/v2/customer/task_templates/translate_en_de",
      rel: "task_template",
      type: "application/json"
     }],
    product: {
     link: [{
      href: "/api/marketplace/v2/products/TextTranslate",
      rel: "product",
      type: "application/json"
     }]
    }
   },
   state: "Queued"
  }]
}
```

Search Tasks by Customer Reference

Request

Request line:

GET [context]/customer/tasks/search

Request parameter:

Name	Type	Synopsis	Mandatory
customer_ref	String	Filters tasks by specified	No

	customer_reference. If parameter is not specified, it filters tasks by customer_reference = NULL

Request body:
    (empty)

Response
Response status
   • 200, if the request could be successfully handled (even if no tasks were found)

Response body
   • XML Example

```xml
<tasks_response>
 <request_status>...</request_status>
 <tasks>
  <link href="/api/marketplace/v2/customer/tasks/123" rel="task"
     type="application/xml" />
  <customer_ref>translate_run_0</customer_ref>
  <created_at>2012-08-28T12:56:08+01:00</created_at>
  <task_template>
   <link
href="/api/marketplace/v2/customer/task_templates/tt_default"
   rel="task_template" type="application/xml"/>
   <product>
    <link href="/api/marketplace/v2/products/TextTranslate"
rel="product" type="application/xml"/>
   </product>
  </task_template>
  <state>Running</state>
 </tasks>
 <tasks>
  <link href="/api/marketplace/v2/customer/tasks/456" rel="task"
     type="application/xml" />
  <customer_ref>translate_run_0</customer_ref>
  <created_at>2012-08-28T12:50:01+01:00</created_at>
  <task_template>
   <link
href="/api/marketplace/v2/customer/task_templates/tt_default"
   rel="task_template" type="application/xml"/>
   <product>
    <link href="/api/marketplace/v2/products/TextTranslate"
rel="product" type="application/xml"/>
   </product>
  </task_template>
  <state>Queued</state>
 </tasks>
</tasks_response>
```

- JSON Representation

```
tasks_response: {
 request_status: ... ,
 tasks: [{
  link: [{
   href: "[context]/customer/tasks/123",
   rel: "task",
   type: "application/json"
  }],
  customer_ref: "translate_run 0",
  template: {
   link: [{
    href:
"/api/marketplace/v2/customer/task_templates/translate_en_de",
    rel: "task_template",
    type: "application/json"
   }],
   product: {
    link: [{
    href: "/api/marketplace/v2/products/TextTranslate",
    rel: "product",
    type: "application/json"
   }]
  },
  state: "Running",
  created_at: "2012-08-28T12:56:08+01:00"
 },{
  link: [{
   href: "[context]/customer/tasks/456",
   rel: "task",
   type: "application/json"
  }],
  customer_ref: "translate_run 0",
  template: {
   link: [{
    href:
"/api/marketplace/v2/customer/task_templates/translate_en_de",
    rel: "task_template",
    type: "application/json"
   }],
   product: {
    link: [{
     href: "/api/marketplace/v2/products/TextTranslate",
     rel: "product",
     type: "application/json"
    }]
   }
  },
```

```
   state: "Queued",
   created_at: "2012-08-28T12:50:01+01:00"
 }]
}
```

Create Task

A customer makes new work available by creating a task. Tasks are automatically confirmed and distributed to the crowd for resolution.

Customers can choose to receive notifications upon task completion.

Request

Request line:

POST [context]/customer/tasks/

Request parameter:

(Common only)

Request body:

- XML Example

```xml
<task>
 <customer_ref>task_batch 1</customer_ref>
 <template>
  <link

href="/api/marketplace/v2/customer/task_templates/translate_en_d
e" rel="task_template" type="application/xml" />
 </template>
 <input>
  <items>
   <code>source</code>
   <content>English Term</content>
  </items>
 </input>
 <notifications>
  <event>CUSTOMER_INPUT_REQUIRED</event>
  <callback_url>http://notification.example.com</callback_url>
  <callback_method>POST</callback_method>
  <payload_format>JSON</payload_format>
 </notifications>
</task>
```

- JSON Example

```json
task: {
 customer_ref: "task_batch 1",
 template: {
  link: [{
    href:
"/api/marketplace/v2/customer/task_templates/translate_en_de",
    rel: "task_template",
```

```
   type: "application/json"
  }]
 },
 input {
  items: [{
   code: "source",
   content: "English Term"
  }]
 },
 notifications: [{
  event: "CUSTOMER_INPUT_REQUIRED",
  callback_url: "http://notification.example.com/",
  callback_method: "POST",
  payload_format: "JSON"
 }]
}
```

<u>Response</u>

Response status

- 201, if the task has successfully been created
- 400, if the referenced Task Template does not exist

Response body

(See example in section "View Task Details")

View Task Details

<u>Request</u>

Request line:

GET [context]/customer/tasks/${id}

Request parameter:

Name	Type	Synopsis	Mandatory
id	Integer	The automatically generated Task id as returned by the Create Task operation	Yes
show_cloned_task	Boolean	If is true and the the task has cloned task, than the data of the cloned task will be shown.	No

Request body:

(Empty)

<u>Response</u>

Response status

- 200, if the requested resource was found
- 404, if the requested resource was not found or the given customer id is not associated with the detected credentials

Response body:

- XML Example

```
<task_response>
```

```
 <request_status>…</request_status>
 <task>
  <link href="/api/marketplace/v2/customer/tasks/123" rel="self"
   type="application/xml" />
  <id>123</id>
  <cloned_task>
   <link href="/api/marketplace/v2/customer/tasks/234"
rel="self" type="application/xml">
  </cloned_task>
  <customer_ref>translate batch a</customer_ref>
  <template>
   <link

href="/api/marketplace/v2/customer/task_templates/translate_en_d
e" rel="task_template" type="application/xml" />
   <product>
    <link href="/api/marketplace/v2/products/TextTranslate"
rel="product" type="application/xml"/>
   </product>
  </template>
  <net_amount>0.00</net_amount>
  <tax_amount>0.00</tax_amount>
  <amount>0.00</amount>
  <currency>EUR</currency>
  <state>Created</state>
  <input>
   <items>
    <code>source</code>
    <content>English Term</item>
   </items>
  </input>
  <result />
  <notifications />
  <progress_logs />
 </task>
</task_response>
```

- JSON Example

```
task_response: {
 request_status: …,
 task: {
  link: [{
   href: "/api/marketplace/v2/customer/tasks/123",
   rel: "self",
   type: "application/json"
  }],
  id: 123,
  cloned_task: {
   href: "/api/marketplace/v2/customer/tasks/234",
```

```
    rel: "self",
    type: "application/json"
   },
   customer_ref: "translate batch a",
   template: {
    link: [{
     href:
"/api/marketplace/v2/customer/task_templates/translate_en_de",
     rel="task_template",
     type="application/json"
    }],
    product: {
     link: [{
      href: "/api/marketplace/v2/products/TextTranslate",
      rel="product",
      type="application/json"
     }]
    }
   },
   net_amount: 0.00,
   tax_amount: 0.00,
   amount: 0.00,
   currency: "EUR",
   state: "Created",
   input: {
    items: [{
     code: "source",
     content: "English Term"
    }]
   },
   result: {},
   notifications: [],
   progress_logs: []
  }
}
```

Delete Task

Request
Request line
> DELETE [context]/customer/tasks/${ id}
> *or*
> POST [context]/customer/tasks/${ id}?_method=DELETE

Request parameter:

Name	Type	Synopsis	Mandatory

id	Integer	The auto-computed Task id as returned by the Create Task operation	Yes

Request body:

(empty)

<u>Response</u>

Response status:

- 204, if the request could be successfully handled
- 404, if the given resource was not found
- 409, if the task could not be deleted

Response body:

(empty)

Clickworker Bonus Payment

A customer has a way to make bonus payments to clickworkers.

NOTE: 40% of the amount will be added to the basic payment.

<u>Request</u>

Request line:

POST [context]/customer/tasks/${ id}/grant_bonus

Request parameter:

Name	Type	Synopsis	Mandatory
id	Integer	The auto-computed Task id as returned by the Create Task operation	Yes

Request body:

- XML Example

```
<bonus>
 <amount>30.0</amount>
 <clickworker_id>2</clickworker_id>
 <comment>Clickworker bonus payment</comment>
</bonus>
```

- JSON Example

```
bonus: {
 amount: 30.0,
 clickworker_id: 2,
 comment: "Clickworker bonus payment"
}
```

<u>Response</u>

Response status

- 201, if the payment has successfully been created
- 422, if the some validation error has occurred

Response body

(See example in section "View Task Details")

- XML Example

```xml
<grant_bonus_response>
  <request_status>
    <id>qcscPfWiEqKsOGLP1kfFVHQKJbQ=</id>
    <valid>true</valid>
    <status_code>201</status_code>
    <status_text>Created</status_text>
  </request_status>
  <bonus>
    <amount>100.0</amount>
    <clickworker_id>2</clickworker_id>
    <comment>Clickworker bonus payment</comment>
  </bonus>
</grant_bonus_response>
```

- JSON Example

```json
grant_bonus_response: {
 request_status: {
   id: "qcscPfWiEqKsOGLP1kfFVHQKJbQ=",
   valid: true,
   status_code: 201,
   status_text: "Created",
 },
 bonus: {
   amount: 30.0,
   clickworker_id: 2,
   comment: "Clickworker bonus payment"
 }
}
```

External form submit (used for iframe tasks)

When the task-form is hosted on your platform, we need a way to know when the clickworker has finished the task. As soon as the clickworker saves the form on the external platform, you should call this endpoint to let us know about the submit.

Request

Request line:

POST [context]/customer/tasks/${ id}/external_submit

Request parameter:

Name	Type	Synopsis	Mandatory
id	Integer	The auto-computed Task id as returned by the Create Task operation	Yes

Request body:

- XML Example

```xml
<submit>
```

```
    <clickworker_id>2</clickworker_id>
</submit>
```
- JSON Example

```
{submit: {
    clickworker_id: 2
}}
```

Response

Response status
- 201, if the result was successfully submitted
- 422, if the some validation error has occurred

Response body

    (See example in section "View Task Details")
- XML Example

```
<external_submit_response>
  <request_status>
    <id>qcscPfWiEqKsOGLP1kfFVHQKJbQ=</id>
    <valid>true</valid>
    <status_code>201</status_code>
    <status_text>Created</status_text>
  </request_status>
  <external_submit>
    <id>1000</id>
    <clickworker_id>2</clickworker_id>
  </external_submit>
</external_submit_response>
```
- JSON Example

```
external_submit_response: {
 request_status: {
    id: "qcscPfWiEqKsOGLP1kfFVHQKJbQ=",
    valid: true,
    status_code: 201,
    status_text: "Created",
 },
 external_submit: {
    id: 1000,
    clickworker_id: 2
 }
}
```

Accept the results of this task (used for iframe tasks)
Accept the results of this task. You can add an optional description.

Request
Request line:

    PUT [context]/customer/tasks/${ id}/accept

Request parameter:

Name	Type	Synopsis	Mandatory
id	Integer	The auto-computed Task id as returned by the Create Task operation	Yes

Request body:

- XML Example

```xml
<accept>
 <description>Well done</description>
</accept>
```

- JSON Example

```json
{accept: {
   description: "Well done"
}}
```

Response

Response status

- 201, if the result was successfully submitted
- 422, if the some validation error has occurred

Response body

(See example in section "View Task Details")

- XML Example

```xml
<accept_response>
  <request_status>
    <id>qcscPfWiEqKsOGLP1kfFVHQKJbQ=</id>
    <valid>true</valid>
    <status_code>201</status_code>
    <status_text>Created</status_text>
  </request_status>
  <accept>
    <id>1000</id>
  </accept>
</accept_response>
```

- JSON Example

```json
accept_response: {
 request_status: {
   id: "qcscPfWiEqKsOGLP1kfFVHQKJbQ=",
   valid: true,
   status_code: 201,
   status_text: "Created",
 },
 accept: {
   id: 1000
 }
}
```

Reject the results of this task (used for iframe tasks)

Reject the results of this task. The description is mandatory.

Request

Request line:

PUT [context]/customer/tasks/${ id}/reject

Request parameter:

Name	Type	Synopsis	Mandatory
id	Integer	The auto-computed Task id as returned by the Create Task operation	Yes

Request body:

- XML Example

```
<reject>
 <description>Well done</description>
</reject>
```

- JSON Example

```
{reject: {
   description: "Well done"
}}
```

Response

Response status

- 201, if the result was successfully submitted
- 422, if the some validation error has occurred

Response body

(See example in section "View Task Details")

- XML Example

```
<reject_response>
  <request_status>
    <id>qcscPfWiEqKsOGLP1kfFVHQKJbQ=</id>
    <valid>true</valid>
    <status_code>201</status_code>
    <status_text>Created</status_text>
  </request_status>
  <reject>
    <id>1000</id>
  </reject>
</reject_response>
```

- JSON Example

```
reject_response: {
 request_status: {
   id: "qcscPfWiEqKsOGLP1kfFVHQKJbQ=",
   valid: true,
   status_code: 201,
   status_text: "Created",
 },
 reject: {
```

```
    id: 1000
  }
}
```

Notify Clickworkers

With this call, you are able to notify clickworkers. Your emails will be delivered to workers as E-mail messages. Clickworker will receive a copy of this email. You may send a message to up to 100 clickworkers at a time.

<u>Request</u>

Request line:

POST [context]/customer/notify_workers

Request body:
- XML Example

```
<notify_workers>
 <subject>Subject</subject>
 <message>Message body</message>
 <clickworker_ids>1</clickworker_ids>
 <clickworker_ids>2</clickworker_ids>
 <clickworker_ids>3</clickworker_ids>
 <clickworker_ids>4</clickworker_ids>
</notify_workers>
```

- JSON Example

```
notify_workers: {
 subject: "Subject",
 message: "Message body",
 clickworker_ids: [1,2,3,4]
}
```

<u>Response</u>

Response status
- 201, if the notification has successfully been created
- 422, if the some validation error has occurred

Response body

(See example in section "View Task Details")
- XML Example

```
<notify_workers_response>
  <request_status>
    <id>qcscPfWiEqKsOGLP1kfFVHQKJbQ=</id>
    <valid>true</valid>
    <status_code>201</status_code>
    <status_text>Created</status_text>
  </request_status>
  <notify_workers>
    <subject>Subject</subject>
```

```
     <message>Message body</message>
     <clickworker_ids>1</clickworker_ids>
     <clickworker_ids>2</clickworker_ids>
     <clickworker_ids>3</clickworker_ids>
     <clickworker_ids>4</clickworker_ids>
  </notify_workers>
</notify_workers_response>
```

- JSON Example

```
notify_workers_response: {
 request_status: {
   id: "qcscPfWiEqKsOGLP1kfFVHQKJbQ=",
   valid: true,
   status_code: 201,
   status_text: "Created",
 },
 notify_workers: {
   subject: "Subject",
   message: "Message body",
   clickworker_ids: [1,2,3,4]
 }
}
```

Notification

Notifications are callbacks to the customer's system that keep customers informed of certain events related to a specific task. Notifications for the following events can be configured:

- TASK_COMPLETED: This event is triggered when all work related to the task, including the customer's review, has been completed.
- TASK_TIMEDOUT: This event is triggered when the maximum duration or due date for the task has elapsed and the task has not been completed.
- CUSTOMER_INPUT_REQUIRED: This event is triggered when additional input is required from the customer, such as approval of an article.

A notification can be registered during task creation.

Attributes

Name	Type	Access	Description
id	Integer	r/-	The Notification id
href	URI	r/-	The Notification unique resource identifier
event	String	r/w	The symbolic event name. The following codes are defined: • TASK_COMPLETED • TASK_TIMEDOUT • CUSTOMER_INPUT_REQUIRED
callback_url	URL	r/w	The callback URL
callback_method	String	r/w	The HTTP method, one of GET or POST

payload_format	String	r/w	Payload format indicator, either XML or JSON. Only supported if method is set to POST.
			The actual payload will be a copy of the selected Task entity in the requested format.

Table 20: Notification Entity Attributes

Representation

- XML Representation

```
<notification>
 <link
href="[context]/customer/tasks/${task_id}/notifications/${id}"
   rel="self" type="application/xml" />
 <event>${event}</event>
 <callback_url>${callback_url}</callback_url>
 <callback_method>${callback_method}</callback_method>
 <payload_format>${payload_format}</payload_format>
</notification>
```

- JSON Representation

```
notification: {
 link: [{
  href:
"[context]/customer/tasks/${task_id}/notifications/${id}",
  rel: "self",
  type: "application/json"
 }],
 event: "${event}",
 callback_url: "${callback_url}",
 callback_method: "${callback_method}",
 payload_format: "${payload_format}"
}
```

Notification Payload

The Notification sent to the given "callback url" will contain the following information:
1. The event code, as described above
2. The URI of the task that triggered the event

If ${callback_method} was set to POST, the Notification payload will be the only element in the body of the request. If the method was set to GET, the notification will be a URL-compliant serialization of the data and become the value of a query string parameter named "payload".

Please note that because of URL length restrictions, the use of the POST method is highly recommended!

- XML Example

```xml
<notification>
 <link href="/api/marketplace/v2/customer/notifications/1234"
rel="self"
    type="application/xml" />
 <event>CUSTOMER_INPUT_REQUIRED</event>
 <task>
  <link href="/api/marketplace/v2/customer/tasks/1234"
rel="task"
      type="application/xml" />
 </task>
</notification>
```

- JSON Example

```json
notification: {
 link: [{
  href: "/api/marketplace/v2/customer/notifications/1234",
  rel: "self",
  type: "application/json"
 }],
 event: "CUSTOMER_INPUT_REQUIRED",
 task: {
  link: [{
   href: "/api/marketplace/v2/customer/tasks/1234",
   rel: "task",
   type: "application/json"
  }]
 }
}
```

Operations

List Registered Notifications
There are two ways to index Notifications associated with the customer:
1. View all Notification instances that are associated with the customer
2. View only Notifications that are associated with a specific task

Request

Request line:

GET [context]/customer/tasks/${id}/notifications/
or
GET [context]/customer/notifications/

Request parameter:

Name	Type	Synopsis	Mandatory
id	Integer	The auto-generated Task id, as returned by the Create Task operation	Yes

Request body:
(empty)

Response

Response status:
- 200, if the request could be handled successfully (even if there are no notifications)
- 404, if the addressed Task does not exist

Response body:
- XML Example

```
<notifications_response>
 <request_status>...</request_status>
 <notifications>
  <link
href="/api/marketplace/v2/customer/tasks/1234/notifications/1"
    rel="self" type="application/xml" />
  <event>TASK_COMPLETED</event>
  <callback_url>http://notification.example.com/</callback_url>
  <callback_method>POST</callback_method>
  <payload_format>XML</payload_format>
  <author_id>111</author_id>
  <task_id>1234</task_id>
 </notifications>
</notifications_response>
```

- JSON Example

```
notification_response: {
 request_status: ... ,
 notifications: [{
  link: [{
   href:
"/api/marketplace/v2/customer/tasks/1234/notifications/1",
   rel: "self",
   type: "application/json"
  }],
  event: "TASK_COMPLETED",
  callback_url: "http://notification.example.com/",
  callback_method: "POST",
  payload_format: "JSON",
  author_id: "111",
  task_id: "1234"
 }]
}
```

Job

Whenever the customer confirms a new task, it is accomplished based on a predefined, product-specific workflow. The steps of this workflow are called "Jobs." This is illustrated by the following workflow, consisting of three Jobs:

1. "create" performed by a clickworker writing a text according to a task's specifications
2. "review" performed by another clickworker who reviews the results of the first job

3. "signoff" performed by the customer, who approves or declines the results.

A task is completed when the entire workflow is completed. This occurs when all jobs are finished.

Using the Marketplace API, customers can only see and manipulate jobs that are assigned to them. Usually, these are jobs for signing off on the results. The product details contain the relevant meta information about the job's input and results.

Attributes

Name	Type	Access	Description
href	URI	r/-	The Job resource identifier
id	String	r/-	The Job Id
task	Task	r/-	The Task with which this Job is associated
customer_ref	String	r/-	Copied from the associated Task
items	List	r/-	The Job's payload, a combination of the task input data provided by the customer and the clickworker output
form	List	r/-	A list of Form Elements that visualize the required and supported input items (see below)
input	List	r/w	A list of items supplied for job completion. The purpose, format, and the available alternatives (if any) are described by the form attribute above.

Table 21: Job Entity Attributes

Representation

- XML Representation (Detail View)

```
<job>
 <link href="[context]/customer/jobs/${id}" rel="self"
   type="application/json" />
 <id>${id}</id>
 <task>
  <link href="[context]/customer/tasks/${task}" rel="task"
type="application/xml" />
 </task>
 <items>
  <code>${item_code}</code>
  <content>${item_content}</content>
 </items>
 <form>
  <elements>

<type>$<text_field|text_area|number|keyword|email|url|select></type>
```

```
 <titles>
  <$<en|fr|de>>${title}</$<en|fr|de>>
 </titles>
 <descriptions>
  <$<en|fr|de>>${description}</$<en|fr|de>>
 </descriptions>
 <item_code>${item_code}</item_code>
 <is_output>${is_output}</is_output>
 <is_mandatory>${is_mandatory}</is_mandatory>
 <options>
  <code>${opt_code}"></code>
  <value><![CDATA[${opt_value}]]></value>
 </options>
</elements>
 </form>
 <input>
  <items>
   <code>${item_code}</code>
   <content>${item_content}</content>
  </items>
 </input>
</job>
```

- XML Representation (Index View)

```
<job>
 <link href="[context]/customer/jobs/${id}" rel="self"
   type="application/xml" />
 <task>
  <link href="[context]/customer/tasks/${task}" rel="task"
type="application/xml"/>
 </task>
</job>
```

- JSON Representation (Detail View)

```
job: {
 link: [{
  href: "[context]/customer/jobs/${id}",
  rel: "self",
  type: "application/json"
 }],
 id: ${id},
 task: {
  link: [{
   href: "[context]/customer/tasks/${task}",
   rel: "task",
   type: "application/json"
  }]
 },
 items: [{
```

```
   code: "${item_code}",
   content: "${item_content}"
 }],
 form: [{
 type:
"$<text_field|text_area|number|keyword|email|url|select>",
 titles: {
  "$<en|fr|de>": "${title}"
 },
 descriptions: {
  "$<en|fr|de>": "${description}"
 },
 item_code: "${item_code}",
 is_output: ${is_output},
 is_mandatory: ${is_mandatory},
 options: [{
  code: "${opt_code}",
  value: ${opt_value}
 }]
}],
 input: {
  items: [{
    code: "${item_code}",
    content: "${item_content}"
  }]
 }
}
```

• JSON Representation (Index View)

```
job: {
 link: [{
  href: "[context]/customer/jobs/${id}",
  rel: "self",
  type: "application/json"
 }],
 task: {
  link: [{
   href: "[context]/customer/tasks/${task}",
   rel: "task",
   type: "application/json"
  }]
 }
}
```

Operations

Index Registered Jobs
There are two ways to index Jobs associated with the customer's account:

1. View all Job instances that are assigned to a customer
2. Only view Jobs that are associated with a specific Task

Request

Request line:

> GET [context]/customer/jobs/
>
> *or*
>
> GET [context]/customer/tasks/${id}/jobs/

Request parameter:

Name	Type	Synopsis	Mandatory
id	Integer	The auto-generated Job id.	Yes

Request body:

> (empty)

Response

Response status:

- 200, if the index was built (even if it does not contain any Jobs)
- 404, if the addressed Task does not exist

Response body:

- XML Example

```
<jobs_response>
 <request_status>...</request_status>
 <jobs>
  <link href="/api/marketplace/v2/customer/jobs/1234" rel="job"
    type="application/xml" />
  <task>
   <link href="/api/marketplace/v2/customer/tasks/5678"
rel="task" type="application/xml" />
  </task>
 </jobs>
</jobs_response>
```

- JSON Example

```
jobs_response: {
 request_status: { … } ,
 jobs: [{
  link: [{
   href: "/api/marketplace/v2/customer/jobs/1234",
   rel: "job",
   type: "application/json"
  }],
  task: [{
   href: "/api/marketplace/v2/customer/tasks/5678",
   rel: "task",
   type: "application/json"
  }]
 }]
}
```

View Job Details

Request

Request line:

GET [context]/customer/tasks/${task_id}/jobs/${id}

Request parameter:

Name	Type	Synopsis	Mandatory
task_id	Integer	The auto-generated Task id, as returned by the Create Task operation	Yes
id	Integer	The auto-generated Job id, as returned by the Create Notification operation	Yes

Request body:

(empty)

Response

Response status:

- 200, if the Job was found
- 404, if the Job or the Task was not found

Response body:

- XML Example

```
<job_response>
 <request_status>...</request_status>
 <job>
  <link href="/api/marketplace/v2/customer/jobs/1234" rel="self"
    type="application/xml" />
  <id>1234</id>
  <task>
   <link href="/api/marketplace/v2/customer/tasks/5678"
rel="task" type="application/xml" />
  </task>
  <items><code>source</code><content>English
Term</content></items>
  <items><code>target</code><content>Deutscher
Begriff</contents></items>
 </job>
</job_response>
```

- JSON Example

```
job_response: {
 request_status: ... ,
 job: {
  link: [{
   href: "/api/marketplace/v2/customer/jobs/1234",
   rel: "self",
   type: "application/json"
  }],
  id: 1234,
  task: {
   link: [{
```

```
    href: "/api/marketplace/v2/customer/tasks/456"
    rel: "task"
    type: "application/json"
   }]
  },
  items: [{
   code: "source",
   content: "English Term"
  },{
   code: "target",
   content: "deutscher Begriff"
  }]
 }
}
```

Update Jobs

To update a Job, the input attribute's content must match the structure of the embedded form.

Request

Request line:

PUT [context]/customer/jobs/${id}

or

POST [context]/customer/jobs/${id}?_method=PUT

Request parameter:

Name	Type	Synopsis	Mandatory
task_id	Integer	The auto-generated Task id, as returned by the Create Task operation	Yes
id	Integer	The auto-generated Job id.	Yes

Request body:
- XML Example

```
<job>
 <input>
  <items>
   <code>accepted</code>
   <content>1</content>
  </items>

  <!-- evaluation points for grammar: 4-no error, 3-less erorrs,
2-many errors, 0-not acceptable-->

  <items>
   <code>grade_gr</code>
   <content>2</content>
```

```xml
  </items>

  <!-- comment for grammar -->
  <items>
   <code>grade_gr_comment</code>
   <content>Here is my grade_gr comment</content>
  </items>

  <!-- evaluation points for spelling and punctuation: 4-no
error, 3-less erorrs, 2-many errors, 0-not acceptable-->

  <items>
   <code>grade_sppu</code>
   <content>2</content>
  </items>

  <!-- comment for für grade_sppu-->
  <items>
   <code>grade_sppu_comment</code>
   <content>Here is my spelling and punctuation
            comment
   </content>
  </items>


  <!-- evaluation points for style: 4-no error, 3-less erorrs,
2-many errors, 0-not acceptable-->

  <items>
   <code>grade_sest</code>
   <content>2</content>
  </items>

  <!-- comment for für style -->
  <items>
   <code>grade_sest_comment</code>
   <content>Here is my style comment</content>
  </items>


  <!-- evaluation points for sentence structure: 4-no error, 3-
less erorrs, 2-many errors, 0-not acceptable-->
  <items>
   <code>grade_st</code>
   <content>2</content>
  </items>

  <!-- comment for für sentence structure -->
```

```xml
  <items>
   <code>grade_st_comment</code>
   <content>Here is my sentence structure comment </content>
  </items>
  <!-- evaluation points for comprehension and clarity: 4-no
error, 3-less erorrs, 2-many errors, 0-not acceptable-->
  <items>
   <code>grade_cocl</code>
   <content>2</content>
  </items>

  <!-- comment for für comprehension and clarity -->
  <items>
   <code>grade_cocl_comment</code>
   <content>Here is my comprehension and clarity comment
   </content>
  </items>

  <!-- evaluation points for follow instructions: 4-no error, 3-
less erorrs, 2-many errors, 0-not acceptable-->
  <items>
   <code>grade_foin</code>
   <content>2</content>
  </items>

  <!-- comment for für follow instructions -->
  <items>
   <code>grade_foin_comment</code>
   <content>Here is my follow instructions comment </content>
  </items>

  <!-- evaluation points for selection of information: 4-no
error, 3-less erorrs, 2-many errors, 0-not acceptable-->
  <items>
   <code>grade_sein</code>
   <content>2</content>
  </items>

  <!-- comment for für selection of information -->
  <items>
   <code>grade_sein_comment</code>
   <content>Here is my selection of information comment
   </content>
  </items>

  <!-- Your comment to author. Required if text is rejected. -->
  <items>
   <code>comment</code>
```

```
      <content>Text is ok.</content>
   </items>

 </input>
</job>
```

- JSON Example

```
job: {
 input: {
  items: [{
    code: "accepted",
    content: "1"
   },{
    code: "grade_gr",
    content: "2"
   },{
    code: "grade_gr_comment",
    content: "Here is my grade_gr comment"
   },{
    code: "grade_sppu",
    content: "2"
   },{
    code: "grade_sppu_comment",
    content: "Here is my spelling and punctuation comment"
   },{
    code: "grade_sest",
    content: "2"
   },{
    code: "grade_sest_comment",
    content: "Here is my style comment"
   },{
    code: "grade_st",
    content: "2"
   },{
    code: "grade_st_comment",
    content: "Here is my sentence structure comment "
   },{
    code: "grade_cocl",
    content: "2"
   },{
    code: "grade_cocl_comment",
    content: "Here is my comprehension and clarity comment"
   },{
    code: "grade_foin",
    content: "2"
   },{
    code: "grade_foin_comment",
    content: "Here is my follow instructions comment "
   },{
```

```
    code: "grade_sein",
    content: "2"
   },{
    code: "grade_sein_comment",
    content: "Here is my selection of information comment"
   },{
    code: "comment",
    content: "Text is ok."
   }
  ]
 }
}
```

Response

Response status:
- 200, if the request was successfully handled
- 404, if the Job does not exist or the Task does not exist.
- 409, if the input items provided by the customer do not match the form definition

Response body:
    (Identical to "View Job Details," as described above)

Clickworker Teams

Teams allow to group clickworkers and include or exclude them from your projects.
Teams are typically used in the following scenarios:
- Excluding clickworkers that have already participated in a previous survey.
- Only addressing specific clickworkers with a survey for follow up surveys
- Excluding authors from your text project (blacklisting)
- Assigning text projects to a list of known authors that have worked for you previously

Attributes

Name	Type	Access	Description
id	Integer	r/-	The Team identifier
name	String	r/w	The Team's unique resource identifier
note	String	r/w	Notes describing this team
hide	Boolean	r/w	When true, the team is hidden from drop-down lists in the marketplace (e.g. in order to hide teams no longer used)

Operations

Create team

Request

Request line:
    POST [context]/customer/teams

Request parameter:

Name	Type	Synopsis	Mandatory
name	String	The team name	Yes
note	String	A note describing the team	No
hide	Boolean	Hide team from drop down lists when true	No

Request body:
    (empty)

Response

Response status
- 200, if the request could be successfully handled (even if no tasks were found)

Request body:
- XML Example

```
<team>
 <name>My first team</name>
 <note>This is my first team</note>
 <hide>false</hide>
</team>
```

- JSON Example

```
team: {
 name: "My first team",
 note: "This is my first team",
 hide: "false"
}
```

Response

Response status
- 201, if the team has successfully been created

Response body
    (See example in section "View Team")

Delete team

Request

Request line
    DELETE [context]/customer/teams/${ id}
    or
    POST [context]/customer/teams/${ id}?_method=DELETE

Request parameter:

Name	Type	Synopsis	Mandatory
id	Integer	The auto-computed Team id as returned by the Create Team operation	Yes

Request body:
    (empty)

Response

Response status:

- 204, if the request could be successfully handled
- 404, if the given resource was not found
- 409, if the task could not be deleted

Response body:

(empty)

Update team details

Request

Request line:

PUT [context]/customer/teams/${ id}

Request parameter:

(Common only)

Request body:

- XML Example

```xml
<team>
 <name>My first team</name>
 <note>This is my first team</note>
 <hide>false</hide>
</team>
```

- JSON Example

```json
team: {
 name: "My first team",
 note: "This is my first team",
 hide: "false"
}
```

Response

Response status

- 200, if the request could be successfully handled (even if no tasks were found)

Response body

- (See example in section "View Team Details")

List teams

Request

Request line:

GET [context]/customer/teams/

Request parameter:

(Common only)

<u>Response</u>

Response body:

-            XML Example

```xml
<teams_response>
  <request_status>
    <id>wybSKEcLKGBNbd+LKX7zXIsvVJc=</id>
    <valid>true</valid>
    <status_code>200</status_code>
    <status_text>OK</status_text>
    <total_count>5</total_count>
    <page_num>1</page_num>
    <page_size>50</page_size>
  </request_status>
  <teams>
    <id>12242</id>
    <name>Content Managers</name>
  </teams>
  <teams>
    <id>232304</id>
    <name>Interpreters</name>
  </teams>
</teams_response>
```

- JSON Representation

```json
teams_response: {
 request_status: ... ,
 teams: [{
  link: [{
   href: "[context]/customer/teams/12242",
   rel: "team",
   type: "application/json"
  }],
  name: "Content Managers"
  created_at: "2012-08-28T12:56:08+01:00"
 },{
  link: [{
   href: "[context]/customer/teams/232304",
   rel: "team",
   type: "application/json"
  }],
  name: "Interpreters"
  created_at: "2012-08-28T12:56:08+01:00"
}
```

Response status

- 200, if the request could be successfully handled (even if no teams were found)

Add clickworkers to a team

Request

Request line:

PUT [context]/customer/teams//${ id}

Request parameter:

(Common only)

Request body:

- XML Example

```
<team>
  <add_user_ids>[USER_ID1]</add_user_ids>
  <add_user_ids>[USER_ID2]</add_user_ids>
  <add_user_ids>[USER_ID3]</add_user_ids>
</team>
```
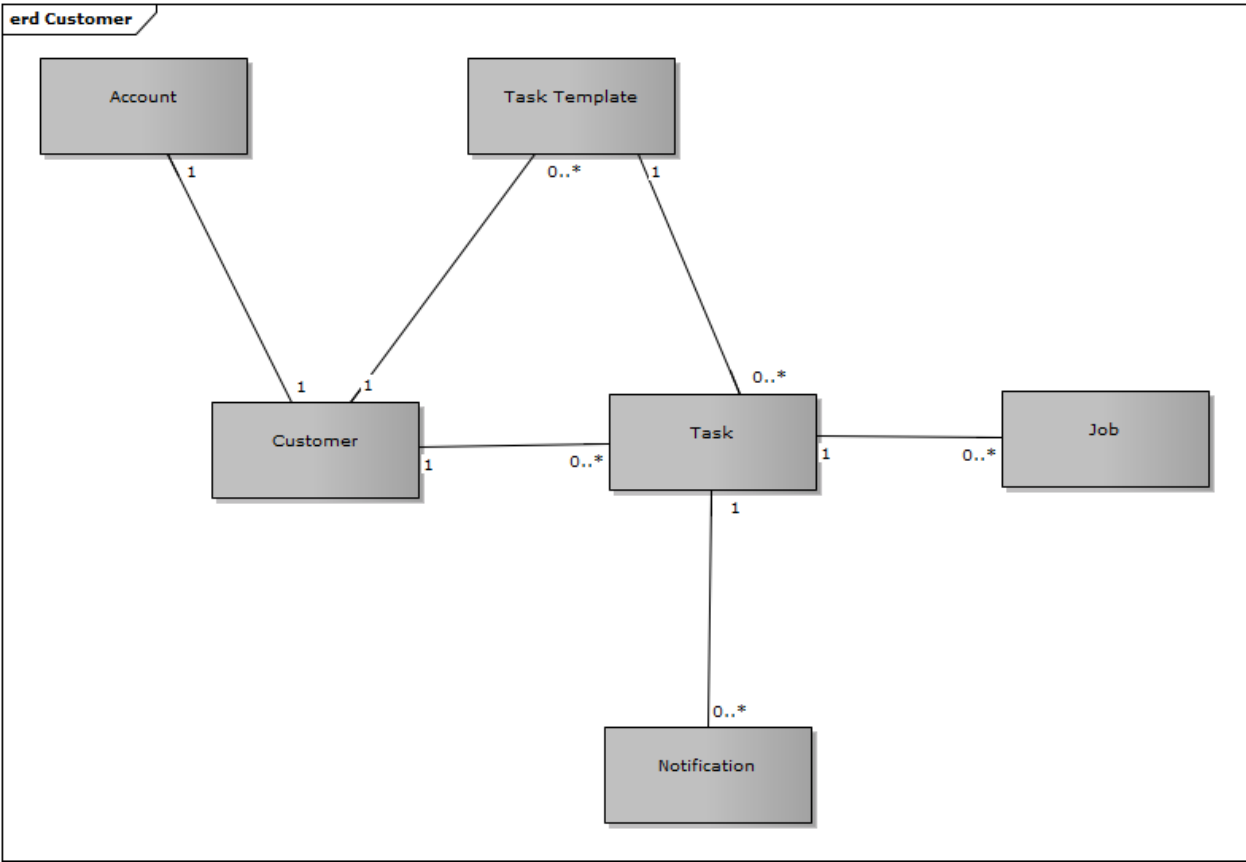
- JSON Representation

```
team: [{
add_user_ids: [
  USER_ID1, USER_ID2, USER_ID3
  ]
}]
```

Response status

- 200, if the request could be successfully handled (even if no teams were found)

Remove clickworkers from a team

Request

Request line:

PUT [context]/customer/teams//${ id}

Request parameter:

(Common only)

Request body:

- XML Example

```
<team>
  <remove_user_ids>[USER_ID1]</remove_user_ids>
  <remove_user_ids>[USER_ID2]</remove_user_ids>
  <remove_user_ids>[USER_ID3]</remove_user_ids>
</team>
```

- JSON Representation

```
team: [{
remove_user_ids: [
  USER_ID1, USER_ID2, USER_ID3
  ]
}]
```

Response status

- 200, if the request could be successfully handled (even if no teams were found)

Entity Relationship Diagram

API Principles

Managing resources using REST

Representational State Transfer[3] (REST) is an architectural principle for distributed systems, such as the World Wide Web. Applications and services following the REST principle apply the following concepts:

- Every resource (and every resource type) is directly reachable using a Uniform Resource Identifier (URI[4]), usually a canonical URL.
- A single resource can have multiple representations, like HTML, XML, or text. This allows clients to use various tools (like browsers) to handle these resources.
- No session state is maintained on the server. Clients must provide all information necessary to handle the given request. This allows atomic operations and scalable services.

REST makes extensive use of the Hypertext Transport Protocol (HTTP) for transporting resource representations. Operations are called using HTTP methods and results are reported using HTTP status codes.

URI

Using REST, a resource is identified by a URI. A URI consists of various parts. The following example demonstrates these parts and their purpose:

<p align="center">https://api.clickworker.com/marketplace/v2/products/TextCreate</p>

The following parts of the URI are static and consistent throughout a given API release. In the following examples, this part is referenced by a pattern as [context]:

- *https://* identifies the protocol to use (HTTP over SSL)
- *api.clickworker.com* is the service host's DNS name
- */marketplace/v2* is the service's context, including an API version marker

The last parts of the URI are specific to the managed entities.

- */products* identifies the resource type
- */TextCreate* is the resource's id (a product code in this example)

As a result, details on the product with code TextCreate are returned.

---

[3] See http://en.wikipedia.org/wiki/Representational_State_Transfer for more.
[4] See http://en.wikipedia.org/wiki/Uniform_Resource_Identifier for more.

Representing Resources Using XML or JSON

As stated above, a single resource may have multiple representations. Currently, the clickworker.com Marketplace API supports Extensible Markup Language (XML) and JavaScript Object Notation (JSON).

Entities may support two different view modes, automatically chosen by the system:
1. A "detail" view that represents the entity and all of its public attributes. This view is selected when an entity is requested directly.
2. An "index" view that represents the entity with a limited set of public attributes. This view is chosen when an entity is requested indirectly, e.g. as part of a collection.

Clients may call one of these representations by providing a valid HTTP Accept header:
- To request a resource as XML, the Accept header must use a MIME-type of "application/xml"
- To request a resource as JSON, the Accept header must use a MIME-type of "application/json"

When clients send json or xml documents as body of post or put requests, the Content-Type Header must be set accordingly:
- To send a XML document, the Content-Type must be set to "application/xml"
- To send a JSON document, the Content-Type must be set to "application/json"

Selecting a resource representation by using a virtual file extension (like ".xml" or ".json") is not supported, as it violates the concept of a single resource URI.

XML documents make use of a published Document Type Definition (DTD) and optionally an XML Schema. In addition to resource-specific elements and attributes, globally defined attributes (like xml:lang) and attributes from other XML namespaces (like xlink:href) may be used.

Calling operations using HTTP methods

The Hypertext Transport Protocol (HTTP) defines 8 different request methods[5]:
- GET
- POST
- PUT
- DELETE
- HEAD
- OPTIONS
- CONNECT
- TRACE

---

[5] See http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html for more.

Of this set of HTTP "verbs," GET, POST, PUT and DELETE are supported by the Clickworker Marketplace API. These verbs are used to call predefined operations on addressed resources.

Due to issues surrounding firewall restrictions and legacy clients, HTTP PUT and DELETE methods may be encapsulated using HTTP POST. To do this, add an additional URL parameter called "_method" to the request, using the HTTP method name in uppercase as the value.

GET: Display or index resources

Calling the GET method will produce two different results:

1. If called on a resource type instead of an identified resource, an index of available resources of the requested type will be returned. The index usually contains a reduced set of resource attributes, along with the URI of the indexed resource.
2. If called on an identified resource, the resource will be printed in detail.

Usually, the request will return the HTTP status code 200 ("OK") and contain the requested resource (or resource index) as its body. See Table 1 for details on available response codes.

POST: Create a new resource

The POST method is used to create a new resource using the addressed resource's type. POST must only be called on URIs specifying a resource type. To modify an existing resource, the PUT method should be used.

Usually, the request will result in an HTTP status code of 201 ("Created") and contain details on the created resource in the response's body.

PUT: Update an existing resource

To modify an existing resource, a PUT request must be sent to the server. The request must use the URI of the resource being modified. The resource is updated with the request's body data, which usually contains a reduced representation of the resource. It only contains attributes that can be updated. For example, the resource identifier cannot be altered after the resource has been created.

Usually, the request will result in an HTTP status code of 200 ("OK") and contain details on the updated resource in the response's body.

DELETE: Remove and existing resource

To remove an existing resource, the DELETE method must be called on the resource's URI.

Usually, the request will result in an HTTP status code of 204 ("No Content") and an empty response body, since the resource is not available any more.

Reporting status using HTTP codes

HTTP already defines an extensive set of numeric status codes that are suitable to report the outcome of request processing. They can be divided into roughly three groups:

- Success codes
- Client error codes
- Server error codes

API operations can report the status in two different ways:

1. By using the predefined HTTP header "Status" (this is the default).
2. By encapsulating the actual operation outcome inside the Request Status entity (see page 14). The HTTP status code will always be 200 OK (unless a server error occurs). To enable this behavior, a special URL parameter named "suppress_http_status" must be set to value "1".

The following tables display the status codes that are used by the Clickworker Marketplace API.

Code	Text	Usage
200	OK	The request was handled successfully and created a response that is available to the client. This status code is mostly used for GET and PUT operations.
201	Created	The request was handled successfully and has created a new resource that is available to the client. This status code is mostly used for POST operations.
204	No Content	The request was handled successfully, but has not created any content. This status code is mostly used for DELETE operations.

Table 22: Success Codes

Code	Text	Usage
400	Bad Request	The request cannot be handled due to formal errors. The client should not repeat the request without making changes.
401	Authorisation Required	Access to the requested resource requires authentication, but no credentials were provided. See section "Authentication and Transport Security" for details on how to authenticate.
403	Forbidden	Access to the requested resource is not allowed due to invalid credentials. See section "Authentication and Transport Security" for details on how to authenticate.
404	Not Found	The requested resource is not available
405	Method Not Allowed	The operation requested (GET, POST, PUT, DELETE) is not allowed for the addressed resource. An HTTP header ("Allow") is provided to name the valid methods.
406	Not Acceptable	The addressed resource cannot be represented in the format requested by the client. This code is returned if the resource was requested in a representation other than XML or JSON.
409	Conflict	The request could not be completed due to a conflict with the current state of the resource. This code is returned if modifying operations are requested on locked resources.
415	Unsupported Media Type	The resource provided in the request's body is in a format other than XML or JSON. This status code can only occur for POST and PUT

Code	Text	Usage
		requests.

Table 23: Client Error Codes

Code	Text	Usage
500	Internal Server Error	The server is not able to handle the request due to an internal error. An appropriate message will be provided inside the response's body.
501	Not Implemented	The server does not support the functionality required to fulfill the request. Unlike a 405, this status is reported for operations that are not available for any resource type.
503	Service Unavailable	The server is currently unable to handle the request due to a temporary error or maintenance.

Table 24: Server Error Codes

Authentication and Transport Security

The Clickworker Marketplace API uses HTTP Authentication[6] ("Basic Auth") for access control. In order to use Clickworker's crowdsourcing services through the API, you need to be registered as a customer on www.clickworker.com.

The credentials used for API authentication are identical to the username and password used at registration. In order to be able to access the API you need to be activated as an API user by our support. In order to request API access please log in with your customer account and go to "Profile" / "API access".

To ensure data security, Clickworker's services require the use of Secure Socket Layers (SSL).

Verifying A Client Setup

For convenience, two variants of a dedicated "Echo" service have been made available:
- Echo incoming message without authentication required
- Echo incoming message with authentication required

Calling the Echo Service

Without Authentication

To verify the availability of our services and the client's basic communication setup, send a plain HTTP POST request to the Echo service.

Request
Request URI:

http://api.clickworker.com/api/marketplace/v2/echo/simple_echo

---

[6] See http://tools.ietf.org/html/rfc2617 for more.

Request body:
  • XML Representation

```
<echo>
 <message>${message}</message>
</echo>
```

  • JSON Representation

```
echo: {
 message: "${message}"
}
```

Request parameter

Name	Type	Synopsis	Mandatory
message	String	Message to be echoed	Yes

Response

Response status:
  • 200, if the request was formally correct

Response body
  • XML Example

```
<simple_echo_response>
 <request_status>…</request_status>
 <echo>
  <message>Your Message</message>
 </echo>
</simple_echo_response>
```

  • JSON Example

```
simple_echo_response: {
 request_status: …,
 echo: {
  message: "Your Message"
 }
}
```

With Authentication

Calling the echo service with authentication not only verifies basic client/server communication, but also ensures the validity of the customer's account.

Technically, the procedure is identical to the one without identification with one major difference: Clients must use an SSL HTTP (https) POST request.

Request

Request URI:

https://api.clickworker.com/api/marketplace/v2/echo/extended_echo

Request body:
  • XML Representation

```
<echo>
 <message>${message}</message>
</echo>
```
- JSON Representation
```
echo: {
 message: "${message}"
}
```

Request parameter

Name	Type	Synopsis	Mandatory
message	String	Message to be echoed	Yes

Response
Response status:
- 200, if the request was formally correct

Response body
- XML Example
```
<entended_echo_response>
 <request_status>…</request_status>
 <echo>
  <message>Your Message</message>
 </echo>
</extended_echo_response>
```
- JSON Example
```
extended_echo_response: {
 request_status: …,
 echo: {
  message: "Your Message"
 }
}
```

Appendix

List of Tables