Collective Knowledge

Grigori Fursin

May 13, 2021

Introduction

Collective Knowledge framework (CK) helps to organize any software project as a database of reusable components (algorithms, datasets, models, frameworks, scripts, experimental results, papers, etc) with common automation actions and extensible meta descriptions based on FAIR principles (findability, accessibility, interoperability, and reusability).

The ultimate goal is to help everyone share, reuse, and extend their knowledge in the form of reusable artifacts and portable workflows with a common API, CLI, and JSON meta description.

See how CK helps to support collaborative and reproducible AI, ML, and systems R&D in some real-world use cases from Arm, General Motors, IBM, MLPerf, the Raspberry Pi foundation, and ACM: https://cKnowledge.org/partners.

CHAPTER 1

CK basics

1.1 Project overview

- Philosophical Transactions of the Royal Society (2020)

1.2 Why CK?

While working in large R&D projects with multiple partners to design efficient machine learning systems, we face the same problems over and over again:

- When we find an interesting GitHub project, a Jupyter/Colab notebook, or a Docker file from a research paper, we want to test it with different data, models, libraries, and AI frameworks. We may also want to compare it with another project or integrate it with some continuous integration service, or even rebuild it with a different compiler to run on a different hardware. However, we quickly get lost in the structure of the project, scripts, and APIs, and spend too much time trying to figure out how to make it compatible with different software and hardware, and how to plug in different datasets and models while correctly setting up numerous paths and variables. Eventually, we either give up or we manage to customize and run it but then we often struggle to reproduce and compare results (speed, throughput, accuracy, energy, costs). By that time, we most likely have to look at another project while facing the same problems again.
- We want to reuse some data, code, and models from our own project or from the project we participated in a few years ago, but we already forgot the details and scripts while the readme files and comments are too vague, our colleagues or students have left the project, and we either give up or start reimplementing everything from scratch.

Eventually, we realized that a possible solution is to organize software projects and folders as a sort of database of components (algorithms, datasets, models, frameworks, scripts, results from experiments, papers, etc) with extensible JSON meta descriptions. In such case we can have a simple tool to automatically find all related components from all projects. We can also implement common automation actions for related components that can be reused across different projects due to a unified API and CLI. We can even implement workflows from these actions that can automatically find and plug in all related components from all compatible projects thus minimizing manual interventions and providing a common interface for all shared projects and components.

We called this project Collective Knowledge (CK) because it helps users share their knowledge, experience, and best practices as reusable automation actions and components with a common API and meta description. Interestingly, the CK concept enables FAIR principles (findability, accessibility, interoperability, and reusability) published in this Nature article.

1.3 What is CK?

We have developed the Collective Knowledge framework (CK) as a small Python library with minimal dependencies to be very portable and have the possibility to be implemented in other languages such as C, C++, Java, and Go. The CK framework has a unified command line interface (CLI), a Python API, and a JSON-based web service to manage ***CK repositories*** and add, find, update, delete, rename, and move ***CK components*** (sometimes called ***CK entries*** or ***CK data***).

CK repositories are human-readable databases of reusable CK components that can be created in any local directory and inside containers, pulled from GitHub and similar services, and shared as standard archive files. CK components simply wrap user artifacts and provide an extensible JSON meta description with ***common automation actions*** for related artifacts.

Automation actions are implemented using ***CK modules*** - Python modules with functions exposed in a unified way via CK API and CLI and using extensible dictionaries for input/output (I/O). The use of dictionaries makes it easier to support continuous integration tools and web services and extend the functionality while keeping backward compatibility. The unified I/O also makes it possible to reuse such actions across projects and chain them together into unified pipelines and workflows.

Since we wanted CK to be non-intrusive and technology neutral, we decided to use a simple 2-level directory structure to wrap user artifacts into CK components:

wrappers

The root directory of the CK repository contains the `.ckr.json` file to describe this repository and specify dependencies on other CK repositories to explicitly reuse their components and automation actions.

CK uses `.cm` directories similar to `.git` to store meta information of all components as well as Unique IDs of all components to be able to find them even if their user-friendly names have changed over time (***CK alias***).

CK modules are always stored in ***module / < CK module name >*** directories in the CK repository. For example, `module/dataset` or `module/program`. They have a `module.py` with associated automation actions (for example, `module/dataset/module.py` or `module/program/module.py`). Such approach allows multiple users to add, improve, and reuse common automation action for related components rather than reimplementing them from scratch for each new project.

CK components are stored in ***< CK module name > / < CK data name >*** directories. For example, `dataset/text1234-for-nlp` or `dataset/some-images-from-imagenet`.

Each CK component has a `.cm` directory with the `meta.json` file describing a given artifact and `info.json` file to keep the provenance of a given artifact including copyrights, licenses, creation date, names of all contributors, and so on.

CK framework has an internal ***default CK repository*** with ***stable CK modules*** and the most commonly used automation actions across many research projects. When CK framework is used for the first time, it also creates a ***local CK repository*** in the user space to be used as a scratch pad.

CK provides a simple command line interface similar natural language to manage CK repositories, entries, and actions:

```
ck <action> <CK module name> (flags) (@input.json) (@input.yaml)
ck <action> <CK module name>:<CK entry name> (flags) (@input.json or @input.yaml)
ck <action> <CK repository name>:<CK module name>:<CK entry name>
```

The next example demonstrates how to compile and run the shared automotive benchmark on any platform, and then create a copy of the ***CK program component***:

```
pip install ck

ck pull repo --url=https://github.com/ctuning/ck-crowdtuning

ck search dataset --tags=jpeg
```

(continues on next page)

(continued from previous page)

```
ck search program:cbench-automotive-*

ck find program:cbench-automotive-susan

ck load program:cbench-automotive-susan

ck help program

ck compile program:cbench-automotive-susan --speed
ck run program:cbench-automotive-susan --env.OMP_NUM_THREADS=4

ck run program --help

ck cp program:cbench-automotive-susan local:program:new-program-workflow

ck find program:new-program-workflow

ck benchmark program:new-program-workflow --record --record_uoa=my-test

ck replay experiment:my-test
```

The CK program module describes dependencies on software detection plugins and meta packages usin simple tags with version ranges that the community has to agree on:

```
{
  "compiler": {
    "name": "C++ compiler",
    "sort": 10,
    "tags": "compiler,lang-cpp"
  },
  "library": {
    "name": "TensorFlow C++ API",
    "no_tags": "tensorflow-lite",
    "sort": 20,
    "version_from": [1,13,1],
    "version_to": [2,0,0],
    "tags": "lib,tensorflow,vstatic"
  }
}
```

CK also provides a Python library with a simple API that can be easily used in web applications or continuous integration services:

```
import ck.kernel as ck

# Equivalent of "ck compile program:cbench-automotive-susan --speed"
r=ck.access({'action':'compile', 'module_uoa':'program', 'data_uoa':'cbench-
↪automotive-susan',
             'speed':'yes'})
if r['return']>0: return r # unified error handling

print (r)

# Equivalent of "ck run program:cbench-automotive-susan --env.OMP_NUM_THREADS=4
r=ck.access({'action':'run', 'module_uoa':'program', 'data_uoa':'cbench-automotive-
↪susan',
             'env':{'OMP_NUM_THREADS':4}})
if r['return']>0: return r # unified error handling

print (r)
```

Based on the feedback from our users, we have recently developed an open ***CK platform*** to help the

community share CK components, create live scoreboards, and participate in collaborative experiments: https://cKnowledge.io.

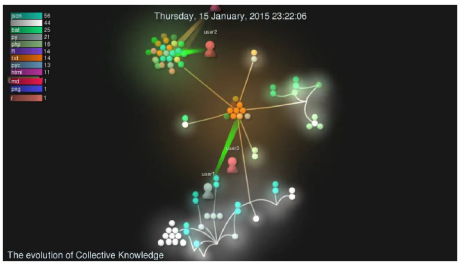
- We suggest you to read this nice blog post from Michel Steuwer about CK basics!
- You can find a partial list of CK-compatible repositories at cKnowledge.io/repos.

1.4 How CK supports collaborative and reproducible ML&systems research

It is a very tedious, ad-hoc, and time consuming process to design complex computational systems that can run AI, ML, and other emerging workloads in the most efficient way due to continuously changing software, hardware, models, data sets, and research techniques.

The first reason why we have developed CK was to connect our colleagues, students, researchers, and engineers from different workgroups to collaboratively solve these problems and decompose complex systems and research projects into reusable, portable, customizable, and non-virtualized CK components with unified automation actions, Python APIs, CLI, and JSON meta description.

We used CK as a common playground to prototype and test different abstractions and automations of many ML&systems tasks in collaboration with our great academic and industrial partners while agreeing on APIs and meta descriptions of all components. Over years the project grew from several core CK modules and abstractions to 150+ CK modules with 600+ actions automating typical, repetitive, and tedious tasks from ML&systems R&D. See this fun video and the knowledge graph showing the evolution of CK over time.

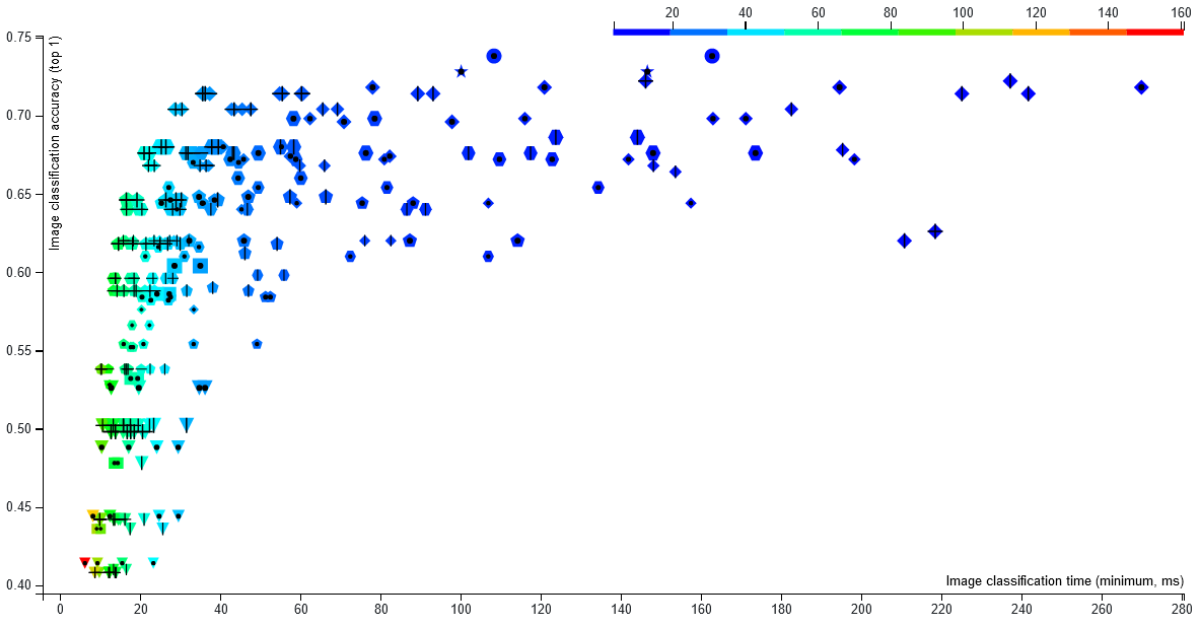


CK evolution

For example, CK now features actions for software detection, package installation and platform/OS detection to automate the detection and installation of all the dependencies including data sets and models required by different research projects.

Thanks to unified automation actions, APIs, and JSON meta descriptions of such components, we could apply the DevOps methodology to connect them into platform-agnostic, portable, customizable, and reproducible program pipelines (workflows). Such workflows can automatically adapt to evolving environments, models, data sets, and non-virtualized platforms by automatically detecting the properties of a target platform, finding all required components on a user platform using CK software detection plugins based on the list of all dependencies, installing missing components using portable CK meta packages, building and running code, and unifying and testing outputs.

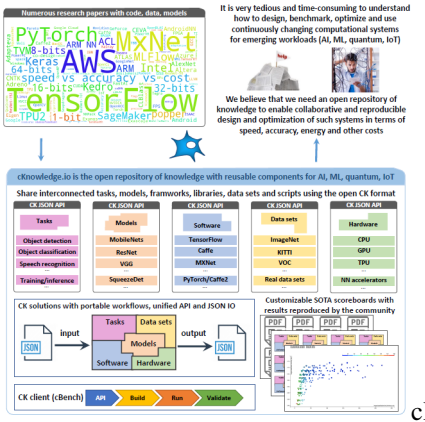
Eventually, CK helped to connect researchers and practitioners to collaboratively co-design, benchmark, optimize, and validate novel AI, ML, and quantum techniques using the open repository of knowledge with live SOTA scoreboards and reproducible papers. Such scoreboards can be used to find and rebuild the most efficient AI/ML/SW/HW stacks on a Pareto frontier across diverse platforms from supercomputers to edge devices while trading off speed, accuracy, energy, size, and different costs. Stable and optimized CK workflows can be then deployed inside Docker and Kubernetes to simplify the integration and adoption of innovative technology in production.



Pareto-efficient ML/SW/HW stacks

Our goal is to use the CK technology to bring DevOps principles to ML&systems R&D, make it more collaborative, reproducible, and reusable, enable portable MLOps, and make it possible to understand what happens inside complex and "black box" computational systems.

Our dream is to see portable workflows shared along with new systems, algorithms, and published research techniques to be able to quickly test, reuse and compare them across different data sets, models, software, and hardware! That is why we support related reproducibility and benchmarking initiatives including artifact evaluation, MLPerf, PapersWithCode, and ACM artifact review and badging.



cKnowledge platform concept

You can learn more about the CK project from CK presentations and white papers.

Even though the CK technology is used in production for more than 5 years, it is still a proof-of-concept prototype requiring further improvements and standardization. Depending on the available resources, we plan to develop a new, backward-compatible and more user-friendly version - please get in touch if you are interested to know more!

1.5 CK platform

- cKnowledge.io: the open portal with stable CK components, workflows, reproduced papers, and SOTA scoreboards for complex computational systems (AI,ML,quantum,IoT):
  - **Browse all CK ML&systems components**
  - Browse CK compatible repositories

  - Browse SOTA scoreboards powered by CK workflows
  - Browse all shared CK components
- Check documentation
- Our reproducibility initiatives for systems and ML conferences

1.6 CK-powered workflows, automation actions, and reusable artifacts for ML&systems R&D

- Real-world use-cases
- Reproducibility initiatives: [methodology], [events]
- Showroom (public projects powered by CK):
  - MLPerf automation
  - Student Cluster Competition automation: SCC18, digital artifacts
  - ML-based autotuning project: reproducible paper demo, MILEPOST
  - Stable Docker containers with CK workflows: MLPerf example, cKnowledge.io, Docker Hub
  - Quantum hackathons
  - ACM SW/HW co-design tournaments for Pareto-efficient deep learning
  - Portable CK workflows and components for:
    * TensorFlow
    * PyTorch
    * TensorRT
    * OpenVino
    * individual NN operators
    * object detection
  - GUI to automate ML/SW/HW benchmarking with MLPerf example (under development)
  - Reproduced papers
  - Live scoreboards for reproduced papers
- Examples of CK components (automations, API, meta descriptions):
  - *program : image-classification-tflite-loadgen* [cKnowledge.io] [GitHub]
  - *program : image-classification-tflite* [GitHub]
  - *soft : lib.mlperf.loadgen.static* [GitHub]
  - *package : lib-mlperf-loadgen-static* [GitHub]
  - *package : model-onnx-mlperf-mobilenet* [GitHub]
  - *package : lib-tflite* [cKnowledge.io] [GitHub]
  - *docker : object-detection-tf-py.tensorrt.ubuntu-18.04* [cKnowledge.io]
  - *docker : ** [GitHub]
  - *docker : speech-recognition.rnnt* [GitHub]
  - *package : model-tf-** [GitHub]
  - *script : mlperf-inference-v0.7.image-classification* [cKnowledge.io]

- *jnotebook : object-detection* [GitHub]

CHAPTER 2

Feedback and feature requests

We rely on your feedback to improve this open technology! If something doesn't work as expected or you have new suggestions and feature requests please do not hesitate to get in touch!

CHAPTER 3

Acknowledgments

We would like to thank all contributors and collaborators for their support, fruitful discussions, and useful feedback!

CHAPTER 4

CK installation

You can install the Collective Knowledge framework on most platforms using PIP as follows:

```
pip install ck
```

You can also install CK using a specific Python version (for example, Python 3.6 or for Python 2.7):

```
python3.6 -m pip install ck
```

or

```
python2.7 -m pip install ck
```

You may need to add flag "–user" to install the client in your user space:

```
pip install ck --user
python3.6 -m pip install ck --user
```

You should now be able to run CK using one of the following alternative commands:

```
ck

python3.6 -m ck
```

If the installation is successful, you will see some internal information about the CK installation and a Python version used:

```
CK version: 1.15.0

Python executable used by CK: /usr/bin/python

Python version used by CK: 2.7.12 (default, Oct  8 2019, 14:14:10)
   [GCC 5.4.0 20160609]

Path to the default repo: /home/fursin/fggwork/ck/ck/repo
Path to the local repo:   /home/fursin/CK/local
Path to CK repositories:  /home/fursin/CK

Documentation:        https://github.com/ctuning/ck/wiki
```

(continues on next page)

(continued from previous page)

```
CK Google group:      https://bit.ly/ck-google-group
CK Slack channel:     https://cKnowledge.org/join-slack
Stable CK components: https://cKnowledge.io
```

4.1 Prerequisites

The CK framework requires minimal dependencies: Python 2.7+ or 3.x, PIP and Git.

4.2 Linux

You need to have the following packages installed (Ubuntu example):

```
sudo apt-get install python3 python3-pip git wget
```

4.3 MacOS

```
brew install python3 python3-pip git wget
```

4.4 Windows

- Download and install Git from git-for-windows.github.io.
- Download and install any Python from www.python.org/downloads/windows.

4.5 Android (Linux host)

These dependencies are needed to cross-compile for Android (tested on Ubuntu 18.04 including Docker and Windows 10 Subsystem for Linux).

```
 sudo apt update
 sudo apt install git wget libz-dev curl cmake
 sudo apt install gcc g++ autoconf autogen libtool
 sudo apt install android-sdk
 sudo apt install google-android-ndk-installer
```

4.6 Docker

We prepared several Docker images with the CK framework and AI/ML CK workflows at the cTuning Docker hub. Select the most relevant image and run it as follows:

```
docker run -p 3344:3344 -it {Docker image name from the above list} /bin/bash
```

4.7 Customization

Please check this wiki to learn more about how to customize your CK installation.

CHAPTER 5

Trying CK

5.1 How CK enables portable and customizable workflows

We originally developed CK to help our partners and collaborators implement modular, portable, customizable, and reusable workflows. We needed such workflows to enable collaborative and reproducible ML&systems R&D while focusing on deep learning benchmarking and ML/SW/HW co-design. We also wanted to automate and reuse tedious tasks that are repeated across nearly all ML&systems projects as described in our FOSDEM presentation.

In this section, we demonstrate how to use CK with portable and non-virtualized program workflows that can automatically adapt to any platform and user environment, i.e. automatically detect target platform properties and software dependencies and then compile and run a given program with any compatible dataset and model in a unified way.

Note that such approach also supports our reproducibility initiatives at ML&systems conferences to share portable workflows along with published papers. Our goal is to make it easier for the community to reproduce research techniques, compare them, build upon them, and adopt them in production.

5.2 CK installation

Follow this guide to install CK on Linux, MacOS, or Windows. Don't hesitate to contact us if you encounter any problem or have questions.

5.3 Pull CK repositories with the universal program workflow

Now you can pull CK repo with the universal program workflow.

```
ck pull repo --url=https://github.com/ctuning/ck-crowdtuning
```

CK will automatically pull all required CK repositories with different automation actions, benchmarks, and datasets in the CK format. You can see them as follows:

```
ck ls repo
```

By default, CK stores all CK repositories in the user space in *$HOME/CK-REPOS*. However, you can change it using the environment variable *CK_REPOS*.

5.4 Manage CK entries

You can now see all shared program workflows in the CK format:

```
ck ls program
```

You can find and investigate the CK format for a given program (such as *cbench-automotive-susan*) as follows:

```
ck find program:cbench-automotive-susan
```

You can see the CK meta description of this program from the command line as follows:

```
ck load program:cbench-automotive-susan
ck load program:cbench-automotive-susan --min
```

It may be more convenient to check the structure of this entry at GitHub with all the sources and meta-descriptions.

You can also see the CK JSON meta description for this CK program entry here. When you invoke automation actions in the CK module *program*, the automation code will read this meta description and perform actions for different programs accordingly.

5.5 Invoke CK automation actions

You can now try to compile this program on your platform:

```
ck compile program:cbench-automotive-susan --speed
```

CK will invoke the function "compile" in the module "program" (you can see it at GitHub or you can find the source code of this CK module locally using "ck find module:program"), read the JSON meta of *cbench-automotive-susan*, and perform a given action.

Note, that you can obtain all flags for a given action as follows:

```
ck compile program --help
```

You can update any above key from the command line by adding "–" to it. If you omit the value, CK will use "yes" by default.

When compiling program, CK will first attempt to automatically detect the properties of the platform and all required software dependencies such as compilers and libraries that are already installed on this platform. CK uses multiple plugins describing how to detect different software, models, and datasets.

Users can add their own plugins either in their own CK repositories or in already existing ones.

You can also perform software detection manually from the command line. For example you can detect all installed GCC or LLVM versions:

```
ck detect soft:compiler.gcc
ck detect soft:compiler.llvm
```

Detected software is registered in the local CK repository together with the automatically generated environment script (*env.sh* or *env.bat*) specifying different environment variables for this software (paths, versions, etc).

You can list registered software as follows:

```
ck show env
ck show env --tags=compiler
```

You can use CK as a virtual environment similar to venv and Conda:

```
ck virtual env --tags=compiler,gcc
```

Such approach allows us to separate CK workflows from hardwired dependencies and automatically plug in the requied ones.

You can now run this program as follows:

```
ck run program:cbench-automotive-susan
```

While running the program, CK will collect and unify various characteristics (execution time, code size, etc). This enables unified benchmarking reused across different programs, datasets, models, and platform. Furthermore, we can continue improving this universal program workflow to monitor CPU/GPU frequencies, performing statistical analysis of collected characteristics, validating outputs, etc:

```
ck benchmark program:cbench-automotive-susan --repetitions=4 --record --record_
↪uoa=ck_entry_to_record_my_experiment
ck replay experiment:ck_entry_to_record_my_experiment
```

Note that CK programs can automatically plug different datasets from CK entries that can be shared by different users in different repos (for example, when publishing a new paper):

```
ck search dataset
ck search dataset --tags=jpeg
```

Our goal is to help researchers reuse this universal CK program workflow instead of rewriting complex infrastructure from scratch in each research project.

5.6 Install missing packages

Note, that if a given software dependency is not resolved, CK will attempt to automatically install it using CK meta packages (see the list of shared CK packages at cKnowledge.io). Such meta packages contain JSON meta information and scripts to install and potentially rebuild a given package for a given target platform while reusing existing build tools and native package managers if possible (make, cmake, scons, spack, python-poetry, etc). Furthermore, CK package manager can also install non-software packages including ML models and datasets while ensuring compatibility between all components for portable workflows!

You can list CK packages available on your system (CK will search for them in all CK repositories installed on your system):

```
ck search package --all
```

You can then try to install a given LLVM on your system as follows:

```
ck install package --tags=llvm,v10.0.0
```

If this package is successfully installed, CK will also create an associated CK environment:

```
ck show env --tags=llvm,v10.0.0
```

By default, all packages are installed in the user space (*$HOME/CK-TOOLS*). You can change this path using the CK environment variable *CK_TOOLS*. You can also ask CK to install packages inside CK virtual environment entries directly as follows:

```
ck set kernel var.install_to_env=yes
```

Note that you can now detect or install multiple versions of the same tool on your system that can be picked up and used by portable CK workflows!

You can run a CK virtual environment to use a given version as follows:

```
ck virtual env --tags=llvm,v10.0.0
```

You can also run multiple virtual environments at once to combine different versions of different tools together:

```
ck show env
ck virtual env {UID1 from above list} {UID2 from above list} ...
```

Another important goal of CK is invoke all automation actions and portable workflows across all operating systems and environments including Linnux, Windows, MacOS, Android (you can retarget your workflow for Andoird by adding *–target_os=android23-arm64* flag to all above commands when installing packages or compiling and running your programs). The idea is to have a unified interface for all research techniques and artifacts shared along with research papers to make the onboarding easier for the community!

5.7 Participate in crowd-tuning

You can even participate in crowd-tuning of multiple programs and data sets across diverse platforms:.

```
ck crowdtune program:cbench-automotive-susan
ck crowdtune program
```

You can see the live scoreboard with optimizations here.

5.8 Use CK python API

You can also run CK automation actions directly from any Python (2.7+ or 3.3+) using one *ck.access* function:

```
import ck.kernel as ck

# Equivalent of "ck compile program:cbench-automotive-susan --speed"
r=ck.access({'action':'compile', 'module_uoa':'program', 'data_uoa':'cbench-
↪automotive-susan',
             'speed':'yes'})
if r['return']>0: return r # unified error handling

print (r)

# Equivalent of "ck run program:cbench-automotive-susan --env.OMP_NUM_THREADS=4
r=ck.access({'action':'run', 'module_uoa':'program', 'data_uoa':'cbench-automotive-
↪susan',
             'env':{'OMP_NUM_THREADS':4}})
if r['return']>0: return r # unified error handling

print (r)
```

5.9 Try the CK ML workflow

You can now try a more complex example with TensorFlow. You should pull the related CK repository and install the prebuilt version of TensorFlow CPU via CK:

```
ck pull repo:ck-tensorflow
ck install package --tags=lib,tensorflow,vcpu,vprebuilt
```

Check that it was successfully installed:

```
ck show env --tags=lib,tensorflow
```

You can find a path to a given entry describing this TF installation as follows:

```
ck find env:{env UID from above list}
```

Run the CK virtual environment and test TF:

```
ck virtual env --tags=lib,tensorflow
ipython
> import tensorflow as tf
>
```

You can try to run the CK image classification workflow example using the installed TF:

```
ck run program:tensorflow --cmd_key=classify
```

You can even try to rebuild TensorFlow via CK for your platform with CUDA:

```
ck install package:lib-tensorflow-1.7.0-cuda
```

CK will attempt detect your CUDA compiler and related libraries and tools including Java, Basel, and will then try to rebuild TF. Note that you may still need to install some extra dependencies yourself as described in this readme.

You can also try to run ML workflows from the MLPerf benchmarking initiative using this CK MLPerf repository.

Finally, you can try our recent MLPerf automation demo to automate submissions and validations of MLPerf results.

5.10 Further information

As you may notice, CK helps to convert ad-hoc research projects into a unified database of reusable components with common automation actions and unified meta descriptions. The goal is to promote artifact sharing and reuse while gradually substituting and unifying all tedious and repetitive research tasks!

You can find shared CK repositories, components, automation actions, and live scoreboards at the open cKnowledge.io platform.

You can also check how the universal CK program workflow was successfully reused in different projects including the ACM REQUEST tournaments to collaboratively co-design SW/HW stack for deep learning (Report about results of the 1st ReQuEST-ASPLOS'18 tournament and next steps and ACM ReQuEST-ASPLOS'18 proceedings with artifact descriptions) and reproducible quantum tournaments.

Finally, check this guide to learn how to add your own repositories, workflows, and components!

5.11 Contact the CK community

If you encounter problems or have suggestions, do not hesitate to contact us!

CHAPTER 6

The most common usage

Here we describe how to create and share new CK program workflows, software detection plugins, and packages either using new (empty) CK repositories or the existing ones. Portable CK program workflow is the most commonly used CK automation to compile, run, validate, and compare different algorithms and benchmarks across different compilers, libraries, models, datasets, and platforms.

We strongly suggest you to check the CK introduction and getting started guide first.

You can also check the following real-world use cases that are based on our portable and customizable CK workflow:

- MLPerf benchmark automation
- Reproducible ACM REQUEST tournaments to co-design Pareto-efficient AI/ML/SW/HW stacks
- Reproducible quantum hackathons
- Student Cluster Competition automation at SuperComputing: SCC18, general SCC
- reproducible and interactive paper for ML-based compilers

6.1 Initialize a new CK repository in the current directory (can be existing Git repo)

If you plan to contribute to already existing CK repositories you can skip this subsection. Otherwise, you need to manually create a new CK repository.

You need to choose some user friendly name such as "my-new-repo" and initialize CK repository in your current directory from the command line (Linux, Windows, and MacOS) as follows:

```
ck init repo:my-new-repo
```

If the current directory belongs to a Git repo, CK will automatically detect the URL. Otherwise you need to specify it from CLI too:

```
ck init repo:my-new-repo --url={Git URL}
```

You can then find where CK created this dummy CK repository using the following command:

```
ck where repo:my-new-repo
```

Note that if you want to share your repository with the community or within workgroups to reuse automations and components, you must create an empty repository at GitHub, GitLab, BitBucket or any other Git-based service. Let's say that you have created *my-new-repo* at *https://github.com/my_name*.

You can then pull this repository using CK as follows:

```
ck pull repo --url=https://github.com/my_name/my-new-repo
```

CK will then create my_name_repo repository locally, add default meta information, and will mark it as shared repository (to semi-automatically synchronize content with the associated Git repository):

```
ck where repo:my-new-repo
```

For example, you can later commit and push updates for this repository back to Git as follows:

```
ck push repo:my-new-repo
```

We then suggest you to make the first commit immediately after you pulled your dummy repository from GitHub to push automatically *.ckr.json* file with some internal meta information and Unique ID back to the GitHub.

Note that you can also commit and push all updates using Git commands directly from the CK repository directory! Do not forget to commit hidden CK directories *.cm/**!

You are now ready to use the newly created repository as a database to add, share, and reuse new components. You can also share this repository with your colleagues or the Artifact Evaluation Committee to test your components and workflows in a unified way:

```
ck pull repo --url=https://github.com/my_name/my-new-repo
```

6.2 Add dependency on other repositories to reuse automation actions and components

When you want to reuse existing CK automation actions and components from other repositories, you need to add a dependency to all these repositories in the *.ckr.json* file in your root CK repository:

```
  ...
  "dict": {
    ...
    "repo_deps": [
      {
        "repo_uoa": "ck-env"
      },
      {
        "repo_uoa": "ck-autotuning"
      },
      {
        "repo_uoa": "ck-mlperf",
        "repo_url": "https://github.com/ctuning/ck-mlperf"
      }
    ]
  }
```

Whenever someones pull your repository, CK will automatically pull all other required CK repositories with automation actions!

6.3 Add a new program workflow

You are now ready to add a new CK workflow to compile and run some algorithm or a benchmark in a unified way.

Since CK concept is about reusing and extending existing components with a common API similar to Wikipedia, we suggest you to look at this index of shared CK programs in case someone have already shared a CK workflows for the same or similar program!

If you found a similar program, for example "cbench-automotive-susan" you can create a working copy of this program in your new CK repository for further editing as follows:

```
ck pull repo:ctuning-programs
ck cp program:cbench-automotive-susan my-new-repo:program:my-copy-of-cbench-
↪automotive-susan
```

You now have a working copy of the CK "cbench-automotive-susan" program entry in your new repository that contains sources and the CK meta information about how to compile and run this program:

```
ck compile program:my-copy-of-cbench-automotive-susan --speed
ck run program:my-copy-of-cbench-automotive-susan
```

You can find and explore the new CK entry from command line as follows:

```
ck find program:my-copy-of-cbench-automotive-susan
```

You will see the following files in this directory:

- *.cm/desc.json* - description of all I/O types in all automation actions (empty by default - we can skip it for now)
- *.cm/info.json* - the provenance for this entry (creation date, author, license, etc)
- *.cm/meta.json* - **main CK meta information about how to compile, run, and validate this program**
- *susan.c" - source code of this program

Once again, do not forget to add .cm directories when committing to Git since *.cm* files are usually not visible from bash in Linux!

If you did not find a similar program, you can then create a new program using shared program templates as follows:

```
ck add my-new-repo:program:my-new-program
```

CK will then ask you to select the most close template:

```
0) C program "Hello world" (--template=template-hello-world-c)
1) C program "Hello world" with compile and run scripts (--template=template-hello-
↪world-c-compile-run-via-scripts)
2) C program "Hello world" with jpeg dataset (--template=template-hello-world-c-
↪jpeg-dataset)
3) C program "Hello world" with output validation (--template=template-hello-world-
↪c-output-validation)
4) C program "Hello world" with xOpenME interface and pre/post processing (--
↪template=template-hello-world-c-openme)
5) C++ TensorFlow classification example (--template=image-classification-tf-cpp)
6) C++ program "Hello world" (--template=template-hello-world-cxx)
7) Fortran program "Hello world" (--template=template-hello-world-fortran)
8) Java program "Hello world" (--template=template-hello-world-java)
9) Python MXNet image classification example (--template=mxnet)
10) Python TensorFlow classification example (--template=image-classification-tf-
↪py)
11) Python program "Hello world" (--template=template-hello-world-python)
```

(continues on next page)

(continued from previous page)

```
12) image-classification-tflite (--template=image-classification-tflite)
13) Empty entry
```

If you select "Python TensorFlow classification example", CK will create a working image classification program in your new repository with software dependencies on TensorFlow AI framework and compatible models.

Since it's a Python program, you do not need to compile it:

```
ck run program:my-new-program
```

Note that you can later make your own program a template by adding the following key to the *meta.json* file:

```
  "template": "yes"
```

6.3.1 Update program sources

If you found a similar program with all the necessary software dependencies, you can now update or change its sources for your own program.

In such case, you must update the following keys in the *meta.json* of this program entry:

- add your source files:

```
  "source_files": [
    "susan.c"
  ],
```

- specify a command line to run your program (see *run_cmd_main*):

```
  "run_cmds": {
    "corners": {
      "dataset_tags": [
        "image",
        "pgm",
        "dataset"
      ],
      "ignore_return_code": "no",
      "run_time": {
        "run_cmd_main": "$#BIN_FILE#$ $#dataset_path#$$#dataset_filename#$ tmp-
↪output.tmp -c"
      }
    },
    "edges": {
      "dataset_tags": [
        "image",
        "pgm",
        "dataset"
      ],
      "ignore_return_code": "no",
      "run_time": {
        "run_cmd_main": "$#BIN_FILE#$ $#dataset_path#$$#dataset_filename#$ tmp-
↪output.tmp -e"
      }
    },
```

Note that you can have more than one possible command line to run this program. In such case, CK will ask you which one to use when you run this program. For example, this can be useful to perform ML model training ("train"), validation ("test"), and classification ("classify").

You can also update meta.json keys to customize program compilation and execution:

```
  "build_compiler_vars": {
    "XOPENME": ""
  },

  "compiler_env": "CK_CC",

  "extra_ld_vars": "$<<CK_EXTRA_LIB_M>>$",

  "run_vars": {
    "CT_REPEAT_MAIN": "1",
    "NEW_VAR":"123"
  },
```

Note that you can update environment variables when running a given program in a unified way from the command line as follows:

```
ck run program:my-new-program --env.OMP_NUM_THREADS=4 --env.ML_MODEL=mobilenet-v3
```

You can also expose different algorithm parameters and optimizations via environment to apply customizable CK autotuner as used in this CK ReQuEST workflow to automatically explore (co-design) different MobileNets configurations in terms of speed, accuracy, and costs.

Here is the brief description of other important keys in CK program *meta.json*:

```
  "run_cmds": {

    "corners": {                # User key describing a given execution command line

      "dataset_tags": [         # dataset tags - will be used to query CK
        "image",                # and automatically find related entries such as␣
↪images
        "pgm",
        "dataset"
      ],

      "run_time": {             # Next is the execution command line format
                                # $#BIN_FILE#$ will be automatically substituted␣
↪with the compiled binary
                                # $#dataset_path#$$#dataset_filename#$ will be␣
↪substituted with
                                # the first file from the CK dataset entry (see␣
↪above example
                                # of adding new datasets to CK).
                                # tmp-output.tmp is and output file of a processed␣
↪image.
                                # Basically, you can shuffle below words to set␣
↪your own CMD

        "run_cmd_main": "$#BIN_FILE#$ $#dataset_path#$$#dataset_filename#$ tmp-
↪output.tmp -c",

        "run_cmd_out1": "tmp-output1.tmp",  # If !='', add redirection of the␣
↪stdout to this file
        "run_cmd_out2": "tmp-output2.tmp",  # If !='', add redirection of the␣
↪stderr to this file

        "run_output_files": [               # Lists files that are produced during
                                            # benchmark execution. Useful when␣
↪program
                                            # is executed on remote device (such as
                                            # Android mobile) to pull necessary
                                            # files to host after execution
```

(continues on next page)

(continued from previous page)

```
          "tmp-output.tmp",
          "tmp-ck-timer.json"
        ],


        "run_correctness_output_files": [   # List files that should be used to␣
↪check
                                            # that program executed correctly.
                                            # For example, useful to check␣
↪benchmark correctness
                                            # during automatic compiler/hardware␣
↪bug detection
          "tmp-output.tmp",
          "tmp-output2.tmp"
        ],

        "fine_grain_timer_file": "tmp-ck-timer.json"  # If XOpenME library is used,
↪ it dumps run-time state
                                                      # and various run-time␣
↪parameters (features) to tmp-ck-timer.json.
                                                      # This key lists JSON files␣
↪to be added to unified
                                                      # JSON program workflow␣
↪output
      },

      "hot_functions": [                  # Specify hot functions of this program
        {                                 # to analyze only these functions during␣
↪profiling
          "name": "susan_corners",        # or during standalone kernel extraction
          "percent": "95"                 # with run-time memory state (see
↪"codelets"
                                          #  shared in CK repository from the␣
↪MILEPOST project
                                          #  and our recent papers for more info)
        }
      ]

      "ignore_return_code": "no"          # Some programs have return code >0 even␣
↪during
                                          # successful program execution. We use␣
↪this return code
                                          # to check if benchmark failed␣
↪particularly during
                                          # auto-tuning or compiler/hardware bug␣
↪detection
                                          #  (when randomly or semi-randomly␣
↪altering code,
                                          #   for example, see Grigori Fursin's PhD␣
↪thesis with a technique
                                          #   to break assembler instructions to␣
↪detect
                                          #   memory performance bugs)
    },
    ...
  },
```

You can also check how to use pre and post-processing scripts before and after running your program in this example from the Student Cluster Competition'18.

6.4 Update software dependencies

If you new program rely on extra software dependencies (compilers, libraries, models, datasets) you must first find the ones you need in this online index of software detection plugins. You can then specify the tags and versions either using *compile_deps* or *run_deps* keys in the *meta.json* of your new program as follows:

```
"compile_deps": {
  "compiler": {
    "local": "yes",
    "name": "C compiler",
    "sort": 10,
    "tags": "compiler,lang-c"
  },
  "xopenme": {
    "local": "yes",
    "name": "xOpenME library",
    "sort": 20,
    "tags": "lib,xopenme"
  }
},
```

```
"run_deps": {
  "lib-tensorflow": {
    "local": "yes",
    "name": "TensorFlow library",
    "sort": 10,
    "tags": "lib,tensorflow",
    "no_tags":"vsrc"
  },
  "tensorflow-model": {
    "local": "yes",
    "name": "TensorFlow model (net and weights)",
    "sort": 20,
    "tags": "tensorflowmodel,native"
  }
},
```

As a minimum, you just need to add a new sub-key such as "lib-tensorflow", a user-friendly name such as "TensorFlow library", one or more tags to specify your software detection plugin from above index (CK will use these tags to find related CK components), and an order in which dependencies will be resolved using the *sort* key.

You can also select version ranges with the following keys:

```
"version_from": [1,64,0], # inclusive
"version_to": [1,66,0]    # exclusive
```

Have a look at a more complex meta.json of the Caffe CUDA package.

Whenever CK compiles or runs programs, it first automatically resolves all software dependencies. CK also registers all detected software or installed packages in the CK virtual environment (see the getting started guide) with automatically generated *env.sh* or *env.bat* batch scripts. These scripts are then loaded one after another based on the above *sort* key to aggregate all required environment variables and pass them either to the compilation or execution scripts. Your scripts and algorithms can then use all these environment variables to customize compilation and execution without any need to change paths manually, i.e. we enable portable workflows that can automatically adapt to a user environment.

6.5 Reuse or add basic datasets

We have developed a simple mechanism in the CK workflow to reuse basic (small) datasets such a individual images.

You can find already shared datasets using this online index.

If you want to reuse them in your program workflow, you can find the related one, check its tags (see the meta.json of the image-jpeg-0001 dataset), and add them to your program meta as follows:

```
  "run_cmds": {
    "corners": {
      "dataset_tags": [
        "image",
        "pgm",
        "dataset"
      ],
      "ignore_return_code": "no",
      "run_time": {
        "run_cmd_main": "$#BIN_FILE#$ $#dataset_path#$$#dataset_filename#$ tmp-
↪output.tmp -c"
      }
    },
```

CK then search for all dataset entries in all pulled CK repositories using these flags, and will ask a user which one to use when multiple entries are found. CK will then substitute *$#dataset_path#$$#dataset_filename#$* with the full path and a file of the dataset from the selected entry.

Such approach allows to get rid of hardwired paths in ad-hoc scripts while easily sharing and reusing related datasets. Whenever you pull a new repository with CK datasets, they can be automatically picked up by a given program workflow!

For example you can see all pgm images available in your CK repositories as follows:

```
ck pull repo:ctuning-datasets-min
ck search dataset --tags=dataset,image,pgm
```

You can add a new dataset in your new repository as follows:

```
ck add my-new-repo:dataset:my-new-dataset
```

You will be asked to enter some tags and to select a file that will be copied into your new CK entry.

Note that for large and complex datasets such as ImageNet, we use CK packages that can download a given dataset and even process it depending on other software dependencies. For example one may need a different procedure when using TensorFlow or PyTorch or MXNet.

6.6 Add new CK software detection plugins

If CK software plugin doesn't exist for a given code, data, or models, you can add a new one either in your own repository or in already existing ones.

We suggest you to find the most close software detection plugin using this online index, pull this repository, and make a copy in your repository as follows:

```
ck copy soft:lib.armcl my-new-repo:soft:lib.my-new-lib
```

or

```
ck copy soft:dataset.imagenet.train my-new-repo:soft:my-new-data-set
```

Alternatively, you can add a new soft entry and select the most relevant template:

```
ck add my-new-repo:soft:my-new-data-set
```

You must then update related keys in the *.cm/meta.json* file of the new CK entry. You can find it as follows:

```
ck find soft:lib.my-new-lib
```

Typical software meta description:

```
{
  "auto_detect": "yes",
  "customize": {
    "check_that_exists": "yes",
    "ck_version": 10,
    "env_prefix": "CK_ENV_LIB_ARMCL",
    "limit_recursion_dir_search": {
      "linux": 4,
      "win": 4
    },
    "soft_file": {
      "linux": "libarm_compute.a",
      "win": "arm_compute.lib"
    },
    "soft_path_example": {
    }
  },
  "soft_name": "ARM Compute Library",
  "tags": [
    "lib",
    "arm",
    "armcl",
    "arm-compute-library"
  ]
}
```

First, you must update *tags* keys for your new software, *soft_name* to provide a user-friendly name for your software, *env_prefix* to expose different environment variables for the detected software in the automatically generated virtual environment script (*env.sh* or *env.bat*), and *soft_file* keys to tell CK which unique filename inside this soft to search for when detecting this software automatically on your system.

If the *soft_file* is the same across all platforms (Linux, Windows, MacOS, etc), you can use the following universal key:

```
    "soft_file_universal": "libGL$#file_ext_dll#$",
```

CK will then substitute *file_ext_dll* with *dll* key from the *file_extensions* dictionary in the target OS (see example for the 64-bit Linux and 64-bit Windows).

You can also tell CK to detect a given soft for a different target such as Android as follows:

```
ck detect soft:compiler.gcc.android.ndk --target_os=android21-arm64
ck detect soft --tags=compiler,android,ndk,llvm --target_os=android21-arm64
```

Next, you may want to update the customize.py file in the new entry. This Python script can have multiple functions to customize the detection of a given software and update different environment variables in the automatically generated "env.sh" or "env.bat" for the virtual CK environment.

For example, *setup* function receives a full path to a found software file specified using the above *soft_name* keys:

```
    cus=i.get('customize',{})
    fp=cus.get('full_path','')
```

It is then used to prepare different environment variables with different paths (see *env* dictionary) as well as embedding commands directly to "env.sh" or "env.bat" using "s" string in the returned dictionary:

```
    return {'return':0, 'bat':s}
```

Here is an example of the automatically generated "env.sh" on a user machine:

```
#! /bin/bash
# CK generated script

if [ "$1" != "1" ]; then if [ "$CK_ENV_LIB_ARMCL_SET" == "1" ]; then return; fi; fi

# Soft UOA           = lib.armcl (fc544df6941a5491)  (lib,arm,armcl,arm-compute-
↪library,compiled-by-gcc,compiled-by-gcc-8.1.0,vopencl,vdefault,v18.05,v18,
↪channel-stable,host-os-linux-64,tar
get-os-linux-64,64bits,v18.5,v18.5.0)
# Host OS UOA        = linux-64 (4258b5fe54828a50)
# Target OS UOA      = linux-64 (4258b5fe54828a50)
# Target OS bits     = 64
# Tool version       = 18.05-b3a371b
# Tool split version = [18, 5, 0]

# Dependencies:
. /home/fursin/CK/local/env/fd0d1d044f44c09b/env.sh
. /home/fursin/CK/local/env/72fa25bd445a993f/env.sh

export CK_ENV_LIB_ARMCL_LIB=/home/fursin/CK-TOOLS/lib-armcl-opencl-18.05-gcc-8.1.0-
↪linux-64/install/lib
export CK_ENV_LIB_ARMCL_INCLUDE=/home/fursin/CK-TOOLS/lib-armcl-opencl-18.05-gcc-8.
↪1.0-linux-64/install/include


export LD_LIBRARY_PATH="/home/fursin/CK-TOOLS/lib-armcl-opencl-18.05-gcc-8.1.0-
↪linux-64/install/lib":$LD_LIBRARY_PATH
export LIBRARY_PATH="/home/fursin/CK-TOOLS/lib-armcl-opencl-18.05-gcc-8.1.0-linux-
↪64/install/lib":$LIBRARY_PATH

export CK_ENV_LIB_ARMCL=/home/fursin/CK-TOOLS/lib-armcl-opencl-18.05-gcc-8.1.0-
↪linux-64/install
export CK_ENV_LIB_ARMCL_CL_KERNELS=/home/fursin/CK-TOOLS/lib-armcl-opencl-18.05-
↪gcc-8.1.0-linux-64/src/src/core/CL/cl_kernels/
export CK_ENV_LIB_ARMCL_DYNAMIC_CORE_NAME=libarm_compute_core.so
export CK_ENV_LIB_ARMCL_DYNAMIC_NAME=libarm_compute.so
export CK_ENV_LIB_ARMCL_LFLAG=-larm_compute
export CK_ENV_LIB_ARMCL_LFLAG_CORE=-larm_compute_core
export CK_ENV_LIB_ARMCL_SRC=/home/fursin/CK-TOOLS/lib-armcl-opencl-18.05-gcc-8.1.0-
↪linux-64/src
export CK_ENV_LIB_ARMCL_SRC_INCLUDE=/home/fursin/CK-TOOLS/lib-armcl-opencl-18.05-
↪gcc-8.1.0-linux-64/src/include
export CK_ENV_LIB_ARMCL_STATIC_CORE_NAME=libarm_compute_core.a
export CK_ENV_LIB_ARMCL_STATIC_NAME=libarm_compute.a
export CK_ENV_LIB_ARMCL_TESTS=/home/fursin/CK-TOOLS/lib-armcl-opencl-18.05-gcc-8.1.
↪0-linux-64/src/tests
export CK_ENV_LIB_ARMCL_UTILS=/home/fursin/CK-TOOLS/lib-armcl-opencl-18.05-gcc-8.1.
↪0-linux-64/src/utils

export CK_ENV_LIB_ARMCL_SET=1
```

All these environment variables will be exposed to the CK program compilation and execution workflow if this software dependency is seleted in a program meta description.

You can also look at how this functionality is implemented in the CK soft module.

There are many options and nuances so we suggest you to have a look at existing examples or contact the CK community for further details. We regularly explain users how to add new software detection plugins and packages.

6.7 Add new CK packages

Whenever a required software is not found, CK will automatically search for existing packages with the same tags for a given target in all installed CK repositories.

CK package module provides a unified JSON API to automatically download, install, and potentially rebuild a given package (software, datasets, models, etc) in a portable way across Linux, Windows, MacOS, Android, and other supported platforms. It is also a unified front-end for other package managers and build tools including make, cmake, scons, Spack, EasyBuild, etc.

If CK packages are not found, CK will print notes from the *install.txt* file from a related software detection plugin about how to download and install such package manually as shown in this example for CUDA.

In such case, you may be interested to provide a new CK package to be reused either in your workgroup or by the broad community to automate the installation.

Similar to adding CK software detection plugins, you must first find the most close package from this online index, download it, and make a new copy in your repository unless you want to share it immediately with the community in already existing CK repositories.

For example, let's copy a CK protobuf package that downloads a given protobuf version in a tgz archive and uses cmake to build it:

```
ck cp package:lib-protobuf-3.5.1-host my-new-repo:package:my-new-lib
```

Note, that all CK packages must be always connected with some software detection plugins such as "soft:lib.my-new-lib" created in the previous section. You just need to find its Unique ID as follows:

```
ck info soft:lib.my-new-lib
```

and add it to the *soft_uoa* key in the package *meta.json*.

Next, copy/paste *the same* tags from the *meta.json* of the soft plugin to the *meta.json* of the package and add extra tags specifying a version. See examples of such tags in existing packages such as lib-armcl-opencl-18.05 and compiler-llvm-10.0.0-universal.

Alternatively, you can add a new CK package using existing templates while specifying a related software plugin in the command line as follows:

```
ck add my-new-repo:package:my-new-lib --soft=lib.my-new-lib
```

In such case, CK will automatically substitute correct values for *soft_uoa* and *tags* keys!

Next, you need to update the *.cm/meta.json* file in the new CK package entry:

```
ck find package:my-new-lib
```

For example, you need to update other keys in the package meta.json to customize downloading and potentially building (building is not strictly required when you download datasets, models, and other binary packages):

```
   "install_env": {
      "CMAKE_CONFIG": "Release",
      "PACKAGE_AUTOGEN": "NO",
      "PACKAGE_BUILD_TYPE": "cmake",
      "PACKAGE_CONFIGURE_FLAGS": "-Dprotobuf_BUILD_TESTS=OFF",
      "PACKAGE_CONFIGURE_FLAGS_LINUX": "-DCMAKE_INSTALL_LIBDIR=lib",
      "PACKAGE_CONFIGURE_FLAGS_WINDOWS": "-DBUILD_SHARED_LIBS=OFF -Dprotobuf_MSVC_
↪STATIC_RUNTIME=OFF",
      "PACKAGE_FLAGS_LINUX": "-fPIC",
      "PACKAGE_NAME": "v3.5.1.tar.gz",
      "PACKAGE_NAME1": "v3.5.1.tar",
      "PACKAGE_NAME2": "v3.5.1",
      "PACKAGE_RENAME": "YES",
```

(continues on next page)

(continued from previous page)

```
      "PACKAGE_SUB_DIR": "protobuf-3.5.1",
      "PACKAGE_SUB_DIR1": "protobuf-3.5.1/cmake",
      "PACKAGE_UNGZIP": "YES",
      "PACKAGE_UNTAR": "YES",
      "PACKAGE_UNTAR_SKIP_ERROR_WIN": "YES",
      "PACKAGE_URL": "https://github.com/google/protobuf/archive",
      "PACKAGE_WGET": "YES"
    },
    "version": "3.5.1"
```

You can specify extra software dependencies using *deps* dictionary if needed.

You must also describe the file which will be downloaded or created at the end of the package installation process using *end_full_path* key to let CK validate that the process was successful:

```
  "end_full_path": {
    "linux": "install$#sep#$lib$#sep#$libprotobuf.a",
    "win": "install\\lib\\libprotobuf.lib"
```

You can add or update a script to download and build a given package. See examples of such scripts in CK package *imagenet-2012-aux*: *install.sh* and install.bat to download ImageNet 2012 auxiliary dataset used in the ACM ReQuEST-ASPLOS tournament and MLPerf submissions.

Note that CK will pass at least 2 environment variables to this script:

- *PACKAGE_DIR* - the path to the CK package entry. This is useful if your script need additional files or subscripts from the CK package entry.
- *INSTALL_DIR* - the path where this package will be installed. Note that *end_full_path* key will be appended to this path.

If you need to know extra CK variables passed to this script, you can just export all environment variable to some file and check the ones starting from *CK_*.

For example, if your package has software dependencies on a specific Python version, all environment variables from the resolved software dependencies will be available in your installation script. This allows you to use the *${CK_ENV_COMPILER_PYTHON_FILE}* environment variable instead of calling python directly to be able to automatically adapt to different python versions on your machine.

At the end of the package installation, CK will check if this file was created, and will pass it to the related software detection plugin to register the CK virtual environment, thus fully automating the process of rebuilding the required environment for a given workflow!

If you need to create a simple package that downloads an archive, uses *configure* to configure it, and builds it using *make*, use this lib-openmpi-1.10.3-universal CK package as example:

```
"PACKAGE_URL": "https://www.open-mpi.org/software/ompi/v1.10/downloads",
"PACKAGE_NAME": "openmpi-1.10.3.tar.gz",
"PACKAGE_NAME1": "openmpi-1.10.3.tar",
"PACKAGE_NAME2": "openmpi-1.10.3",
"PACKAGE_SUB_DIR": "openmpi-1.10.3",
"PACKAGE_SUB_DIR1": "openmpi-1.10.3",
"linux": "install/lib/libmpi.so"
```

Note that we described only a small part of all available functions of the CK package manager that we have developed in collaboration with our [http://cKnowledge.org/partners.html partners and users]. We continue documenting them and started working on a user-friendly GUI to add new software and packages via web. You can try it here.

6.8 Pack CK repository

You can pack a given repository as follows:

```
ck zip repo:my-new-repo
```

This command will create a *ckr-my-new-repo.zip* file that you can archive or send to your colleagues and artifact evaluators.

Other colleagues can then download it and install it on their system as follows:

```
ck add repo --zip=ckr-my-new-repo.zip
```

They can also unzip entries to an existing repository (local by default) as follows:

```
ck unzip repo --zip=ckr-my-new-repo.zip
```

This enables a simple mechanism to share repositories, automation actions, and components including experimental results and reproducible papers with the community. We also hope it will help to automate the tedious Artifact Evaluation process.

6.9 Prepare CK repository for Digital Libraries

During the ACM ReQuEST-ASPLOS'18 tournament the authors needed to share the snapshots of their implementations of efficient deep learning algorithms for the ACM Digital Library.

We have added a new automation to the CK to prepare such snapshots of a given repository with all dependencies and the latest CK framework in one zip file:

```
ck snapshot artifact --repo=my-new-repo
```

It will create a *ck-artifacts-{date}.zip* archive with all related CK repositories, the CK framework, and two scripts:

- *prepare_virtual_ck.bat*
- *run_virtual_ck.bat*

The first script will unzip all CK repositories and the CK framework inside your current directory.

The second script will set environment variables to point to above CK repositories in such a way that it will not influence you existing CK installation! Basically it creates a virtual CK environment for a given CK snapshot. At the end, this script will run *bash* on Linux/MacOS or *cmd* on Windows allowing you to run CK commands to prepare, run, and validate a given CK workflow while adapting to your platform and environment!

6.10 Prepare a Docker container with CK workflows

One of the CK goals is to be a plug&play connector between non-portable workflows and containers.

CK can work both in native environments and containers. While portable CK workflows can fail in the latest environment, they will work fine inside a container with a stable environment.

We have added the CK module *docker* to make it easier to build, share, and run Docker descriptions. Please follow the Readme in the ck-docker for more details.

Please check examples of the CK Docker entries with CK workflows and components in the following projects:

- https://github.com/ctuning/ck-mlperf/tree/master/docker
- https://github.com/ctuning/ck-request-asplos18-caffe-intel/tree/master/docker
- https://github.com/ctuning/ck-docker/tree/master/docker

You can find many of these containers ready for deployment, usage, and further customization at the cTuning Docker hub.

6.11 Create more complex workflows

Users can create even more complex CK workflows that will automatically compile, run, and validate multiple applications with different compilers, datasets, and models across different platforms while sharing, visualizing, and comparing experimental results via live scoreboards.

See the following examples:

- https://cKnowledge.org/rpi-crowd-tuning
- https://github.com/SamAinsworth/reproduce-cgo2017-paper (see CK workflow module)
- https://github.com/ctuning/ck-scc18/wiki
- https://github.com/ctuning/ck-scc
- https://github.com/ctuning/ck-request-asplos18-results

Users can create such workflows using two methods:

6.11.1 Using shell scripts

We have added CK module *script* that allows you to add a CK entry where you can store different system scripts. Such scripts can call different CK modules to install packages, build and run programs, prepare interactive graphs, generate papers, etc.

You can see examples of such scripts from the reproducible CGO'17 paper. You can also check this unified Artifact Appendix and reproducibility checklist at the end of this article describing how to run those scripts.

You can add your own CK script entry as follows:

```
$ ck add my-new-repo:script:my-scripts-to-run-experiments
$ ck add my-new-repo:script:my-scripts-to-generate-articles
```

You can also write Python scripts calling CK APIs directly. For example, check this ReQuEST-ASPLOS'18 benchmark script to prepare, run, and customize ACM REQUEST experiments:

```
#! /usr/bin/python

import ck.kernel as ck
import os

...

def do(i, arg):

    # Process arguments.
    if (arg.accuracy):
        experiment_type = 'accuracy'
        num_repetitions = 1
    else:
        experiment_type = 'performance'
        num_repetitions = arg.repetitions

    random_name = arg.random_name
    share_platform = arg.share_platform

    # Detect basic platform info.
    ii={'action':'detect',
```

(continues on next page)

(continued from previous page)

```
        'module_uoa':'platform',
        'out':'con'}
    if share_platform: ii['exchange']='yes'
    r=ck.access(ii)
    if r['return']>0: return r

...
```

6.11.2 Using CK modules

You can also add and use any new module "workflow.my-new-experiments" as a workflow with different functions to prepare, run, and validate experiments. This is the preferred method that allows you to use unified CK APIs and reuse this module in other projects:

```
ck add my-new-repo:module:workflow.my-new-experiments
```

Note, that CK module and entry names are global in the CK. Therefore, we suggest you to find a unique name.

You can then add any function to this workflow. For example, let's add a function "run" to run your workflow:

```
ck add_action my-new-repo:module:workflow.my-new-experiments --func=run
```

CK will create a working dummy function in the python code of this CK module that you can test immediately:

```
ck run workflow.my-new-experiments
```

You can then find the *module.py* from this CK module and update *run* function to implement your workflow:

```
ck find module:workflow.my-new-experiments
cd `ck find module:workflow.my-new-experiments`
ls *.py
```

Don't hesitate to get in touch with the [[Contacts|CK community]] if you have questions or comments.

6.12 Generate reproducible and interactive articles

Unified CK APIs and portable CK workflows can help to automate all experiments as well as the generation of papers with all tables and graphs.

As a proof-of-concept, we collaborated with the Raspberry Pi foundation to reproduce results from the MILEPOST project and develop a Collective Knowledge workflow for collaborative research into multi-objective autotuning and machine learning techniques.

We have developed a MILEPOST GCC workflow, shared results in the CK repository, created live CK dashboards to crowdsource autotuning, and automatically generated a live and interactive article where results can be automatically updated by the community. The stable snapshot of such article can still be published as a traditional PDF paper.

However, it is still a complex process. We have started documenting this functionality here and plan to gradually improve it. When we have more resources, we plan to add a web-based GUI to the cKnowledge.io platform to make it easier to create such live, reproducible, and interactive articles.

6.13 Publish CK repositories, workflows, and components

We are developing an open cKnowledge.io platform to let users share and reuse CK repositories, workflows, and components similar to PyPI. Please follow this guide to know more.

6.14 Contact the CK community

We continue improving the CK technology, components, automation actions, workflows, and this documentation! If you have questions, encounter problems or have some feedback, do not hesitate to contact us!

CHAPTER 7

CK CLI and API

Most of the CK functionality is implemented using CK modules with automation actions and associated CK entries (components).

Here we describe the main CK functionality to manage repositories, modules, and actions. Remember that you can see all flags for a given automation action from the command line as follows:

```
ck {action} {CK module} --help
```

You can set the *value* of any *key* for the automation action as follows:

```
ck {action} ... key=value
ck {action} ... -key=value
ck {action} ... --key=value
ck {action} ... --key=value
```

If the value is omitted, CK will use "yes" string.

You can also use JSON or YAML files as inputs to a given action:

```
ck {action} ... @input.json
ck {action} ... @input.yaml
```

7.1 CLI to manage CK repositories

- Automation actions are implemented using the internal CK module *repo*.
- See the list of all automation actions and their API at cKnowledge.io platform.

7.1.1 Init new CK repository in the current path

```
ck init repo
```

CK will ask user for a repo name and will also attempt to detect Git URL from .git/config.

Extra options:

```
ck init repo:{CK repo name}
ck init repo --url={Git URL with the CK repository}
ck init repo --url={Git URL with the CK repository} --deps={list of CK repos}
```

Example:

```
ck init repo:asplos21-artifact123 --url=https://github.com/ctuning/ck-asplos21-
↪artifact123 --deps=ck-autotuning
ck init repo:mlperf-submission --url=https://github.com/ctuning/ck-mlperf-
↪submission321 --deps=ck-mlperf
```

7.1.2 Pull existing repository using Git URL

```
ck pull repo --url={Git URL with the CK repository}
```

7.1.3 Pull existing repository from cTuning GitHub

```
ck pull repo:{CK repo name}
```

In this case, CK will use *https://github.com/ctuning/{CK repo name}*

7.1.4 Download a repository as a zip file

```
ck add repo --zip={URL to the zip file with the CK repository}
```

7.1.5 Update all local CK repositories from Git

```
ck pull all
```

7.1.6 Create a dummy CK repository locally

Quick mode with minimal questions:

```
ck add repo:{user-friendly name} --quiet
```

Advanced mode with many questions to configure repository:

```
ck add repo:{user-friendly name}
```

7.1.7 Import existing local repository from current directory

```
ck import repo --quiet
```

7.1.8 Import existing local repository from some local directory

```
ck import repo --path={full path to the local CK repository} --quiet
```

7.1.9 List local CK repositories

```
ck ls repo
```

or

```
ck list repo
```

7.1.10 Delete a given CK repository

Unregister CK repository but do not delete the content (you can later import it again to reuse automation actions and components):

```
ck rm repo:{CK repo name from above list}
```

or

```
ck remove repo:{CK repo name from above list}
```

or

```
ck delete repo:{CK repo name from above list}
```

Delete CK repository completely with the content:

```
ck rm repo:{CK repo name from above list} --all
```

7.1.11 Find a path to a given CK repository

```
ck where repo:{CK repo name}
```

or

```
ck find repo:{CK repo name}
```

7.1.12 Pack a given CK repository to a zip file

```
ck zip repo:{CK repo name}
```

7.1.13 Add CK entries from a zip file to an existing CK repository

To a local repository:

```
ck unzip repo:{CK repo name} --zip={path to a zip file with the CK repo}
```

To a given repository:

```
ck unzip repo:{CK repo name} --zip={path to a zip file with the CK repo}
```

7.2 CLI to manage CK entries

CK repository is basically a database of CK modules and entries. You can see internal CK commands to manage CK entries as follows:

```
ck help
```

CID is the Collective Identifier of the following formats:

- {CK module name or UID}:{CK entry name or UID}
- {CK repo name or UID}:{CK module name or UID}:{CK entry name or UID}

Note that wildcards are allowed in CID when appropriate!

Here are the most commonly used commands to manage CK modules and entries.

7.2.1 List CK modules from all local CK repositories

```
ck ls module
```

or

```
ck list module
```

Show full CID (repository:module:entry):

```
ck ls module --all
```

7.2.2 List some CK modules with a wildcard from all local CK repositories

```
ck ls module:{wildcard}
```

Example:

```
ck ls module:re* --all

 default:module:repo
 ck-analytics:module:report
 ...
```

7.2.3 List CK entries for a given CK module in a given repository

```
ck ls {CK repo}:{CK module}:
```

or

```
ck ls {CK repo}:{CK module}:*

With a wildcard:
```bash
ck ls {CK repo}:{CK module}:{wildcard for CK entries}
```

Example:
```

```
ck ls ctuning-datasets-min:dataset:*jpeg*dnn*

  ctuning-datasets-min:dataset:image-jpeg-dnn-cat
  ctuning-datasets-min:dataset:image-jpeg-dnn-cat-gray
  ctuning-datasets-min:dataset:image-jpeg-dnn-computer-mouse
  ctuning-datasets-min:dataset:image-jpeg-dnn-cropped-panda
  ctuning-datasets-min:dataset:image-jpeg-dnn-fish-bike
  ctuning-datasets-min:dataset:image-jpeg-dnn-snake-224
  ctuning-datasets-min:dataset:image-jpeg-dnn-surfers
```

7.2.4 Search for CK entries by tags

```
ck search {CK module} --tags={list of tags separated by comma}
```

or

```
ck search {CK module}:{wildcard for CK entries} --tags={list of tags separated by␣
↪comma}
```

or

```
ck search {CK repo}:{CK module}:{wildcard for CK entries} --tags={list of tags␣
↪separated by comma}
```

Example:

```
ck search dataset --tags=jpeg
ck search dataset:*dnn* --tags=jpeg
```

7.2.5 Search for CK entries by a string

You can search CK entries by the occurrence of a given string in values of keys in JSON meta descriptions

```
ck search {CK module} --search_string={string with wildcards}
```

Note that CK supports transparent indexing of all CK JSON meta descriptions by ElasticSearch to enable fast search and powerful queries. This mode is used in our cKnowledge.io platform. Please check these pages to know how to configure your CK installation with ES:

- https://github.com/ctuning/ck/wiki/Customization
- https://github.com/ctuning/ck/wiki/Indexing-entries
- https://github.com/ctuning/ck/wiki/Searching-entries

7.2.6 Find a path to a given CK entry

```
ck find {CK module}:{CK entry}
```

or

```
ck find {CK repo}:{CK module}:{CK entry}
```

Example:

```
ck find module:repo
ck find dataset:image-jpeg-dnn-snake-224
ck find ctuning-datasets-min:dataset:image-jpeg-dnn-snake-224
```

7.2.7 Show JSON meta description of a given entry

```
ck load {CK module}:{CK entry} --min
```

or

```
ck load {CK repo}:{CK module}:{CK entry} --min
```

7.2.8 Delete a given CK entry

```
ck rm {CK module}:{CK entry (can be with wildcard)}
```

or

```
ck rm {CK repo}:{CK module}:{CK entry can be with wildcard}
```

or

```
ck remove {CK repo}:{CK module}:{CK entry can be with wildcard}
```

or

```
ck delete {CK repo}:{CK module}:{CK entry can be with wildcard}
```

Example:

```
ck rm ctuning-datasets-min:dataset:image-jpeg-dnn-snake-224
ck rm dataset:*dnn*
```

7.2.9 Create an empty CK entry

Create a CK entry in a *local* repository (CK scratch-pad):

```
ck add {CK module}:{CK entry name}
```

Create CK entry in a given repository:

```
ck add {CK repo}:{CK module}:{CK entry name}

If CK entry name is omitted, CK will create an entry with a UID:
```bash
ck add {CK module}
```
```

Example:

```
ck add tmp

  Entry  (2eab7af343d399d1, /home/fursin/CK-REPOS/local/tmp/2eab7af343d399d1)␣
↪added successfully!

ck add tmp:xyz

  Entry xyz (44812ba5445a0a52, /home/fursin/CK-REPOS/local/tmp/xyz) added␣
↪successfully!
```

Note that CK always generate Unique IDs for all entries!

7.2.10 Rename a given CK entry

```
ck ren {CK module}:{CK entry} :{new CK entry name}
```

or

```
ck ren {CK repo}:{CK module}:{CK entry} :{new CK entry name}
```

or

```
ck rename {CK repo}:{CK module}:{CK entry} :{new CK entry name}
```

Note that CK keeps the same global UID for a renamed entry to be able to always find it!

Example:

```
ck ren ctuning-datasets-min:dataset:image-jpeg-dnn-snake-224 :image-jpeg-dnn-snake
```

7.2.11 Move a given CK entry to another CK repository

```
ck mv {CK repo}:{CK module}:{CK entry name} {CK new repo}::
```

or

```
ck move {CK repo}:{CK module}:{CK entry name} {CK new repo}::
```

Example:

```
ck mv ctuning-datasets-min:dataset:image-jpeg-dnn-computer-mouse local::
```

7.2.12 Copy a given CK entry

With a new name within the same repository:

```
ck cp {CK repo}:{CK module}:{CK entry name} ::{CK new entry name}
```

With a new name in a new repository:

```
ck cp {CK repo}:{CK module}:{CK entry name} {CK new repo}::{CK new entry name}
```

Example:

```
ck cp ctuning-datasets-min:dataset:image-jpeg-dnn-computer-mouse local::new-image
```

7.3 CLI to manage CK actions

All the functionality in CK is implemented as automation actions in CK modules.

All CK modules inherit default automation actions from the previous section to manage associated CK entries.

A new action can be added to a given CK module as follows:

```
ck add_action {module name} --func={action name}
```

CK will ask you a few questions and will create a dummy function in the given CK module. You can immediately test it as follows:

```
ck {action name} {module name}
```

It will just print the input as JSON to let you play with the command line and help you understand how CK converts the command line parameters into the dictionary input for this function.

Next, you can find this module and start modifying this function:

```
ck find module:{module name}
```

For example, you can add the following Python code inside this function to load some meta description of the entry ''my data" when you call the following action:

```
ck {action name} {module name}:{my data}
```

```
    def {action name}(i):

    action=i['action']      # CK will substitute 'action' with {action name}
    module=i['module_uoa'] # CK will substitute 'module_uoa' with {module name}
    data=i['data_uoa']      # CK will substitute 'data_uoa' with {my data}

    # Call CK API to load meta description of a given entry
    # Equivalent to the command line: "ck load {module name}:{data name}"
    r=ck.access({'action':'load',
                 'module_uoa':work['self_module_uid'], # Load the UID of a given␣
↪module
                 'data_uoa':data})
    if r['return']>0: return r # Universal error handler in the CK

    meta=r['dict']       # meta of a given entry
    info=r['info']       # provenance info of this entry
    path=r['path']       # local path to this entry
    uoa=r['data_uoa']    # Name of the CK entry if exists. Otherwise UID.
    uid=r['data_uid']    # Only UID of the entry
```

Note that *uoa* means that this variable accepts Unique ID or Alias (user-friendly name).

Here *ck* is a CK kernel with various productivity functions and one unified *access* function to all CK modules and actions with unified dictionary (JSON) I/O.

You can find JSON API for all internal CK actions from the previous section that manage CK entries from the command line as follows:

```
ck get_api --func={internal action}
```

For example, you can check the API of the "load" action as follows:

```
ck get_api --func=load
```

For non-internal actions, you can check their API as follows:

```
ck {action name} {module name} --help
```

You can also check them at the cKnowledge.io platform.

When executing the following command

```
ck my_action my_module --param1=value1 --param2 -param3=value3 param4 @input.json␣
↪...
```

CK will convert the above command line parameters to the following Python dictionary ''i" for a given action:

```
i={
    "action":"my_action",
    "module_uoa":"my_module",
    "param1":"value1",
    "param2":"yes",
    "param3":"value3"

    #extra keys merged from the input.json

    ...
}
```

Note that when adding a new action to a given module, CK will also create a description of this action inside the *meta.json* of this module. You can see an example of such descriptions for the internal CK module "repo" here. When CK calls an action, it is not invoked directly from the given Python module but CK first checks the description, tests inputs, and then passes the control to the given Python module.

Also note that we suggest not to use aliases (user-friendly names) inside CK modules but CK UIDs. The reason is that CK names may change while CK UIDs stay persistent. We specify dependencies on other CK modules in the *meta.json* of a given module using the *module_deps* key. See an example in the CK module *program*:

- program meta.json
- how it is used in the CK module program

Such approach also allows us to visualize the growing knowledge graph: interactive graph, video.

Finally, a given CK module has an access to the 3 dictionaries:

- *cfg* - this dictionary is loaded from the *meta.json* file from the CK module
- *work* - this dictionary has some run-time information:
  - self_module_uid: UID of the module
  - self_module_uoa: Alias (user-friendly name of the module) or UID
  - self_module_alias: Alias (user-friendly name of the module) or empty
  - path: path to the CK module
- *ck.cfg* - CK global cfg dictionary that is updated at run-time with the meta description of the "kernel:default" entry. This dictionary is used to customize the local CK installation.

7.4 CK Python API

One of the goals of the CK framework was to make it very simple for any user to access any automation action. That is why we have developed just one unified Python "access" function that allows one to access all automation actions with a simple I/O (dictionary as input and dictionary as output).

You can call this function from any Python script or from CK modules as follows:

```
import ck.kernel as ck

i={'action': # specify action
   'module_uoa': # specify CK module UID or alias

   check keys from a given automation action for a given CK module (ck action␣
↪module --help)
  }

r=ck.access(i)

if r['return']>0: return r # if used inside CK modules to propagate to all CK␣
↪callers
```

(continues on next page)

(continued from previous page)

```
#if r['return']>0: ck.err(r) # if used inside Python scripts to print an error and␣
↪exit

#r dictionary will contain keys from the given automation action.
# See API of this automation action (ck action module --help)
```

Such approach allows users to continue extending different automation actions by adding new keys while keeping backward compatibility. That's how we managed to develop 50+ modules with the community without breaking portable CK workflows for our ML&systems R&D.

At the same time, we have implemented a number of "productivity" functions in the CK kernel that are commonly used by many researchers and engineers. For example, you can load JSON files, list files in directories, copy strings to clipboards. At the same time, we made sure that these functions work in the same way across different Python versions (2.7+ and 3+) and different operating systems thus removing this burden from developers.

You can see the list of such productivity functions here. For example, you can load a json file in your script or CK module in a unified way as follows:

```
import ck.kernel as ck

r=ck.load_json_file({'json_file':'some_file.json'})
if r['return']>0: ck.err(r)

d=r['dict']

d['modify_some_key']='new value'

r=ck.save_json_to_file({'json_file':'new_file.json', 'dict':d, 'sort_keys':'yes'})
if r['return']>0: ck.err(r)
```

7.5 More resources

- CK wiki

CHAPTER 8

CK specs

8.1 CK repository

Here we describe the structure of a CK repository. You may want to look at any CK repository such as ck-env to better understand this structure.

Note that CK creates this structure automatically when you use CK CLI or Python API.

8.1.1 Root files

- *.ckr.json* : JSON meta description of this repository including UIDs and dependencies on other repositories. Most of this information is automatically generated when a new CK repository is created.

```
{
  "data_alias": # repository name (alias) such as "ck-env"
  "data_name": # user-friendly repository name such as "CK environment"
  "data_uid": # CK UID for this repository (automatically generated)
  "data_uoa": # repository alias or repository UID if alias is empty
  "dict": {
    "desc": # user-friendly description of this repository
    "repo_deps": [
      {
        "repo_uoa": # repository name
        "url": # Git URL of this repo
      }
      ...
    ],
    "shared": # =="git" if repository is shared
    "url": # Git URL of this repository
  }
}
```

8.1.2 Root directories (CK modules)

Root of the CK repository can contain any sub-directories to let users gradually convert their ad-hoc projects into the CK format. However, if a directory is related to CK entry, it should have the same name as an associated CK module and two files in the *.cm* directory:

- *CK module name*
- *.cm/alias-a-{CK module name}* : contains UID of the CK module
- *.cm/alias-u-{UID}* : contains CK module name (alias)

These 2 files in .cm help CK to understand that a given directory inside CK repository is associated with some CK entry! They also support fast such for a given CK entry by UIDs or aliases.

However, in the future, we may want to remove such files and perform automatic indexing when CK pulls repositories (similar to Git). See this ticket.

8.1.3 Sub-directories for CK entries

If the directory in the CK repository is a valid CK module name, it can contain CK entries associated with this CK module.

If CK entry does not have a name (an alias), it will be stored as a CK UID (16 lowercase hexadecimal characters):

- *UID* : holder for some artifacts

If CK entry has a name (an alias), there will be two more files in the *.cm* directory:

- *CK entry name* : holder for some artifacts
- *.cm/alias-a-{CK entry name}* : contains UID of the CK entry
- *.cm/alias-u-{UID}* : contains CK entry name (alias)

Once again, these .cm files allow CK to quickly find CK entries by UID and aliases in all CK repositories without the need for any indexing.

8.1.4 CK entry

Each valid CK entry has at least 3 files in the *.cm* directory:

- *.cm/meta.json* : JSON meta description of a given CK entry
- *.cm/info.json* : provenance for a given CK entry (date of creation, author, copyright, license, CK used, etc)
- *.cm/desc.json* : meta description SPECs (under development)

This entry can also contain any other files and directories (for example models, data set files, algorithms, scripts, papers and any other artifacts).

CHAPTER 9

Automating ML&systems R&D

After releasing CK we started working with the community to gradually automate the most common and repetitive tasks for ML&systems R&D (see the FastPath'20 presentation).

We started adding the following CK modules and actions with a unified API and I/O.

9.1 Platform and environment detection

These CK modules automate and unify the detection of different properties of user platforms and environments.

- *module:os* [API] [components]
- *module:platform* [API]
- *module:platform.os* [API]
- *module:platform.cpu* [API]
- *module:platform.gpu* [API]
- *module:platform.gpgpu* [API]
- *module:platform.nn* [API]

Examples:

```
ck detect platform
ck detect platform.gpgpu --cuda
```

9.2 Software detection

This CK module automates the detection of a given software or files (datasets, models, libraries, compilers, frameworks, tools, scripts) on a given platform using CK names, UIDs, and tags:

- *module:soft* [API] [components]

It helps to understand a user platform and environment to prepare portable workflows.

Examples:

```
ck detect soft:compiler.python
ck detect soft --tags=compiler,python
ck detect soft:compiler.llvm
ck detect soft:compiler.llvm --target_os=android23-arm64
```

9.3 Virtual environment

- *module:env* [API]

Whenever a given software or files are found using software detection plugins, CK creates a new "env" component in the local CK repository with an env.sh (Linux/MacOS) or env.bat (Windows).

This environment file contains multiple environment variables with unique names usually starting from *CK_* with automatically detected information about a given soft such as versions and paths to sources, binaries, include files, libraries, etc.

This allows you to detect and use multiple versions of different software that can easily co-exist on your system in parallel.

Examples:

```
ck detect soft:compiler.python
ck detect soft --tags=compiler,python
ck detect soft:compiler.llvm

ck show env
ck show env --tags=compiler
ck show env --tags=compiler,llvm
ck show env --tags=compiler,llvm --target_os=android23-arm64

ck virtual env --tags=compiler,python
```

9.4 Meta packages

When a given software is not detected on our system, we usually want to install related packages with different versions.

That's why we have developed the following CK module that can automate installation of missing packages (models, datasets, tools, frameworks, compilers, etc):

- *module:package* [API] [components]

This is a meta package manager that provides a unified API to automatically download, build, and install packages for a given target (including mobile and edge devices) using existing building tools and package managers.

All above modules can now support portable workflows that can automatically adapt to a given environment based on soft dependencies.

Examples:

```
ck pull repo:ck-mlperf
ck install package --tags=lib,tflite,v2.1.1
ck install package --tags=tensorflowmodel,tflite,edgetpu
```

See an example of variations to customize a given package: lib-tflite.

9.5 Scripts

We also provided an abstraction for ad-hoc scripts:

- *module:script* [API] [components]

See an example of the CK component with a script used for MLPerf benchmark submissions: GitHub

9.6 Portable program pipeline (workflow)

Next we have implemented a CK module to provide a common API to compile, run, and validate programs while automatically adapting to any platform and environment:

- *module:program* [API] [components]

A user describes dependencies on CK packages in the CK program meta as well as commands to build, pre-process, run, post-process, and validate a given program.

Examples:

```
ck pull repo:ck-crowdtuning

ck compile program:cbench-automotive-susan --speed
ck run program:cbench-automotive-susan --repeat=1 --env.OMP_NUM_THREADS=4
```

9.7 Reproducible experiments

We have developed an abstraction to record and reply experiments using the following CK module:

- *module:experiment* [API] [components]

This module records all resolved dependencies, inputs and outputs when running above CK programs thus allowing to preserve experiments with all the provenance and replay them later on the same or different machine:

```
ck benchmark program:cbench-automotive-susan --record --record_uoa=my_experiment

ck find experiment:my_experiment

ck replay experiment:my_experiment

ck zip experiment:my_experiment
```

9.8 Dashboards

Since we can record all experiments in a unified way, we can also visualize them in a unified way. That's why we have developed a simple web server that can help to create customizable dashboards:

- *module:web* [API]

See examples of such dashboards:

- view online at cKnowledge.io platform
- view locally (with or without Docker)

9.9 Interactive articles

One of our goals for CK was to automate the (re-)generation of reproducible articles. We have validated this possibility in this proof-of-concept project with the Raspberry Pi foundation.

We plan to develop a GUI to make the process of generating such papers more user friendly!

9.10 Jupyter notebooks

It is possible to use CK from Jupyter and Colab notebooks. We provided an abstraction to share Jupyter notebooks in CK repositories:

- *module:jnotebook* [API] [components]

You can see an example of a Jupyter notebook with CK commands to process MLPerf benchmark results here.

9.11 Docker

We provided an abstraction to build, pull, and run Docker images:

- *module:docker* [API] [components]

You can see examples of Docker images with unified CK commands to automate MLPerf benchmark here.

CHAPTER 10

Further info

During the past few years we converted all the workflows and components from our past ML&systems R&D including the MILEPOST and cTuning.org project to the CK format.

There are now 150+ CK modules with actions automating and abstracting many tedious and repetitive tasks in ML&systems R&D including model training and prediction, universal autotuning, ML/SW/HW co-design, model testing and deployment, paper generation and so on:

- A high level overview of portable CK workflows
- A Collective Knowledge workflow for collaborative research into multi-objective autotuning and machine learning techniques (collaboration with the Raspberry Pi foundation)
- A summary of main CK-based projects with academic and industrial partners
- cKnowledge.io platform documentation

Don't hesitate to contact us if you have a feedback or want to know more about our plans!

CHAPTER 11

Notes

Users extend the CK functionality via external GitHub reposities in the CK format. See docs for more details.

If you want to extend the CK core, please note that we plan to completely rewrite it based on the OO principles (we wrote the first prototype without OO to be able to port to bare-metal devices in C but we decided not to do it at the end). We also plan to relicense the framework to Apache 2.0. In the meantime, please check this documentation.

CHAPTER 12

How to contribute

Current Collective Knowledge framework (CK) uses standard Apache 2.0 license. Contributions are very welcome to improve the existing functionality of the CK framework, CK modules, and CK components.

You can easily contribute to CK and all related repositories by forking them on GitHub and then submitting a pull request. We strongly suggest you to discuss your PR with the community using the public CK google group.

Since most CK modules and components are reused in different projects, please make sure that you thoroughly tested your contributions and they are backward compatible!

When sending PRs, please briefly explain why they are needed, how their work, and how you tested backward compatibility to make sure that dependent projects continue working correctly.

Please submit bug reports, feedback, and ideas as GitHub issues.

Note that we plan to rewrite the CK kernel and make it more pythonic when we have more resources! Feel free to get in touch if you would like to know more about our future R&D plans.

Thank you very much for supporting this community project!

CHAPTER 13

Auto-generated CK Python API

13.1 Submodules

13.2 ck.kernel module

`ck.kernel.access(i)`

Universal access to all CK actions with unified I/O as dictionaries Target audience: end users

NOTE: If input is a string and it will be converted to the dictionary as follows (the same as CK command line):

key1=value1 -> converted to {key1:value1}

-key10 -> converted to {key10:"yes"}

-key11=value11 -> converted to {key11:value11}

—key12 -> converted to {key12:"yes"}

–key13=value13 -> converted to {key13:value13}

@file_json -> JSON from this file will be merged with INPUT

@@ -> CK will ask user ot enter manually JSON from console and merge with INPUT

@@key -> Enter JSON manually from console and merge with INPUT under this key

@@@cmd_json -> convert string to JSON (special format) and merge with INPUT

—xyz -> add everything after – to "unparsed_cmd" key in INPUT

When string is converted to INPUT dictionary, "cmd" variable is set to True

Parameters **Unified input as dictionary or string** (*converted to dict*) – action (str): automation action

module_uoa (str): CK module UOA for the automation action or

(cid1) (str): if doesn't have = and doesn't start from – or - or @ -> appended to cids[] (cid2) (str): if doesn't have = and doesn't start from – or - or @ -> appended to cids[] (cid3) (str): if doesn't have = and doesn't start from – or - or @ -> appended to cids[]

(repo_uoa) (str): CK repo UOA if action is applied to some CK entry (data_uoa) (str): CK entry name(s)

(out) (str): output for a given action

- if '', none
- if 'con', console interaction (if from CMD, default)
- if 'json', print return dict as json to console
- if 'json_with_sep', separation line and return dict as json to console
- if 'json_file', save return dict to JSON file

(out_file) (str): Name of the file to save return dict if 'out'=='json_file'

(con_encoding) (str): force encoding for I/O (ck_profile) (str): if 'yes', profile CK

Keys for a given CK automation action

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

Output from the given CK automation action

`ck.kernel.`**`access_index_server`**(*i*)

Access index server (usually ElasticSearch) Target audience: CK kernel and low-level developers

Parameters

- **request** (*str*) – request type ('PUT' | 'DELETE' | 'TEST' | 'GET')
- **(path)** (*str*) – ES "path" with indexing info
- **(dict)** (*dict*) – send this query as dict
- **(limit_size)** (*int*) – limit queries using this number (if 'GET')
- **(start_from)** (*int*) – start from a given entry in a query

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

dict (dict): dictionary from ElasticSearch with all entries

`ck.kernel.`**`add`**(*i*)

CK action: create CK entry with a given meta-description in a CK repository Target audience: should use via ck.kernel.access

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA

- **module_uoa** (*str*) – CK module UOA
- **data_uoa** (*str*) – CK entry (data) UOA
- **(data_uid)** (*str*) – CK entry (data) UID (if UOA is an alias)
- **(data_name)** (*str*) – User-friendly name of this entry
- **(dict)** (*dict*) – meta description for this entry (will be recorded to meta.json)
- **(update)** (*str*) – if == 'yes' and CK entry exists, update it
- **(substitute)** (*str*) – if 'yes' and update=='yes' substitute dictionaries, otherwise merge!
- **(dict_from_cid)** (*str*) – if !="", merge dict to meta description from this CID (analog of copy)
- **(dict_from_repo_uoa)** (*str*) – merge dict from this CK repo UOA
- **(dict_from_module_uoa)** (*str*) – merge dict from this CK module UOA
- **(dict_from_data_uoa)** (*str*) – merge dict from this CK entry UOA
- **(desc)** (*dict*) – under development - defining SPECs for meta description in the CK flat format
- **(extra_json_files)** (*dict*) – dict with extra json files to save to this CK entry (keys in this dictionary are filenames)
- **(tags)** (*str*) – list or comma separated list of tags to add to entry
- **(info)** (*dict*) – entry info to record - normally, should not use it!
- **(extra_info)** (*dict*) –

  enforce extra info such as

  - author
  - author_email
  - author_webpage
  - license
  - copyright

  If not specified then take it from the CK kernel (prefix '**default_**')
- **(updates)** (*dict*) – entry updates info to record - normally, should not use it!
- **(ignore_update)** (*str*) – if 'yes', do not add info about update
- **(ask)** (*str*) – if 'yes', ask questions, otherwise silent
- **(unlock_uid)** (*str*) – unlock UID if was previously locked
- **(sort_keys)** (*str*) – by default, 'yes'
- **(share)** (*str*) – if 'yes', try to add via GIT
- **(skip_indexing)** (*str*) – if 'yes', skip indexing even if it is globally on
- **(allow_multiple_aliases)** (*str*) – if 'yes', allow multiple aliases for the same UID (needed for cKnowledge.io to publish renamed components with the same UID)

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

Output from the 'create_entry' function

`ck.kernel.add_action(i)`

Add a new action to the given CK module Target audience: should use via ck.kernel.access

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA
- **module_uoa** (*str*) – must be "module"
- **data_uoa** (*str*) – UOA of the module for the new action
- **func** (*str*) – action name
- **(desc)** (*str*) – action description
- **(for_web)** (*str*) – if 'yes', make it compatible with the CK web API, i.e. allow an access to this function in the CK server
- **(skip_appending_dummy_code)** (*str*) – if 'yes', do not append code

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

Output from the 'update' function for the given CK module

`ck.kernel.add_index(i)`

Index CK entries using ElasticSearch or similar tools Target audience: CK kernel and low-level developers

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA with wild cards
- **(module_uoa)** (*str*) – CK module UOA with wild cards
- **(data_uoa)** (*str*) – CK entry (data) UOA with wild cards
- **(print_full)** (*str*) – if 'yes', show CID (repo_uoa:module_uoa:data_uoa)
- **(print_time)** (*str*) – if 'yes'. print elapse time at the end
- **(time_out)** (*float*) – time out in sec. (default -1, i.e. no timeout)

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.kernel.browser(i)`

Open web browser with the API if exists Target audience: CK kernel and low-level developers

Parameters

- **(template)** (*str*) – use this web template (CK wfe module)
- **(repo_uoa)** (*str*) – CK repo UOA
- **(module_uoa)** (*str*) – CK module UOA
- **(data_uoa)** (*str*) – CK entry (data) UOA
- **(extra_url)** (*str*) – Extra URL

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.kernel.cd(i)`

CK action: print 'cd {path to CID}' Target audience: end users

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA
- **module_uoa** (*str*) – CK module UOA
- **data_uoa** (*str*) – CK entry (data) UOA or
- **cid** (*str*) – CK CID

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

Output from the 'load' function

string (str): prepared string 'cd {path to entry}'

`ck.kernel.cdc(i)`

CK action: print 'cd {path to CID}' and copy to clipboard Target audience: end users

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA
- **module_uoa** (*str*) – CK module UOA
- **data_uoa** (*str*) – CK entry (data) UOA or
- **cid** (*str*) – CK CID

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

Output from the 'load' function

`ck.kernel.check_lock(i)`

Check if a given path is locked. Unlock if requested. Target audience: CK kernel and low-level developers

Parameters

- **path** (*str*) – path to be checked/unlocked
- **(unlock_uid)** (*str*) – UID of the lock to release it

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.kernel.check_version(i)`

Compare a given version with the CK version Target audience: CK kernel and low-level developers

Parameters **version** (*str*) – your version

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

ok (str): if 'yes', your CK kernel version is outdated

current_version (str): your CK kernel version

`ck.kernel.check_writing(i)`

Check is writing to a given repo with a given module is allowed Target audience: CK kernel and low-level developers

Parameters

- **(module_uoa)** (*str*) – module UOA
- **(module_uid)** (*str*) – module UID
- **(repo_uoa)** (*str*) – repo UOA
- **(repo_uid)** (*str*) – repo UID
- **(repo_dict)** (*dict*) – repo meta description with potential read/write permissions
- **(delete)** (*str*) – if 'yes', check if global delete operation is allowed

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

(repo_dict) (dict): repo meta description if available

`ck.kernel.cid(i)`

CK action: get CID from the current path or from the input Target audience: end users

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA
- **(module_uoa)** (*str*) – CK module UOA
- **(data_uoa)** (*str*) – CK entry (data) UOA
- **If above is empty, detect CID in the current path !**

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

data_uoa (str): CK entry (data) UOA

module_uoa (str): CK module UOA

(repo_uoa) (str): CK repo UOA

`ck.kernel.cli()`

`ck.kernel.compare_dicts(i)`

Compare two dictionaries recursively Target audience: end users

Note that if dict1 and dict2 has lists, the results will be as follows:

- dict1={"key":['a','b','c']} dict2={"key":['a','b']} EQUAL
- dict1={"key":['a','b']} dict2={"key":['a','b','c']} NOT EQUAL

Parameters

- **dict1** (*dict*) – dictionary 1
- **dict2** (*dict*) – dictionary 2
- **(ignore_case)** (*str*) – if 'yes', ignore case of letters

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

equal (str); if 'yes' then dictionaries are equal

`ck.kernel.compare_flat_dicts(i)`

Compare two CK flat dictionaries Target audience: end users

Parameters

- **dict1** (*dict*) – dictionary 1
- **dict2** (*dict*) – dictionary 2
- **(ignore_case)** (*str*) – if 'yes', ignore case of letters
- **(space_as_none)** (*str*) – if 'yes', consider "" as None

- **(keys_to_ignore)** (*list*) – list of keys to ignore (can be wildcards)

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

equal (str); if 'yes' then dictionaries are equal

`ck.kernel.`**`convert_ck_list_to_dict`**(*i*)

Convert CK list to CK dict with unicode in UTF-8 (unification of interfaces) Target audience: CK kernel and low-level developers

Parameters **(list)** – list from the 'action' function in this kernel

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

ck_dict (dict):

action (str): CK action

cid (str): CK module UOA or CID (x means that it may not be really CID and has to be processed specially

cids (list): a list of multple CIDs from CMD (commands like copy, move, etc) [cid1, cid2, cid3, . . .]

key1 (str): value1 from –key1=value1 or -key1=value1 or key1=value

key2 (str):

. . .

key10 (str):

. . .

keys (str): keys/values from file specified by "file_json"; if file extension is .tmp, it will be deleted after read!

keys (str): keys/values from cmd_json

unparsed (str): unparsed command line after –

`ck.kernel.`**`convert_cm_to_ck`**(*i*)

List files in a given CK entry Target audience: internal

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA with wild cards
- **(module_uoa)** (*str*) – CK module UOA with wild cards
- **(data_uoa)** (*str*) – CK entry (data) UOA with wild cards
- **(print_full)** (*str*) – if 'yes', show CID (repo_uoa:module_uoa:data_uoa)
- **(print_time)** (*str*) – if 'yes'. print elapse time at the end

- **(ignore_update)** (*str*) – if 'yes', do not add info about update
- **(time_out)** (*float*) – time out in sec. (default -1, i.e. no timeout)

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.kernel.`**`convert_entry_to_cid`**(*i*)

Convert info about CK entry to CID Target audience: CK kernel and low-level developers

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA
- **(repo_uid)** (*str*) – CK repo UID
- **(module_uoa)** (*str*) – CK module UOA
- **(module_uid)** (*str*) – CK module UID
- **(data_uoa)** (*str*) – CK entry (data) UOA
- **(data_uid)** (*str*) – CK entry (data) UID

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

cuoa (str); module_uoa:data_uoa (substituted with ? if can't find)

cid (str): module_uid:data_uid (substituted with ? if can't find)

xcuoa (str): repo_uoa:module_uoa:data_uoa (substituted with ? if can't find)

xcid (str): repo_uid:module_uid:data_uid (substituted with ? if can't find)

`ck.kernel.`**`convert_file_to_upload_string`**(*i*)

Convert file to a string for web-based upload Target audience: end users

Parameters **filename** (*str*) – file name to convert

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

file_content_base64 (str): string that can be transmitted through Internet

`ck.kernel.`**`convert_iso_time`**(*i*)

Convert iso text time to a datetime object Target audience: end users

Parameters **iso_datetime** (*str*) – date time as string in ISO standard

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

datetime_obj (obj): datetime object

`ck.kernel.`**`convert_json_str_to_dict`**(*i*)

Convert string in a special format to dict (JSON) Target audience: end users

Parameters **str** (*str*) – string (use ' instead of ", i.e. {'a':'b'} to avoid issues in CMD in Windows and Linux!)

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

dict (dict): dict from json file

`ck.kernel.`**`convert_str_key_to_int`**(*key*)

Support function for safe convert str to int Target audience: end users

Parameters **key** (*str*) – variable to be converted to int

Returns *(int)* – int(key) if key can be converted to int, or 0 otherwise

`ck.kernel.`**`convert_str_tags_to_list`**(*i*)

Split string by comma into a list of stripped strings Target audience: end users

Used to process and strip tags

Parameters **i** (*list or string*) – list or string to be splitted and stripped

Returns *(list)* – list of stripped strings

`ck.kernel.`**`convert_upload_string_to_file`**(*i*)

Convert upload string to file Target audience: end users

Parameters

- **file_content_base64** (*str*) – string transmitted through Internet
- **(filename)** (*str*) – file name to write (if empty, generate tmp file)

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

filename (str): filename with full path

filename_ext (str): filename extension

`ck.kernel.copy(i)`

CK action: copy or move CK entry Target audience: should use via ck.kernel.access

Parameters **See "cp" function**

Returns See "cp" function

`ck.kernel.copy_path_to_clipboard(i)`

Copy current path to clipboard (productivity function) Target audience: CK kernel and low-level developers

Parameters **(add_quotes)** (*str*) – if 'yes', add quotes

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.kernel.copy_to_clipboard(i)`

Copy string to clipboard if supported by OS (requires Tk or pyperclip) Target audience: end users

Parameters **string** (*str*) – string to copy

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.kernel.cp(i)`

CK action: copy or move CK entry Target audience: should use via ck.kernel.access

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA
- **module_uoa** (*str*) – CK module UOA
- **data_uoa** (*str*) – CK entry (data) UOA
- **xcids** (*list*) – use original name from xcids[0] and new name from xcids[1] ({'repo_uoa', 'module_uoa', 'data_uoa'}) or
- **(new_repo_uoa)** (*str*) – new CK repo UOA
- **(new_module_uoa)** (*str*) – new CK module UOA
- **new_data_uoa** (*str*) – new CK data alias
- **(new_data_uid)** (*str*) – new CK entry (data) UID (leave empty to generate the new one)
- **(move)** (*str*) – if 'yes', remove the old entry
- **(keep_old_uid)** (*str*) – if 'yes', keep the old UID
- **(without_files)** (*str*) – if 'yes', do not move/copy files

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

Output from the "add" function

`ck.kernel.create_entry(i)`

Create a CK entry with UID or alias in the given path Target audience: CK kernel and low-level developers

Parameters

- **path** (*str*) – path where to create an entry
- **(split_dirs)** (*int*) – number of first characters to split directory into subdirectories to be able to handle many entries (similar to Mediawiki)
- **(data_uoa)** (*str*) – CK entry UOA
- **(data_uid)** (*str*) – if data_uoa is an alias, we can force data UID
- **(force)** (*str*) – if 'yes', force to create CK entry even if related directory already exists
- **(allow_multiple_aliases)** (*str*) – (needed for cKnowledge.io to publish renamed components with the same UID)

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

path (str): path to the created CK entry

data_uid (str): UID of the created CK entry

data_alias (str): alias of the created CK entry

data_uoa (str): alias or UID (if alias=="") of the created CK entry

`ck.kernel.debug_out(i)`

Universal debug print of a dictionary while removing unprintable parts Target audience: end users

Parameters **i** (*dict*) – dictionary to print

Returns

(dict) –

Unified CK dictionary:

return (int): 0

`ck.kernel.delete(i)`

CK action: delete CK entry or CK entries Target audience: should use via ck.kernel.access

Parameters **See "rm" function**

Returns See "rm" function

`ck.kernel.delete_alias(i)`

Delete the CK entry alias from a given path Target audience: CK kernel and low-level developers

Parameters

- **path** (*str*) – path to the CK entry
- **data_uid** (*str*) – CK entry UID
- **(data_alias)** (*str*) – CK entry alias
- **(repo_dict)** (*str*) – meta description of a given CK repository to check if there is an automatic sync with a Git repository
- **(share)** (*str*) – if 'yes', try to delete using the Git client

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.kernel.delete_directory(i)`

Delete a given directory with all sub-directories (must be very careful) Target audience: CK kernel and low-level developers

Parameters **path** (*str*) – path to delete

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.kernel.delete_file(i)`

Delete file from the CK entry Target audience: CK kernel and low-level developers

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA
- **module_uoa** (*str*) – CK module UOA
- **data_uoa** (*str*) – CK entry (data) UOA
- **filename** (*str*) – filename to delete including relative path
- **(force)** (*str*) – if 'yes', force deleting without questions

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.kernel.delete_index(i)`

Delete index for a given CK entry in the ElasticSearch or a similar services Target audience: CK kernel and low-level developers

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA with wild cards
- **(module_uoa)** (*str*) – CK module UOA with wild cards
- **(data_uoa)** (*str*) – CK entry (data) UOA with wild cards
- **(print_time)** (*str*) – if 'yes'. print elapse time at the end
- **(time_out)** (*float*) – in sec. (default -1, i.e. no timeout)

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.kernel.`**`detect_cid_in_current_path`**(*i*)

Detect CID in the current directory Target audience: CK kernel and low-level developers

Parameters **(path)** (*str*) – path, or current directory if path==""

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

repo_uoa (str): CK repo UOA

repo_uid (str): CK repo UID

repo_alias (str): CK repo alias

(module_uoa) (str): CK module UOA

(module_uid) (str): CK module UID

(module_alias) (str): CK module alias

(data_uoa) (str): CK entry (data) UOA

(data_uid) (str): CK entry (data) UID

(data_alias) (str): CK entry (data) alias

`ck.kernel.`**`download`**(*i*)

Download CK entry from remote host (experimental) Target audience: end users

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA
- **(module_uoa)** (*str*) – CK module UOA
- **(data_uoa)** (*str*) – CK data UOA
- **(version)** (*str*) – version (the latest one if skipped)
- **(new_repo_uoa)** (*str*) – target CK repo UOA, "local" by default

- **(skip_module_check)** (*str*) – if 'yes', do not check if module for a given component exists
- **(all)** (*str*) – if 'yes', download dependencies
- **(force)** (*str*) – if 'yes, force download even if components already exists
- **(tags)** (*str*) – download components using tags separated by comma (usually soft/package)

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.kernel.`**`dump_json`**(*i*)

Dump dictionary (json) to a string Target audience: end users

Parameters

- **dict** (*dict*) – dictionary to convert to a string
- **(skip_indent)** (*str*) – if 'yes', skip indent
- **(sort_keys)** (*str*) – if 'yes', sort keys

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

string (str): JSON string

`ck.kernel.`**`dumps_json`**(*i*)

Dump dictionary (json) to a string Target audience: end users

Parameters

- **dict** (*dict*) – dictionary to convert to a string
- **(skip_indent)** (*str*) – if 'yes', skip indent
- **(sort_keys)** (*str*) – if 'yes', sort keys

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

string (str): JSON string

`ck.kernel.`**`edit`**(*i*)

CK action: edit data meta-description through external editor Target audience: should use via ck.kernel.access

Parameters

- **(repo_uoa)** (*str*) – repo UOA
- **module_uoa** (*str*) – module UOA
- **data_uoa** (*str*) – data UOA
- **(ignore_update)** (*str*) – (default==yes) if 'yes', do not add info about update
- **(sort_keys)** (*str*) – (default==yes) if 'yes', sort keys
- **(edit_desc)** (*str*) – if 'yes', edit description rather than meta (useful for compiler descriptions)

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.kernel.`**`eout`**`(s)`

Universal print of a unicode error string in the UTF-8 or other format to stderr Target audience: end users

Supports: Python 2.x and 3.x

Parameters **s** (*str*) – unicode string to print

Returns None

`ck.kernel.`**`err`**`(r)`

Print error to stderr and exit with a given return code Target audience: end users

Used in Bash and Python scripts to exit on error

Example: import ck.kernel as ck

r=ck.access({'action':'load', 'module_uoa':'tmp', 'data_uoa':'some tmp entry'})

if r['return']>0: ck.err(r)

Parameters **r** (*dict*) – output dictionary of any standard CK function:

- return (int): return code
- (error) (str): error string if return>0

Returns None - exits script!

`ck.kernel.`**`filter_add_index`**`(i)`

`ck.kernel.`**`filter_convert_cm_to_ck`**`(i)`

`ck.kernel.`**`filter_delete_index`**`(i)`

`ck.kernel.`**`find`**`(i)`

CK action: find CK entry via the 'load' function Target audience: should use via ck.kernel.access

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA
- **module_uoa** (*str*) – CK module UOA
- **data_uoa** (*str*) – CK entry (data) UOA

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

Output from the 'load' function

number_of_entries (int): total number of found entries

`ck.kernel.find2(i)`

`ck.kernel.find_path_to_data(i)`

Find path to CK sub-directory Target audience: CK kernel and low-level developers

First search in the default repo, then in the local repo, and then in all installed repos

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA
- **module_uoa** (*str*) – CK module UOA
- **data_uoa** (*str*) – CK data UOA

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

path (str): path to CK entry (CK data)

path_module (str): path to CK module entry (part of the CK entry)

path_repo (str): path to the CK repository with this entry

repo_uoa (str): CK repo UOA

repo_uid (str): CK repo UID

repo_alias (str): CK repo alias

module_uoa (str): CK module UOA

module_uid (str): CK module UID

module_alias (str): CK module alias

uoa (str): CK sub-directory UOA

uid (str): CK sub-directory UID

alias (str): CK sub-directory alias

`ck.kernel.find_path_to_entry(i)`

Find path to CK entry (CK data) while checking both UID and alias. Target audience: CK kernel and low-level developers

Parameters

- **path** (*str*) – path to a data entry
- **data_uoa** (*str*) – CK entry UOA (CK data)

- **(split_dirs)** (*int/str*) – number of first characters to split directory into subdirectories to be able to handle many entries (similar to Mediawiki)

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

path (str): path to CK entry

data_uid (str): CK entry UID

data_alias (str): CK entry alias

data_uoa (str): CK entry alias of UID, if alias is empty

`ck.kernel.`**`find_path_to_repo`**(*i*)

Find path for a given CK repo Target audience: end users

Parameters **(repo_uoa)** (*str*) – CK repo UOA. If empty, get the path to the default repo (inside CK framework)

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

dict (dict): CK repo meta description from the cache path (str): path to this repo

repo_uoa (str): CK repo UOA

repo_uid (str): CK repo UID

repo_alias (str): CK repo alias

`ck.kernel.`**`find_repo_by_path`**(*i*)

Find CK repo info by path Target audience: CK kernel and low-level developers

Parameters **path** (*str*)

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

repo_uoa (str): CK repo UOA

repo_uid (str): CK repo UID

repo_alias (str): CK repo alias

`ck.kernel.`**`find_string_in_dict_or_list`**(*i*)

Find a string in a dict or list Target audience: end users

Parameters

- **dict** (*dict or list*) – dict or list to search
- **(search_string)** (*str*) – search string
- **(ignore_case)** (*str*) – if 'yes' then ignore case of letters

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

found (str): if 'yes', string found

`ck.kernel.`**`flatten_dict`**(*i*)

Any list item is converted to @number=value Any dict item is converted to #key=value # is always added at the beginning

Input: {

dict - python dictionary

(prefix) - prefix (for recursion)

(prune_keys) - list of keys to prune (can have wildcards)

}

Output: {

return - return code = 0, if successful > 0, if error

(error) - error text if return > 0 dict - flattened dictionary

}

`ck.kernel.`**`flatten_dict_internal`**(*a*, *aa*, *prefix*, *pk*)

Convert dictionary into the CK flat format Target audience: internal use for recursion

Parameters

- **a** (*any*)
- **aa** (*dict*) – target dict
- **prefix** (*str*) – key prefix
- **pk** – aggregated key?

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

dict (dict): pass dict a from the input

`ck.kernel.`**`flatten_dict_internal_check_key`**(*prefix*, *pk*)

Convert dictionary into the CK flat format Target audience: internal use

Parameters

- **prefix** (*str*) – key prefix

- **pk** – aggregated key?

Returns *(bool)* – key must be added if True

`ck.kernel.gen_tmp_file(i)`

Generate temporary files Target audience: end users

Parameters

- **(suffix)** (*str*) – temp file suffix
- **(prefix)** (*str*) – temp file prefix
- **(remove_dir)** (*str*) – if 'yes', remove dir

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

file_name (str): temp file name

`ck.kernel.gen_uid(i)`

Generate valid CK UID Target audience: end users

Parameters **None**

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

data_uid (str): UID in string format (16 lowercase characters 0..9,a..f)

`ck.kernel.get_api(i)`

Print API from the CK module for a given action Target audience: CK kernel and low-level developers

Parameters

- **(path)** (*str*) – path to a CK module, if comes from the access function or
- **(module_uoa)** (*str*) – if comes from CMD
- **(func)** – API function name
- **(out)** – how to output this info

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

title (str): title string

desc (str): original description

module (str): CK module name

api (str): api

line (str): description string in the CK module

`ck.kernel.get_by_flat_key(i)`

Get a value from a dict by the CK flat key Target audience: end users

Parameters

- **dict** (*dict*) – dictionary
- **key** (*str*) – CK flat key

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

value (any): value or None, if key doesn't exist

`ck.kernel.get_current_date_time(i)`

Get current date and time Target audience: end users

Parameters **(dict)** – empty dict

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

array (dict); dict with date and time

- date_year (str)
- date_month (str)
- date_day (str)
- time_hour (str)
- time_minute (str)
- time_second (str)

iso_datetime (str): date and time in ISO format

`ck.kernel.get_default_repo(i)`

Print path to the default repo Target audience: CK kernel and low-level developers

Args:

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

path (str): path

`ck.kernel.get_from_dicts`(*dict1*, *key*, *default_value*, *dict2*, *extra=”*)

Get value from one dict, remove it from there and move to another Target audience: end users

Parameters

- **dict1** (*dict*) – first check in this dict (and remove if there)
- **key** (*str*) – key in the dict1
- **default_value** (*str*) – default value if not found
- **dict2** (*dict*) – then check key in this dict

Returns *(any)* – value from the dictionary

`ck.kernel.get_os_ck`(*i*)

Get host platform name (currently win or linux) and OS bits Target audience: end users

Parameters **(bits)** (*int*) – force OS bits

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

platform (str): ‘win’ or ‘linux’

bits (str): OS bits in string (32 or 64)

python_bits (str): Python installation bits (32 or 64)

`ck.kernel.get_split_dir_number`(*repo_dict*, *module_uid*, *module_uoa*)

Support function for checking splitting entry number Target audience: CK kernel and low-level developers

Parameters

- **repo_dict** (*dict*) – dictionary with CK repositories
- **module_uid** (*str*) – requested CK module UID
- **module_uoa** (*str*) – requested CK module UOA

Returns

(int) –

number of sub-directories for CK entries - useful when holding millions of entries

`ck.kernel.get_version`(*i*)

Get CK version Target audience: end users

Args: None

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

version (list): list of sub-versions starting from major version number

version_str (str): version string

`ck.kernel.`**`guide`**(*i*)

Open web browser with the user/developer guide wiki Target audience: CK kernel and low-level developers

Parameters **(dict)** – empty dict

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.kernel.`**`help`**(*i*)

CK action: print help for a given module Target audience: end users

Parameters **(module_uoa)** (*str*) – CK module UOA

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

help (str): string with the help text

`ck.kernel.`**`index_module`**(*module_uoa*, *repo_uoa*)

Support function for checking whether to index data using ElasticSearch or not ... Target audience: CK kernel and low-level developers

Useful to skip some sensitive data from global indexing.

Parameters

- **module_uoa** (*str*) – CK module UID or alias
- **repo_uoa** (*str*) – CK repo UID or alias

Returns *(bool)* – True if needs to index

`ck.kernel.`**`info`**(*i*)

CK action: print CK info about a given CK entry Target audience: end users

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA

- **module_uoa** (*str*) – CK module UOA
- **(data_uoa)** (*str*) – CK entry (data) UOA

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

Keys from the "load" function

`ck.kernel.`**`init`**(*i*)

Inititalize CK (current instance - has a global state!) Target audience: internal use

Parameters **(dict)** – empty dict

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.kernel.`**`inp`**(*i*)

Universal input of unicode string in UTF-8 or other format Target audience: end users

Supports Python 2.x and 3.x

Parameters **text** (*str*) – text to print before the input

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

string (str): entered string

`ck.kernel.`**`input_json`**(*i*)

Input JSON from console (double enter to finish) Target audience: end users

Parameters **text** (*str*) – text to print

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

string (str): entered string

dict (str): dictionary from JSON string

`ck.kernel.`**`is_uid`**(*s*)

Check if a string is a valid CK UID Target audience: end users

Parameters **s** (*str*) – string

Returns *(bool)* – True if a string is a valid CK UID

`ck.kernel.`**`is_uoa`**`(s)`

Check if string is correct CK UOA, i.e. it does not have special characters including *, ? Target audience: end users

Parameters **s** (*str*) – string

Returns *(bool)* – True if a string is a valid CK UID or alias

`ck.kernel.`**`jerr`**`(r)`

Print error message for CK functions in the Jupyter Notebook and raise KeyboardInterrupt Target audience: end users

Used in Jupyter Notebook

Example: import ck.kernel as ck

r=ck.access({'action':'load', 'module_uoa':'tmp', 'data_uoa':'some tmp entry'})

if r['return']>0: ck.jerr(r)

Parameters **r** (*dict*) – output dictionary of any standard CK function:

- return (int): return code
- (error) (str): error string if return>0

Returns None - exits script with KeyboardInterrupt!

`ck.kernel.`**`list_actions`**`(i)`

List actions in the given CK module Target audience: should use via ck.kernel.access

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA
- **module_uoa** (*str*) – must be "module"
- **data_uoa** (*str*) – UOA of the module for the new action

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

actions (dict): dict with actions in the given CK module

`ck.kernel.`**`list_all_files`**`(i)`

List all files recursively in a given directory Target audience: all users

Parameters

- **path** (*str*) – top level path
- **(file_name)** (*str*) – search for a specific file name
- **(pattern)** (*str*) – return only files with this pattern

- **(path_ext)** (*str*) – path extension (needed for recursion)
- **(limit)** (*str*) – limit number of files (if directories with a large number of files)
- **(number)** (*int*) – current number of files
- **(all)** (*str*) – if 'yes' do not ignore special directories (like .cm)
- **(ignore_names)** (*list*) – list of names to ignore
- **(ignore_symb_dirs)** (*str*) – if 'yes', ignore symbolically linked dirs (to avoid recursion such as in LLVM)
- **(add_path)** (*str*)

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

list (dict): dictionary of all files: {"file_with_full_path":{"size":.., "path":..}

sizes (dict): sizes of all files (the same order as above "list")

number (int): (internal) total number of files in a current directory (needed for recursion)

`ck.kernel.`**`list_data`**(*i*)

List CK entries Target audience: CK kernel and low-level developers

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA with wildcards
- **(module_uoa)** (*str*) – CK module UOA with wildcards
- **(data_uoa)** (*str*) – CK entry (data) UOA with wildcards
- **(repo_uoa_list)** (*list*) – list of CK repos to search
- **(module_uoa_list)** (*list*) – list of CK modules to search
- **(data_uoa_list)** (*list*) – list of CK entries to search
- **(filter_func)** (*str*) – name of the filter function to customize search
- **(filter_func_addr)** (*obj*) – Python address of the filter function
- **(add_if_date_before)** (*str*) – add only entries with date before this date
- **(add_if_date_after)** (*str*) – add only entries with date after this date
- **(add_if_date)** (*str*) – add only entries with this date
- **(ignore_update)** (*str*) – if 'yes', do not add info about update (when updating in filter)
- **(search_by_name)** (*str*) – search by name
- **(search_dict)** (*dict*) – search if this dict is a part of the entry
- **(ignore_case)** (*str*) – ignore string case when searching!
- **(print_time)** (*str*) – if 'yes', print elapsed time at the end
- **(do_not_add_to_lst)** (*str*) – if 'yes', do not add entries to lst
- **(time_out)** (*float*) – in secs, default=30 (if -1, no timeout)
- **(limit_size)** (*int*) – if >0, limit the number of returned entries

- **(print_full)** (*str*) – if 'yes', show CID (repo_uoa:module_uoa:data_uoa) or
- **(all)** (*str*) – the same as above
- **(print_uid)** (*str*) – if 'yes', print UID in brackets
- **(print_name)** (*str*) – if 'yes', print name (and add info to the list) or
- **(name)** (*str*) – the same as above
- **(add_info)** (*str*) – if 'yes', add info about entry to the list
- **(add_meta)** (*str*) – if 'yes', add meta about entry to the list

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

lst (list): [{'repo_uoa', 'repo_uid',

'module_uoa', 'module_uid', 'data_uoa','data_uid', 'path' (,meta) (,info) . . .

}]

elapsed_time (float): elapsed time in string

(timed_out) (str): if 'yes', timed out or limited by size

}

`ck.kernel.`**`list_data2`**(*i*)

`ck.kernel.`**`list_files`**(*i*)

List files in a given CK entry Target audience: end users

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA
- **(module_uoa)** (*str*) – CK module UOA
- **(data_uoa)** – CK entry (data) UOA
- **See other keys for the "list_all_files" function**

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

Output from the "list_al_files" function

`ck.kernel.`**`list_tags`**(*i*)

CK action: list tags in found CK entries (uses search function) Target audience: should use via ck.kernel.access

Parameters **The same as in "search" function**

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

tags (list): sorted list of all found tags

The same as from "search" function

}

`ck.kernel.load(i)`

CK action: load meta description from the CK entry Target audience: should use via ck.kernel.access

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA
- **module_uoa** (*str*) – CK module UOA
- **data_uoa** (*str*) – CK entry (data) UOA
- **(get_lock)** (*str*) – if 'yes', lock this entry
- **(lock_retries)** (*int*) – number of retries to aquire lock (default=5)
- **(lock_retry_delay)** (*float*) – delay in seconds before trying to aquire lock again (default=10)
- **(lock_expire_time)** (*float*) – number of seconds before lock expires (default=30)
- **(skip_updates)** (*str*) – if 'yes', do not load updates
- **(skip_desc)** (*str*) – if 'yes', do not load descriptions
- **(load_extra_json_files)** (*str*) – list of files to load from the entry
- **(unlock_uid)** (*str*) – UID of the lock to release it
- **(min)** (*str*) – show minimum when output to console (i.e. meta and desc)
- **(create_if_not_found)** (*str*) – if 'yes', create, if entry is not found - useful to create and lock entries

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

dict (dict): CK entry meta description

(info) (dict): CK entry info

(updates) (dict): CK entry updates

(desc) (dict): CK entry description

path (str): path to the CK entry

path_module (str): path to the CK module entry for this CK entry

path_repo (str): path to the CK repository for this CK entry

repo_uoa (str): CK repo UOA

repo_uid (str): CK repo UID

repo_alias (str): CK repo alias

module_uoa (str): CK module UOA

module_uid (str): CK module UID

module_alias (str): CK module alias

data_uoa (str): CK entry (data) UOA

data_uid (str): CK entry (data) UID

data_alias (str): CK entry (data) alias

data_name (str): CK entry user friendly name

(extra_json_files) (dict): merged dict from JSON files specified by 'load_extra_json_files' key

(lock_uid) (str): unlock UID, if locked successfully

`ck.kernel.`**`load_json_file`**(*i*)

Load json from file into dict Target audience: end users

Parameters **json_file** (*str*) – name of a json file

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

dict (dict or list): dict or list from the json file

`ck.kernel.`**`load_meta_from_path`**(*i*)

Load CK meta description from a path Target audience: CK kernel and low-level developers

Parameters

- **path** (*str*) – path to a data entry
- **(skip_updates)** (*str*) – if 'yes', do not load updates
- **(skip_desc)** (*str*) – if 'yes', do not load descriptions to be able to handle many entries (similar to Mediawiki)

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

dict (dict): dict with CK meta description

path (str): path to json file with meta description

(info) (dict): dict with CK info (provenance) if exists

(path_info) (str): path to json file with info

(updates) (dict): dict with updates if exists

(path_updates) (str): path to json file with updates

(path_desc) (str): path to json file with API description

`ck.kernel.load_module_from_path(i)`

Load (CK) python module Target audience: end users

Parameters

- **path** (*str*) – path to a Python module
- **module_code_name** (*str*) – Python module name
- **(cfg)** (*dict*) – CK module configuration if exists
- **(skip_init)** (*str*) – if 'yes', skip init of the CK module
- **(data_uoa)** (*str*) – CK module UOA (useful when printing errors)

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

code (obj): Python code object

path (str): full path to the module

cuid (str): automatically generated unique ID for the module in the internal cache of modules

`ck.kernel.load_repo_info_from_cache(i)`

Load repo meta description from cache Target audience: CK kernel and low-level developers

Parameters **repo_uoa** (*str*) – CK repo UOA

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

repo_uoa (str): CK repo UOA

repo_uid (str): CK repo UID

repo_alias (str): CK repo alias

all other keys from repo dict

`ck.kernel.load_text_file(i)`

Load a text file to a string or list Target audience: end users

Parameters

- **text_file** (*str*) – name of a text file
- **(keep_as_bin)** (*str*) – if 'yes', return only bin
- **(encoding)** (*str*) – by default 'utf8', however sometimes we use utf16

- **(split_to_list)** (*str*) – if 'yes', split to list
- **(convert_to_dict)** (*str*) – if 'yes', split to list and convert to dict
- **(str_split)** (*str*) – if !=", use as separator of keys/values when converting to dict
- **(remove_quotes)** (*str*) – if 'yes', remove quotes from values when converting to dict
- **(delete_after_read)** (*str*) – if 'yes', delete file after read (useful when reading tmp files)

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

bin (byte): loaded text file as byte array

(string) (str): loaded text as string with removed

(lst) (list): if split_to_list=='yes', split text to list

(dict) (dict): if convert_to_dict=='yes', return as dict

`ck.kernel.`**`load_yaml_file`**(*i*)

Load YAML file to dict Target audience: end users

Parameters **yaml_file** (*str*) – name of a YAML file

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

dict (dict): dict from a YAML file

`ck.kernel.`**`lower_list`**(*lst*)

Support function to convert all strings into lower case in a list Target audience: internal

Parameters **lst** (*list*) – list of strings

Returns *(list)* – list of lowercased strings

`ck.kernel.`**`merge_dicts`**(*i*)

Merge intelligently dict1 with dict2 key by key in contrast with dict1.update(dict2) Target audience: end users

It can merge sub-dictionaries and lists instead of substituting them

Parameters

- **dict1** (*dict*) – merge this dict with dict2 (will be directly modified!)
- **dict2** (*dict*) – dict to be merged

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

dict1 (dict): dict1 passed through the function

`ck.kernel.move(i)`

CK action: move CK entry to another CK repository Target audience: should use via ck.kernel.access

Parameters **See "mv" function**

Returns See "mv" function

`ck.kernel.mv(i)`

CK action: move CK entry to another CK repository Target audience: should use via ck.kernel.access

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA
- **module_uoa** (*str*) – CK module UOA
- **data_uoa** (*str*) – CK entry (data) UOA
- **xcids** (*list*) – use original name from xcids[0] and new name from xcids[1] ({'repo_uoa', 'module_uoa', 'data_uoa'}) or
- **(new_repo_uoa)** (*str*) – new CK repo UOA
- **(new_module_uoa)** (*str*) – new CK module UOA
- **(new_data_uoa)** (*str*) – new CK data alias
- **(new_data_uid)** (*str*) – new CK entry (data) UID (leave empty to generate the new one)

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

Output from the "copy" function

`ck.kernel.out(s)`

Universal print of a unicode string in UTF-8 or other format Target audience: end users

Supports: Python 2.x and 3.x

Parameters **s** (*str*) – unicode string to print

Returns None

`ck.kernel.parse_cid(i)`

Convert CID to a dict and add missing parts in CID from the current path Target audience: CK kernel and low-level developers

Parameters

- **cid** (*str*) – in format (REPO_UOA:)MODULE_UOA:DATA_UOA
- **(cur_cid)** (*str*) – output from the "detect_cid_in_current_path" function

- **(ignore_error)** (*str*) – if 'yes', ignore wrong format

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

data_uoa (str): CK data UOA

module_uoa (str): CK module UOA

(repo_uoa) (str): CK repo UOA

`ck.kernel.`**`path`**(*i*)

CK action: get CID from the current path Target audience: end users

Parameters **(dict)** – empty dict

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

Keys from the "detect_cid_in_current_path" function

`ck.kernel.`**`perform_action`**(*i*)

Perform an automation action via CK kernel or from the kernel Target audience: CK kernel and low-level developers

Parameters

- **()** – all parameters from the "access" function
- **(web)** (*str*) – if 'yes', called from the web
- **(common_func)** (*str*) –

  if 'yes', ignore search for modules and call common func from the CK kernel

  or
- **(kernel)** (*str*) – the same as above
- **(local)** (*str*) – if 'yes', run locally even if remote repo . . .

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

(out) (str): if action changes output, log it

Output from a given action

}

`ck.kernel.`**`perform_remote_action`**(*i*)

Perform remote action via CK web service Target audience: CK kernel and low-level developers

Parameters **See "perform_action" function**

Returns See "perform_action" function

`ck.kernel.`**`prepare_special_info_about_entry`**(*i*)

Prepare provenance for a given CK entry (CK used, author, date, etc) Target audience: end users

Parameters **i** (*dict*) – empty dict

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

dict (dict): dictionary with provenance information

`ck.kernel.`**`print_input`**(*i*)

Print input dictionary to screen for debugging Target audience: CK kernel and low-level developers

Used in console and web applications

Parameters **(dict)** – input

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

html (str): input as JSON string

`ck.kernel.`**`pull`**(*i*)

Pull CK entries from the CK server Target audience: CK kernel and low-level developers

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA
- **module_uoa** (*str*) – must be "module"
- **data_uoa** (*str*) – UOA of the module for the new action
- **(filename)** (*str*) –

  filename (with path) (if empty, set archive to 'yes'). If empty, create an archive of the entry

  or
- **(cid[0])** (*str*)
- **(archive)** (*str*) – if 'yes' pull whole entry as zip archive using filename or ck_archive.zip
- **(all)** (*str*) – if 'yes' and archive, add even special directories (.cm, .svn, .git, etc)
- **(out)** (*str*) – if 'json' or 'json_file', encode file and return in r

- **(skip_writing)** (*str*) – if 'yes', do not write file (not archive) to current directory
- **(pattern)** (*str*) – return only files with this pattern
- **(patterns)** (*str*) – multiple patterns (useful to pack mutiple points in experiments)
- **(encode_file)** (*str*) – if 'yes', encode file
- **(skip_tmp)** (*str*) – if 'yes', skip tmp files and directories

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

actions (dict): dict with actions in the given CK module

(file_content_base64) (str): if i['to_json']=='yes', encoded file

(filename) (str): filename to record locally

`ck.kernel.push(i)`

Push CK entry to the CK server Target audience: CK kernel and low-level developers

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA
- **module_uoa** (*str*) – must be "module"
- **data_uoa** (*str*) – UOA of the module for the new action
- **(filename)** (*str*) –

  filename (with path) (if empty, set archive to 'yes'). If empty, create an archive of the entry

  or
- **(cid[0])** (*str*)
- **(extra_path)** (*str*) – extra path inside entry (create if doesn't exist)
- **(file_content_base64)** (*str*) – if !='', take its content and record into filename
- **(archive)** (*str*) – if 'yes' push to entry and unzip . . .
- **(overwrite)** (*str*)

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.kernel.pwiki(i)`

Open web browser with the private discussion wiki page for a given CK entry Target audience: CK kernel and low-level developers

URL is taken from default kernel configuration cfg['private_wiki_data_web']

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA
- **(module_uoa)** (*str*) – CK module UOA
- **(data_uoa)** (*str*) – CK entry (data) UOA

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.kernel.`**`python_version`**`(i)`

CK action: print python version used by CK Target audience: end users

Parameters **(dict)** – empty dict

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

version (str): sys.version

version_info (str): sys.version_info

`ck.kernel.`**`reinit`**`()`

Reinitialize CK Target audience: end users

Parameters **None**

Returns *(dict)* – output from the "init" function

`ck.kernel.`**`reload_repo_cache`**`(i)`

Reload cache with meta-descriptions of all CK repos Target audience: CK kernel and low-level developers

Parameters **(force)** (*str*) – if 'yes', force recaching

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.kernel.`**`remove`**`(i)`

CK action: delete CK entry or CK entries Target audience: should use via ck.kernel.access

Parameters **See "rm" function**

Returns See "rm" function

`ck.kernel.`**`remove_action`**`(i)`

Remove an action from the given module Target audience: should use via ck.kernel.access

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA
- **module_uoa** (*str*) – must be "module"
- **data_uoa** (*str*) – UOA of the module for the new action
- **func** (*str*) – action name

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

Output from the 'update' function for the given CK module

`ck.kernel.`**`ren`**(*i*)

CK action: rename CK entry Target audience: should use via ck.kernel.access

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA
- **module_uoa** (*str*) – CK module UOA
- **data_uoa** (*str*) – CK entry (data) UOA
- **new_data_uoa** (*str*) – new CK entry (data) alias or
- **new_data_uid** (*str*) – new CK entryt (data) UID (leave empty to keep the old one) or
- **xcids** (*list*) – take new CK entry UOA from xcids[0]['data_uoa']
- **(new_uid)** (*str*) – if 'yes', generate new UID
- **(remove_alias)** (*str*) – if 'yes', remove alias
- **(add_uid_to_alias)** (*str*) – if 'yes', add UID to alias
- **(share)** (*str*) – if 'yes', try to remove the old entry via GIT and add the new one

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.kernel.`**`rename`**(*i*)

CK action: rename CK entry Target audience: should use via ck.kernel.access

Parameters **See "ren" function**

Returns See "ren" function

`ck.kernel.`**`restore_flattened_dict`**(*i*)

Restore flattened dict Target audience: end users

Parameters **dict** (*dict*) – CK flattened dictionary

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

dict (dict): restored dict

`ck.kernel.`**`restore_state`**(*r*)

Restore CK state Target audience: end users

Parameters **r** (*dict*) – saved CK state

Returns *(dict)* – output from the "init" function

`ck.kernel.`**`rm`**(*i*)

CK action: delete CK entry or CK entries Target audience: should use via ck.kernel.access

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA
- **module_uoa** (*str*) – CK module UOA
- **data_uoa** (*str*) – CK entry (data) UOA
- **(force)** (*str*) – if 'yes', force deleting without questions or
- **(f)** (*str*) – to be compatible with rm -f
- **(share)** (*str*) – if 'yes', try to remove via GIT
- **(tags)** (*str*) – use these tags in format tags=x,y,z to prune rm or
- **(search_string)** (*str*)

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.kernel.`**`rm_read_only`**(*f*, *p*, *e*)

`ck.kernel.`**`run_and_get_stdout`**(*i*)

Run command and log stdout and stdout Target audience: end users

Parameters

- **cmd** (*list*) – list of command line arguments, starting with the command itself
- **(shell)** (*str*) – if 'yes', reuse shell environment

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

return_code (int): return code from the os.system call

stdout (str): standard output of the command

stderr (str): standard error of the command

`ck.kernel.`**`safe_float`**(*i*, *d*)

Support function for safe float (useful for sorting function) Target audience: end users

Parameters

- **i** (*any*) – variable with any type
- **d** (*float*) – default value

Returns *(float)* – returns i if it can be converted to float or d otherwise

`ck.kernel.`**`safe_get_val_from_list`**(*lst*, *index*, *default_value*)

Support function to get value from list without error if out of bounds Target audience: end users

Useful for sorting functions.

Parameters

- **lst** (*list*) – list of values
- **index** (*int*) – index in a list
- **default_value** (*any*) – if index inside list, return lst[index] or default value otherwise

Returns *(int)* – returns i if it can be converted to int, or d otherwise

`ck.kernel.`**`safe_int`**(*i*, *d*)

Support function for safe int (useful for sorting function) Target audience: end users

Parameters

- **i** (*any*) – variable with any type
- **d** (*int*) – default value

Returns *(int)* – returns i if it can be converted to int, or d otherwise

`ck.kernel.`**`save_json_to_file`**(*i*)

Save dict to a json file Target audience: end users

Parameters

- **json_file** (*str*) – filename to save dictionary
- **dict** (*dict*) – dict to save
- **(sort_keys)** (*str*) – if 'yes', sort keys
- **(safe)** (*str*) – if 'yes', ignore non-JSON values (only for Debugging - changes original dict!)

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.kernel.save_repo_cache(i)`

Save cache with meta-descriptions of all CK repos Target audience: CK kernel and low-level developers

Parameters (dict) – empty dict

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.kernel.save_state()`

Save CK state Target audience: end users

FGG: note that in the future we want to implement CK kernel as a Python class where we will not need such save/restore state ...

Parameters None

Returns *(dict)* – current CK state

`ck.kernel.save_text_file(i)`

Save string to a text file with all removed

Target audience: end users

Args: text_file (str): name of a text file string (str): string to write to a file (all

will be removed)

(append) (str): if 'yes', append to a file

Returns: (dict): Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.kernel.save_yaml_to_file(i)`

Save dict to a YAML file Target audience: end users

Parameters

- **yaml_file** (*str*) – name of a YAML file
- **dict** (*dict*) – dict to save

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.kernel.search(i)`

CK action: search CK entries Target audience: should use via ck.kernel.access

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA with wildcards
- **(module_uoa)** (*str*) – CK module UOA with wildcards
- **(data_uoa)** (*str*) – CK entry (data) UOA with wildcards
- **(repo_uoa_list)** (*list*) – list of CK repos to search
- **(module_uoa_list)** (*list*) – list of CK modules to search
- **(data_uoa_list)** (*list*) – list of CK entries to search
- **(filter_func)** (*str*) – name of the filter function to customize search
- **(filter_func_addr)** (*obj*) – Python address of the filter function
- **(add_if_date_before)** (*str*) – add only entries with date before this date
- **(add_if_date_after)** (*str*) – add only entries with date after this date
- **(add_if_date)** (*str*) – add only entries with this date
- **(ignore_update)** (*str*) – if 'yes', do not add info about update (when updating in filter)
- **(search_by_name)** (*str*) – search by name
- **(search_dict)** (*dict*) – search if this dict is a part of the entry
- **(ignore_case)** (*str*) – ignore string case when searching!
- **(print_time)** (*str*) – if 'yes', print elapsed time at the end
- **(do_not_add_to_lst)** (*str*) – if 'yes', do not add entries to lst
- **(time_out)** (*float*) – in secs, default=30 (if -1, no timeout)
- **(print_full)** (*str*) – if 'yes', show CID (repo_uoa:module_uoa:data_uoa) or
- **(all)** (*str*) – the same as above
- **(print_uid)** (*str*) – if 'yes', print UID in brackets
- **(print_name)** (*str*) – if 'yes', print name (and add info to the list) or
- **(name)** (*str*) – the same as above
- **(add_info)** (*str*) – if 'yes', add info about entry to the list
- **(add_meta)** (*str*) – if 'yes', add meta about entry to the list
- **(internal)** (*str*) – if 'yes', use internal search even if indexing is on
- **(limit_size)** (*int*) – limit the number of returned entries. Use 5000 by default or set to -1 if no limit
- **(start_from)** (*int*) – start from a specific entry (only for ElasticSearch)
- **(debug)** (*str*) – if 'yes', print debug info

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

lst (list): [{'repo_uoa', 'repo_uid',

'module_uoa', 'module_uid', 'data_uoa','data_uid', 'path' (,meta) (,info) . . .

}]

elapsed_time (float): elapsed time in string

(timed_out) (str): if 'yes', timed out or limited by size

}

`ck.kernel.search2(i)`

`ck.kernel.search_filter(i)`

Search filter Target audience: CK kernel and low-level developers

Parameters

- **repo_uoa** (*str*) – CK repo UOA
- **module_uoa** (*str*) – CK module UOA
- **data_uoa** (*str*) – CK entry (data) UOA
- **path** (*str*) – path to the current entry
- **(search_dict)** (*dict*) – check if this dict is a part of the entry meta description
- **(ignore_case)** (*str*) – if 'yes', ignore case of letters

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

skip (str): if 'yes', skip this entry from search

`ck.kernel.search_string_filter(i)`

Search filter Target audience: CK kernel and low-level developers

Parameters

- **repo_uoa** (*str*) – CK repo UOA
- **module_uoa** (*str*) – CK module UOA
- **data_uoa** (*str*) – CK data UOA
- **path** (*str*) – path to the current CK entry
- **(search_string) - search with expressions *?**

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

skip (str): if 'yes' then skip this entry from search

`ck.kernel.select(i)`

Universal selector of a dictionary key Target audience: end users

Note: advanced version available in the CK module "choice"

Parameters

- **dict** (*dict*) – dict with values being dicts with 'name' as string to display and 'sort' as int (for ordering)
- **(title)** (*str*) – print title
- **(error_if_empty)** (*str*) – if 'yes' and just Enter, return error
- **(skip_sort)** (*str*) – if 'yes', do not sort dictionary keys

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

string (str): selected dictionary key

`ck.kernel.`**`select_uoa`**(*i*)

Universal CK entry UOA selector Target audience: end users

Note: advanced version available in the CK module "choice"

Parameters

- **choices** (*list*) – list from the search function
- **(skip_enter)** (*str*) – if 'yes', do not select 0 when a user presses Enter
- **(skip_sort)** (*str*) – if 'yes', do not sort list

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

choice (str): CK entry UOA

`ck.kernel.`**`set_by_flat_key`**(*i*)

Set a value in a dictionary using the CK flat key Target audience: end users

Parameters

- **dict** (*dict*) – dictionary
- **key** (*str*) – CK flat key
- **value** (*any*) – value to set

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

dict (dict): modified dict

`ck.kernel.set_lock(i)`

Set a lock in a given path (to handle parallel writes to CK entries) Target audience: CK kernel and low-level developers

Parameters

- **path** (*str*) – path to be locked
- **(get_lock)** (*str*) – if 'yes', lock this entry
- **(lock_retries)** (*int*) – number of retries to aquire lock (default=11)
- **(lock_retry_delay)** (*float*) – delay in seconds before trying to aquire lock again (default=3)
- **(lock_expire_time)** (*float*) – number of seconds before lock expires (default=30)
- **(unlock_uid)** (*str*) – UID of the lock to release it

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

(lock_uid) (str): lock UID, if locked successfully

`ck.kernel.short_help(i)`

Print short CK help Target audience: end users

Parameters **(dict)** – empty dict

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

help (str): string with the help text

`ck.kernel.split_name(name, number)`

Support function to split entry name (if needed) Target audience: CK kernel and low-level developers

Parameters

- **name** (*str*) – CK entry name
- **number** (*int*) – Split number (do not split if 0)

Returns

(name1 (str): first part of splitted name

name2 (str): second part of splitted name

)

`ck.kernel.status(i)`

CK action: check CK server status Target audience: CK kernel and low-level developers

Parameters **(dict)** – empty dict

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

outdated (str): if 'yes', newer version exists

`ck.kernel.substitute_str_in_file(i)`

Substitute string in a file Target audience: end users

Parameters

- **filename** (*str*) – filename
- **string1** (*str*) – string to be replaced
- **string2** (*str*) – replacement string

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.kernel.system_with_timeout(i)`

os.system with time out Target audience: end users

Parameters

- **cmd** (*str*) – command line
- **(timeout)** (*float*) – timeout in seconds (granularity 0.01 sec) - may cause some overheads . . .

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

return_code (int): return code from the os.system call

`ck.kernel.system_with_timeout_kill(proc)`

Support function to safely terminate a given process Target audience: end users

Parameters **proc** (*obj*) – process object

Returns None

`ck.kernel.uid(i)`

CK action: generate CK UID Target audience: end users

Parameters **(dict)** – empty dict

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

data_uid (str): UID in string format (16 lowercase characters 0..9,a..f)

`ck.kernel.unzip_file(i)`

Unizip archive file to a given path Target audience: end users

Parameters

- **archive_file** (*str*) – full path to a zip file
- **(path)** (*str*) – path where to unzip (use current path if empty)
- **(overwrite)** (*str*) – if 'yes', overwrite existing files
- **(delete_after_unzip)** (*str*)

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

skipped (list): list of files that were not overwritten

`ck.kernel.update(i)`

CK action: update CK entry meta-description Target audience: should use via ck.kernel.access

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA
- **module_uoa** (*str*) – CK module UOA
- **data_uoa** (*str*) – CK entry (data) UOA
- **(data_uid)** (*str*) – CK entry (data) UID (if UOA is an alias)
- **(data_name)** (*str*) – User-friendly name of this entry
- **(dict)** (*dict*) – meta description for this entry (will be recorded to meta.json)
- **(substitute)** (*str*) – if 'yes' and update=='yes' substitute dictionaries, otherwise merge!
- **(dict_from_cid)** (*str*) – if !="", merge dict to meta description from this CID (analog of copy)
- **(dict_from_repo_uoa)** (*str*) – merge dict from this CK repo UOA
- **(dict_from_module_uoa)** (*str*) – merge dict from this CK module UOA
- **(dict_from_data_uoa)** (*str*) – merge dict from this CK entry UOA

- **(desc)** (*dict*) – under development - defining SPECs for meta description in the CK flat format
- **(extra_json_files)** (*dict*) – dict with extra json files to save to this CK entry (keys in this dictionary are filenames)
- **(tags)** (*str*) – list or comma separated list of tags to add to entry
- **(info)** (*dict*) – entry info to record - normally, should not use it!
- **(extra_info)** (*dict*) –

  enforce extra info such as

  - author
  - author_email
  - author_webpage
  - license
  - copyright

  If not specified then take it from the CK kernel (prefix '**default_**')
- **(updates)** (*dict*) – entry updates info to record - normally, should not use it!
- **(ignore_update)** (*str*) – if 'yes', do not add info about update
- **(ask)** (*str*) – if 'yes', ask questions, otherwise silent
- **(unlock_uid)** (*str*) – unlock UID if was previously locked
- **(sort_keys)** (*str*) – by default, 'yes'
- **(share)** (*str*) – if 'yes', try to add via GIT
- **(skip_indexing)** (*str*) – if 'yes', skip indexing even if it is globally on
- **(allow_multiple_aliases)** (*str*) – if 'yes', allow multiple aliases for the same UID (needed for cKnowledge.io to publish renamed components with the same UID)

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

Output from the "add" function (the last "add" in case of wildcards)

`ck.kernel.`**`version`**(*i*)

CK action: print CK version Target audience: end users

Parameters **(dict)** – empty dict

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

version (list): list of sub-versions starting from major version number

version_str (str): version string

`ck.kernel.webapi(i)`

Open web browser with the API page if exists Target audience: CK kernel and low-level developers

Parameters **(dict)** – from the "access" function(repo_uoa) (str): CK repo UOA

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.kernel.webhelp(i)`

Open web browser with the help page for a given CK entry Target audience: CK kernel and low-level developers

Parameters **(dict)** – from the "access" function

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.kernel.wiki(i)`

Open web browser with the discussion wiki page for a given CK entry Target audience: CK kernel and low-level developers

URL is taken from default kernel configuration cfg['wiki_data_web']

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA
- **(module_uoa)** (*str*) – CK module UOA
- **(data_uoa)** (*str*) – CK entry (data) UOA

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.kernel.zip(i)`

Zip CK entries Target audience: CK kernel and low-level developers

Parameters

- **(repo_uoa)** (*str*) – CK repo UOA with wild cards
- **(module_uoa)** (*str*) – CK module UOA with wild cards
- **(data_uoa)** (*str*) – CK entry (data) UOA with wild cards
- **(archive_path)** (*str*) – if '' create inside repo path

- **(archive_name)** (*str*) – if !=" use it for zip name
- **(auto_name)** (*str*) – if 'yes', generate name name from data_uoa: ckr-<repo_uoa>.zip
- **(bittorent)** (*str*) – if 'yes', generate zip name for BitTorrent: ckr-<repo_uid>-YYYYMMDD.zip
- **(overwrite)** (*str*) – if 'yes', overwrite zip file
- **(store)** (*str*) – if 'yes', store files instead of packing

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

13.3 ck.files module

`ck.files.load_json_file(i)`

Load json from file into dict Target audience: end users

Parameters **json_file** (*str*) – name of a json file

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

dict (dict or list): dict or list from the json file

`ck.files.load_text_file(i)`

Load a text file to a string or list Target audience: end users

Parameters

- **text_file** (*str*) – name of a text file
- **(keep_as_bin)** (*str*) – if 'yes', return only bin
- **(encoding)** (*str*) – by default 'utf8', however sometimes we use utf16
- **(split_to_list)** (*str*) – if 'yes', split to list
- **(convert_to_dict)** (*str*) – if 'yes', split to list and convert to dict
- **(str_split)** (*str*) – if !=", use as separator of keys/values when converting to dict
- **(remove_quotes)** (*str*) – if 'yes', remove quotes from values when converting to dict
- **(delete_after_read)** (*str*) – if 'yes', delete file after read (useful when reading tmp files)

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

bin (byte): loaded text file as byte array

(string) (str): loaded text as string with removed

(lst) (list): if split_to_list=='yes', split text to list

(dict) (dict): if convert_to_dict=='yes', return as dict

`ck.files.`**`load_yaml_file`**(*i*)

Load YAML file to dict Target audience: end users

Parameters **yaml_file** (*str*) – name of a YAML file

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

dict (dict): dict from a YAML file

`ck.files.`**`save_json_to_file`**(*i*)

Save dict to a json file Target audience: end users

Parameters

- **json_file** (*str*) – filename to save dictionary
- **dict** (*dict*) – dict to save
- **(sort_keys)** (*str*) – if 'yes', sort keys
- **(safe)** (*str*) – if 'yes', ignore non-JSON values (only for Debugging - changes original dict!)

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.files.`**`save_text_file`**(*i*)

Save string to a text file with all removed

Target audience: end users

Args: text_file (str): name of a text file string (str): string to write to a file (all

will be removed)

(append) (str): if 'yes', append to a file

Returns: (dict): Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.files.save_yaml_to_file(i)`

Save dict to a YAML file Target audience: end users

Parameters

- **yaml_file** (*str*) – name of a YAML file
- **dict** (*dict*) – dict to save

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

13.4 ck.net module

`ck.net.access_ck_api(i)`

Universal web request to the CK server (usually cKnowledge.io) Target audience: CK kernel and low-level developers

Parameters

- **url** (*str*) – URL API
- **(dict)** (*dict*) – dict to send to above URL

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

dict (dict): dictionary from the CK server

`ck.net.request(i)`

Web request to cKnowledge.org server Target audience: CK kernel and low-level developers

Parameters

- **get** (*dict*) – GET parameters
- **post** (*dict*) – POST parameters

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

string (str): returned string from the server dict (dict): JSON string converted to dict (if possible)

13.5 ck.strings module

`ck.strings.convert_json_str_to_dict(i)`

Convert string in a special format to dict (JSON) Target audience: end users

Parameters **str** (*str*) – string (use ' instead of ", i.e. {'a':'b'} to avoid issues in CMD in Windows and Linux!)

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

dict (dict): dict from json file

`ck.strings.copy_to_clipboard(i)`

Copy string to clipboard if supported by OS (requires Tk or pyperclip) Target audience: end users

Parameters **string** (*str*) – string to copy

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

`ck.strings.dump_json(i)`

Dump dictionary (json) to a string Target audience: end users

Parameters

- **dict** (*dict*) – dictionary to convert to a string
- **(skip_indent)** (*str*) – if 'yes', skip indent
- **(sort_keys)** (*str*) – if 'yes', sort keys

Returns

(dict) –

Unified CK dictionary:

return (int): return code = 0, if successful > 0, if error

(error) (str): error text if return > 0

string (str): JSON string

13.6 Module contents

CHAPTER 14

Miscellaneous

- CK Wiki
- cKnowledge.io docs

CHAPTER 15

Index

- genindex

Python Module Index

C

Index

U

V

W

Z