Tile-in-ONE
An integrated framework for the data quality assessment and database management for the ATLAS Tile Calorimeter

R. Cunha 1, C. Solans 2, A. Sivolella 1, F. Ferreira 1 and C. Maidantchik 1

1 Universidade Federal do Rio de Janeiro, Av. Athos da Silveira Ramos, 149 - Cidade Universitária Rio de Janeiro, Brazil
2 CERN, Geneve 23, CH-1211 Switzerland

Abstract.
In order to ensure the proper operation of the ATLAS Tile Calorimeter and assess the quality of data, many tasks are performed by means of several tools which have been developed independently. The features are displayed into standard dashboards, dedicated to each working group, covering different areas, such as Data Quality and Calibration.

1. Introduction
The ATLAS detector [1] construction, commissioning, operation and maintenance have been conducted by an international collaboration, of students and researchers from 177 institutions [1]. The ATLAS Tile calorimeter [2] detects hadronic particles and jets, measuring their energy and position. The members of the Tile calorimeter system are responsible for the operation and the quality of the data by means of several tools and web interfaces that have been developed throughout the different phases of the experiment. Most of these tools were developed independently, without following the same guidelines, sometimes lacking basic software infrastructure and implemented without a global perspective of the calorimeter. Therefore, in many cases, they overlap in the objectives and resources with each other. Besides, in order to perform a single task, the collaborator must navigate through different tools, databases and software, which is time consuming and susceptible to mistakes. It is therefore necessary to have an infrastructure which allows the implementation of any functionality without having to duplicate the effort while being possible to integrate with an overall view of the detector status. Tile-in-ONE provides a unified interface, which integrates all the web systems and tools used by the Tile calorimeter, with a standard development technology and access to common services.

2. The Architecture
The framework is composed basically of three elements: the system core, the services and the plug-ins. The system core is the basic code that loads the configuration, manages user settings and loads the system components at runtime. It is also responsible for displaying the basic page...
structure and its content. The relationship between the system entities is shown in the Figure 1. Both dashboards and plug-ins are handled by a base API in order to keep the system modular. Furthermore, a user space is created on each session where the accesses to the same resources are streamlined to avoid overloading. This is achieved by the usage of services that are independent from the core.

Services provide common access to the resources. They have two purposes: to facilitate the development of new plug-ins, by reducing the amount of code necessary to perform the same operation, and minimize the impact of multiple access to the resources. There are three types of services in Tile-in-ONE: the connectors, which provide connectivity to some database or remote machine, the libraries which provide functionalities for user interaction or page stylization and finally interfaces to external software. For example, Tile-in-ONE provides a connector to a computer containing files with results from Data Quality (DQ) automatic validation of calibration runs. Another one allows access to the ATLAS conditions database and there is also a connector for the database containing the physics runs data. A set of libraries provides basic functionality for general user interaction and another set provide graphical designs such as the Tile Barrels image. Finally, there is a service through which it is possible to use features from TUCS, which is a software that allows the experts to perform general analysis and monitoring of calibration data.

A set of components, which together provide a well defined functionality, corresponds to a plug-in. Depending on its output, a component can be defined as a function or as a gadget. A gadget should always return a content to be displayed. On the other hand, a function returns a json [3] string containing information to be used by any other system entity. This relationship is represented in the Figure 1, in which the two classes Function and Gadget inherit from the class PluginComponent. A dashboard is composed by a set of gadgets, which are needed for a specific working group or user to accomplish their tasks. So, in the same figure, the class DashboardGadget defines which gadgets belong to each dashboard.

![Figure 1. The system core database](image)

The communication between the system core, the plug-ins and the resources is operated by services, through a well defined API. Thus, the services receive HTTP requests made by other system elements, and respond with the required resource output, as shown in Figure 2.

3. The Web Interface
The user interface is made out of several dashboards. Some of them are implemented to support a specific interest group.

The DQ Assessment dashboard provides the tools to judge the quality of the data. The Tile calorimeter is represented as a set of four concentric rings, each divided into 64 wedges or
modules, as presented on Figure 3. The summary status of the whole module is given by a color code. This status is generated by an automatic check done by an external software. Several layers allow the user to compare the status of each module across different runs. By selecting a module the user can access detailed information in the form of plots and tables. For example, Figure 3 shows the histograms and the channel per channel status. The Data Quality validation can be performed by entering decisions, comments or even a validation status, into a report through other specific plug-ins.

The Calibration dashboard allows experts to perform stability studies and comparisons of the constants throughout the different calibration stages. A single plug-in provides the functionality to select date ranges, calibration type and different style settings.

The Hot Points dashboard displays the plug-ins that provide a general overview of the detector, such as the one that shows the channels that are masked, and therefore not used for physics analysis. Another Hot Point example shows the gain variation over time per read-out cell.

In addition, each user has its own dashboard that is created the first time he logs in to the system. The user can add any of the available plug-ins through an interface listing the plug-in names, their description and developer name as shown on Figure 4. When the user clicks the Add button, he can see which gadgets are offered by the selected plug-in and can add it to any of the dashboards controlled by him.

4. The Plug-ins development

Tile-in-ONE also allows the possibility to add external plug-ins to the framework. The user has to register the plug-in in the system by informing the name, description, the underlying programming language and the category, through the form shown in Figure 5. Then, gadgets and functions can be registered by filling the form shown in Figure 6, where it is important to insert, beyond the name, title and description, the method in which the referred gadget will be defined, the type and the dimensions in which it will be displayed in the interface.

For each available programming language there is a different way to write the code, so that the system can recognize and load it properly. If the plug-in is written in Python, a simple class is needed, as shown in the example script below. The constructor of this class parses parameters in the form of json strings. Only methods properly registered in the system will be called, such as the OutputGadget as shown in the code, which defines the previously registered gadget output. The special function callService allows any class to have access to any service by specifying the service name and the required parameters.
Figure 3. The DQ Assessment Dashboard

Figure 4. The plug-ins management for the user
class MyPlugin(OneBasePlugin):
    def __init__(self, params):
        self.params=json.loads(params)
        self.bar_name=self.params['bar_name']
        self.ch_number=self.params['ch_number']
        self.mod_number=self.params['mod_number']

    def Request(self):
        module=self.bar_name+self.mod_number
        params={'module':module,'channel':self.ch_number}
        cur_cell=callService("TilecalMapping","GetCellByChannel",params)
        cur_pmt=callService("TilecalMapping","GetPMTByChannel",params)
        result_list=[cur_pmt[0],cur_cell[0]]
        return result_list

    def OutputGadget(self):
        data=self.Request()
        html += "<h3>Result</h3>"
        html+="<p>PMT: " + data[0] + "</p>"
        html+="<p>Cell: " + data[1] + "</p>"
        return html

Figure 5. Registering a plug-in

Figure 6. Registering a plug-in gadget

5. Conclusions
Tile-in-One provides the infrastructure to integrate the Tile calorimeter web systems. It makes the maintenance of existing tools and the development of new ones easier and more efficient. Currently, the services cover the current needs for Data Quality and Calibration tasks. In order to extend the functionality to other areas, such as DCS [4], other services should be offered.

The better the framework promotes plug-ins reuse of the existing code, the higher is the probability that tools will get easier to implement. This also represents a benefit for the user who can profit from a more complete tool.

References
[1] The ATLAS Collaboration 2008 Journal of Instrumentation 3 S08003
[2] The ATLAS Collaboration 2010 Eur. Phys. J. C 70 1193-1236. 64 p
[3] 2013 Json URL http://www.json.org
[4] ICALEPCS2011, Grenoble, France 2011 Detector control system of the ATLAS Tile Calorimeter