clarizen
the way to work

Clarizen Web Services

API Guide

Version 3.2

TABLE OF CONTENTS

GETTING STARTED

Clarizen provides programmatic access to your organization information using simple and powerful application programming interface - Clarizen Web Services API.

Using Clarizen Web Services you will be able to develop various types of the applications, such as:

- Custom applications: create custom application to have additional benefits of your organization and business data that resides in the Clarizen repository. Provide your employees with specific familiar User Interface for specific organization processes.
- Integrations with Desktop tools: create integrations with desktop authoring tools to bring task management close to the employee desktop
- Integrations with legacy systems: create integrations with your internal data management systems to exchange relevant data.

Use this section to learn about:

- Introducing CLARIZEN Web Services
- Quick Start
- Object Basics
- API Calls Basics
- Error Handling
- Security & Licensing
- Web Service Governance

Introducing CLARIZEN Web Services

Web Services (API) access is enabled for all Enterprise organizations. Certified partners can request a partner Id to access Clarizen's Web Services. Please contact your sales representative or Clarizen technical support for more information.

Use this section to get quick introduction to Web services API:

- Supported Editions
- Standard Compliance
- Development Platforms
- API Support Policy

Supported Editions

Web Services API is provided beginning from version V3 of Clarizen,
To use Web Services API your organization must use Enterprise Edition. If you are already customer of Clarizen and do not have this edition, please, contact your sales representative.

Business and development partners of Clarizen can use Developer edition. Developer edition provides access to all functions available in the Enterprise edition.

Standard Compliance

The API is implemented to give compliance to the following specifications:

Standard Name	Website
Simple Access Object Protocol (SOAP) 1.1	**http://www.w3.org/TR/2000/NOTE-SOAP-20000508**
Web Service Description Language (WSDL) 1.1	**http://www.w3.org/TR/2001/NOTE-wsdl-20010315**
WS-I Basic Profile 1.1	**http://www.w3-i.org/Profiles/BasicProfile-1.1-2004-08-24.html**

Development Platforms

Clarizen Web Services API supports SOAP development environments including, but not limited to .NET 2.0 and Apache Axis. In this document we mainly provide examples in C# (.NET), some of the examples are provided for Java.

Note: If you work with .NET environment, you can use .NET 2003 and upper versions.

API Support Policy

We would recommend that you use the latest version of Clarizen Web services WSDL to benefit of richer features. See "How to Obtain Web Services WSDL" and "Import WSDL File into Development Platform" of the "Quick Start" chapter to learn how to update WSDL file upon release of the new version of Clarizen.

Quick Start

In this section you can find details on how to use Microsoft .NET or Java to build an application that uses Clarizen Web services and how to run the sample application provided by Clarizen (https://api.clarizen.com/v1.0/ClarizenClient.zip).

To make your experience of using Web Services smooth and efficient, we would recommend that you read the rest of this document.

How to Obtain the Web Service WSDL

In order to access the Clarizen Web Service you must first import the Clarizen WSDL into your development environment. The WSDL is an XML document supported by most development tools which describes the messages and operations supported by a web service.
The current version of the WSDL is located at https://api.clarizen.com/v1.0/Clarizen.svc.

Import WSDL File into Development Platform

Visual Studio
To access the Clarizen web services from .NET you need to import the Clarizen WSDL into visual studio. This operation generates proxy classes that you can reference in your client code to access the services.

Important: To import the WSDL you must first have a Visual Studio project opened.

To import the WSDL follow these steps:

1. On the Project menu select "Add Service Reference"
2. Click "Advanced" and select "Add Web Reference"
3. In the URL text box enter the Clarizen web services URL: https://api.clarizen.com/v1.0/Clarizen.svc and click "Go"
4. In the web reference name text box enter "ClarizenSvc" and click Add Reference.
5. Visual Studio will generate the proxy classes and add them to your project.

Java (Apache Axis)
This section shows how to import the Clarizen WSDL using the Apache Axis WSDL2Java tool. WSDL2Java will generate java proxy classes that you can use in your code to access the Clarizen web service.

To run the WSDL2Java tool your java classpath must be set correctly. The following command line assumes you are running the tool from the Apache Axis 1.4 directory:

```
set classpath=lib/axis.jar;lib/jaxrpc.jar;lib/saaj.jar;lib/commons-logging-
1.0.4.jar;lib/commons-discovery-0.2.jar;lib/wsdl4j-1.5.1.jar
java.exe org.apache.axis.wsdl.WSDL2Java -a
https://api.clarizen.com/v1.0/Clarizen.svc
```

The above command line will generate a set of Java source files that can be compiled and used in your client code.

Sample Code

The following sample code provides a basic template you can use when creating a client that accesses the Clarizen web service.

C# Sample Code

```
static void Main(string[] args)
{
        //Create a proxy to the Clarizen Web Service
        clarizenClient = new Clarizen();
        clarizenClient.Timeout = 60 * 1000;
        //Enhance performance by compressing traffic
        clarizenClient.EnableDecompression = true;

        //Set your applicationId if applicable
        LoginOptions options = new LoginOptions();
        options.ApplicationId = "Sample c# client";

        //Call the Login method
        LoginResult lr = clarizenClient.Login(userName, password, options);
}
```

Java Sample Code

```
public static void main(String[] args) throws Exception {
       //Create a proxy to the Clarizen Web Service
       ClarizenHttpStub binding;
       binding = (ClarizenHttpStub)new ClarizenLocator().getClarizenHttp();

       // Time out after a minute
       binding.setTimeout(60000);
       // prompt for user\password details
       String userName = readFromConsole("User:");
       String password = readFromConsole("Password:");

       //Set your applicationId
       LoginOptions opts=new LoginOptions();
       opts.setApplicationId("Sample Java client");

       //Call the Login method
       LoginResult lr=binding.login(userName,password,opts);

       //Create a header that will hold the Session ID returned fron the login
operation
       SessionHeader sh=new SessionHeader();
       sh.setID(lr.getSessionId());
       String sessionNamespace =new
ClarizenLocator().getServiceName().getNamespaceURI();

       //Add the Session ID header to the proxy so the header will
       //be sent on every request
       binding.setHeader(sessionNamespace,"Session",sh);
}
```

PHP Sample Code

```
<?php

function displayErrors(stdClass $result)
{
       $serverErrorObj = $result->ExecuteResult->Result->Error;
       if($serverErrorObj)
       {
              echo '<ul>';
              echo '<li>ErrorCode : '.$serverErrorObj->ErrorCode.'</li>';
              echo '<li>Message : '.$serverErrorObj->Message.'</li>';
              echo '<li>ReferenceId : '.$serverErrorObj->ReferenceId.'</li>';
              echo '</ul>';
              return true;
       }
       return false;
}

//LOGIN PROCCESS
$soapUrl = 'https://api.clarizen.com/v1.0/Clarizen.svc?WSDL';
$soapApiUrl = 'http://clarizen.com/api';
```

```
$soapConfig = array('exceptions' => 1);
$request = array();
$params = array(
       'userName'          => 'username',
       'password'          => 'password'
);
try{
       $client = new SoapClient($soapUrl, $soapConfig);
       $response = $client->Login($params);

       $sessionId = $response->LoginResult->SessionId;
       $userId = $response->LoginResult->UserId;

       $header = new SoapHeader($soapApiUrl, 'Session', array("ID"=>$sessionId));
       $client->__setSoapHeaders($header);

       $request[] = ... //Create additional request to the API
       $result = $client->Execute(array("request"=>$request));
       if($result)
       {
              if (!displayErrors($result))
                 var_dump($result);
       }

}
catch(SoapFault $e){
       var_dump($result);
       var_dump($e->getMessage());
}
```

Object Basics

Clarizen Data Model is represented as a set of entity types. Each entity type represents specific type of data. Clarizen entity types correspond to a database table that keeps your organization data.
For example, one of the central entity types in Clarizen is "Work Items". "Work Items" is an entity type that keeps data on Projects, Milestones and Tasks that construct Projects you manage in Clarizen application.
The term "entity" in this document is used to describe particular occurrence of specific object, for example, specific project "Release V2.0" or task "Check the book". "Entity" or "Object" corresponds to a row in a database table.

Read more about Clarizen entity types and entities in "Data Model" chapter.

Most of the objects accessible through the API are Read - Write objects. There are cases when such objects are Read - Only by definition, such cases will be specifically indicated in the document.

Note: Access to specific entity types through API is controlled by the authorization rights of the user that runs the application.

Primitive Field Types

Note: You can use all field names of the entity types directly as given in this document.

Web Services API uses following primitive field types:

Name	Description
Integer	*Integer* fields contain numerical data without fractional portion after decimal place. *Integer* field is 32 bits long and can contain signed data in the range of: −2,147,483,648 to +2,147,483,647. Examples of such fields are `SequenceOrder`, `Priority` and others.
Long	*Long* fields contain numerical data without fractional portion after decimal place. *Long* field is 64 bits long and can contain signed data in the range of: −9,223,372,036,854,775,808 to +9,223,372,036,854,775,807. Examples of such fields are `FileSize`, `TotalSpace`, `UsedSpace` (used in the storage description) and others.

Double	*Double* fields contain numerical data of type double with fractional portion after the decimal place. Examples of such fields are `ExpectedProgress`, `PercentCompleted`, `ImpactWeight` and others. Specific restriction may be applied to the Double data: • Scale – maximum number of digits to the right of the decimal dot. • Precision – total number of digits to the right and left of the decimal dot.
Date	*Date* field contains Date value. Unlike *DateTime* field, *Date* field does not contain the time portion of the value. Since the time portion of the date is irrelevant for the *Date* type, the *Date* value is always set to the midnight.
DateTime	*DateTime* field contains date/time (timestamps) values with the precision to the second. Examples of such fields are *`CreatedOn`* and *`ModifiedOn`* that appear in many objects. *DateTime* values are kept in the database converted to the Coordinated Universal Format (UTC). In your client application you need to translate your local time to the UTC format. There are two special *DateTime* fields that undergo additional special treatment: • *`StartDate`* – is always set to 08:00 AM to represent that work should start at the beginning of the working day • *`DueDate`* – is always set to start of the working day plus working day duration to represent that work should end at the end of the working day In your client application you do not need to do anything special to support correctness of the values in these fields.
Boolean	*Boolean* type fields can have one of two possible values: True or False. Examples of such fields are `BelongToCriticalPath` or `PercentCompleteManuallySet` and others.
String	*String* type fields contain character strings (text). The maximum size of the string is restricted for specific fields and generally depends on the specific requirements applied in specific case, but cannot be more than 4000 B. Examples of such fields are `Name`, `Description` and others.
Text	*Text* type fields contain character strings (text). The maximum size of the string is not restricted programmatically and depends only the restriction that is applied by the Oracle on the maximum size of the CLOB column type. Examples of such fields are `Recommendation`, `Comment` and others.

CLARIZEN Specific Field types

Web Services API introduces following specific Clarizen field types:

Name	Description
Entity	"*Entity*" type is used to uniquely identify specific entity. Entity type field consists of: • Entity type designation • Identifier – string that ensures uniqueness of the entity in the entity type class. This field type is also used to represent a reference to another entity. For example, the "Comment" Entity Type has a field called "`AttachedTo`" which is a reference to the item that comment was made on. See below sample code on how to work with *Entity* field.
Duration	*Duration* field type is used mainly for the fields that represent time period or duration of the work item. Example of such field is `Duration` field that can be found in all types of work items: projects, milestones and tasks. *Duration* field consists of three (3) sub-fields: • *Value* – double that contains duration of the time period • *Unit* – Clarizen supports following measurement types for duration: Minutes, Hours, Days, Weeks and Months. • *Duration Type* – Optional enumeration that represents specifics of the corresponding time period relatively to working and non-working days and can be one of the following: ○ *Default* – time period contains only working days ○ *Consecutive* – all over ○ *OverrideStart* – time period can start on non-working day ○ *OverrideEnd* – time period can end on non-working day ○ *OverrideBoth* – time period can start and end on non-working day See below sample code on how to work with *Duration* field.

Data Types Used in the API Calls

This section describes special Web Services runtime data types that are used in the API requests.

BaseEntity and GenericEntity

An entity in Clarizen is represented in Web Services using the BaseEntity class. BaseEntity is the abstract base class of all Entity classes. In the current version of Clarizen Web Services API the only subclass of BaseEntity is GenericEntity, GenericEntity is a representation of a Clarizen entity using a generic collection of name value pairs where (represented as a FieldValue object) where each name represents the name of a field belonging to that Entity.

See the description of RetrieveMessage\ RetrieveMultipleMessage for sample code of how to retrieve a task with a known Id.

```
static void RetrieveTask(string taskId)
{
	//Create the retrieve message
	RetrieveMessage retrieveMsg = new RetrieveMessage();
	retrieveMsg.Id = new EntityId();
	retrieveMsg.Id.TypeName = "Task";
	retrieveMsg.Id.Value = taskId;

	//Retrieve only the fields you need.
	retrieveMsg.Fields = new string[] { "Name" , "StartDate" };

	BaseMessage[] messages = new BaseMessage[] { retrieveMsg };

	//Send the retrieve message to the server and cast the result to correct 
type
	RetrieveResult result = (RetrieveResult)client.Execute(messages)[0];

	//Always check the result status
	if (result.Success)
	{
		//Cast the returned entity to GenericEntity
		GenericEntity task = (GenericEntity)result.Entity;

		//Now you can access the specific fields that were retrieved
		string taskName = (string)task.Values[0].Value;
		DateTime startDate = (DateTime)task.Values[1].Value;
	}
}
```

Using External IDs & References

Entity Identifier

Entity identifier is an external identifier that provides a means to designate and reference an object.

Field `ID` is a String of 40 characters length. It is implemented using database mechanism of Primary Key.

When creating an entity you can either let the system create an Entity Id for you or explicitly set it yourself. If not set specifically during entity creation by the application, field `ID` will be automatically filled with GUID value to keep uniqueness of the field value.

Use external identifiers in various scenarios where you need to control entity identification already at the stage of entity creation. For example:

- Create and link two objects. Define external IDs of the entities at the stage of their creation and then use these IDs at the consequent stage of link (reference) creation.

Note: External identifiers can be used by your organization to maintain single meaningful format of entity designation.

Working with References

The term *Reference* is used to define reference to any existing entity in Clarizen.

Following are the examples of the mechanism of references:

- Build relations (links) having various application's meaning between two different entities, for example:
  - Documents attached to the task
  - Milestone that impacts completeness of the other milestone
  - User working as a resource in specific task
  - Others
- Reference specific entities in the same or other entity type, for example
  - Reference pick up values
  - Discussion around specific work item

See the description of UpdateMessage for sample code of how to update a reference between a Task and its Manager.

The following example shows how to create 2 tasks and make one of them the parent of the 2nd class by creating a link between them:

```
//Create an entity that represents a task
GenericEntity parentTask = new GenericEntity();
parentTask.Id = new EntityId();
parentTask.Id.TypeName = "Task";
parentTask.Id.Value = Guid.NewGuid().ToString();

FieldValue parentName = new FieldValue();
parentName.FieldName = "Name";
parentName.Value = "Parent Task";

parentTask.Values = new FieldValue[] { parentName };

//Create an entity that represent a child task
GenericEntity childTask = new GenericEntity();
childTask.Id = new EntityId();
childTask.Id.TypeName = "Task";
childTask.Id.Value = Guid.NewGuid().ToString();

FieldValue childName = new FieldValue();
childName.FieldName = "Name";
childName.Value = "Child Task";

childTask.Values = new FieldValue[] { childName };

//Create an entity that represents the link between the parent and the child task
GenericEntity link = new GenericEntity();
link.Id = new EntityId();
link.Id.TypeName = "WorkItemHierarchyLink";
```

```
//Create a field that references the parent task Id
FieldValue parentRef = new FieldValue();
parentRef.FieldName = "Parent";
parentRef.Value = parentTask.Id;

//Create a field that references the child task Id
FieldValue childRef = new FieldValue();
childRef.FieldName = "Child";
childRef.Value = childTask.Id;

link.Values = new FieldValue[] { parentRef, childRef };

//Now create 3 CreateMessage objects that perform the creation of the tasks
//and the link between them
CreateMessage msg1 = new CreateMessage();
msg1.Entity = parentTask;
CreateMessage msg2 = new CreateMessage();
msg2.Entity = childTask;
CreateMessage msg3 = new CreateMessage();
msg3.Entity = link;

Result[] results = client.Execute(new BaseMessage[] { msg1, msg2, msg3 });
//checks all requests succeeded
…
```

Usage of External Identifiers in the Standard Clarizen Operations

Note: As a rule external identifiers are required in the operations that deal with specific entity / entities.

Following table shows examples of requirements of External Identifier in the selected Clarizen operations:

Operation	ID Required	ID Returned on Response
Add	No	Yes
Update	Yes	No
Delete	Yes	No
Retrieve	Yes	Yes
Lifecycle	Yes	No
Search	Yes / No	Yes

API Calls Basics

Using API calls you can perform specific operations applied to the Clarizen entities and metadata. Using API you can:

- Add, update, delete entities that belong to different entity types
- Login into the system
- Query entities from different entity types
- Retrieve administrative information
- Other

Synchronous Execution of the API

Execution of the API requests is synchronous. API request is submitted by the client application to the Clarizen Web Service, Clarizen processes request and returns relevant response. Client application should wait for response on submitted request. Asynchronous requests are not supported.

Auto Commit Policy

Every request that is submitted by the client application is automatically committed. All operations, such as Create, Update and Delete are treated as a separate transaction. Client application does not need to send explicitly Commit statements.

It is recommended that client application will avoid situation when several operations should be handled as one transactional package. If impossible, such situations should be handled by the client application specifically. For example, if there are two consequent operations that should be both performed successfully and the second operation failed, application must “roll back” itself results of the first operation.

IClarizen Interface

IClarizen interface contains five (5) members:

- `Execute (BaseMessage[])`
- `Login (Username, Password, LoginOptions)`
- `Logout`
- `Metadata (MetadataMessage)`
- `Query (QueryExpression)`

Execute

`Execute` interface is used to perform majority of the operations over Clarizen entities: Create, Delete, Update, Retrieve, Upload, Download and others.
See the Working with References section for some sample code using the `Execute` method.

Login and Logout

`Login` interface is used to login into the system with user credentials and additional options such as `partnerId` and `applicationId`.
`Logout` closes user’s session.

```
 static void Main(string[] args)
{
        //Create a proxy to the Clarizen Web Service
        clarizenClient = new Clarizen();
        clarizenClient.Timeout = 60 * 1000;
        //Enhance performance by compressing traffic
        clarizenClient.EnableDecompression = true;

        //Call the Login method
        LoginResult lr = clarizenClient.Login(userName, password, null);
        //Use the web service
        …
        …
        //Make sure to Logout after you have finished using the web service
        clarizenClient.Logout();
}
```

Metadata

`Metadata` interface is an interface that allows access to various administrative data and information about Clarizen entity types that could be essential at runtime.

The following example shows how to retrieve the metadata description of the `Task` entity type:

```
DescribeEntitiesMessage msg = new DescribeEntitiesMessage();
msg.TypeNames = new string[] { "Task" };
DescribeEntitiesResult result = client.Metadata(msg);
EntityDescription description = result.EntityDescriptions[0];
// description.Fields contains the fields that are part of the Task entity type
…
```

Query

Query interface is used to access pre-saved queries or construct new queries for the Clarizen entities.
The following example shows how to use the built in `MyWorkItemsQuery` to query for active tasks you are currently managing.

```
MyWorkItemsQuery query = new MyWorkItemsQuery();
query.ItemsType = WorkItemType.Task;
query.ItemsTypeSpecified = true;
query.ItemsFilter = WorkItemFilter.IAmAManager;
query.ItemsFilterSpecified = true;
query.ItemsState = WorkItemState.Active;
query.ItemsStateSpecified = true;

query.Fields = new string[]
{
        "Name",
        "Importance",
        "DueDate",
         "StartDate",
         "PercentCompleted"
};

QueryResult qr = client.Query(query);
```

Error Handling

Web Services API returns error data that can help you to designate and identify erroneous situations created during managing Web Services requests.
There are two kinds of errors that may occur during a web service request:

- Fault – An exception that causes the entire operation to fail. Usually in this case the request processing does not even start. An example of such an exception is a session timeout
- Error – A failure to process one or more of the requests sent to the web service due to missing or invalid data. For example, trying to set a negative %completed value for a task will result in such an error.

In most programming languages when a fault is returned, the proxy generated on the client will throw an exception.

Handling Session Timeout

The following example shows one way to handle session timeouts in your code:

```
try
{
        //Call web service methods
}
catch (System.Web.Services.Protocols.SoapException e)
{
        if (e.Code.Name == "SessionTimeout")
        {
        //Here you can either re-login and retry the last operation
        //or show a message to the user.
        }
        else
                throw;
}
```

Handling Errors during API Requests Execution

Every operation performed against the Clarizen web service return a status representing whether the operation succeeded or failed. The `Result` Object that is returned from each call has the following properties to help you identify whether the operation has succeeded or failed and details of failure. Result contains the following properties:

- `Success` – A boolean value indicating whether the operation succeeded or failed.
- `Error` – An object of type `Error` that provides information on why the operation failed.

The `Error` object contains the following properties:

- `ErrorCode` – A value from the `ErrorCode` enumeration that indicates the type of error that occurred.
- `Message` – A string suitable for display that describes the error and how to fix it.
- `ReferenceId` – A string containing a unique identifier representing this error. Please provide this Id when contacting support so we can easily track this error.

The following sample code shows how to check for errors when accessing the Clarizen web service:

```
static void HandleError()
{
        ...
        RetrieveResult result = (RetrieveResult)client.Execute(messages)[0];
        If (result.Success)
        {
                //The operation succeeded – continue as usual
        }
        else
        {
                if (result.Error.ErrorCode == ErrorCode.EntityNotFound)
                {
                        Console.WriteLine("The entity does not exist");
                }
                else
                        Console.WriteLine(result.Error.Message);
        }
}
```

Security

Web Services API leverages the latest industry security standards in order to ensure high-levels of security around your organization data. All Web service requests are controlled by the same security mechanisms as used in the Clarizen application plus additional identification procedure at the authentication phase:

- Authentication
- Authorization
- Session management
- Encryption

Authentication

Authentication is a process of confirming identity of specific person and / or process. During Login process client application written using Web Services should provide correct user credentials and API identification number:

User credentials are:

- User Name
- Password

Clarizen supports two (2) different login options:

- Customers – Customers who want to use the API to access their data should purchase an Enterprise license. You do not have to pass anything special during login.
- Partners – Partners are required to provide a *Partner Id* and *Application Id* during login so they can be identified.
    - *Partner ID* – Identification of a Clarizen partner that develops applications using Clarizen Web Services API that are supposed to be used by various Clarizen customers.
    - *Application ID* – Identification of a specific partner application that can be used for licensing purposed. This Id must be registered with Clarizen.

Important:
Please, contact your sales manager to get your “Partner ID” before you start using Clarizen Web Services.
Please, contact technical assistant to agree on the “Application ID”, if relevant.

Authorization

From Wikipedia: Authorization is the concept of allowing access to resources only to those permitted to use them.

All API calls generated through the Clarizen Web Services undergo the same permission checking as applied in the Clarizen application.

Session Management

After a user was successfully authenticated, the system generates a random and unique identifier known as the Session ID. This identifier is returned to the client as part of the Login return value and the client must pass this Session Id on subsequent requests (See the sample code for details on how to pass the session ID)
A session lifetime is controlled by

- Session Timeouts
- Sessions Numbers Limit

Session Timeout

Similar to the UI, Web services sessions automatically time-out after 60 minutes of inactivity, requiring a login to resume activity. However, if the client submits a request the inactivity time is reset.

Note: For security reasons, we recommend that you close your session using the Logout operation as soon as you are done using the service.

If you explicitly logout of a session, and then attempt to utilize the same session, a SessionTimeout error code is returned. Your code should be prepared to handle session timeouts by re-login when a session timeout occurs. If you need to eliminate open sessions for security reasons, you should call logout upon completion of an operation.

Session Number Limits

Number of the concurrent sessions authenticated with the same user credentials is limited to two (2).

Licensing

Regular application written using Web Services API requires Standard Clarizen license.

Recommendation: You can create special user that will play the role of the “Web Service” user in a non interactive applications and assign him a standard Clarizen license. Use this user’s credentials to login into non interactive application. Be sure that you treat correctly Session tine out case in non interactive applications.

Web Services Governance

In order to optimize performance of the Clarizen application and database servers, Clarizen API engine provides several mechanisms to control consumption of the processes initiated by Web Services. These mechanisms monitor and control requests from the Web services to ensure that: the user experience is not extensively impacted by the possible heavy processes and the burden of the Web services is shared among all users.

Web Services Governance is made up of following areas:

- Request batch limiting: The total number of requests that can be processed in one call to the web service is 100.
- Request limiting: The total size of the requests sent in one call to the web service must not exceed 25 MB.
- Call limiting: 1,000 API calls per paid license per day with a maximum of 1,000,000 calls per day.

INTRODUCTION TO REFERENCE GUIDE

Please, see detailed reference guide for Clarizen Web Services API calls in the "Clarizen Web Services Reference Guide" document available in the Developer page of Clarizen Web site.

Data Model

This section contains description of the data elements and relationships between these elements within Clarizen

Clarizen data model has an object oriented nature. At the logical level it is represented by a set of entity types (can also be referenced as *Classes*) and their relationships.
At the physical level Clarizen uses the underlying database units, such as tables, indexes, constraints definitions, linking tables, partitioned tables or clusters. Physical data model is derived from a logical data model.

Some of the Web Services APIs manipulates with the metadata notions. Metadata is a data about data model, essentially a description of the logical & physical data model & derivation rules between logical & physical representations, such as

- Entity type name, description, type, attributes, database table, etc.
- Collection of fields in specific entity type
- Collection of possible relations between various entity types
- others

In this section you will find definition for:

- Classification of the entity types in use
- Commonly used fields
- Standard Clarizen objects

Entity Types Classification

Entity Type or *Class* is a named collection of the fields and the methods that fully describes specific object type and rules according to which the object of this entity type behaves.

Different *Entity Types* can share and inherit same fields, methods and rules of entity behavior. This concept is reflected in the hierarchical relations between classes.

Hierarchical relationship between entity types / classes involves *Subclasses* (child-classes) and *Superclasses* (parent-classes) concepts.

Regular Entity Types

Regular Entity type is used to store regular objects, such as Projects, Task, Milestones and others.

Link Entity Types

Link Entity types are used to store relations between entities that belong to the same or different entity types. Clarizen allows:

- One-to-One relationship.
- One-to-Many relationships - in one-to-many relationship one object can be linked to many other objects which in turn will be linked to this and only this object.
- Many-to-Many relationships – in many-to-many relationship one object can be linked to many other objects and vice versa.

Link Entity Type always contains following obligatory fields:

- RelatedEntitiy1– reference to the first linked object
- RelatedEntitiy2- reference to the second linked object

Links or relationships between entities can be directed (for example, parent – child relationship) or non-directed links. Direction of specific type of relation is defined by its business logic and is described below in the relevant sections.

Pick up Tables

Pick up tables are similar to the Regular entity types. Pick up tables usually contain limited set of fields and limited number of entities and are referenced from the other types of classes.

Other References between Entity Types

Clarizen data model supports also references between objects that belong to the regular or link entity types. These types of references are implemented using One-to-Many relationship model.

Example of such reference is, `CreatedBy` field that appears in all Clarizen entity types. This field references entity of specific user in the `User` entity type.

References from the regular or link entity types to the pick up tables are implemented using the same mechanism.

Commonly Used Fields

Unique Entity Identifier

Name	Type	Description
SYSID	String(40	Unique entity identifier given by the system to a specific entity.

Common Fields

Following table presents fields that appear in all Clarizen Entity types:

Name	Type	Description
Entity	Entity	Entity identifier that uniquely designates the object. See "Entity Identifier" section for detailed description.
CreatedOn	DateTime	Date and time when specific entity was created

CreatedBy	Entity	Reference to the entity that keeps properties of the user that created the object
ModifiedOn	DateTime	Date and time when specific entity was last modified
ModifiedBy	Entity	Reference to the entity that keeps properties of the user that last modified the object. Note: In case specific entity was last modified by the system process or job this field will keep reference to the *System* user. For example, the job that performs scheduled calculation of the *Task* track status will change value of the `TrackStatus` field to "Off Track" in case the *Task* passed it's `DueDate`.

Standard Fields

Following table presents fields that appear in the majority of the Clarizen entity types, such as Tasks, Projects, Milestones, Documents and others.

Name	Type	Description
Name	String (256)	Name of the entity
Description	String (512)	Description of the entity

Note: In the pickup tables `Name` uniquely represents specific value in the pickup table.

Lifecycle Fields

Lifecycle fields appear in the majority of the Clarizen entity types, such as Tasks, Projects, Milestones, Documents, Documents and others.
Lifecycle fields serve to trace progress of the corresponding entity through its lifecycle. Following table represents the Lifecycle fields:

Name	Type	Description
State	Entity	Represents lifecycle State of the entity. For example, possible states of the `Work Item` objects can be *Draft, Active, Cancelled, Completed, On Hold*. Value is a reference to `State` pickup table.
Phase	Entity	Represents lifecycle Phase of the entity. For example, possible states of the `Work Item` objects can be *Concept, Preliminary Design, Implementation, etc*. Value is a reference to `Phase` pickup table.

Standard Entity Types

Work Item

`Work Item` entity type is a superclass that is currently used to represent three (3) different types of working items:

- `Project`
- `Milestone`
- `Task`

`Project`, `Milestone` and `Task` are subclasses of `Work Item` superclass. All three subclasses inherit fields defined at the level of the `Work Item` entity type. Inherited (shared) fields reside in the `Work Item` class, while fields that differ between these entity types reside in the corresponding subclass,

While creating actual object of one of the work item types you should use entity type name of `Project`, `Milestone` or `Task` respectively to designate which entity type you are going to create.

See the description of CreateMessage for a sample code of how to create a Task.

Work Items of all types have following set of frequently used fields:

- UniqueEntityIdentifier
- Common
- Standard
- Lifecycle

Following table represents specific fields of Work Item entity type tree:

Name	Type	Mandatory	Description
Name		✓	*Inherited from Standard*
ID			*Inherited from UniqueEntityIdentifier. System generated.*
Project	Entity		Represents project to which work item belongs. Reference to the corresponding project entity.
Planning fields			
StartDate	DateTime		Planned start date of the work item
DueDate	DateTime		Planned end date of the work item
Duration	Duration		Planned duration of the work item
ActualEffort	TimeEffort		Actual effort that was already invested into work item
EstimatedEffort	TimeEffort		Estimated effort that should be invested into the work item
RemainingEffort	TimeEffort		Effort that still should be invested into the work item
ActualEndDate	DateTime		Date when the work item was actually finished
TrackStatus	Entity		Indicates track status of the work item*: On Track, On Risk or Off Track*. Reference to the `Track Status` pickup table
ExpectedProgress	Double		Completeness progress expected for the current date
Importance	Entity		Importance of the work item. Reference to the `Importance` pickup table
Priority	Integer		Priority of the work item
Completing Percentage			
PercentCompleted	Double		Percent of progress completeness correct for current date
Responsibility Fields			
Manager	Entity		Functional manager of the work item. Reference to the corresponding user in the `User` entity type

Budget fields			
ActualCost	Double		Actual cost invested to the work item
ActualCostManuallySet	Boolean		*True* – indicates that actual cost was set manually rather than calculated from the “child” work items
PlannedBudget	Double		Planned budget to be invested into the work item
PlannedBudgetManuallySet	Boolean		*True* – indicates that planned budget was set manually rather than calculated from the “child” work items
PlannedRevenue	Double		Income planned for the work item
ActualRevenue	Double		Actual income received
Counter Fields			
ChildrenCount	Integer		Number of work items that are direct children of this work item
ResourcesCount	Integer		Number of resources assigned to this work item
AttachmentsCount	Integer		Number of documents attached to this work item
PostsCount	Integer		Number of Posts made on this work item
NotesCount	Integer		Number of Notes made on this work item

Project

Following table represents `Project` entity type specific fields:

Name	Type	Description
ProjectManager	Entity	Represents the user that is a project manager of specific project. Reference to the user entity type.
ProjectTarget	String(512)	Represents text that is used to describe target of the project
ProjectType	Entity	Represents type of the project. Reference to `Project Type` pickup table.
DirtyFlag	Long	Service field for internal needs. Used to represent that project requires specific type of recalculation.
Parent	Entity	Direct reference to the parent entity

Task

Following table represents `Task` entity type specific fields:

Name	Type	Description
Type	Entity	Represents type of the project. Reference to `Tasks Type` pickup table.
Parent	Entity	Direct reference to the parent entity

Milestone

Following table represents `Milestone` entity type specific fields:

Name	Type	Description
Type	Entity	Represents type of the project. Reference to `Milestone Type` pickup table.

Users

`User` entity type represents user working with the system.
`User` has following set of frequently used fields:

- Common
- Lifecycle

Following table represents specific fields of `User` entity type tree:

Name	Type	Mandatory	Description
DisplayName	String(256)	✓	Represents display name of the user used in various places of the application UI
UserName	String(256)	✓	Represents login name of the user. By default is equal to user's email
FirstName	String(256)	✓	Represents first name of the user
LastName	String(256)	✓	Represents last name of the user
Email	String(256)	✓	Represents email of the user
OfficePhone	String(256)		Represents office phone of the user
OfficeFax	String(256)		Represents office fax of the user
MobilePhone	String(256)		Represents mobile phone of the user
BusinessAddress	String(1024)		Represents business address of the user
LastLogin	DateTime		Represents date and time when user last logged in into the application
AllowEmails	Boolean		*True* indicates that user would like to receive emails from the system
CostRateperHour	Double		Cost rate of the user per hour

Organizations

`Organization` entity type represents Organizations working with Clarizen.
`Organization` has following set of frequently used fields:

- Common
- Standard
- Lifecycle

Following table represents specific fields of `Organization` entity type tree:

Name	Type	Mandatory	Description
Name		✓	*Inherited from Standard*
Country	Entity		Represents country. Reference to the pickup table "`Countries`".
CountryState	Entity		Represents country state (relevant only for USA). Reference to the pickup table "`Country States`".
IndustryType	Entity		Represents industry type of the organization. Reference to the pickup table "`Industry type`".
CurrencyType	Entity		Represents currency type used in the organization. Reference to the pickup table "`Currency type`".
OrganizationLanguage	Entity		Represents language used organization wide. Reference to the pickup table "`NLSLanguage`".
ResourceRateperHour	Double		Represents default rate of user per hour.

Note: All fields of Organization have read-only access through API.

Customers

`Customer` entity represents customers working with specific Clarizen organization.
`Customers` have following set of frequently used fields:

- UniqueEntityIdentifier
- Common
- Standard
- Lifecycle

Following table represents specific fields of Customers entity type:

Name	Type	Mandatory	Description
Name		✓	*Inherited from Standard*
Description			*Inherited from Standard*
ID			*Inherited from UniqueEntityIdentifier. System generated.*
BusinessAddress	String(1024)		Customer Business address
BillingAddress	String(1024)		Customer Billing Address

Following example shows how to get all customers of your organization:

```
EntityQuery customersQuery = new EntityQuery();
customersQuery.TypeName = "Customer";
// Retrieve only name of each customer
customersQuery.Fields = new string[] { "Name" };
// Order results by customer name
customersQuery.Orders = new OrderBy[] { new OrderBy() { FieldName = "Name" } };

QueryResult queryResult = clarizen.Query(customersQuery);
```

Contacts

Contact Person (API Name = ContactPerson) entity represents people that serve contact points between an organization and a specific customers.
Contact Person entity has following set of frequently used fields:

- Standard

Following table represents specific fields of Contacts entity type:

Name	Type	Mandatory	Description
Email	String(256)		Email of the contact person

OfficePhone	String(256)		
MobilePhone	String(256)		
FaxNumber	String(256)		

Following example shows how to get all contacts of a specific customer:

```
string customerId = "210a7ced-a440-45cd-9a30-1ff251b2b1fd";

EntityQuery contactByCustomerQuery = new EntityQuery();
contactByCustomerQuery.TypeName = "ContactPerson";
contactByCustomerQuery.Fields = new string[] { "Name", "Email" };
Compare condition = new Compare();
condition.LeftExpression = new FieldExpression() { FieldName = "Customer" };
condition.Operator = Operator.Equal;
condition.RightExpression = new ConstantExpression() { Value = new EntityId {
TypeName = "Customer", Value = customerId } };

contactByCustomerQuery.Where = condition;
QueryResult result = client.Query(contactByCustomerQuery);
```

All Issue Types

`All Issue Types` entity type is a superclass that is currently used to represent four (4) different types of issues:

- `Issue`
- `Bug`
- `Risk`
- `Enhancement Request`

`Issue`, `Bug`, `Risk` and `Enhancement Request` are subclasses of `All Issue Types` superclass. All four subclasses inherit fields defined at the level of the `All Issue Types` entity type. Inherited (shared) fields reside in the `All Issue Types` class, while fields that differ between these entity types reside in the corresponding subclass,

While creating actual object of one of the issue types you should use entity type name of `Issue`, `Bug`, `Risk` or `Enhancement Request` (API name = `EnhancementRequest`) respectively to designate which entity type you are going to create.

API name of the entity is `Case`.
Issues of all types have following set of frequently used fields:

- UniqueEntityIdentifier
- Common
- Standard
- Lifecycle

Following table represents specific fields of All Issue Type entity type:

Name	Type	Mandatory	Description
Title		✓	*Inherited from Standard 'Name' field*
ID			*Inherited from UniqueEntityIdentifier. System generated.*

Description	Text		Text of Description as shown to the user. Contains "rich" text characters
PlainText	Text		Text of Description as stored in the database for purposes of searches
Severity	Entity		Represents severity of the issue. Reference to "`Severity`" pickup table
Priority	Integer		Represents priority of the issue
Mandatory	Boolean		Checked if issue's resolution is mandatory
Owner	Entity		User that owns the issue. Reference to the Users entity
DueDate	DateTime		Target date to provide solution
AssignedTo	Entity		Who is currently assigned to the issue. Reference to the Users entity
AssignmentDate	DateTime		Date of the current user assignment, Read only field, set by the system
SubmittedBy	Entity		Who submitted the case. Reference to the Users entity
SubmissionDate	DateTime		Date of the case submission
EvaluatedBy	Entity		Who evaluated the case. Reference to the Users entity
EvaluationDate	DateTime		Date of the issue evaluation
OpenedBy	Entity		Who opened the issue for resolution. Reference to the Users entity
OpeningDate	DateTime		Date of opening the issue for resolution
ResolvedBy	Entity		Who resolved an issue? Reference to the Users entity
ResolvedDate	DateTime		Date when the case was resolved
ResolutionDetails	String(2000)		Description of resolution
ClosedBy	Entity		Who closed an issue? Reference to the Users entity
ClosureDate	DateTime		Date of the issue closure
RejectedBy	Entity		Who rejected an issue? Reference to the Users entity
RejectionDate	DateTime		Date of the issue rejection
RejectDetails	String(2000)		Description of reasons and other details to reject the case
ReopenedBy	Entity		Who reopened the case? Reference to the Users entity
ReopeningDate	DateTime		Date when the case was reopened
ReopenReasons	String(2000)		Description of reasons to reopen the case
DeferredBy	Entity		Who deferred the case? Reference to the Users entity
DeferringDate	DateTime		Date of the case deferring
DeferReasons	String(2000)		Description of reasons to defer the issue
Comment	String(2000)		Free Comments
PlannedFor	Entity		When an issue is planned to be resolved. Reference to the Work Items entity.
ReportedbyCustomer	Boolean		Reported by the customer
Category	String(256)		Category of the issue
DuplicateTo	String(256)		ID of the duplicate issue

Issue

No specific fields for the `Issue` entity type.

Bug

Following table represents `Bug` entity type specific fields:

Name	Type	Description
Regression	Boolean	Whether bug is a regression from the previous versions.
ImpactArea	String(2000)	Description of the resolution impact area
FoundInBuild	String(256)	In which build was found
IntegratedInBuild	String(256)	In which build interated a fix
ClosedInBuild	String(256)	In which build closed
ReopenedInBuild	String(256)	In which build reopened back for fix due to fix falure

Risk

Following table represents `Risk` entity type specific fields:

Name	Type	Description
PercentProbability	Double	Probability of occurrence.
Impact	Integer	Number from 1 to 5 representing the impact of the risk
TriggerDate	DateTime	Trigger Date of the incident
MitigationPlan	String(2000)	Description of the mitigation plan
ContingencyPlan	String(2000)	Description of the contingency plan
RiskRate	Double	Risk rating is measured by Impact * %Probability. Read only, Calculated.

Enhancement Request

API name of the entity is `EnhancementRequest`.

No specific fields for the `EnhancementRequest` entity type.

Following example shows how to add / modify an issue of the Risk type:

```
// Create new Risk
public static void CreateRisk(string riskId)
{
       //Create an entity representing the new task
       GenericEntity task = new GenericEntity();
       risk.Id = new EntityId();
       risk.Id.TypeName = "Risk";
       risk.Id.Value = riskId;

       //Set risk title
       FieldValue titleField = new FieldValue();
       titleField.FieldName = "Title";
       titleField.Value = "My Risk";

       //Assign the fields to the risk
       risk.Values = new FieldValue[] { titleField };

       CreateMessage createRiskMessage=new CreateMessage();
       createRiskMessage.Entity = risk;

       //Send the CreateMessage to the server for execution
       client.Execute(new BaseMessage[] {createRiskMessage});
}
```

Following example shows how to get all issues of a specific type reported by you:

```
string myUserId = loginResult.UserId; //LoginResult is returned after login

EntityQuery query = new EntityQuery();
query.TypeName = "Case"; //Change to Risk,Issue or Bug to get specific
cases
query.Fields = new string[] { "Title" };

Compare condition = new Compare();
condition.LeftExpression = new FieldExpression() { FieldName =
"SubmittedBy" };
condition.Operator = Operator.Equal;
condition.RightExpression = new ConstantExpression() { Value = new EntityId
{
  TypeName = "User", Value = myUserId } };

query.Where = condition;
QueryResult result = client.Query(query);
```

Documents

`Document` entity type represents document object that can be attached to other entities in the system. Document itself can have several datasets attached.
`Document` has following set of frequently used fields:

- UniqueEntityIdentifier
- Common
- Standard
- Lifecycle

Following table represents specific fields of `Document` entity type tree:

Name	Type	Description
DocumentType	Entity	Represents type of the document. Reference to the pickup table “`Document Type`”.

The following example shows how to upload a new document to Clarizen

```
public void AttachFileToTask(string fileName, EntityId entityId)
{
///Attaching new file to work item is a 3 step procedure:
///1. Create a Document entity
///2. Create a link between the entity and the WorkItem
///3. Upload a file for that document

//Step 1:
GenericEntity document = new GenericEntity();
document.Id = new EntityId();
document.Id.TypeName = "Document";
document.Id.Value = Guid.NewGuid().ToString();

FieldValue nameValue = new FieldValue();
nameValue.FieldName = "Name";
//You can also prompt the user for a doucment name
nameValue.Value = Path.GetFileName(fileName);

document.Values = new FieldValue[] { nameValue };
CreateMessage createDocument = new CreateMessage();
createDocument.Entity = document;


//Step 2: Create a link between WorkItem and Document
GenericEntity link = new GenericEntity();
link.Id = new EntityId();
link.Id.TypeName = "AttachmentLink";

FieldValue entityIdField = new FieldValue();
entityIdField.FieldName = "Entity";
entityIdField.Value = entityId; //A reference to the task

FieldValue documentId = new FieldValue();
documentId.FieldName = "Document";
documentId.Value = document.Id; //A reference to the new document

link.Values = new FieldValue[] { entityIdField, documentId };
CreateMessage createLink = new CreateMessage();
createLink.Entity = link;

//Step 3: Upload the file
UploadMessage upload = new UploadMessage();
upload.DocumentId = document.Id;
upload.FileInformation = new FileInformation();

//Sorage.Server - File is physically stored on the server.
//Other options are Sorage.Url and Sorage.Link where the file is not actually
uploaded but
//a reference to the file path is stored in Clarizen
upload.FileInformation.Storage = Storage.Server;
upload.FileInformation.FileName = fileName;
//Read all file content as an array of bytes
upload.FileInformation.Content = File.ReadAllBytes(fileName);

//Perform all 3 steps in 1 call to the web service:
results = client.Execute(new BaseMessage[] { createDocument, createLink, upload });
}
```

The following example shows how to download a document from Clarizen:

```
public void Download(string documentId)
{
    DownloadMessage download = new DownloadMessage();
    download.DocumentId = new EntityId();
    download.DocumentId.TypeName = "Document";
    download.DocumentId.Value = documentId;

    DownloadResult r = (DownloadResult)client.Execute(new BaseMessage[] { download
})[0];
    if (r.FileInformation.Storage == Storage.Server)
    {
        byte[] fileContent = r.FileInformation.Content;
        //The contents of the file is now in the fileContent variable
        ...
    }
    else //If document is a reference to URL - open it in browser
    {
        string fileUrl = r.FileInformation.Url;
        //The file was not uploaded to the server.
        //fileUrl contains a reference to the file location
        ...
    }
}
```

Comments - Notes & Discussions

`Comment` entity type is a superclass for three (3) subclasses:

- `Note`
- `Email`
- `Discussion`

`Comment` has following set of frequently used fields:

- Common
- Standard
- IsTemplate

Following table represents specific fields of `Document` entity type tree:

Name	Type	Description
Subject		*Inherited from Standard 'Name' field*
Comment	Text	Text of Comment as shown to the user. Contains "rich" text characters
PlainText	Text	Text of Comment as stored in the database for purposes of searches
AttachedTo	Entity	Represents entity on which note or discussion are written. Reference to corresponding entity: Task, Document, other
Visibility	Entity	Represents whether specific Note is Public or Private. Reference to the pickup table "`Comment Type`".

Relations between Entity Types

All entity types of type "Link" or "Relation" have two fields that references two (2) objects that create specific relationship:

Name	Type	Description
RelatedEntitiy1	Entity	Represents first related object.
RelatedEntitiy2	Entity	Represents second related object.

Basic “Link” type allows building bidirectional many-to-many links. Direction of specific type of relation is defined by its business logic and is described below in the relevant sections.

Work Item Progress Impact

`Work Item Progress Impact` represents complex relationships between two work items:

- Hierarchical
- Progress impact
- Shortcuts

These relations mainly represent business rules of progress impact between two work items.

Hierarchical (contained) relation represents structure of the work item tree. A child work item affects parent’s completion progress, start and due dates, status, budget and alert calculation.

Progress Impact (reflected) relation allows building relations that reflect impact on the completion progress, status, dates of the work items that do not have hierarchical relations.

Shortcut relation represents reference to a reflection of another work item, similar to a shortcut in Windows File system. A shortcut has the same effect on its parent as if it was a “real” work item – it affects its completion progress, start and due dates, status and alert calculation.

`Parent` item in this relation always plays role of the impacted by the `Child` work item.

Following table represents specific fields of `Work Item Progress Impact` entity type:

Name	Type	Description
Parent	Entity	Represents Work Item that plays the role of *Parent* in the relation, uses `RelatedEntity1` placeholder
Child	Entity	Represents a Work Item that plays the role of *Child* in the relation, uses `RelatedEntity2` placeholder
LinkType	Entity	Represents specific type of the link. Reference to pick up table “`Contained Link type`” that has two values “Contained” or “Reflected”
IsShortcut	Boolean	*False* indicates relation between two “real” work items. *True* indicates that `Child` field represents “shortcut” of the original work item.
SequenceOrder	Integer	Represents sequence order of the child work item within all children of the same parent.
ImpactWeight	Double	Represents impact weight of the Child work item on the progress of the parent work item.

Dependency

`Dependency` entity type represents relations between work items of “Scheduling dependency” types. This link is many-to-many link that represents following types of scheduling dependency:

- Finish to Start
- Finish to Finish

- Start to Start
- Start to Finish

Following table represents specific fields of `Dependency` entity type:

Name	Type	Description
WorkItem	Entity	Represents Work Item - successor in the dependency relation, uses `RelatedEntity1` placeholder
DependsOn	Entity	Represents a Work Item - predecessor in the dependency relation, uses `RelatedEntity2` placeholder
DependencyType	Entity	Represents type of scheduling dependency. Reference to pickup table `Dependency Type.`
Lag	TimeEffort	Positive vale of `Lag` field represents delay between two dependant work items, Negative value – leading overlap.

Human Resource

`Human Resource` entity type represents working relationships between `Users` and `Work Items`. `Human Resource` link is many-to-many link that serves to build various types of work relations between users and work item, such as:

- Resource that fulfills the work item
- Reviewer that has view access to work item for review purposes

Following table represents specific fields of `Human Resource` entity type:

Name	Type	Description
WorkItem	Entity	Represents related Work Item, uses `RelatedEntity1` placeholder
Resource	Entity	Represents related User that fulfills work for specific Work Item, uses `RelatedEntity2` placeholder
ResourceRole	Entity	Represents role of the user in specific work item. Reference to entity type `Role`.
Units	Double	Represents invested percent of work of specific user. Default value = 100.00

Customer - Project Link

Customer – Project link represents the customers that were assigned for a specific project.

API name of the entity type – `CustomerLink.`

Following table represents specific fields of the entity type:

Name	Type	Description
Project	Entity	Represents related project, uses `RelatedEntity1` placeholder
Customer	Entity	Represents related customer, uses `RelatedEntity2` placeholder
CostAllocation	Double	Represents percent of cost allocation per customer. 100% by default

Customer - Issues Link

Customer – Issues link represents the customers that submitted specific issue and/or the issue is committed for resolution to the customer.
API name of the entity type – `CustomerLink`.

Following table represents specific fields of the entity type:

Name	Type	Description
Issue	Entity	Represents related issue, uses `RelatedEntity1` placeholder
Customer	Entity	Represents related customer, uses `RelatedEntity2` placeholder
Submitted	Boolean	Represents whether specific issue was submitted by a specific customer
Committed	Boolean	Resolution of the issue was committed to the customer
CommittedDate	DateTime	Target date of commitment

Following example shows how to get all issues of all types reported by the customer:

```
string customerId = <customer id>;

IssuesQuery query = new IssuesQuery();
query.TypeName = "Case";
query.Fields = new string[] { "Title" };
Compare customersCondition = new Compare();
customersCondition.LeftExpression = new EntityIdExpression();
customersCondition.Operator = Operator.Equal;
customersCondition.RightExpression = new ConstantExpression() { Value =
  customerId };

query.CustomerCondition = customersCondition;
QueryResult result = client.Query(query);
```

Issue Related Work

Issue Related Work represents the work, in the terms of Projects, Milestones or Tasks, that should be fulfilled to resolve an issue.
API name of the entity type – `RelatedWork`.

Following table represents specific fields of the entity type:

Name	Type	Description
Issue	Entity	Represents related issue, uses `RelatedEntity1` placeholder
Work Item	Entity	Represents related work item, uses `RelatedEntity2` placeholder

Issue Team Members

Issue Team Members entity represents the stakeholder that are interested in the life cycle of a specific issue. API name of the entity type – `IssueTeamMembers`.

Following table represents specific fields of the entity type:

Name	Type	Description
Issue	Entity	Represents related issue, uses `RelatedEntity1` placeholder
User	Entity	Represents related user, uses `RelatedEntity2` placeholder

Attachment

`Attachment` entity type represents relations between `Work Items or Issues` and `Documents`. This link is one-to-many link in current implementation, meaning that specific document can be linked only to one work item / issue, while there can be several documents linked to one work item/issue.

Following table represents fields of `Attachment` entity type:

Name	Type	Description
WorkItem	Entity	Represents related Work Item, uses `RelatedEntity1` placeholder. Reference to `Work Item` entity type.
Document	Entity	Represents document attached to specific Work Item, uses `RelatedEntity2` placeholder. Reference to `Document` entity type.

Pick up Entities

All entities of Pickup Table type have following set of frequently used fields:

- Common
- Standard

Following table represents specific fields of the Pickup Table entity types:

Name	Type	Description
ValueOrder	Integer	Used to indicate order of how values of the pickup table should appear in UI. Used for pickup tables in which order of values is not alpha-numeric.
IsDefaultValue	Boolean	*True* – indicates value that is given by default during creation of the object that references pickup table

Core Requests

The following table lists the core requests that can be sent to the web service using the *Execute* method and provides a brief description of each request.
A more detailed description for the mostly used requests is provided in the sections below. Details on other requests can be found in the "Web Services Reference Guide".

Request	Response	Description
CreateMessage	CreateResult	Add a new entity to your organization's data
UpdateMessage	Result	Update an existing entity
RetrieveMessage	RetrieveResult	Retrieve an entity
RetrieveMultipleMessage	RetrieveMultipleResult	Retrieve multiple entities from the same entity type
DeleteMessage	Result	Delete an entity
LifecycleMessage	Result	Perform life cycle operations on an entity or group of entities
DownloadMessage	DownloadResult	Downloads a file that is attached to a document object
UploadMessage	Result	Uploads or replaces a file that is attached to a document object

CreateMessage

Use the `CreateMessage` to add an entity such as a `Task` or `Document` to your organization's data.
The `CreateMessage` has one property named `Entity.` Use this property to provide details of the entity you'd like to add.
Depending on the type of entity you are adding, you may need to provide data for certain fields which are marked as "mandatory". For example, when creating a `Task`, the field `Name` must have a value.

Important Do not forget to fill mandatory fields

A response of type `CreateResult` is returned for this message.
If the operation succeeds, the `CreateResult` will contain the `Entity Id` of the new Entity. If you provided an `Entity Id` during the creation of the object, the same Entity will be returned. If you did not provide an `Entity Id`, a unique `Entity Id` will be generated and returned. You can use this `Entity Id` later to identity in operations such as Retrieve, Update or Delete.

The following example shows how to create a task and set its name and start date fields:

```
// Create new Task
public static void CreateTask(string taskId)
{
        //Create an entity representing the new task
        GenericEntity task = new GenericEntity();
        task.Id = new EntityId();
        task.Id.TypeName = "Task";
        task.Id.Value = taskId;

        //Set task name
        FieldValue nameField = new FieldValue();
        nameField.FieldName = "Name";
        nameField.Value = "My Task";

        //Set task start date
        FieldValue startDateField = new FieldValue();
        startDateField.FieldName = "StartDate";
        startDateField.Value = DateTime.Now;

        //Assign the fields to the task
        task.Values = new FieldValue[] { nameField, startDateField };

        CreateMessage createTaskMessage=new CreateMessage();
        createTaskMessage.Entity = task;

        //Send the CreateMessage to the server for execution
        client.Execute(new BaseMessage[] {createTaskMessage});
}
```

PHP Sample Code

```
        //Create an entity representing the new task
        $newTask = new stdClass();
        $newTask->Id = new stdClass();
        $newTask->Id->TypeName = 'Issue';
        $newTask->Id->Value = null;

        //Set issue name
        $nameField = new stdClass();
        $nameField->FieldName = "Title";
        $nameField->Value = new SoapVar("My Issue", XSD_STRING, "string",
"http://www.w3.org/2001/XMLSchema");

        //Set task start date
        $startDateField = new stdClass();
        $startDateField->FieldName = "DueDate";
        $date = '2012-05-01';
        $startDateField->Value = new SoapVar($date, XSD_DATETIME, "dateTime",
"http://www.w3.org/2001/XMLSchema");

        //Assign the fields to the task
        $newTask->Values = array($nameField, $startDateField);

          $createMeesage = new stdClass();
          $createMeesage->Entity = new SoapVar($newTask, SOAP_ENC_OBJECT,
"GenericEntity", 'http://clarizen.com/api');

        $request[] = new SoapVar($createMeesage, SOAP_ENC_OBJECT, 'CreateMessage',
$soapApiUrl);
        $result = $client->Execute(array("request"=>$request));
```

UpdateMessage

Use the `UpdateMessage` request when you need to update the fields of an entity. `UpdateMessage` has one property named Entity. Use this property to fill new values of the fields you want to update. To indicate which object should be updated, you must provide an `Entity Id` for the Entity you pass.

Important When updating an entity, make sure you pass **only** the fields that need to be updated. Doing so, you will avoid possible performance implications, as well as erroneous situations similar to example below.

Following is an example of erroneous case, caused by redundant fields in the `UpdateMessage`:

If you retrieve a `Task` entity with its `Name` and `StartDate` and `EndDate` fields populated, and you want to update the task `StartDate`, you should create a new Entity, set its `Entity Id`, add only the `StartDate` field and set its new value and then send the `UpdateMessage` request. If, for example, you will send the `EndDate` field too, the following problems may occur:

- You might overwrite changes made to the `EndDate` field by another user.
- If `EndDate` has to be calculated by the system according to the `StartDate` and `Duration` of a task, it will now be considered "Manually Set" by the user.

A response of type `Result` will be returned for `UpdateMessage` requests. The result does not contain properties specific to the Update operation.

The following example shows how to update a reference between a task and its Manager using the `UpdateMessage` request:

```
static void UpdateManager(string taskId, string managerId)
{
        //Create an entity representing the task to update
        GenericEntity task = new GenericEntity();
        task.Id = new EntityId();
        task.Id.TypeName = "Task";
        task.Id.Value = taskId;

        //Create a FieldValue representing the Manager field
        FieldValue managerField = new FieldValue();
        managerField.FieldName = "Manager";
        //Set the field value to the external identifier of the new manager
        managerField.Value = managerId;

        task.Values = new FieldValue[] { managerField };

        UpdateMessage updateMsg = new UpdateMessage();
        updateMsg.Entity = task;

        //Update the entity
        Result result = client.Execute(new BaseMessage[] { updateMsg })[0];
        if (result.Success)
        {
        ...
        }
}
```

PHP Sample Code

```
        $updatedIssue = new stdClass();
        $updatedIssue->Id = new stdClass();
        $updatedIssue->Id->TypeName = 'Issue';
        $updatedIssue->Id->Value = '84842b75-b2e0-4a05-a44c-1f581d8d73ab';
```

```
       //Set issue name
       $nameField = new FieldValue();
       $nameField->FieldName = "Title";
       $nameField->Value = new SoapVar("My Updated Issue", XSD_STRING, "string", 
"http://www.w3.org/2001/XMLSchema");

        $updatedIssue->Values = array($nameField);

        $updateMessage = new stdClass();
       $updateMessage->Entity = new SoapVar($updatedIssue, SOAP_ENC_OBJECT, 
"GenericEntity", 'http://clarizen.com/api');


       $request[] = new SoapVar($updateMessage, SOAP_ENC_OBJECT, 'UpdateMessage', 
$soapApiUrl);
       $result = $client->Execute(array("request"=>$request));
```

RetrieveMessage\ RetrieveMultipleMessage

Use the `RetrieveMessage` and `RetrieveMultipleMessage` requests to retrieve a single or multiple entities by specifying their IDs.
Be sure that you request entities that belong to the same `Entity type` through `RetrieveMultipleMessage` request. If you need to retrieve entities that belong to different `Entity types`, you should issue several `RetrieveMultipleMessage` requests.

Important Only entities that belong to the same `Entity type` can be retrieved in one `RetrieveMultipleMessage request`.

Both request types allow you to indicate which fields to retrieve. You should retrieve only minimal required fields to keep good performance during the database query and to minimize the amount of data transferred from server to the client.

Important To avoid performance implications retrieve only fields essential to cover required functionality.

A response of type `RetrieveResult` or `RetrieveMultipleResult` is returned containing the requested entities with the requested fields populated. If the entity cannot be retrieved for one of the reasons listed below, `null` will be returned and the Error field will contain the reason:

- Entity does not exist, could be deleted by the other user
- User does not have sufficient privileges to access it

`RetrieveMultipleResult` contains an array of `RetrieveResult` objects each correlated with the `Entity IDs` requested in the `RetrieveMultipleMessage` request.

Note: `RetrieveMultipleResult` will always contain the same number of `RetrieveResult` objects, placed in the same order as was requested.
For example, if you request 10 entities, `RetrieveMultipleResult` will always contain 10 `RetrieveResult` objects in the same order as requested, also in case when specific entity(s) was not found due to one of the reasons listed above.

The following sample code shows how to retrieve a task with a known `Entity Id` and access its `Name` and `StartDate` fields:

```
static void RetrieveTask(string taskId)
{
	//Create the retrieve message
	RetrieveMessage retrieveMsg = new RetrieveMessage();
	retrieveMsg.Id = new EntityId();
	retrieveMsg.Id.TypeName = "Task";
	retrieveMsg.Id.Value = taskId;

	//Retrieve only the fields you need.
	retrieveMsg.Fields = new string[] { "Name" , "StartDate" };

	BaseMessage[] messages = new BaseMessage[] { retrieveMsg };

	//Send the retrieve message to the server and cast the result to correct 
type
	RetrieveResult result = (RetrieveResult)client.Execute(messages)[0];

	//Always check the result status
	if (result.Success)
	{
		//Cast the returned entity to GenericEntity
		GenericEntity task = (GenericEntity)result.Entity;

		//Now you can access the specific fields that were retrieved
		string taskName = (string)task.Values[0].Value;
		DateTime startDate = (DateTime)task.Values[1].Value;
	}
}
```

PHP Sample Code

```
//Retrieve an Issue Title and DueDate

	$retreiveIssue = new stdClass();
	$retreiveIssue->Id = new stdClass();
	$retreiveIssue->Id->TypeName = 'Issue';
	$retreiveIssue->Id->Value = '84842b75-b2e0-4a05-a44c-1f581d8d73ab'; //ID of 
the issue
	 $retreiveIssue->Fields =  array('Title','DueDate');

	$request[] = new SoapVar($retreiveIssue, SOAP_ENC_OBJECT, 'RetrieveMessage', 
$soapApiUrl);
	$result = $client->Execute(array("request"=>$request));
```

DeleteMessage

Use the `DeleteMessage` request to delete an entity by specifying its ID.

PHP Sample Code

```
	$deleteIssue = new stdClass();
	$deleteIssue->Id = new stdClass();
	$deleteIssue->Id->TypeName = 'Issue';
	$deleteIssue->Id->Value = '84842b75-b2e0-4a05-a44c-1f581d8d73ab';

	$request[] = new SoapVar($deleteIssue, SOAP_ENC_OBJECT, 'DeleteMessage', 
$soapApiUrl);
	$result = $client->Execute(array("request"=>$request));
```