# rtcclient Documentation

*Release 0.1.dev95*

**Di Xu**

**Aug 17, 2023**

# Contents

IBM® Rational Team Concert™, is built on the Jazz platform, allowing application development teams to use one tool to plan across teams, code, run standups, plan sprints, and track work. For more info, please refer to here.

**IMPORTANT NOTE: This is NOT an official-released Python-based RTC Client.**

This library can help you:

- Interacts with an RTC server to retrieve objects which contain the detailed information/configuration, including **Project Areas**, **Team Areas**, **Workitems** and etc;
- Creates all kinds of **Workitems** through self-customized templates or copies from some existing **Workitems**;
- Performs some actions on the retrieved **Workitems**, including get/add **Comments**, get/add/remove **Subscribers/Children/Parent** and etc;
- Query **Workitems** using specified filtered rules or directly from your saved queries;
- Logs all the activities and messages during your operation;

CHAPTER 1

# Python & Rational Team Concert Versions

The project has been tested against `Rational Team Concert` **5.0.1** and **5.0.2** on Python 2.6, 2.7 and 3.3.

CHAPTER 2

# Important Links

- Support and bug-reports: https://github.com/dixudx/rtcclient/issues?q=is%3Aopen+sort%3Acomments-desc
- Project source code: https://github.com/dixudx/rtcclient
- Project documentation: https://readthedocs.org/projects/rtcclient/

CHAPTER 3

# User Guide

## 3.1 Authors

Carsten Sauerbrey <carsten.sauerbrey@rohde-schwarz.com> Di Xu <dixudx@users.noreply.github.com> Di Xu <stephenhsu90@gmail.com> Felipe Ruhland <felipe.ruhland@gmail.com> Siddharth Kaul <siddharth.kaul@rail.bombardier.com> stephenhsu <stephenhsu90@gmail.com>

## 3.2 Introduction

In this section, some common terminologies are introduced. For more information, please visit Rational Collaborative Lifecycle Management Solution

### 3.2.1 Project Area

Project Area is, quite simply, an area in the repository where information about the project is stored.

In each of the Collaborative Lifecycle Management (CLM) applications, teams perform their work within the context of a project area. A project area is an area in the repository where information about one or more software projects is stored. **A project area defines the project deliverables, team structure, process, and schedule**. You access all project artifacts, such as iteration plans, work items, requirements, test cases, and files under source control within the context of a project area. Each project area has a process, which governs how members work.

For example, the project area process defines:

- User roles
- Permissions assigned to roles
- Timelines and iterations
- Operation behavior (preconditions and follow-up actions) for Change and Configuration Management and Quality Management

- Work item types and their state transition models (for Change and Configuration Management and Quality Management)

A project area is stored as a top-level or root item in a repository. A project area references project artifacts and stores the relationships between these artifacts. Access to a project area and its artifacts is controlled by access control settings and permissions. A project area cannot be deleted from the repository; however, it can be archived, which places it in an inactive state.

### 3.2.2 Team Area

You can create a team area to assign users in particular roles for work on a timeline or a particular set of deliverables. You can create a team area within an existing project area or another team area to establish a team hierarchy.

### 3.2.3 Component[3]

A configuration or set of configurations may be divided into components representing some user-defined set of object versions and/or sub-configurations; for example, components might be used to represent physical components or software modules. A provider is not required to implement components; they are used only as a way of limiting the scope of the closure over links. Components might or might not be resources; they might be dynamic sets of object versions chosen by other criteria such as property values. A provider can also treat each configuration and sub-configuration in a hierarchy as being separate components.

### 3.2.4 Change set[3]

A set of changes to be made to one or more configurations, where each change is described in terms of members (direct or indirect) that should be added to, replaced in, or removed from some configurations.

### 3.2.5 Role

Each project area and each team area can define a set of roles. The defined roles are visible in the area where they're declared and in all child areas. Roles defined in the project area can be assigned to users for the whole project area or they can be assigned in any team area. Roles defined in a team area can similarly be assigned in that team or in any child team. The ordering of roles in this section determines how they will be ordered in other sections of the editor, but it does not affect the process runtime.

### 3.2.6 Administrator

If you require permissions, contact an administrator. Project administrators can modify and save this project area and its team areas.

### 3.2.7 PlannedFor

In modern software development, a release is divided into a series of fixed-length development periods, typically ranging from two to six weeks, called iterations. Planning an iteration involves scheduling the work to be done during an iteration and assigning individual work items to members of the team.

Iteration planning takes place in the context of a project area. Each project area has a development line that is divided into development phases or iterations. For each iteration, you can create an iteration plan.

[3] SCM Data Model

The project *plannedfor* defines a start and end date along with an iteration breakdown.

### 3.2.8 Workitem

You can use work items to manage and track not only your work, but also the work of your team.

### 3.2.9 Workitem Type

A workitem type is a classification of work items that has a specific set of attributes. Each predefined process template includes the work item types that allow users to work in that process. For example, the Scrum process includes work item types such as *Epic*, *Story*, *Adoption Item*, *Task*, and *Retrospective*, which support an agile development model. The Formal Project Management process, which supports a more traditional development model, includes workitem types such as *Project Change Request*, *Business Need*, and *Risk*. Some work item types, such as *Defect* and *Task*, are used by multiple processes.

### 3.2.10 Workitem Type Category

Each work item type belongs to a work item category. Multiple work item types can belong to the same work item category. The work item types of a work item type category share workflow and custom attributes. When you create a work item type, you must associate it with a category. If you intend to define a unique workflow for the new work item type, create a new category and associate it with the work item type. Otherwise, you can associate the work item type with an existing category.

## 3.3 Workitem Attributes[1]

Attributes identify the information that you want to capture when users create and modify work items. Attributes are similar to fields in records. Work item types include all the built-in attributes that are listed in below Table. Note, however, that not every ready-to-use work item presentation is configured to display all of the built-in attributes in the Rational Team Concert™ Eclipse and web clients. You can customize the attributes that a work item type contains and the presentations that are used to display these attributes. For example, you can customize attributes to add behavior. Such behaviors can include validating an attribute value, or setting an attribute value that is based on other attribute values.

All the attributes of the `rtcclient.workitem.Workitem` can be accessed through **dot notation** and **dictionary**.

### 3.3.1 Built-in Attributes

Table1. Built-in Attributes

[1] Workitem Customization Overview

| Name | Type | ID | Description |
|---|---|---|---|
| Archived | Boolean | archived | Specifies whether the work item is archived. |
| Comments | Comments | comments | Comments about the work item. |
| Corrected Estimate | Duration | correctedEstimate | Correction to the original time estimate (as specified by the Estimate attribute) to resolve the work item. |
| Created By | Contributor | creator | User who created the work item. |
| Creation Date | Timestamp | created | Date when the work item was created. |
| Description | Large HTML | description | Detailed description of the work item. For example, the description for a defect might include a list of steps to follow to reproduce the defect. Any descriptions that are longer than 32 KB are truncated, and the entire description is added as an attachment. |
| Due Date | Timestamp | due | Date by which the resolution of the work item is due. |
| Estimate | Duration | estimate | Estimated amount of time that it takes to resolve the work item. |
| Filed Against | Category | filedAgainst | Category that identifies the component or functional area that the work item belongs to. For example, your project might have GUI, Build, and Documentation categories. Each category is associated with a team area; that team is responsible for responding to the work item. |
| Found In | Deliverable | foundIn | Release in which the issue described in the work item was identified. |
| Id | Integer | identifier | Identification number that is associated with the work item. |
| Modified By | Contributor | modifiedBy | User who last modified the work item. |
| Modified Date | Timestamp | modified | Date when the work item was last modified. |
| Owned By | Contributor | ownedBy | Owner of the work item. |
| Planned For | Iteration | plannedFor | Iteration for which the work item is planned. |
| Priority | Priority | priority | Ranked importance of a work item. For example, *Low*, *Medium*, or *High*. |
| Project Area | ProjectArea | projectArea | Area in the repository where information about the project is stored. |
| Resolution | Small String | resolution | How the work item was resolved. |
| Resolution Date | Timestamp | resolved | Date when the work item was resolved. |
| Resolved By | Contributor | resolvedBy | User who resolved the work item. |

Table2. Built-in Attributes (cont'd)

| Name | Type | ID | Description |
|---|---|---|---|
| Restricted Access | UUID | contextId | Scope of access to the work item. |
| Severity | Severity | severity | Indication of the impact of the work item. For example, *Minor*, *Normal*, *Major*, or *Critical*. |
| Start Date | Timestamp | startDate | Date when work began on the work item. |
| Status | Small String | state | Status of the work item. For example, *New*, *In Progress*, or *Resolved*. |
| Subscribed By | Subscriptions | subscribers | Users who are subscribed to the work item. |
| Summary | Medium HTML | title | Brief headline that identifies the work item. |
| Tags | Tag | subject | Tags that are used for organizing and querying on work items. |
| Time Spent | Duration | timeSpent | Length of time that was spent to resolve the work item. |
| Type | Type | type | Type of work item. Commonly available types are *Defect*, *Task*, and *Story*. |

## 3.4 Installation

This part of the documentation covers the installation of rtcclient. The first step to using any software package is getting it properly installed.

### 3.4.1 Distribute & Pip

Installing rtcclient is simple with pip, just run this in your terminal:

```
$ pip install rtcclient
```

or, with easy_install:

```
$ easy_install rtcclient
```

### 3.4.2 Get from the Source Code

RTCClient is actively developed on GitHub, where the code is always available.

You can either clone the public repository and checkout released tags (e.g. tag 0.1.dev95):

```
$ git clone git://github.com/dixudx/rtcclient.git
$ cd rtcclient
$ git checkout tags/0.1.dev95
```

Once you have a copy of the source, you can embed it in your Python package, or install it into your site-packages easily:

```
$ python setup.py install
```

## 3.5 Quick Start

Eager to get started? This page gives a good introduction in how to get started with rtcclient.

First, make sure that:

- rtcclient is *installed*
- rtcclient is up-to-date

RTCClient is intended to map the objects in RTC (e.g. Project Areas, Team Areas, Workitems) into easily managed Python objects

Let's get started with some simple examples.

### 3.5.1 Setup Logging

You can choose to enable logging during the using of rtcclient. Default logging is for console output. You can also add your own *logging.conf* to store all the logs to your specified files.

```
>>> from rtcclient.utils import setup_basic_logging
# you can remove this if you don't need logging
>>> setup_basic_logging()
```

### 3.5.2 Add a Connection to the RTC Server

Adding a connection with RTC Server is very simple.

Begin by importing the *RTCClient* module:

```
>>> from rtcclient import RTCClient
```

Now, let's input the url, username and password of this to-be-connected RTC Server. For this example,

```
>>> url = "https://your_domain:9443/jazz"
>>> username = "your_username"
>>> password = "your_password"
>>> myclient = RTCClient(url, username, password)
```

If your url ends with **ccm**, set *ends_with_jazz* to *False*, refer to **issue #68** for detailed explanation.

### 3.5.3 About Proxies

If your RTC Server is behind a proxy, you need to set *proxies* explicitly.

#### HTTP Proxies

```
>>> url = "https://your_domain:9443/jazz"
>>> username = "your_username"
>>> password = "your_password"
# example http proxy
>>> proxies = {
        'http': 'http://10.10.1.10:3128',
```

(continues on next page)

(continued from previous page)

```
        'https': 'http://10.10.1.10:1080',
    }
>>> myclient = RTCClient(url, username, password, proxies=proxies)
```

#### SOCKS Proxies

In addition to basic HTTP proxies, proxies using the SOCKS protocol are also supported.

```
>>> url = "https://your_domain:9443/jazz"
>>> username = "your_username"
>>> password = "your_password"
 # example socks proxy
>>> proxies = {
        "http": "socks5://127.0.0.1:1080",
        "https": "socks5://user:pass@host:port"
    }
>>> myclient = RTCClient(url, username, password, proxies=proxies)
```

### 3.5.4 Get a Workitem

You can get a workitem by calling `rtcclient.workitem.Workitem.getWorkitem`. The attributes of a workitem can be accessed through **dot notation** and **dictionary**.

Some common attributes are listed in *Built-in Attributes*.

For example,

```
>>> wk = myclient.getWorkitem(123456)
# get a workitem whose id is 123456
# this also works: getting the workitem using the equivalent string
>>> wk2 = myclient.getWorkitem("123456")
# wk equals wk2
>>> wk == wk2
True
>>> wk
<Workitem 123456>
>>> str(wk)
'141488'
>>> wk.identifier
u'141488'
# access the attributes through dictionary
>>> wk["title"]
u'title demo'
# access the attributes through dot notation
>>> wk.title
u'title demo'
>>> wk.state
u'Closed'
>>> wk.description
u'demo description'
>>> wk.creator
u'tester1@email.com'
>>> wk.created
u'2015-07-16T08:02:30.658Z'
```

(continues on next page)

(continued from previous page)

```
>>> wk.comments
[u'comment test 0', u'add comment test 1', u'add comment test 2']
```

### 3.5.5 About Returned Properties

You can also customize your preferred properties to be returned by specifying **returned_properties** when the called methods have this optional parameter, which can also **GREATLY IMPROVE** the performance of this client especially when getting or querying lots of workitems.

For the meanings of these attributes, please refer to *Built-in Attributes*.

Important Note: **returned_properties** is an advanced parameter, the returned properties can be found in *instance_obj.field_alias.values()*, e.g. *myworkitem1.field_alias.values()*. If you don't care the performance, just leave it alone with *None*.

```
>>> import pprint
# print the field alias
>>> pprint.pprint(wk2.field_alias, width=1)
{u'affectedByDefect': u'calm:affectedByDefect',
 u'affectsExecutionResult': u'calm:affectsExecutionResult',
 u'affectsPlanItem': u'calm:affectsPlanItem',
 u'apply_step': u'rtc_cm:apply_step',
 u'archived': u'rtc_cm:archived',
 u'blocksTestExecutionRecord': u'calm:blocksTestExecutionRecord',
 u'comments': u'rtc_cm:comments',
 u'contextId': u'rtc_cm:contextId',
 u'correctedEstimate': u'rtc_cm:correctedEstimate',
 u'created': u'dc:created',
 u'creator': u'dc:creator',
 u'description': u'dc:description',
 u'due': u'rtc_cm:due',
 u'elaboratedByArchitectureElement': u'calm:elaboratedByArchitectureElement',
 u'estimate': u'rtc_cm:estimate',
 u'filedAgainst': u'rtc_cm:filedAgainst',
 u'foundIn': u'rtc_cm:foundIn',
 u'identifier': u'dc:identifier',
 u'implementsRequirement': u'calm:implementsRequirement',
 u'modified': u'dc:modified',
 u'modifiedBy': u'rtc_cm:modifiedBy',
 u'ownedBy': u'rtc_cm:ownedBy',
 u'plannedFor': u'rtc_cm:plannedFor',
 u'priority': u'oslc_cm:priority',
 u'progressTracking': u'rtc_cm:progressTracking',
 u'projectArea': u'rtc_cm:projectArea',
 u'relatedChangeManagement': u'oslc_cm:relatedChangeManagement',
 u'relatedExecutionRecord': u'calm:relatedExecutionRecord',
 u'relatedRequirement': u'calm:relatedRequirement',
 u'relatedTestCase': u'calm:relatedTestCase',
 u'relatedTestPlan': u'calm:relatedTestPlan',
 u'relatedTestScript': u'calm:relatedTestScript',
 u'relatedTestSuite': u'calm:relatedTestSuite',
 u'resolution': u'rtc_cm:resolution',
 u'resolved': u'rtc_cm:resolved',
 u'resolvedBy': u'rtc_cm:resolvedBy',
 u'schedule': u'oslc_pl:schedule',
```

(continues on next page)

(continued from previous page)

```
 u'severity': u'oslc_cm:severity',
 u'startDate': u'rtc_cm:startDate',
 u'state': u'rtc_cm:state',
 u'subject': u'dc:subject',
 u'subscribers': u'rtc_cm:subscribers',
 u'teamArea': u'rtc_cm:teamArea',
 u'testedByTestCase': u'calm:testedByTestCase',
 u'timeSheet': u'rtc_cm:timeSheet',
 u'timeSpent': u'rtc_cm:timeSpent',
 u'title': u'dc:title',
 u'trackedWorkItem': u'oslc_cm:trackedWorkItem',
 u'tracksChanges': u'calm:tracksChanges',
 u'tracksRequirement': u'calm:tracksRequirement',
 u'tracksWorkItem': u'oslc_cm:tracksWorkItem',
 u'type': u'dc:type'}
```

Note: these field aliases may differ due to the type of workitems. But most of the common-used attributes will stay unchanged.

The *returned_properties* is a string **composed by the above values with comma separated**.

It will run faster if *returned_properties* is specified. Because the client will only get/request the attributes you specified.

```
>>> returned_properties = "dc:title,dc:identifier,rtc_cm:state,rtc_cm:ownedBy"
# specify the returned properties: title, identifier, state, owner
# This is optional. All properties will be returned if not specified
>>> wk_rp = myclient.getWorkitem(123456,
                                 returned_properties=returned_properties)
>>> wk_rp.identifier
u'141488'
# access the attributes through dictionary
>>> wk_rp["title"]
# access the attributes through dot notation
u'title demo'
>>> wk_rp.title
u'title demo'
>>> wk_rp.state
u'Closed'
>>> wk_rp.ownedBy
u'tester1@email.com'
```

### 3.5.6 Add a Comment to a Workitem

After getting the `rtcclient.workitem.Workitem` object, you can add a comment to this workitem by calling `addComment`.

```
>>> mycomment = wk.addComment("add comment test 3")
>>> mycomment
<Comment 3>
>>> mycomment.created
u'2015-08-22T03:55:00.839Z'
>>> mycomment.creator
u'tester1@email.com'
>>> mycomment.description
u'add comment test 3'
```

(continues on next page)

(continued from previous page)

```
>>> str(mycomment)
'3'
```

### 3.5.7 Get all Workitems

All workitems can be fetched by calling `rtcclient.client.RTCClient.getWorkitems`. It will take a long time to fetch all the workitems in some certain project areas if there are already many existing workitems.

If both *projectarea_id* and *projectarea_name* are None, all the workitems in all project areas will be returned.

```
>>> workitems_list = myclient.getWorkitems(projectarea_id=None,
                                           projectarea_name=None,
                                           returned_properties=returned_properties)
# get all workitems in a specific project area
>>> projectarea_name = "my_projectarea_name"
>>> workitems_list2 = myclient.getWorkitems(projectarea_name=projectarea_name,
                                            returned_properties=returned_properties)
```

### 3.5.8 Query Workitems

After customizing your query string, all the workitems meet the conditions will be fetched.

```
>>> myquery = myclient.query # query class
>>> projectarea_name = "my_projectarea_name"
# customize your query string
# below query string means: query all the workitems with title "use case 1"
>>> myquerystr = 'dc:title="use case 1"'
>>> returned_prop = "dc:title,dc:identifier,rtc_cm:state,rtc_cm:ownedBy"
>>> queried_wis = myquery.queryWorkitems(myquerystr,
                                         projectarea_name=projectarea_name,
                                         returned_properties=returned_prop)
```

More detailed and advanced syntax on querying, please refer to *query syntax*.

### 3.5.9 Query Workitems by Saved Query

You may have created several customized queries through RTC Web GUI or got some saved queries created by other team members. Using these saved queries

```
>>> myquery = myclient.query # query class
>>> saved_query_url = 'http://test.url:9443/jazz/xxxxxxxx&id=xxxxx'
>>> projectarea_name = "my_projectarea_name"
# get all saved queries
# WARNING: now the RTC server cannot correctly list all the saved queries
#          It seems to be a bug of RTC. Recommend using `runSavedQueryByUrl` to
#          query all the workitems if the query is saved.
>>> allsavedqueries = myquery.getAllSavedQueries(projectarea_name=projectarea_name)
# saved queries created by tester1@email.com
>>> allsavedqueries = myquery.getAllSavedQueries(projectarea_name=projectarea_name,
                                                 creator="tester1@email.com")
# my saved queries
>>> mysavedqueries = myquery.getMySavedQueries(projectarea_name=projectarea_name)
```

(continues on next page)

(continued from previous page)

```
>>> mysavedquery = mysavedqueries[0]
>>> returned_prop = "dc:title,dc:identifier,rtc_cm:state,rtc_cm:ownedBy"
>>> queried_wis = myquery.runSavedQuery(mysavedquery,
                                        returned_properties=returned_prop)
```

### 3.5.10 Query Workitems by Saved Query Url

You can also query all the workitems directly using your saved query's url.

```
>>> myquery = myclient.query # query class
>>> saved_query_url = 'http://test.url:9443/jazz/xxxxxxxx&id=xxxxx'
>>> returned_prop = "dc:title,dc:identifier,rtc_cm:state,rtc_cm:ownedBy"
>>> queried_wis = myquery.runSavedQueryByUrl(saved_query_url,
                                             returned_properties=returned_prop)
```

## 3.6 Advanced Usage

This document covers some of rtcclient more advanced features.

### 3.6.1 Query Syntax[2]

The following section describes the basic query syntax.

**Comparison Operators**

- = : test for equality of a term,
- != : test for inequality of a term,
- < : test less-than,
- > : test greater-than,
- <= : test less-than or equal,
- >= : test greater-than or equal,
- `in` : test for equality of any of the terms.

**Boolean Operators**

- `and` : conjunction

**Query Modifiers**

- `/sort` : set the sort order for returned items

**BNF**

```
query      ::= (term (boolean_op term)*)+ modifiers
term       ::= (identifier operator)? value+ | (identifier "in")? in_val
operator   ::= "=" | "!=" | "<" | ">" | "<=" | ">="
boolean_op ::= "and"
modifiers  ::= sort?
```

(continues on next page)

[2] Change Management Query Syntax

(continued from previous page)

```
sort       ::= "/sort" "=" identifier
identifier ::= word (":" word)?
in_val     ::= "[" value ("," value)* "]"
value      ::= (integer | string)
word       ::= /any sequence of letters and numbers, starting with a letter/
string     ::= '"' + /any sequence of characters/ + '"'
integer    ::= /any sequence of integers/
```

**Notes**

1. a word consists of any character with the Unicode class Alpha (alpha-numeric) as well as the characters ".", "-" and "_".
2. a string may include the quote character if preceded by the escape character "", as in "my "quoted" example".

### 3.6.2 Compose your Query String

Based on the above *query syntax*, it is easy to compose your own query string.

**Important Note**: For the *identifier* in *query syntax*, please refer to *field alias* and *Built-in Attributes*.

Here are several examples.

**Example 1**: Query all the defects with tags "bvt" whose state is not "Closed"

Note: here defects' state "default_workflow.state.s1" means "Closed". This may vary in your customized workitem type.

```
>>> query_str = ('dc:type="defect" and '
                 'rtc_cm:state!="default_workflow.state.s1" and '
                 'dc:subject="bvt"')
```

**Example 2**: Query all the defects which are modified after 18:42:30 on Dec. 02, 2008

Note: here defects' state "default_workflow.state.s1" means "Closed".

```
>>> query_str = 'dc:type="defect" and dc:modified>="12-02-2008T18:42:30"'
```

**Example 3**: Query all the defects with tags "bvt" or "testautomation"

```
>>> query_str = 'dc:type="defect" and dc:subject in ["bvt", "testautomation"]'
```

**Example 4**: Query all the defects owned/created/modified by "tester@email.com"

```
>>> user_url = "https://your_domain:9443/jts/users/tester@email.com"
>>> query_str = 'dc:type="defect" and rtc_cm:ownedBy="%s"' % user_url
>>> query_str = 'dc:type="defect" and dc:creator="%s"' % user_url
>>> query_str = 'dc:type="defect" and rtc_cm:modifiedBy="%s"' % user_url
```

Note: please replace *your_domain* with your actual RTC server domain.

**Example 5**: Query all the defects whose severity are "Critical"

```
>>> projectarea_name="My ProjectArea"
>>> severity = myclient.getSeverity("Critical",
                                    projectarea_name=projectarea_name)
>>> query_str = 'dc:type="defect" and oslc_cm:severity="%s"' % severity.url
```

**Example 6**: Query all the defects whose priority are "High"

```
>>> projectarea_name="My ProjectArea"
>>> priority = myclient.getPriority("High",
                                    projectarea_name=projectarea_name)
>>> query_str = 'dc:type="defect" and oslc_cm:priority="%s"' % priority.url
```

**Example 7**: Query all the defects whose FiledAgainst are "FiledAgainstDemo"

```
>>> projectarea_name="My ProjectArea"
>>> filedagainst = myclient.getFiledAgainst("FiledAgainstDemo",
                                            projectarea_name=projectarea_name)
>>> query_str = 'dc:type="defect" and rtc_cm:filedAgainst="%s"' % filedagainst.url
```

CHAPTER 4

# API Documentation

CHAPTER 5

# Indices and tables

- genindex
- modindex
- search