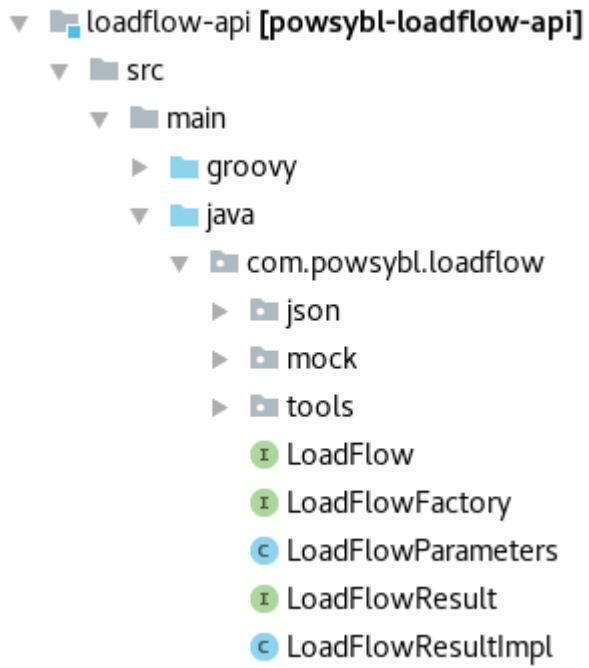
How to : Implement the load-flow API

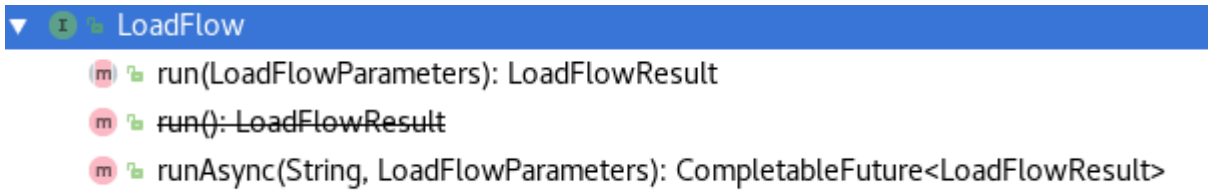
SEBASTIEN MURGEY (RTE)

Load-flow API overview



- Three interfaces to be implemented
- One default implementation for LoadFlowResult interface
- LoadFlowParameters extension is possible

Load-flow interface contract



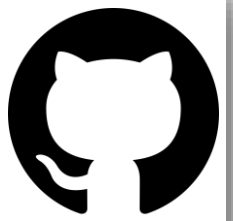
run() : synchronous execution of load-flow computation

runAsync() : asynchronous execution of load-flow computation

But : It does not appear, but load-flow computation is also supposed to modify / update the associated network state…

Load-flow interface : PyPSA implementation

```
 *
 * @author Sebastien Murgey <sebastien.murgey at rte-france.com>
 */
public class PyPsaLoadFlow implements LoadFlow {
    private static final Logger LOGGER = LoggerFactory.getLogger(PyPsaLoadFlow.class);
    private static final String WORKING_DIR_PREFIX = "powsybl_pypsa_";
    private final Network network;
    private final ComputationManager computationManager;

    /**
     * Loadflow implementation constructor
     *
     * @param network the network to use as input for loadflow computation
     * @param computationManager the computation manager
     * @param priority priority of computation
     */
    public PyPsaLoadFlow(Network network, ComputationManager computationManager, int priority) {
        this.network = Objects.requireNonNull(network, "network is null");
        this.computationManager = Objects.requireNonNull(computationManager, "computation manager is null");
    }

    @Override
    public LoadFlowResult run(LoadFlowParameters loadFlowParameters) throws Exception {
        Objects.requireNonNull(loadFlowParameters);
        LOGGER.debug("Using parameters {}", loadFlowParameters);
```

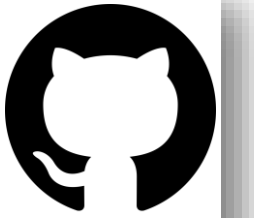


Load-flow factory interface contract

create() : create load-flow instance for given network

Load-flow factory interface : PyPSA implementation

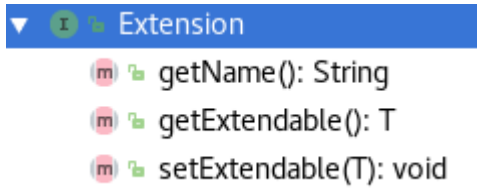
```java
/**
 * Copyright (c) 2018, RTE (http://www.rte-france.com)
 * This Source Code Form is subject to the terms of the Mozilla Public
 * License, v. 2.0. If a copy of the MPL was not distributed with this
 * file, You can obtain one at http://mozilla.org/MPL/2.0/.
 */
package eu.itesla_project.pypsa_loadflow;

import com.powsybl.computation.ComputationManager;
import com.powsybl.iidm.network.Network;
import com.powsybl.loadflow.LoadFlow;
import com.powsybl.loadflow.LoadFlowFactory;

/**
 * PyPSA loadflow instance factory
 *
 * @author Sebastien Murgey <sebastien.murgey at rte-france.com>
 */
public class PyPsaLoadFlowFactory implements LoadFlowFactory {
    @Override
    public LoadFlow create(Network network, ComputationManager computationManager, int i) {
        return new PyPsaLoadFlow(network, computationManager, i);
    }
}
```



Loadflow parameters extension (if needed)

LoadFlowParameters class inherits from **AbstractExtandable**, allowing to extend the load-flow standard parameters with a mechanism that integrates it nicely in PowSyBl framework.

Adding implementation specific parameters can be done easily by creating custom extension of LoadFlowParameters.

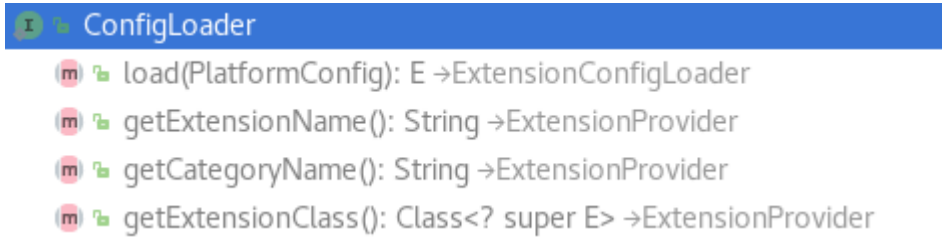


It’s easier to inherit directly from **AbstractExtension<LoadFlowParameters>**

Loadflow parameters extension « Config loader »

To be able to load extra parameters from standard PowSyBl configuration files, it is needed to provide a configuration loader, inheriting from **LoadFlowParameters.ConfigLoader<>.**

Use of **@AutoService(LoadFlowParameters.ConfigLoader.class)** annotation from com.google.auto.service package is encouraged.



load() : load parameters from PowSyBl platform configuration
getExtensionName() : returns extension name
getCategoryName() : returns category of the extensions provided
→ should return ``*loadflow-parameters*``
getExtensionClass() : returns the class of custom extension

Loadflow parameters extension PyPSA implementation

```java

/**
 * PyPSA loadflow additional parameters
 *
 * @author Sebastien Murgey <sebastien.murgey at rte-france.com>
 */
public class PyPsaLoadFlowParameters extends AbstractExtension<LoadFlowParameters> {
    public static final boolean DEFAULT_DEBUG_ACTIVATED = false;
    public static final boolean DEFAULT_DC_LOAD_FLOW = false;
    public static final float DEFAULT_RELAXATION_COEFF = 1f;

    private boolean debugActivated = DEFAULT_DEBUG_ACTIVATED;
    private boolean dcLoadFlow = DEFAULT_DC_LOAD_FLOW;
    private float relaxationCoeff = DEFAULT_RELAXATION_COEFF;

    @Override
    public String getName() {
        return "PyPSALoadflowParameters";
    }

    /**
     * Get a flag triggering the activation of debug mode
     *
     * While in debug mode, working directory generated for computation
     * is not removed at the end of the computation, and some extra consistency checks
     * are performed before powerflow computation by PyPSA.
     *
     * @return the flag triggering the activation of debug mode
     */
    public boolean isDebugActivated() {
        return debugActivated;
    }

    /**
     * Get a flag triggering the activation of debug mode.
     * Default value is false.
     *
     * @param debugActivated the flag to set, true is debug mode activated, false otherwise
     * @return this
     */
```

```java
/**
 * PyPSA loadflow additional parameters configuration loader
 *
 * Used to load PyPSA loadflow additional parameters from PowSyBl configuration
 *
 * @author Sebastien Murgey <sebastien.murgey at rte-france.com>
 */
@AutoService(LoadFlowParameters.ConfigLoader.class)
public class PyPsaLoadFlowParametersConfigLoader implements LoadFlowParameters.ConfigLoader<PyPsaLoadFlowParameter

    static final String MODULE_NAME = "pypsa-default-loadflow-parameters";

    @Override
    public String getExtensionName() {
        return "PyPSALoadflowParameters";
    }

    @Override
    public String getCategoryName() {
        return "loadflow-parameters";
    }

    @Override
    public Class<? super PyPsaLoadFlowParameters> getExtensionClass() {
        return PyPsaLoadFlowParameters.class;
    }

    @Override
    public PyPsaLoadFlowParameters load(PlatformConfig platformConfig) {
        Objects.requireNonNull(platformConfig);
        PyPsaLoadFlowParameters parameters = new PyPsaLoadFlowParameters();
        ModuleConfig config = platformConfig.getModuleConfigIfExists(MODULE_NAME);
        if (config != null) {
            parameters.setDebugActivated(config.getBooleanProperty("debugActivated", DEFAULT_DEBUG_ACTIVATED));
            parameters.setDcLoadFlow(config.getBooleanProperty("dcLoadFlow", DEFAULT_DC_LOAD_FLOW));
            parameters.setRelaxationCoeff(config.getFloatProperty("relaxationCoeff", DEFAULT_RELAXATION_COEFF)
        }
        return parameters;
    }
}
```

Configuring PowSyBl to use your own load-flow implementation

For choosing which loadflow implementation to use, modify your configuration file (by default $HOME/.itools/config.xml).

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
  <componentDefaultConfig>
    <LoadFlowFactory>eu.itesla_project.pypsa_loadflow.PyPsaLoadFlowFactory</LoadFlowFactory>
  </componentDefaultConfig>

  <load-flow-default-parameters>
    <voltageInitMode>DC_VALUES</voltageInitMode>
  </load-flow-default-parameters>

  <pypsa-default-loadflow-parameters>
    <debugActivated>true</debugActivated>
    <dcLoadFlow>false</dcLoadFlow>
    <relaxationCoeff>0.5</relaxationCoeff>
  </pypsa-default-loadflow-parameters>
</config>
```

Ressources

PowSyBl core source code : https://github.com/powsybl/powsybl-core

PyPSA loadflow integration source code : https://github.com/murgeyseb/pypsa-loadflow

Modified PyPSA source code : https://github.com/murgeyseb/PyPSA

Thank you for your attention !