MapLayout2D PLUGGABILITY

Guida rivolta agli svluppatori. Questa guida permette di rendere disponibili, qual'ora il plugin fosse presente nell'ambiente deployato, delle funzionalità aggiuntive che possono essere configurate, gwMap per gwMap, nell'apposita sezione del webadmin (vedi apposita guida)

Un primo meccanismo di pluggabilità dell'OlMapWidget (**LEGACY**), che verrà descritto in fondo, è presente fin dalla versione 4.0.0. Dalla versione 4.5.0 e successive è stato introdotto un ulteriore meccanismo di plugin, sulla falsa riga del 3D. Di base, nell'xml del gwMap si possono configurare (rendere o meno visibili) le seguenti entità:

- ButtonGroup (gruppo di bottoni), in genere omogenei per ambito d'azione ('selection', 'navigation', 'info', etc..)
- Tab, fra quelli disponibili di default, e quelli introdotti da eventuali plugin

Tipicamente un plugin di geoweb per il MapLayout2D può:

- aggiungere un ButtonGroup (gruppo di bottoni) alla toolbar ed aggiungerci contestualmente uno o più mapCommand (comandi-mappa)
- aggiungere una tipologia di TAB a quelle disponibili
- registrare un marker gestito con varie convenzioni, per l'esecuzione di codice in hook-point, utili a predisporre funzionalità usate nel plugin

Queste operazioni partono sempre lato java, tipicamente in un [pluginName]initializerService.class, dentro un metodo annotato @PostCostruct, eseguito all'avvio del framework.

postConstruct.java

```
@PostConstruct
    public void init(){
        try {

//registering resurces
gwRegistry.addJsResource("debug/customDojo/pluginCommands.js");

            gwRegistry.addCssResource("css/furniture/gwPlugin.css");

    //adding mapCommand button in a buttonGroup
            ArrayList<String> twoDSelectionButtonList = new
ArrayList<String>();
            twoDSelectionButtonList.add("CustomPluginCommand");
            registerTwoDMainToolbarButtons(twoDSelectionButtonList,
"pluginCommandSection");

            //in aim to evaluate pluginLayout2DPlugin.jsp (es: tooltip
handling)
            gwRegistry.registerTwoDPlugin("plugin");

//adding custom tab
            List<String> tabList = new ArrayList<String>();
```

```
            tabList.add("pluginTab");
            gwRegistry.registerTwoDTabList(tabList);

        }catch(java.lang.Throwable e){
            log.error(e.getMessage(), e);
            throw new RuntimeException(e);
        }
    }
```

AGGIUNGERE BUTTON-GROUP E RELATIVI MAP-COMMAND ALLA TOOLBAR

Nel [pluginName]initializerService.java registrare ButtonGroup e commandList:

ArrayList<String> twoDSelectionButtonList = new ArrayList<String>();

```
twoDSelectionButtonList.add("CustomPluginCommand");
registerTwoDMainToolbarButtons(twoDSelectionButtonList,
"pluginCommandSection");
```

CustomPluginCommand è tipicamente definito in un file javascript “debug/customDojo/pluginCommands.js” , registrato sempre nell’ [pluginName]initializerService. java, con eventualmente le sue risorse css con i soliti meccanismi generali dei plugin in geoweb. Esempio pluginCommands.js:

pluginCommands.js

```
require([
    'dojo/topic',
    'customDojo/map/OLMapWidget',
gwContextPath+'/resources/'+gwRevision+'/debug/customDojo/map/plugin/GwCustomPlugin.js' //this way it works even with useCompressedImports=true
], function(topic, OLMapWidget, Furniture){
    //do nothing, simply require plugin to add to OLMapWidget
});

/**
 * F U R N I T U R E   C O M M A N D
 */
require(
    ['dojo/_base/declare', 'dojo/ready',
gwContextPath+'/resources/'+gwRevision+'/'+importPath+'customDojo/map/mapCommands.js'],
    function(declare, ready){
        ready(function(){

            console.log('pluginCommands);
```

```
            var doDeclare = function(){

                console.log('pluginCommands - doDeclare()');

                //Dummy mapCommand
                declare('CustomPluginCommand', MapCommand, {
                    name: 'CustomPluginCommand',
                    iconClass: 'iconDone',
                    exclusive: false,
                    category: 'edit-custom-plugin,
                    buttonType:  'CommandButton',
                    execute: function(/*Object*/ params){
                        console.log('CustomPluginCommand
\''+this.name+'\'- execute()');
                        this.inherited(arguments);

                        var that = this;
                        var olMapWidget = this._map.olMapWidget;

                        //TODO
                    }
                });
            }

            if(window['MapCommand']!=null){
                doDeclare();
            }else{
                var handle = window.setInterval(function(){
                    if(window['MapCommand']!=null){
                        window.clearInterval(handle);
                        doDeclare();
                    }
                }, 0);
            }
        });
    }
);
```

AGGIUNGERE TAB A QUELLI DISPONIBILI

Nel [pluginName]initializerService.class registrar ButtonGroup e commandList:

example.java

```
List<String> tabList = new ArrayList<String>();
    tabList.add("pluginTab");
```

```
    gwRegistry.registerTwoDTabList(tabList);
```

La lista di tutti i tab registrati viene elaborata nel jsp della scheda mappa cosi:

```
<c:forEach items="${tabList}" var="tab">
<jsp:include
page="/layout2D/tab/${tab.type}.html?tabName=${tab.name}&tabLabel=${tab.labe
l}&tabIcon=${tab.icon}&tabIcon16=${tab.icon16}&tabIcon32=${tab.icon32}">
</jsp:include>
</c:forEach>
```

Deve quindi essere previsto nel plugin un qualche XController.java, come da esempio:

pluginTabMAV.java

```
@RequestMapping("layout2D/tab/pluginTab")
    public ModelAndView pluginTabMAV(
        @RequestParam(value="tabName", required=false) String tabName,
        @RequestParam(value="tabLabel", required=false) String
tabLabel,
        @RequestParam(value="tabIcon", required=false) String tabIcon,
        @RequestParam(value="tabIcon16", required=false) String
tabIcon16,
        @RequestParam(value="tabIcon32", required=false) String
tabIcon32,
        Locale locale
    ) throws JsonProcessingException{
        ModelAndView modelAndView = new
ModelAndView("layout2D/tab/pluginTab");

        GwLogMessageBuilder gwLogMessageBuilder = new
GwLogMessageBuilder()
            .info()
            .forJava()
            .setLogger(log)
            .addMessage("PluginController - pluginTabMAV ()")
            .addIncomingParamMapEntry("tabName", "tabName")
            .addIncomingParamMapEntry("tabLabel", "tabLabel")
            .addIncomingParamMapEntry("tabIcon", "tabIcon")
            .addIncomingParamMapEntry("tabIcon16", "tabIcon16")
            .addIncomingParamMapEntry("tabIcon32", "tabIcon32");

        try{

            modelAndView.addObject("tabName", tabName);
            modelAndView.addObject("tabLabel",
gwmDictionaryServiceImpl.selectValueByKeyAndLang(tabLabel, locale));
            modelAndView.addObject("tabIcon", tabIcon);
```

```
            modelAndView.addObject("tabIcon16", tabIcon16);
            modelAndView.addObject("tabIcon32", tabIcon32);

        }catch(Exception e){
            if(gwLogMessageBuilder!=null)
                gwLogMessageBuilder.error().setThrowable(e);
            throw e; //throwing e is optional
        }finally{ //always executed
            if(gwLogMessageBuilder!=null)
                gwLogService.logMessage(gwLogMessageBuilder.build());
        }

        return modelAndView;
    }
```

E relativo jsp, come da esempio (qui il contenuto del tab è gestito da un apposito componente customDojo):

pluginTab.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions"%>

<html>

    <head>

        <c:set var="context" value="<%=request.getContextPath()%>" />

        <spring:message code="map.plugin.pluginLabel" var="pluginLabel"
javaScriptEscape="true"/>


        <script type="dojo/require">
            registry: 'dijit/registry',
            topic: 'dojo/topic',
            query: 'dojo/query',

            Furniture:
'${context}/resources/'+gwRevision+'/debug/customDojo/map/plugin/GwPlug
in.js'
        </script>

    </head>

    <body class="claro">
```

```
        <div
            data-dojo-type="customDojo.map.plugin.GwPlugin"
            data-dojo-props="
                sectionType: 'gw_plugin',
                tabName: '${tabName}',
                tabIcon: '${tabIcon}',
                tabIcon16: '${tabIcon16}',
                tabIcon32: '${tabIcon32}'
            "
        >
            <script type="dojo/method">

                this.title = '${not empty tabLabel ?
fn:escapeXml(tabLabel) : pluginLabel}';

//optional stuff below

                <c:if test="${not empty tabToAttributeListMapJSON}">
                    this.tabToAttributeListMap =
${tabToAttributeListMapJSON};
                </c:if>

                <c:if test="${not empty commandNameMapJSON}">
                    this.commandNameMap = ${commandNameMapJSON};
                </c:if>

                <c:if test="${not empty param.mapLayoutId}">
                    this.mapLayout = ${param.mapLayoutId};
                    this.mapLayoutId = '${param.mapLayoutId}';
                </c:if>

                <c:if test="${not empty param.oLMapWidgetId}">
                    this.oLMapWidget = ${param.oLMapWidgetId};
                    this.oLMapWidgetId = '${param.oLMapWidgetId}';
                </c:if>

            </script>
        </div>
    </body>

</html>
```

A determinare il contenuto del tab sarà di fatto customDojo/map/plugin/GwPlugin.js

REGISTRARE PLUGIN MARKER PER HOOKPOINT

Nel [pluginName]initializerService.class registrar ButtonGroup e commandList:

```
//in aim to evaluate furnitureLayout2DPlugin.jsp (tooltip handling)
gwRegistry.registerTwoDPlugin("furniture");
```

Per ogni plugin-marker, all'apertura del MapLayout2D, vengono in automatico importati i javascript, con nome e path seguenti il pattern:

```
<c:forEach items="${pluginList}" var="plugin">
    ${plugin}Commands:
'${context}/resources/'+gwRevision+'/debug/customDojo/${plugin}Commands.js',
</c:forEach>
```

Questo è un meccanismo di import js parallelo al registrare le risorse sul [pluginName]initializerService.

Inoltre, per ogni plugin-marker, viene eseguito uno script "dojo/method" relativo al widget customDojo/map/MapLayout

```
<c:forEach items="${pluginList}" var="plugin">
    <script type="dojo/method">
    <jsp:include page="/${plugin}Layout2DPlugin.html" ></jsp:include>
    </script>
</c:forEach>
```

Qui si riesce, predisponendo un apposito controller ad eseguire codice javascript, ed altre possibili inizializzazioni JSTL, presenti su un .jsp fornito dal plugin stesso.

Controller di supporto

furnitureLayout2DPlugin.java

```
@RequestMapping(value = "furniture"+"Layout2DPlugin")
    public ModelAndView furnitureLayout2DPlugin(
        Locale locale
    ){
        ModelAndView modelAndView = new
ModelAndView("furnitureLayout2DPlugin");
        return modelAndView;
    }

File .jsp, coordinato con il controller (jsp/pluginLayout2DPlugin.jsp)


<%-- This code will to be executed inside a customDojo/map/MapLayout
<script type="dojo/method"></script> --%>

<spring:message code="map.plugin.myLabel " var="myLabel"
javaScriptEscape="true"/>

var doStuff = function(){
    if(CustomPluginCommand)
CustomPluginCommand.prototype._my_prop_label =
```

```
'${fn:escapeXml(myLabel)}';
};

if(window['FurniturePluginCommand']!=null){
    doStuff();
}else{
    var handle = window.setInterval(function(){
        if(window[CustomPluginCommand]!=null){
            window.clearInterval(handle);
            doStuff();
        }
    }, 0);
}

window['plugin_globallyAvailableLabel] = '${myLabel}';
```

Qui, per esempio, assegnano un label localizzata al CustomPluginCommand, che potrà essere usata al suo interno. La stessa label è anche messa resa disponibile nel namespace globale Rimane il fatto che siamo nel dojo/method del MapLayout e possiamo eseguire tutta una serie di cose, in base alle necessità

MECCANISMO DI PLUGIN LEGACY

Si tratta di un meccanismo precedente alla revisione del componente MapLayout2D. E' ancora pienamente supportato, e lavora in parallelo a quello precedentemente descritto. E' reso pluggabile il componente customDojo.map.OLMapWidget. Funziona in maniera molto similare al meccanismo di plugin della griglia dojo dojox.grid.EnhancedGrid (al quale è stata solo tolta la gestione specifica dei meccanismi della griglia). Ad essere stata mantenuta è la possibilità di

- avere punti di intervento multipli dove opera il widget:
  - onPreInit, prima del postCreate del OLMapWidget (solo con parametro preInit==true)
  - onPostInit, dopo del postCreate del OLMapWidget (solo con parametro preInit==false)
  - onStartUp, allo startup OLMapWidget, è quello dove di norma verrà fatto il grosso delle modifiche apportate dal plugin
    - aggiungendo tasti
    - sovrascrivendo funzioni tramite prototype
    - etc..
- la possibilità di collegarsi, alla creazione del plugin, a specifici eventi o invocazioni di metodo lanciati dall'oggetto pluggato (OLMapWidget), e fare azioni ad hoc
- la possibilità di registrare plugin che a loro volta dipendono da altri plugin, e risolvere coerentemente le dipendenze:

```
    olMapWidget.registerPlugin(
customDojo.map.plugins.PluginExample, // full class name of a plugin
{"preInit": false, "dependency": ["aaa_bbb"]} // properties
);
// properties:
//        - "preInit": boolean, whether a plugin should be created before
```

```
EnhancedGrid.postCreate(),
//           false by default(plugins are created after
EnhancedGrid.postCreate()).
//        - "dependency": array or string, plugin(s) indicated by
"dependency" will be created before the current one.
//           Note: recursive cycle dependencies are not supported e.g.
following dependency is invalid:
//           pluginA -> pluginB -> pluginA
```

I plugin vengono registrati in automatico all OLMapWidget alla sola richiesta fatta tramite require. Anche su EnhancedGrid i plugin si autoregistrano alla sola richiesta con require, ma in più vanno esplicitamente abilitati popolando adeguatamente la variabile plugins di ogni nuova istanza di EnhancedGrid. (ex: plugins: {indirextSelection: true} Cio' puo essere fatto in automatico alla registrazione semplicemente sovrascrivendo registerPlugin()

```
        registerPlugin: function(clazz, props){
            //added part
            //autoaddind registered plugin
            if(!this.plugins) this.plugins = {};
            this.plugins[clazz.prototype.name] = true;

            _PluginManager.registerPlugin(clazz, props);
        }
```

Esempio Implementazione PLUGIN

Mettiamo di voler creare ex novo, il plugin PluginExample, definito magari su un plugin di geoweb.

PluginExample.js

```
define([
    "dojo/_base/declare",
    "dojo/_base/lang",
    "customDojo/map/OLMapWidget",
    "../_Plugin"
], function(declare, lang, OLMapWidget, _Plugin){

    var PluginExample = declare("customDojo.map.plugins.PluginExample",
_Plugin, {
        // summary:

        // name: String
        //        plugin name
        name: "pluginExample",

        constructor: function(olMapWidget, args){
            // summary:
            //        See constructor of dojox.grid.enhanced._Plugin.
            this.olMapWidget = olMapWidget;
```

```
            this.nls = nls;

            args = this.args = lang.isObject(args) ? args : {};
            //eventually use args
            // ...

            //eventually connect to olMapWidget function
            //this.connect(olMapWidget, "olMapWidgetFunctionName",
function(obj){
            //    ...
            //});
        },
        onPreInit: function(){
            //add if needed
        },

        onPostInit: function(){
            //add if needed
        },

        onStartUp: function(){
            //add if needed
            //TODO
        }
    });
    //registering plugin
    var props = null; //Object, optional
    OLMapWidget.registerPlugin(PluginExample, props);
    //ex:
    //olMapWidget.registerPlugin(customDojo.map.plugins.PluginExample
/*definition of a plugin*/, {"preInit": false, "dependency":
["aaa_bbb"]} /*properties*/)
    return PluginExample;
});
```