

Measure Platform Developers Guide

Measurement and Data Analysis Platform

.....

Extend the Measure Platform

The Measure platform can be extended in order to support new data sources, new ways to visualise collected data or now data analysis services. The Measure Platform developers guide provides the required key to the development of these extensions. The Platform can be extended as follows:

Development of New Measure

A Measure is a small and autonomous java program based on the SMM specification which allow to collect measurements. The Measures make the link between a Measurement Tool, a Remote Service, a Captor or any others kind of data sources and the Measure Platform.

If you plan to monitor measures that are not currently supported by the measure platform, you will probably have to develop your own Measures.

There two kind of Measures:

- The **Direct Measure** (Collect of measurement in physical world), a Proxy (Ensure communication between a Measurement Tool and the Platform) or
- The **Derived Measure** (Measure calculated by the aggregation of existing Measures).

Development of New Measurement Applications

An Application is a set of Measures aggregated together in order to address functional requirements. The application is associated with a visual dashboard which directly integrated into the Decision-Making platform when the Application is deployed on a project.

Development of New Analysis Service

In order to support a large set of analyses services and do not limit to it a specific technology, the Analysis Tools are external processes. Although external, we wanted a deep integration between the platform and the analysis tools. We solved this issue in the following way:

- The Measure platform provides a REST API which allows an analysis tool to register it on the platform, to receive notification from the platform and access to information related to project defined and measure collected by the platform.
- On its side, the analysis tool provides some web pages which will be embedded into the platform web application.

Measure Development

Measure Architecture

A SMM Measure is a small and independent software component which allows retrieving or calculating a measurement. The Measure make the link between a remote measurement or service and the MeasurePlatform. The implementation of a measure is based on a library developed in parallel with the Measure Platform: the "SMMMeasureApi". We identify two main kinds of measures: The Direct Measures and the Derived Measures.

- A **Direct Measure** is used to collect data in physical world. This kind of measure can be executed on the platform on the Client Side. To define a Direct Measure, the IDirectMeasure has to be implemented. This interface will be called by the Measure Platform to retrieve the measurements.

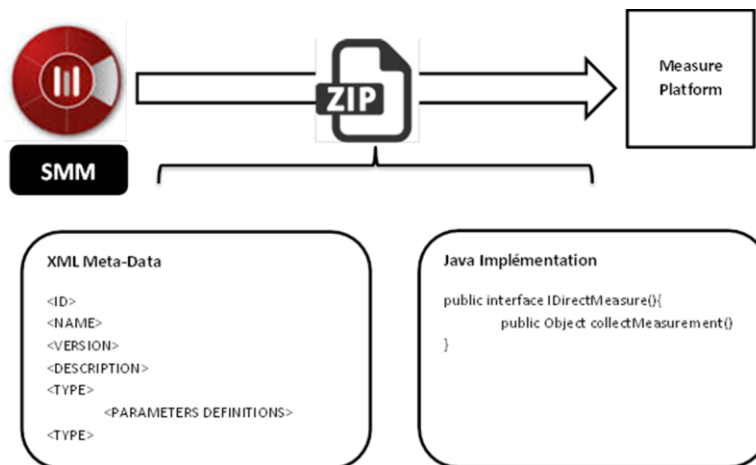
- A **Derived Measure** is used to define a combined measure which can calculate new measurements using one or more measurements stored on the Measure platform. To define a Derived Measure, the `IDerivedMeasure` has to be implemented.

In this section, we will describe how to specify, implement and package a new measure which will be deployed on the Measure platform. The Modelio Modeling tool can be used for the specification, implementation and packaging of measures, but it is also possible to implement the measure manually.

In SMM, a direct or derived measure definition is associated with an Operation which represents the implementation of the measure. These operations can be expressed in natural language or may contain executable code. In order to be able to collect direct measures and to execute calculated measure, we have to choose a common executable language. For that, we currently support Java.

An SMM Measure is a zip file containing:

- A **Jar file**: The Java implementation of the measure
- A **lib folder**: Java libraries used by the measure implementation
- A **MetaData.xml file**: Metadata related to the measure



The Jar file contained the implementation the measure itself. In order to be executed by the platform, this implementation is based on the `SMMMeasureApi` available at this URL:

<https://github.com/ITEA3-Measure/SMMMeasureApi/>

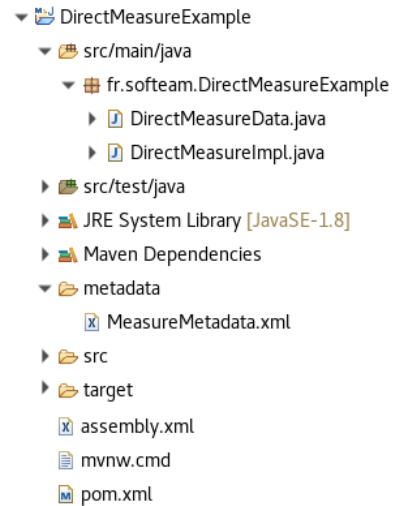
The metadata file is an xml file containing several information's related to the measure and used by the measure platform to load dynamically the Measure. It allows to define the scope (dynamic properties provided wen the measure is deploy on a project of the Measure Platform) and the data model returned by the measure when executed.

Develop a Measure Using Maven

In order to help you to start the development of a new Measure, a Maven Archetype is available on our Maven repository: <http://repository.modelio.org>

To create the implementation project from the Maven Archetype in Eclipse:

- Create a new Maven project using an Archetype
- Register the Modelio maven repository as new remote maven catalogue
<http://repository.modelio.org>
- Select the **direct-measure-archetype** Archetype or the **derived-measure-archetype** Archetype depending of the kind of measure you which to implement.
- A preconfigured maven development project dedicated to measure is created



Use an existing project template which will allow you to start the implementation of a new measure is also available at this address:

<https://github.com/ITEA3-Measure/Measures/tree/master/Examples/TemplateMeasure>

Once the implementation of the measure is completed, you can package your measure as ZIP in a format compatible with the Measure Platform using the Maven Install compilation target.

Develop a Measure using the Modelio Modelling Tool

The Modelio Modeling tool supports the Structured Metrics Model (SMM) standard. This specification defines a meta-model for representing measurement information related to any model-based information with an initial focus on software, its operation, and its design. Referred to as the Structured Metrics Meta-model (SMM), this specification is an extensible meta-model for exchanging both measures and measurement information concerning artefacts contained or expressed by structured models, such as MOF.

The SMMLibrary Module is an extension for Modelio 3.8 tool, which allows to model, specify, implement, and package new catalogue of measures in SMM format.

1. **Download the Modelio Open Source 3.8.0:** <https://www.modelio.org/downloads/download-modelio.html>
2. **Download the last SMMDesigner Module:** <https://github.com/ITEA3-Measure/SMM-Designer/releases/tag/1.0.00> Download file : SMM_1.0.00.jmdac
3. **Start Modelio and create a new Project.**
4. **Add the SMM_1.0.00.jmdac module into the project**

This module will allow you to:

- Specify scope, data model and dependency of the measure using Models
- Generate a Maven implementation project based on this specification.
- Help you to implement the measure using Model Driven Development Approach
- Package the measure in a format supported by the Measure Platform

Please refer to the documentation of the SMM Module for more details about the development of measures using Modelio.

Measure Metadata File

The Metadata.xml file contains meta-data related to the SMM Measure:

- Name, description, category and provider of the Measure
- Type of the Measure

- Unite (data model) of the measure
- List of properties of the measure
- List of references for Derived Measure (inputs form other measures)
- Defaults view descriptions associated with a measure.

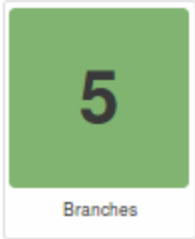
Element	Owner	Attribute	Description
Measure			The Measure
		name	Name / Id of the Measure
		type	SMM Type of the measure: [DIRECT, COLLECTIVE, RANCKING, GRADE, BINARY, COUNTING, ESCALED, RATIO]
		category	Classification of the measure by category
		provider	People / Entity which developed the measure
description	Measure (1)		Description of the Measure
unite	Measure (1)		Data Model of measurements returned by the measure
fields	Unite (*)		A Field of the measure unite
		fieldName	Name of the field
		fieldType	Type of the field: [u_text, u_integer, u_long, u_date, u_boolean, u_float, u_geo_point, ...]
scopeProperties	Measure (*)		A property user to configure the execution of the measure
		name	Name of the property
		defaultValue	Default value of the property
		type	Type of the property: :[STRING, INTEGER, FLOAT, DATE, ENUM, PASSWORD, DESABLE]
description	scopeProperties (1)		Description of the scope property
enumType	scopeProperties (1)		Emum definition for scopeProperties of type ENUM
enumvalue	enumType (*)		Emum values

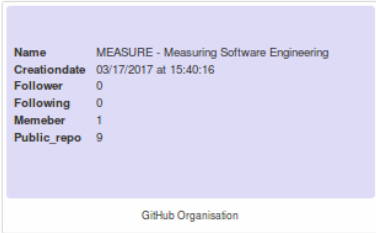
		label	label of the enum entry
		value	value of the enum entry
references	Measure (*)		References to inputs required by Derived Measures
		measureRef	Name of the required measure
		number	Default Number of instances of this input required
		expirationDelay	Filter old measurement
references-role	References (1)		Role of impute in current Measurement. This role allows to identify several instance of the same Measure in a DerivedMeasure.
views	Measure		List of default visualisaiton
view	Views(*)		Default way to display measure data
		name	Nane of the visualisation
		type	Type of the visualisation: [VALUE, TABLE, AREA,BAR,LIGNE,CUSTOM]
		default	Indicate if this visualisation will be created automatically when the measure is activated [true,false]
		autoRefresh	Indicate if this visualisation will be updated periodically [true,false]
description	view (1)		Description of the visualisation
datasource	view (1)		Configure data display by the view
		dataIndex	Unite fields display by the view
		dateIndex	Date index used to collect view datas
		timePeriode	Time period display by the view. <ul style="list-style-type: none"> • Format: Number + Period • Periode [m,h,d,w,M,Y]

			Example : 1m = 1 minute
		timeAggregation	Values display are average values of data on specified period [s,m,h,d,w,M,Y]
layout	view (1)		Layout configuration
		width	with in pixel of the view
		height	Height in pixel of the view
		color	Color of the view : Example : %23E24D42
		fontSize	Font size of the view text
customData	View (1)		Custom View, see section dedicated to custom visualisation in this documentation

Example of Measure Visualisations

dfgdfgdfgdfgd

Simple Value	
<pre><view name="Members" type="VALUE" default="true" autoRefresh="false"> <datasource dataIndex="memeber" timePeriode="1y"/> <layout width="150" height="150" fontSize="60" color="#7EB26D" /> </view></pre>	

Multiple Values	
<pre><view name="GitHub Organisation" type="VALUE" default="true" autoRefresh="false"> <datasource dataIndex="name,creationdate,compagny,follower,following,memeber,public_repo" timePeriode="1y"/> <layout width="500" height="260" fontSize="16" color="#DEDAF7" /></pre>	

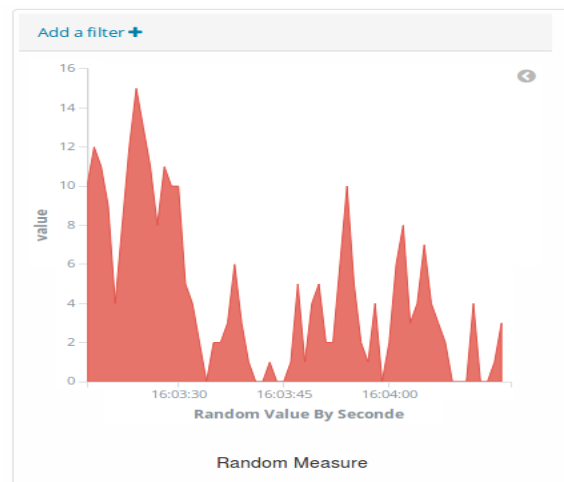
</view>

Data Table

PostDate	Author	CommitMessage	LigneAdd	LigneChanged	LigneDelete
01/30/2019 at 11:11:30	Antonin Abherve	Fix index cration of applications,Change role required to upload a measure in the platform	18	32	14
01/29/2019 at 16:31:40	Antonin Abherve	code cleaning	1015	4641	3626
01/28/2019 at 16:34:02	Amin Boudeffa	Merge pull request #72 from ITEA3-Measure/Feat_right_access Feat right access	20	29	9
01/28/2019 at 16:32:43	Amin Boudeffa	Merge remote-tracking branch 'origin/master' into Feat_right_access	65	97	32
01/28/2019 at 16:31:05	Amin Boudeffa	Add analysis tool routing configuration	20	29	9
01/28/2019 at 14:40:28	Antonin Abherve	null	65	97	32
01/28/2019 at 10:58:12	Amin Boudeffa	Merge pull request #71 from ITEA3-Measure/Feat_right_access Clean existing users platform and activate mailing service	8	21	13
01/28/2019 at 10:57:21	Amin Boudeffa	Clean existing users platform and activate mailing service	8	21	13
01/25/2019 at 17:18:48	Amin Boudeffa	Merge pull request #70 from ITEA3-Measure/Feat_right_access Feat right access	579	668	89
01/25/2019 at 17:18:13	Amin Boudeffa	Fix dashboards SQL query and fix GUI bugs	3	7	4

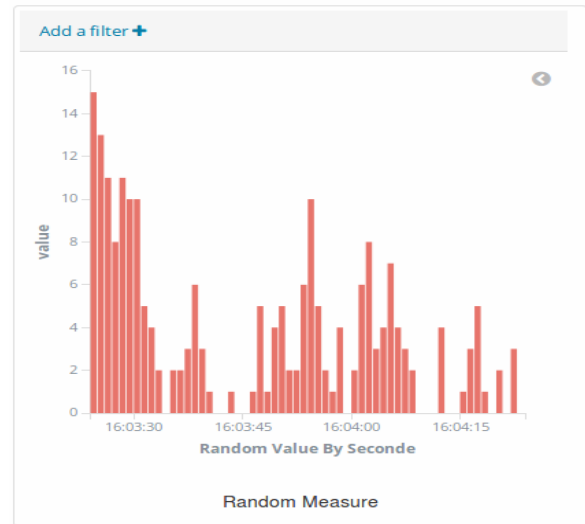
```
<view name="GitHub Issues" type="TABLE" default="true" autoRefresh="false">  
  <description>Commits</description>  
  <datasource dataIndex="UpdatedAt,Tittle,Body,State,Author,ClosedAt,ClosedBy"  
dateIndex="UpdatedAt" timePeriode="2y"/>  
  <layout width="1380" height="600" fontSize="10" color="#ffadad" />  
</view>
```

Area Chart



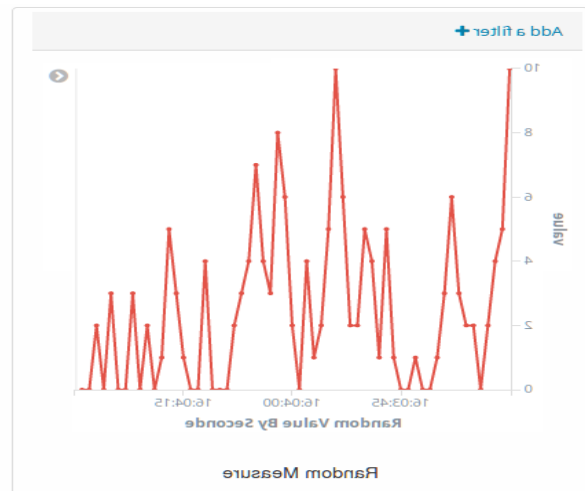
```
<view name="Random Measure" type="AREA" default="true" autoRefresh="true">  
  <description>Random Value By Seconde</description>  
  <datasource dataIndex="value" dateIndex="postDate" timePeriode="1m" timeAggregation="s"/>  
  <layout width="400" height="300" color="%23E24D42" />  
</view>
```

Bar Chart



```
<view name="Random Measure" type="BAR" default="true" autoRefresh="true">
  <description>Random Value By Seconde</description>
  <datasource dataIndex="value" dateIndex="postDate" timePeriod="1m" timeAggregation="s"/>
  <layout width="400" height="300" color="%23E24D42" />
</view>
```

Ligne Chart



```
<view name="Random Measure" type="LIGNE" default="true" autoRefresh="true">
  <description>Random Value By Seconde</description>
  <datasource dataIndex="value" dateIndex="postDate" timePeriod="1m" timeAggregation="s"/>
  <layout width="400" height="300" color="%23E24D42" />
</view>
```

Custom View

The custom views are default measure visualisation created from custom kibana Visualisation. This view are created form Kibana Embedded URL which has bean made customizable and encoded to store it in the MetaData XML file.

Custom View

```

<view name="Repository Activity" default="true">
  <description>Repository Activity</description>
  <datasource timePeriode="60d"/>
  <layout width="450" height="450"/>
  <customData>PGImcmFtZSBzcmM9Imh0dHA6Ly97UExBaWZyYW1lPg==</customData>
</view>

```

To define a custom view :

- Using Kibana visualisation tool, create your own visualisation based on dataforce from the measure in development
- Export this view as “Embedded Code” URL.
- In the URL , replace static values listed below by parameters.
- Encode the string using a standard Base64 encoder like <https://www.base64encode.org/>
- Copy this encoded string in <customData></customData> tag of the metadata file.

Value	Parameter
<iframe src="http://{PLATFORM_URL}/app/	{PLATFORM_URL}
&indexPattern={PLATFORM_INDEX}	{PLATFORM_INDEX}
,time:({PLATFORM_TIMEPERIODE}))	{PLATFORM_TIMEPERIODE}
height="{PLATFORM_HEIGHT }"	{PLATFORM_HEIGHT}
width="{PLATFORM_WIDTH}"	{PLATFORM_WIDTH}

Example of a MeasureMetaData.xml file

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Measure name="RandomMeasure" type="DIRECT" category="Test" provider="MeasurePlatform">
  <description>Return a random measure and his variation</description>

```

```

<scopeProperties defaultValue="100" name="MaxRange" type="INTEGER">
  <description>MaxRange</description>
</scopeProperties>
<scopeProperties defaultValue="0" name="MinRange" type="FLOAT">
  <description>MinRange</description>
</scopeProperties>
<scopeProperties defaultValue="0" name="PreviousValue" type="DESABLE">
  <description>PreviousValue</description>
</scopeProperties>
<scopeProperties defaultValue="Bornd" name="Kind" type="ENUM">
  <description>Kind</description>
  <enumType>
    <enumvalue label="Is Bornd" value="Bornd"/>
    <enumvalue label="Not Bornd" value="UnBornd"/>
  </enumType>
</scopeProperties>
<scopeProperties name="TestDate" type="DATE">
  <description>TestDate</description>
</scopeProperties>
<scopeProperties name="TestPassword" type="PASSWORD">
  <description>TestPassword</description>
</scopeProperties>
<scopeProperties name="TestString" type="STRING">
  <description>TestString</description>
</scopeProperties>
<unit name="RandomMeasurement">
  <fields fieldName=" value" fieldType="u_double"/>
  <fields fieldName="variation" fieldType="u_integer"/>
  <fields fieldName=" postDate" fieldType="u_date"/>
</unit>
<views>
  <view name="Random Measure" type="AREA" default="true" autoRefresh="true">
    <description>Random Value By Seconde</description>
    <datasource dataIndex="value" dateIndex="postDate" timePeriode="1m"
timeAggregation="s"/>
    <layout width="400" height="300" color="%23E24D42" />
  </view>
</views>
</Measure>

```

Direct Measure Implementation

A direct measure is used to collect data in physical world. This kind of measure can be executed on the platform on Client Side. To define a Direct Measure, implement the IDirectMeasure interface. This interface will be called by the MeasurePlatform to retrieve the measurements.

```

public interface IDirectMeasure {
    public List<IMeasurement> getMeasurement() throws Exception;
    public Map<String,String> getProperties();
}

```

To implement a direct measure, please extend the DirectMeasure class:

- **getMeasurement():** calculates and returns a list of measurements.
- **getProperties():** provides a way for the Measure platform to communicate properties to DirectMeasure implementation.

Example: RandomGenerator, a toy measure which returns a random number between MinRange and MaxRange value at each call.

```

public class RandomGenerator extends DirectMeasure {

    @Override
    public List<IMeasurement> getMeasurement() throws Exception {
        List<IMeasurement> result = new ArrayList<>();

```

```

// Retrive Platform Properties by her name
int maxRange = Integer.valueOf(getProperty("MaxRange"));
int minRange = Integer.valueOf(getProperty("MinRange"));

// Collect Measure
Random gen = new Random();
int value = gen.nextInt(maxRange - minRange) + minRange;

// Create Measurement : In this case, a simple IntegerMeasurement
IntegerMeasurement measurement = new IntegerMeasurement();
measurement.setValue(value);
result.add(measurement);
return result;
}
}

```

Derived Measure implementation

A derived measure is used to define a combined measure which calculates new measurements using one or more measurements stored on the Measure platform. To define a derived measure, implement the `IDerivedMeasure` interface. This interface will be called by the Measure Platform to calculate the measurement.

```

public interface IDerivedMeasure {
    public List<IMeasurement> calculateMeasurement() throws Exception;
    public void addMeasureInput(String reference,String role, IMeasurement value);
    public Map<String,String> getProperties();
}

```

To implement a derived measure, please extend the **DerivedMeasure** class:

- **calculateMeasurement():** Calculate and return a list of measurements based on provided measurement inputs.
- **addMeasureInput():** Provide a way for the Measure Platform to communicate input measurements to the DerivedMeasure implementation.
- **getProperties():** Provide a way for the Measure Platform to communicate properties to the Derived Measure implementation.

A Derived Measure allows to combine measurement provided by other measures (Direct or Derived). Required inputs measure are defined on the `MetaData.xml`. These references are identified by a **measureRef** and a **role**.

- The **measureRef** is the id of the measure which can provide a measurement as input.
- The **role** is the role of the input in current measurement. This role allows to identify several instances of the same measure of a Derived Measure.
- The **expirationDelay** property allows to filter as input the measures which has been calculated recently
- The **number** property allows to select the number of inputs of this type which will be communicated to the derived measure implementation by the platform.

```

<references expirationDelay="60000" measureRef="RandomGenerator" number="1">
    <role>RandomNumber A</role>
</references>
<references expirationDelay="60000" measureRef="RandomGenerator" number="1">
    <role>RandomNumber B</role>
</references>

```

Inputs are defined when an instance on the measure is deployed on the Measure Platform.

Example: RandomBinaryMeasure, a toy measure which returns the result of a binary operation between two RandomGenerator result

```

public class RandomBinaryMeasure extends DerivedMeasure {
    @Override

```

```

public List<IMeasurement> calculateMeasurement() {
    Integer result = 0;

    // Retrive input Measurements by her Role
    List<IMeasurement> op1 = getMeasureInputByRole("RandomNumber A");
    List<IMeasurement> op2 = getMeasureInputByRole("RandomNumber B");

    // Calculate result
    if(op1.size() == 1 && op2.size() == 1){
        String oper = "+";

        // Retrive the operator as Property
        oper = getProperty("Operation");

        Integer val1 = (Integer) op1.get(0).getValues().get("value");
        Integer val2 = (Integer) op2.get(0).getValues().get("value");

        if(oper.equals("+")){
            result = val1 + val2;
        }else if(oper.equals("-")){
            result = val1 - val2;
        }else if(oper.equals("*")){
            result = val1 * val2;
        }else if(oper.equals("/")){
            result = val1 / val2;
        }
    }

    // Return result as new IntegerMeasurement
    IntegerMeasurement measurement = new IntegerMeasurement();
    measurement.setValue(result);

    List<IMeasurement> measurements = new ArrayList<>();
    measurements.add(measurement);

    return measurements;
}
}

```

Example: RandomSumMeasure, a toy measure which returns the sum of measurements provided by the RandomGenerator measure.

```

public class RandomSumImpl extends DerivedMeasure {
    @Override
    public List<IMeasurement> calculateMeasurement() throws Exception {
        Integer result = 0;
        for (IMeasurement operande : getMeasureInputByRole("RandomNumber")) {
            try {
                result = result + (Integer) operande.getValues().get("value");
            } catch (NumberFormatException e) {
                System.out.println("Non Numeric Operande");
            }
        }

        IntegerMeasurement measurement = new IntegerMeasurement();
        measurement.setValue(result);

        List<IMeasurement> measurements = new ArrayList<>();
        measurements.add(measurement);
        return measurements;
    }
}

```

Measurement

A Measurement is a data model used as input and output of SMM measure. A measurement has to extend the IMeasurement interface. A measurement is presented as set of Java elements which can be accessed via a Map. All values are accessed using a String identifier defined in MetaData.xml file.

```
public interface IMeasurement {
    public Map<String, Object> getValues();
    public String getLabel();
}
```

Predefined Measurements: The API provides some predefined measurements which can be used in the measure implementation

- IntegerMeasurement: allows to manipulate numbers in the measure implementation

```
int value = 10;
IntegerMeasurement measurement = new IntegerMeasurement();
measurement.setValue(value);
```

Custom Measurements Example: A Measure developer can define custom measurements to manage her own set of data.

The **SVNMeasurement** is used by a measure which collects COMMIT information provided by an SVN repository. It manages data related to the author of the commit, the message on the commit and the date of the commit.

```
public SVNMeasurement(String author,String message,Date postDate){
    super();
    this.valueMap.put("Author", author);
    this.valueMap.put("Message", message);
    this.valueMap.put("postDate",new Date(postDate.getTime()));
}
```

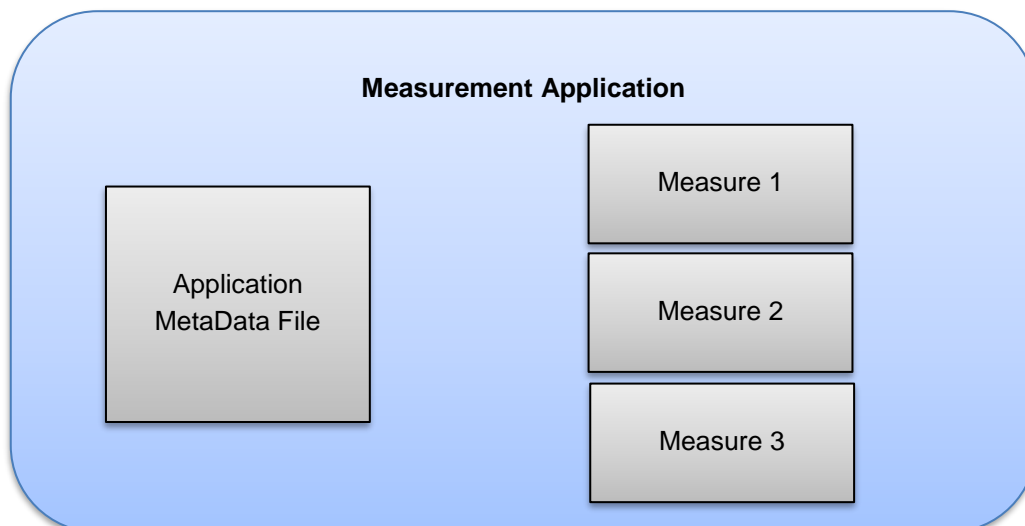
Measures Example

More than 200 measures are available on open source on the GitHub of the Measure project:

<https://github.com/ITEA3-Measure/Measures>

Measurement Application Development

A Measurement Application is a set of measure packaged together to address a clearly defined functional Goal and, when deploy, provide an auto configures dashboard. An application can be deployed on Measure Platform like others measures.



- When the application is deployed on the project, the measure which compose the application are instantiate in a transparent way for the end user
- When the application is activated, the measure which compose the application are collected and a specific dashboard is created
- This specific dashboard is composed form defaults visualisation packaged in the different sub measures.

It is recommended to collect metrics in one application covering the same functional spectrum, or even the same tools. When the application is deployed the list of parameters required to deploy it on a project is the sum of the parameters of the metrics which compose the application. If two metrics have a parameter with the same name, it's considered as the same parameter.

ApplicationMetadata.xml File

The ApplicationMetadata file contain information's related to the list of metrics involved on this application and specification of the composition of dashboards provided by this application.

Element	Owner	Attribute	Description
Application			The Application
		name	Name / Id of the Application
		provider	The developer of the application
description	Application (1)		Description of the Application
measures	Application (1)		List of measures which composed the application
measure	measures (*)		A measures which composed the application
		name	Name of the Measure
		scheduling	Int representing the interval of time between two execution of the measure
		schedulingUnit	Unite of scheduling periode (s=Seconde,...) [s,m,h,d,w,M,Y]
dashboards	Application (1)		List of dashboards provided by the application
dashboard	dashboards (*)		A dashboard provided by the application
		label	Label of the dashboard
view	dashboard (*)		View include in the dashboard
		measure	Name of the measure which provide the view

		view	Name of the view to include in the dashboard
--	--	------	----------------------------------------------

Example of ApplicationMetadata.xml file

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Application name="GitHubRepository" provider="Softeam">
  <description>Monitoring of a GitHub repository including commits supervision and issue
traking</description>
  <measures>
    <measure name="GitHubCommit" scheduling="10" schedulingUnit="m"/>
    <measure name="GitHubIssue" scheduling="10" schedulingUnit="m"/>
    <measure name="GitHubIssueStat" scheduling="10" schedulingUnit="m"/>
    <measure name="GitHubRepository" scheduling="10" schedulingUnit="m"/>
  </measures>
  <dashboards>
    <dashboard label="GitHub Repository">
      <view measure="GitHubRepository" view="Name"/>
      <view measure="GitHubRepository" view="Owner"/>
      <view measure="GitHubRepository" view="LastUpdate"/>
      <view measure="GitHubRepository" view="Language"/>
      <view measure="GitHubRepository" view="Stars"/>
      <view measure="GitHubRepository" view="Suscribers"/>
      <view measure="GitHubRepository" view="Forks"/>
      <view measure="GitHubRepository" view="Branches"/>
      <view measure="GitHubCommit" view="Commits History"/>
      <view measure="GitHubCommit" view="Repository Activity"/>
      <view measure="GitHubCommit" view="Commits By Users"/>
      <view measure="GitHubCommit" view="Commits"/>
    </dashboard>
    <dashboard label="Issus">
      <view measure="GitHubIssueStat" view="Issues"/>
      <view measure="GitHubIssueStat" view="Average Open Duration"/>
      <view measure="GitHubIssue" view="GitHub Issues"/>
    </dashboard>
  </dashboards>
</Application>

```

Application Packaging

A Measurement Application is packaged as a zip containing:

- The ApplicationMetaData.xml file
- Unzipped measures which compose the application

▼ GitHubRepository	5 éléments	Dossier	13:32
▼ GitHubCommit	3 éléments	Dossier	13:15
▶ lib	10 éléments	Dossier	24 janv.
📄 GitHubCommit-0.0.1-SNAPSHOT.jar	5,0 ko	Archive	12:57
📄 MeasureMetadata.xml	7,8 ko	Balisage	13:15
▶ GitHubIssue	3 éléments	Dossier	12:57
▶ GitHubIssueStat	3 éléments	Dossier	12:57
▶ GitHubRepository	3 éléments	Dossier	13:36
📄 ApplicationMetadata.xml	1,5 ko	Balisage	13:32

Analysis Tool Development

The main objective of the analysis platform is to implement analytics algorithms, to correlate the different phases of software development and perform the tracking of metrics and their value. The platform also connects and assure interoperability among the tools and define actions for improvement and possible countermeasures.

Integration Mechanism of Analysis Tool into

In order to ensure the integration of various kind of analysis tool into the measure platform, the Analysis component provide an integration mechanism of external tool to register it into the measure platform, to access to measurement data, to received notification form the platform and finally to provide analysis results. This integration is based on a set of REST services used to manage communication between analysis Tools and the Platform.

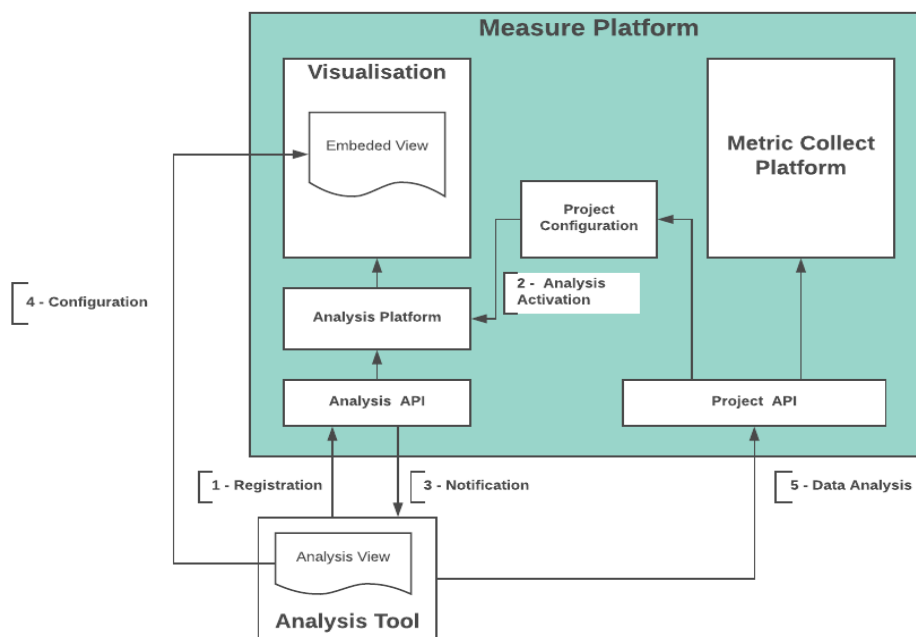


Figure 1 : Registration process of an external analysis tool into the platform

- **Registration:** At start-up of the Analysis Tool, it must register itself to the platform using the Registration service. This would allow the project to activate the analysis tools.

Registration Rest Service:

PUT	/api/analysis/register
Input Data (json)	<pre>{ "configurationURL": "string", "description": "string", }</pre>


```

    "name": "string"
  }

```

- Wait for Notifications:** The Analysis Tool must listen to notifications from the platform in order to know when a project requests the usage of the analysis tool. The notification (Alert) system is based on pooling system. The Analysis tool pool the platform periodically using the alert service to received notifications. The Platform send several kinds of notifications listed below:

Alert Type	Description	Properties
ANALYSIS_ENABLE	A Project sends an activation request for the Analysis Tool. It's not required for analysis tool to subscribe to this alert, the subscription is automatic.	<ul style="list-style-type: none"> ANALYSISID: Id of the instance of analysis associated with this request on platform side
ANALYSIS_DESABLE	A Project indicate that the analysis service is not required anymore. It's not required for analysis tool to subscribe to this alert, the subscription is automatic.	<ul style="list-style-type: none"> ANALYSISID: Id of the instance of analysis associated with this request on platform side.
MEASURE_ADDED	A new Measure is added the the project	<ul style="list-style-type: none"> MEASUREID: Id of the Measure
MEASURE_REMOVED	A Measure is removed from the project	<ul style="list-style-type: none"> MEASUREID: Id of the Measure
MEASURE_SCHEDULED	A Measure is not collected periodically for the project	<ul style="list-style-type: none"> MEASUREID: Id of the Measure
MEASURE_UNCHEDULED	A Measure is not collected anymore by the project	<ul style="list-style-type: none"> MEASUREID: Id of the Measure

By default, all register project subscribe automatically to ANALYSIS_ENABLE and ANALYSIS_DESABLE Notifications.

Retrieve Platform Alerts REST Service: This service retrieves the alerts form the platform for a specific analysis tool

GET	/api/analysis/alert/list/{AnalysisToolName}
Parameter	AnalysisToolName : Name of the Analysis Tool (provided in registration service)
Output Data (json)	<pre> { "alerts": [{ "alertType": "string", "projectId": 0, "properties": [{ "property": "string", "value": "string" }] }] }, </pre>

	<pre>"from": "2018-03-13T12:16:33.164Z" }</pre>
--	-------------------------------------------------

- **Configure Analysis:** When a project activates an analysis tool, the analysis tool must configure it for the project and provide URLs for the project-specific configuration page, the project main view and optionally the dashboard cards.

Configuration REST Service

Warning: The analysis configuration input data required a projectAnalysisId. This id is provided by the platform as properties of the ANALYSIS_ENABLE and ANALYSIS_DISABLE notification message.

PUT	/api/analysis/configure
Input Data (json)	<pre>{ "cards": [{ "cardUrl": "string", "label": "string", "preferedHeight": 0, "preferedWidth": 0 }], "configurationUrl": "string", "projectAnalysisId": 0, "viewUrl": "string" }</pre>

- **Analyse the Project:** When configured, the analysis tool can start its analysis work for the specific project. In order to perform this work, the analysis tool can explore the project configuration using the various services provided by the Measure platform. It can also configure new Alerts to receive notifications when the project configuration has changed.

Alert Subscription REST Service: This service allows an analysis tool to subscribe to a new alert related to a specific project

PUT	PUT /api/analysis/alert/subscribe
Input Data (json)	<pre>{ "analysisTool": "string", "eventType": "ANALYSIS_ENABLE", "projectId": 0, "properties": [{ "property": "string", "value": "string" }] }</pre>

Alert Unsubscribe REST Service : This service allows the analysis tool to unsubscribe to an alert.

PUT	PUT /api/analysis/alert/unsubscribe
Input Data (json)	<pre>{ "analysisTool": "string", "eventType": "ANALYSIS_ENABLE", "projectId": 0, }</pre>

```

"properties": [
  {
    "property": "string",
    "value": "string"
  }
]
}

```

User Interface integration using Embedded View

In order to integrate deeply the analysis tool to the Measure Platform, the analysis tools have to provide some web pages which will be embedded to the platform web application. Each of these views are defined on the platform side by a specific URL. For project specific views, this URL is different for each project. You will see below the list of view which can be provided by the analysis tool and embedded into the Measure Platform.

- **Global Configuration Page (optional):** If the analysis tool requires a way to provide some configuration interface which will be shared by all project, it can provide a global configuration web page.
- **Project Specific Analysis Configuration page:** Configuration page which are specific for each project. This page is embedded into project configuration page and allow to configure the analysis service provided by the external analysis tool.

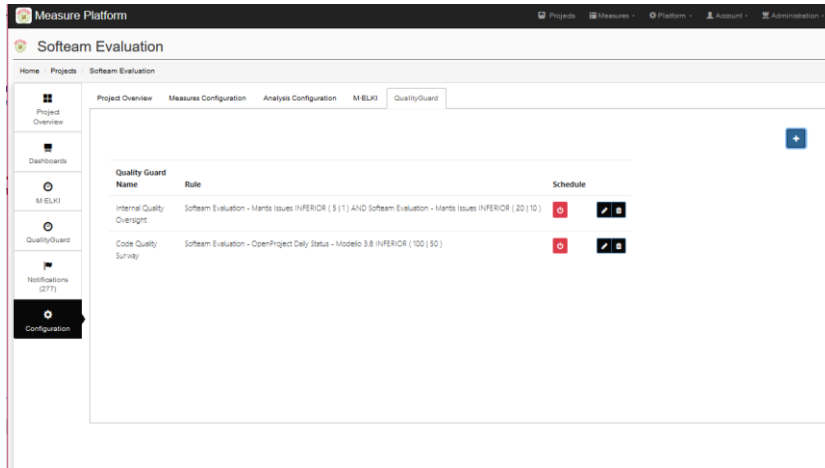


Figure 2 : Analysis tool configuration page of Quality Guard Analysis tool

- **Analysis Tool Main View:** Main view of the analysis tool which are specific for each project. In this view, the analysis service.

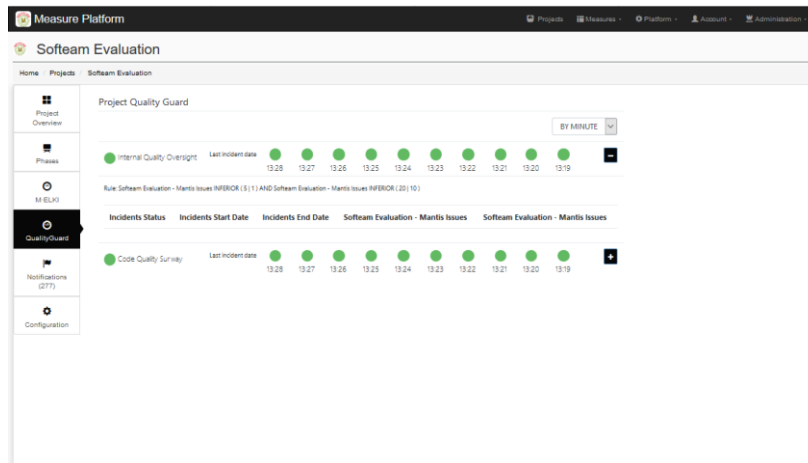


Figure 3 : Main view of the Quality Guard Analysis Tool

- **Dashboard Card:** Optional small view which can be integrated to projects dashboards in order to provide some key information to project managers related to the service provided by the analysis tool.

Platform Querying Services

The platform provides several other services which can be used by the analysis tools to retrieve platform and project configurations data, information related to measures and measurements and more.

The list of available services can be consulted via Swagger directly on deployed Measure platform. To access this specification, one must be connected as Administrator to the platform. The complete API specification is available on Administration > API menu

Some example of available HTTP services:

- GET /api/measure/findall : List all measures
- GET /api/measure/{id} : information related to a specific measure
- GET /api/measure-properties/{id} : List of scope properties associated with one measure
- GET /api/projects : List all projects
- GET /api/projects/{id} : Information related to a specific project
- GET /api/phases/byproject/{id} : Get phases of a specific project
- GET /api/phases/{id} : Information of a specific phase
- GET /api/measure-instances : List of all measure instances
- GET /api/measure-instances/{id} : Information of a specific measure instance
- GET /api/project-measure-instances/{id} : List of measure instances of a specified project
- GET /api/measure-instance/scheduling/execute/{id} : Execute a specific measure
- GET /api/measure-instance/scheduling/start/{id} : Activate scheduling of a specific measure
- GET /api/measure-instance/scheduling/stop/{id} : Deactivate scheduling of a specific measure

Run Measure Platform From Source

The measure platform is an open source product. The current section of the documentation present how to run the platform in developer mode from source.

Prerequisites

The Measure Platform can be executed both on Linux or Windows systems. For that, the platform requires the installation of: MySQL, Elasticsearch, Kibana and Java 1.8.

MySQL Installation

- Download MySQL Community Server 5.7 or above : <https://dev.mysql.com/downloads/mysql/>
- Install MySQL using the following instruction : <https://dev.mysql.com/doc/refman/5.7/en/installing.html>
- Create a new database named "measureplatform".

Elasticsearch Installation

- Download Elasticsearch 5.6 (as zip): <https://www.elastic.co/downloads/elasticsearch>
- Unzip the application in your tool directory.

Kibana Installation

- Download Kibana v 5.6 (as zip): <https://www.elastic.co/downloads/kibana>
- Unzip the application in your tool directory.

Java 1.8 Installation

- Download and install the jdk8 in your environment:
<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

Eclipse IDE

- Download and install the last version of Eclipse IDE for Java EE Developers:
<https://www.eclipse.org/downloads/eclipse-packages/>

Retrieve the MeasurePlatform Source Code

The Measure Platform source code is hosted on GitHub. To retrieve it, you can:

- Download it as zip file : <https://github.com/ITEA3-Measure/MeasurePlatform>
- Clone the Git repository
 - Install git: <https://git-scm.com/downloads>
 - Clone the repository: `git clone https://github.com/ITEA3-Measure/MeasurePlatform.git`

You can now import the Measure Platform as a new Maven project in Eclipse.

Start the Application in developers Mode

1. **Start MySQL**
 2. **Start Elasticsearch** : `./elasticsearch-5.4.0/bin/elasticsearch`
 3. **Start Kibana**: `./kibana-5.4.0/bin/kibana`
 4. **Start the Measure platform:**
- **From Eclipse IDE:** Select the "MeasurePlatformApp.java" file and Right Click > Run as > Java Application