

JMap 6.5

Manuel du développeur

JMap[®]

Table des matières

Introduction	1
Concepts généraux	2
Géométries	2
Développement JMap Pro et JMap Server	11
Introduction	11
Préparation de l'environnement	11
Exemples	17
JMap Pro	18
Modèle de classes	18
Éléments cartographiques	20
Systèmes de coordonnées	24
Vues et gestionnaire de vues	27
Couches et gestionnaire de couches	33
Application JMap Pro	40
Communication client-serveur	41
Liste des composantes d'interface graphique	44
Outils JMap	46
Extensions de JMap Pro	52
Introduction	52
Programmation des extensions de JMap Pro	52
Programmation des requêtes d'extension	55
Intégration dans l'interface graphique	56
Déploiement des extensions de JMap Pro	63
Signature des extensions	65
Intégration de JMap Pro avec d'autres applications	66
Intégration avec applications client	66
Intégration avec applications Web	70
Références	73
Paramètres de démarrage de JMap Pro	73
JMap Server	76
Préparation de l'environnement	76
Extensions de JMap Server	78
Introduction	78
Programmation des extensions serveur	79
Services de JMap Server	80
Déploiement des extensions serveur	87
Interfaces de configuration pour les extensions serveur	88
API Java de JMap 6.5	89
Générateur d'extensions de JMap Pro	89
Développement JMap Web (6.5)	92
Introduction	92

Information générale	92
API Public JMap Web	94
Le processus de démarrage de JMap Web	94
JMap Web Extensions	100
Introduction	100
Programmer des extensions JMap Web	100
Envoyer des requêtes au serveur et actions personnalisées	110
Les actions par défaut de JMap Web	115
Déployer des extensions JMap Web	129
Intégrer JMap Web dans votre propre application	130
Développement JMap Web (7.0 Preview)	134
Introduction	134
Information générale	134
API Public JMap Web	136
Le processus de démarrage de JMap Web	136
JMap Web Extensions	141
Introduction	141
Programmer des extensions JMap Web	141
Envoyer des requêtes au serveur et actions personnalisées	151
Les actions par défaut de JMap Web	156
Déployer des extensions JMap Web	170
Intégrer JMap Web dans votre propre application	171
Nous joindre	175

Introduction

Le SDK de JMap est composé de documents, d'exemples de code source et d'outils pour aider les développeurs à personnaliser les applications JMap et à en étendre le fonctionnement.

JMap est composé de plusieurs parties distinctes. Les principales sont JMap Server, JMap Admin, JMap Pro, JMap Web et JMap Mobile. Chaque partie est basée sur un environnement technologique qui lui est propre et cela détermine la manière d'effectuer la programmation.

Le tableau suivant décrit l'environnement technologique de chaque composante de JMap.

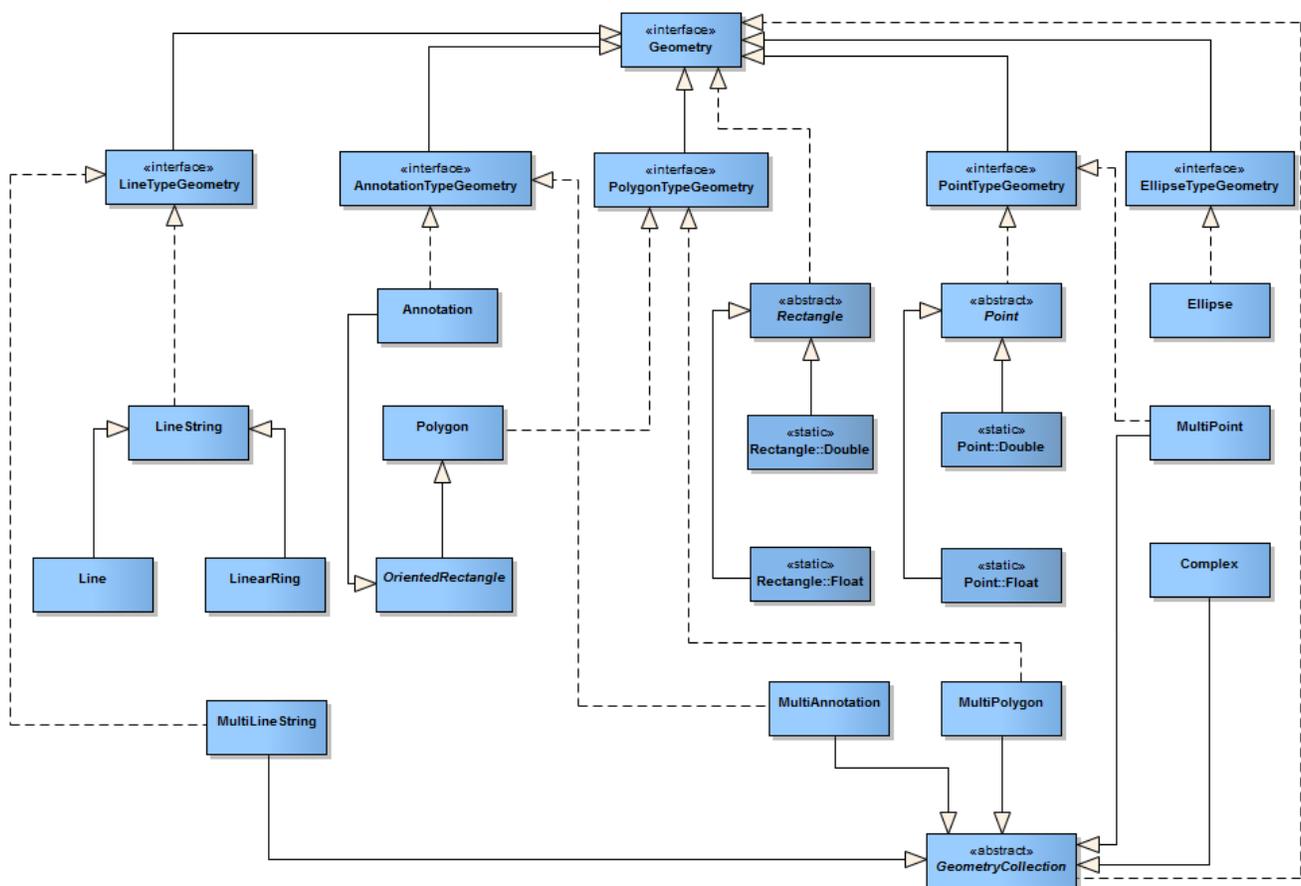
Composante	Technologies
JMap Server	Java
JMap Admin	Java Server Faces (JSF)
JMap Pro	Java
JMap Web	HTML 5, CSS, Javascript, JSON

Note : Il n'est actuellement pas possible de développer pour JMap Mobile à l'aide du SDK de JMap 6.5.

Concepts généraux

L'API des géométries de JMap est utilisée pour le développement dans JMap Pro et dans JMap Server.

Les géométries dans JMap sont à la base de tous les éléments vectoriels qui s'affichent sur la carte. Il s'agit de classes simples représentant les types géométriques de base tels que les points, les lignes et les surfaces. Le modèle de classes des géométries utilisé dans JMap est grandement inspiré du modèle de géométries publié par l' *Open Geospatial Consortium* . Les géométries ne contiennent ni attributs, ni identifiants, ni propriétés d'affichage mais seulement des coordonnées en 2 dimensions (x,y).



Modèle de classes simplifié des géométries de JMap

Dans JMap, les classes de géométries sont dans le package `com.kheops.jmap.spatial` . L'interface que toutes les géométries implémentent est `Geometry` .

Le type de géométrie de base dont toutes les géométries sont composées est le `Point` . Ce type contient uniquement une coordonnée (x,y). Cette classe est abstraite et il est donc nécessaire d'utiliser ses dérivées `Point.Float` (simple précision) et `Point.Double` (double précision).

```

// Creating a new point specifying coordinates
Point pt1 = new Point.Double(-73., 45.);

// Creating a new point by cloning an existing point
Point pt2 = (Point) pt1.clone();

// Changing the location of an existing point
pt1.setLocation(-74., 45);

```

Les principales classes de géométries sont les suivantes :

Classes de géométries	
<i>Point</i>	Coordonnée (x,y) dont les autres géométries sont composées.
<i>Curve</i>	Type abstrait dont sont dérivées toutes les géométries linéaires (<i>Line</i> , <i>LineString</i> , etc.).
<i>Line</i>	Ligne simple définie par 2 points.
<i>LineString</i>	Ligne à plusieurs parties. Composée de N noeuds et N-1 segments.
<i>LinearRing</i>	<i>LineString</i> dont le premier noeud et le dernier noeud sont égaux. Forme une boucle fermée.
<i>Surface</i>	Type abstrait dont sont dérivées toutes les géométries surfaciques (<i>Rectangle</i> , <i>Polygon</i> , etc.)..
<i>Polygon</i>	Polygone. Composé d'une <i>LinearRing</i> extérieure et de 0, 1 ou plusieurs <i>LinearRing</i> qui sont des trous.
<i>Rectangle</i>	Surface rectangulaire non orientée (côtés horizontaux et verticaux).
<i>OrientedRectangle</i>	Surface rectangulaire qui peut être orientée.
<i>Ellipse</i>	Ellipse composée d'un point central, d'un rayon a et d'un rayon b.
<i>Annotation</i>	Dérivée d' <i>OrientedRectangle</i> , la classe <i>Annotation</i> définit l'emprise d'un texte à afficher sur la carte.

Des versions composites des géométries existent afin de supporter les collections de géométries de même type. Ces classes sont: *MultiPoint*, *MultiCurve*, *MultiLineString*, *MultiSurface* et *MultiPolygon*. De plus, la classe *Complex* est un type spécial de collection de géométries de types variés.

Afin de gérer plus facilement tous ces types de géométries, 3 interfaces sont disponibles : *PointTypeGeometry* , *LineTypeGeometry* , *PolygonTypeGeometry* , *AnnotationTypeGeometry* et *EllipseTypeGeometry* . Ces interfaces regroupent respectivement tous les types de géométries, incluant les collections.

Précision des géométries

Les géométries dans JMap peuvent être en double précision ou en simple précision. La simple précision, combinée avec d'autres stratégies, est utilisée principalement dans un but de compression des données, afin d'optimiser les performances du système. Comme programmeur, vous devriez utiliser uniquement la double précision.

```
// Creating a new point with single precision
Point pt1 = new Point.Float(-73., 45.);

// Creating a new point with double precision
Point pt2 = new Point.Double(-73., 45.);
```

Opérations sur les géométries

Opérateurs spatiaux

Les opérateurs spatiaux permettent de faire des calculs géométriques sur les géométries. Ces calculs peuvent donner des résultats de 3 types différents:

- résultats numériques (p. e. calculer la distance entre 2 géométries)
- résultats booléens (ex.: tester si 2 géométries s'intersectent)
- résultats géométriques (calculer l'union de 2 géométries).

La classe *GeometryUtil* fournit des méthodes simples pour effectuer des calculs sur les géométries. Les méthodes les plus souvent utilisées sont présentées dans le tableau suivant:

Tests de relations spatiales (booléen)	
<i>contains(Geometry, Geometry, PrecisionModel)</i>	Teste si la première géométrie contient la deuxième géométrie. L'ordre des géométries est important.
<i>crosses(Geometry, Geometry, PrecisionModel)</i>	Teste si la première géométrie croise la deuxième géométrie. L'ordre des géométries est important.
<i>disjoint(Geometr, Geometry, PrecisionModel)</i>	Teste si la première géométrie est disjointe de la deuxième géométrie. L'ordre des géométries est important.

<i>intersects(Geometry, Geometry, PrecisionModel)</i>	Teste si les deux géométries s'intersectent. L'ordre des géométries n'a pas d'importance.
<i>overlaps(Geometry, Geometry, PrecisionModel)</i>	Teste si les deux géométries se chevauchent. L'ordre des géométries n'a pas d'importance.
<i>relate(Geometry, Geometry, char[], PrecisionModel)</i>	Teste si les deux géométries possèdent les relations spatiales définies dans la matrice en paramètre. L'ordre des géométries est important. Pour plus d'information, consultez http://en.wikipedia.org/wiki/DE-9IM .
<i>spatiallyEquals(Geometry, Geometry, PrecisionModel)</i>	Teste si les deux géométries sont égales spatialement (toutes les coordonnées sont identiques). L'ordre des géométries n'a pas d'importance.
<i>touches(Geometry, Geometry, PrecisionModel)</i>	Teste si les deux géométries se touchent. L'ordre des géométries n'a pas d'importance.
<i>within(Geometry, Geometry, PrecisionModel)</i>	Teste si la première géométrie est à l'intérieur de la deuxième géométrie. L'ordre des géométries est important.
Opérations géométriques ayant pour résultat des nouvelles géométries	
<i>buffer(Geometry, double, double, PrecisionModel)</i>	Retourne une géométrie qui représente la zone tampon autour de la géométrie spécifiée. La grandeur de la zone est spécifiée en paramètre.
<i>convexHull(Geometry, boolean, PrecisionModel)</i>	Retourne la géométrie convexe fabriquée à partir de la géométrie spécifiée.
<i>difference(Geometry, Geometry, PrecisionModel)</i>	Retourne la géométrie résultant de la différence entre la première géométrie et la deuxième géométrie. L'ordre des paramètres est important.
<i>diffSym(Geometry, Geometry, PrecisionModel)</i>	Retourne la géométrie résultant de la différence symétrique entre les deux géométries. L'ordre des paramètres n'a pas d'importance.

<code>intersection(Geometry, Geometry, PrecisionModel)</code>	Retourne la géométrie résultant de l'intersection entre les deux géométries. L'ordre des paramètres n'a pas d'importance.
<code>intersectionMultiple (Geometry[], PrecisionModel)</code>	Retourne la géométrie résultant de l'intersection entre toutes les géométries passées en paramètre. L'ordre des paramètres n'a pas d'importance.
<code>union(Geometry, Geometry, PrecisionModel)</code>	Retourne la géométrie résultant de l'union entre les deux géométries. L'ordre des paramètres n'a pas d'importance.
<code>unionMultiple(Geometry[], PrecisionModel)</code>	Retourne la géométrie résultant de l'union entre toutes les géométries passées en paramètre. L'ordre des paramètres n'a pas d'importance.
Calculs	
<code>distance(Geometry, Geometry, PrecisionModel)</code>	Calcule la plus courte distance séparant les deux géométries. L'ordre des paramètres n'a pas d'importance.

Modèles de précision

La classe `PrecisionModel` permet de définir la tolérance considérée pour effectuer les calculs utilisant les géométries. Le fait d'utiliser un modèle de précision bien adapté aux géométries permet d'obtenir des résultats plus précis lors des calculs.

Le modèle de précision varie en général selon le type d'unité des données (mètres, degrés, etc.). La classe `PrecisionModelFactory` permet d'obtenir une instance de `PrecisionModel` pour une unité donnée. Chaque instance de la classe `Layer` possède aussi sa propre instance de `PrecisionModel` qu'il est possible d'utiliser. Il est aussi possible de demander à JMap de déterminer le modèle de précision optimal pour une géométrie spécifique.

L'exemple suivant montre comment obtenir un modèle de précision de différentes manières.

```
// Obtain the optimal precision model for data of the current project
Project project = ...
PrecisionModel precisionModel1 = PrecisionModelFactory.getInstance(project.getMapUnit())

// Obtain the optimal precision model for unit Meter
PrecisionModel precisionModel2 = PrecisionModelFactory.METER;

// Obtain the optimal precision model of a layer
```

```

Layer layer = ...
PrecisionModel precisionModel3 = layer.getPrecisionModel();

// Obtain the optimal precision model for the specified geometry
Geometry geometry = ...
PrecisionModel precisionModel4 = PrecisionModelFactory.getInstance(geometry);

```

Transformations

Les transformations permettent d'appliquer des transformations de toute nature sur les géométries. L'interface *Geometry* contient une méthode *transform(Transformation)* qui reçoit une instance de *Transformation*. Cette méthode crée un clone de la géométrie, applique la transformation spécifiée sur celui-ci et retourne la nouvelle géométrie transformée. Les transformations peuvent modifier les coordonnées qui composent la géométrie et même modifier la nature elle-même de la géométrie. Les transformations peuvent par exemple servir à appliquer une projection cartographique sur des géométries ou à effectuer une généralisation sur celles-ci.

Il est aussi possible d'appliquer des transformations en utilisant la méthode *transform()* de la classe *Transformation*. Dans ce cas, la géométrie n'est peut-être pas toujours clonée, selon le type de transformation. Consulter la documentation de chaque transformation pour en connaître davantage.

La classe *UnaryTransformation* est la classe de base des transformations qui ne modifient que les coordonnées qui composent la géométrie. Si vous devez implémenter votre propre classe de transformation, vous devrez probablement dériver de cette classe.

Le tableau suivant montre les transformations qui sont disponibles avec l'API de JMap.

Transformations disponibles	
ProjectionTransformation	Applique la projection spécifiée en paramètre du constructeur à une géométrie. La projection peut aussi être appliquée en sens inverse.
OffsetTransformation	Applique une translation en x et en y, tel que spécifié en paramètre du constructeur, à une géométrie.
SmoothTransformation	Applique une généralisation, selon la tolérance spécifiée en paramètre du constructeur, à une géométrie.
PrecisionTransformation.TO_FLOAT	Transforme une géométrie double précision en une géométrie simple précision.
PrecisionTransformation.TO_DOUBLE	Transforme une géométrie simple précision en une géométrie double précision.

L'exemple de code suivant montre comment utiliser la transformation de changement de précision.

```
Point[] points = new Point[]
{
    new Point.Double(10., 10.),
    new Point.Double(20., 20.),
    new Point.Double(30., 30.)
};

LineString doubleLineString = new LineString( points );

// The transformation can be done by the geometry. After the execution, doubleLineString
// will still contain single precision coordinates.
LineString singleLineString1 = (LineString)doubleLineString.transform(PrecisionTransform

// The transformation is made directly on the geometry. After the execution, doubleLine
// will contain single precision coordinates. Moreover, singleLineString2 refers to the
// instance as doubleLineString.
LineString singleLineString2 = (LineString)PrecisionTransformation.TO_FLOAT.transform(d
```

L'exemple de code suivant montre comment utiliser la transformation de changement de projection cartographique.

```
Point[] points = new Point[]
{
    new Point.Double(-73.12345, 45.12345),
    new Point.Double(-73.54321, 45.54321)
};

LineString lineString = new LineString( points );

// Define the source and the destination projections
Projection fromProjection = new LongitudeLatitude();

Projection toProjection = new Mtm();
toProjection.setParams("8"); // zone 8

// Create a combined projection object that converts points from one projection
// to the other
final CombinedProjection combinedProj = new CombinedProjection(fromProjection, toProjec

// Apply the projection transformation
lineString = (LineString)lineString.transform(new ProjectionTransformation(combinedProj
```

L'exemple de code suivant montre comment créer une classe de transformation anonyme en utilisant l'adaptateur `TransformationAdapter`. La transformation convertit les points en ellipses.

```
Point[] points = new Point[]
{
    new Point.Double(10., 10.),
    new Point.Double(20., 20.),
    new Point.Double(30., 30.)
};

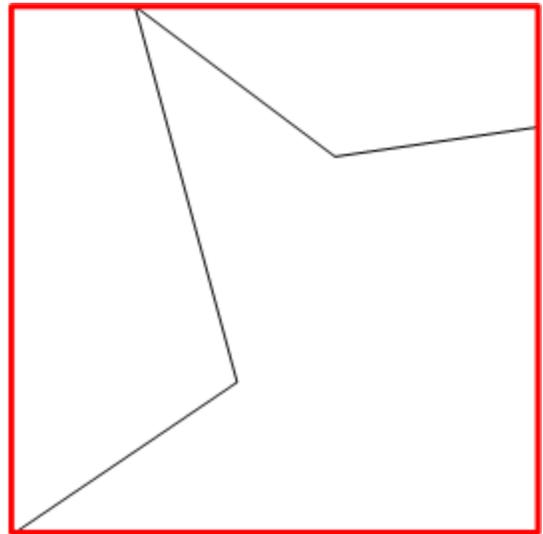
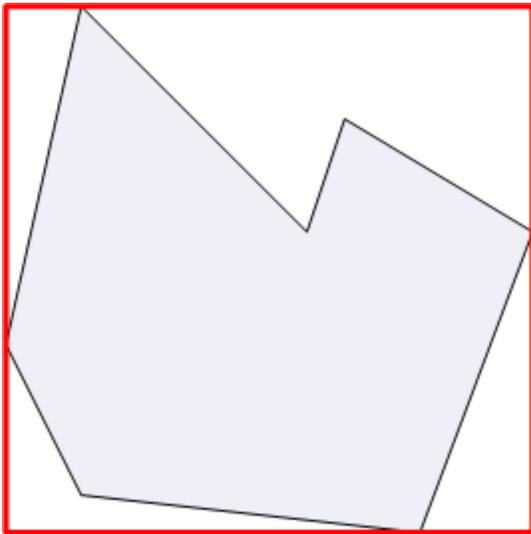
// Create a new transformation which creates a new ellipse at the
// specified location.
Transformation tr = new TransformationAdapter()
{
    public Geometry transform(Point point)
    {
        return new Ellipse(
            (Point)point.clone(), new Dimension.Double(5, 3), 0
        );
    }
};

// Apply the transformation on all points
Ellipse[] ellipses = new Ellipse[points.length];
for (int i = 0; i < ellipses.length; i++)
    ellipses[i] = (Ellipse)tr.transform(points[i]);
```

Rectangle englobant

Le rectangle englobant (MBR pour Minimum Bounding Rectangle) d'une géométrie est le plus petit rectangle orthogonal qui englobe totalement la géométrie. Le MBR est utilisé dans JMap pour effectuer des analyses spatiales rapides sur une grande quantité de géométries avant d'utiliser des algorithmes plus précis (mais plus lents) sur un nombre plus restreint de géométries. Tel que prescrit par l'interface *Geometry* de JMap, tous les types de géométries implémentent la méthode *getBounds()* qui retourne le MBR de la géométrie.

Le MBR d'un point est un rectangle de largeur 0 et de hauteur 0. Le MBR d'une ligne verticale est un rectangle de largeur 0 et le MBR d'une ligne horizontale est un rectangle de hauteur 0.



Exemples de rectangles englobants

Développement JMap Pro et JMap Server

Cette section a pour objectif de guider les développeurs Java dans le développement d'applications cartographiques basées sur JMap Pro et JMap Server 6.5 en utilisant l'API Java de JMap.

Bien qu'il soit possible de personnaliser l'application JMap Pro, il est fortement recommandé d'implémenter toutes les nouvelles fonctions et personnalisations par le développement d'extensions en utilisant l'API Java de JMap.

L'application JMap Pro est une application client développée en Java. Elle peut s'exécuter sous la forme d'une applet Java dans un navigateur, d'une application autonome JavaWebStart (JNLP) ou d'une application autonome lancée à la ligne de commande (surtout utile en développement). Dans tous les cas, il s'agit de la même application, et les extensions développées sont compatibles.

JMap Server est une application serveur développée en Java. Vous pouvez développer des extensions pour JMap Server en utilisant l'API Java de JMap.

Ant

Le SDK de JMap 6.5 utilise, pour la programmation Java, l'outil *Ant* pour effectuer différentes tâches à l'aide de scripts. *Ant* peut être téléchargé à partir du site <http://ant.apache.org>.

Lorsque *Ant* est utilisé depuis *Eclipse*, il est possible que *Ant* ne puisse pas trouver le compilateur java pour la compilation des classes. Si c'est le cas, vous devez ajouter une référence vers la librairie *tools.jar* qui est fournie avec le JDK de Java dans la configuration de *Ant*. Pour plus d'information, consultez cet article http://wiki.eclipse.org/FAQ_Why_can't_my_Ant_build_find_javac%3F.

Eclipse

L'environnement de développement Java privilégié pour JMap est *Eclipse*. Ce dernier peut être téléchargé à partir de cette adresse: <http://www.eclipse.org>. Par contre, il est tout à fait possible de développer des applications JMap dans l'environnement et avec les outils de votre choix. Les informations qui suivent concernent l'environnement *Eclipse Luna* (version 4.4).

Intégration du SDK de JMap 6.5 comme projet Eclipse

Pour faciliter le développement, il est conseillé d'intégrer le SDK de JMap 6.5 comme un projet *Eclipse*. Vous devriez faire votre développement dans des projets séparés du SDK afin de conserver celui-ci intact comme source de référence.

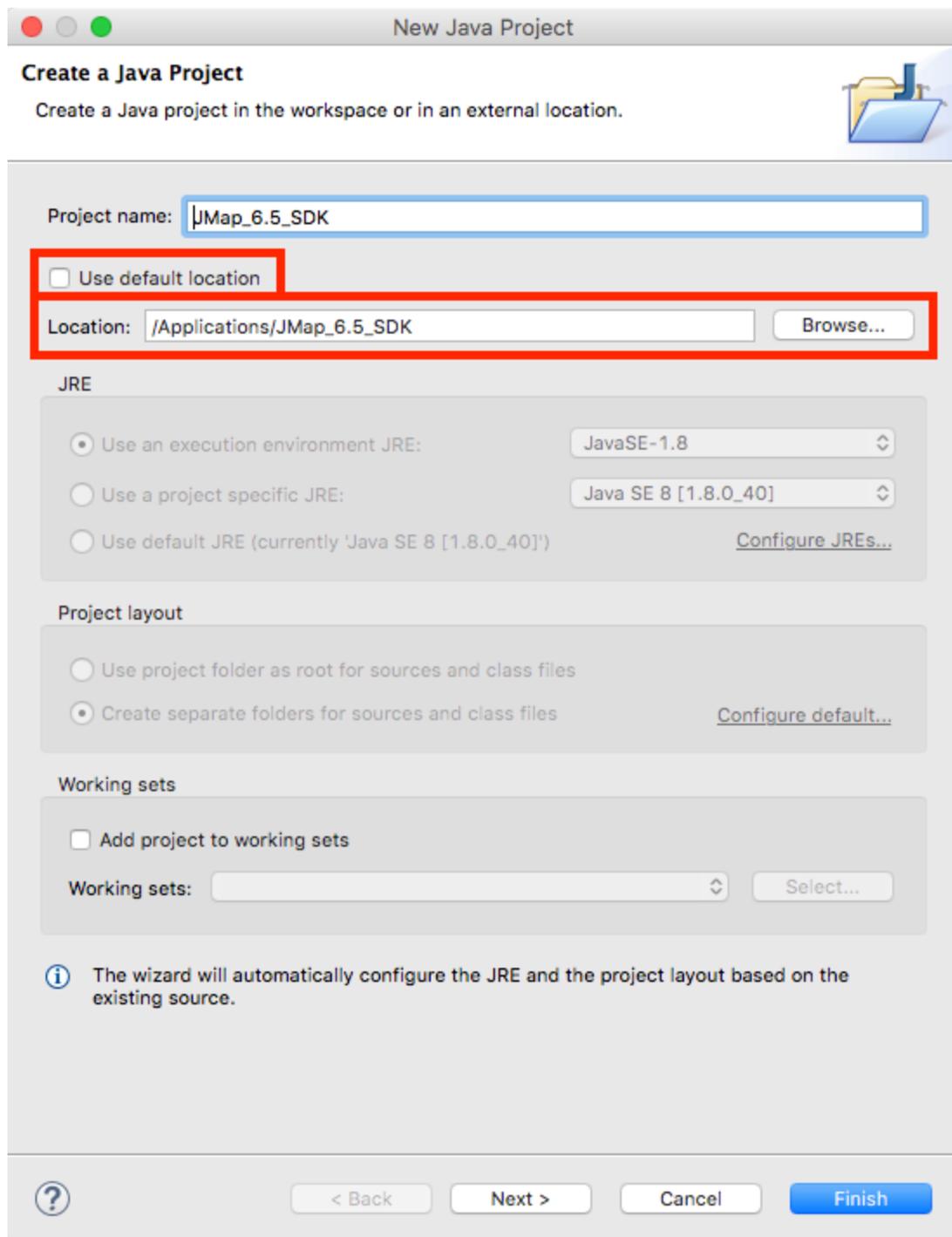
Étape 1

Dans un *workspace Eclipse* (existant ou nouveau), créez un projet Java pour le SDK de JMap 6.5.

Menu **File -> New -> Java Project**

Étape 2

Sélectionnez l'emplacement du SDK de JMap 6.5 tel que spécifié lors de son installation. Pour ce faire, vous devez désactiver l'option *Use default location* afin de permettre à *Eclipse* de sélectionner un répertoire en dehors du *workspace* courant. Notez que le nom du projet est renseigné automatiquement lors vous sélectionnez le répertoire. Appuyez ensuite sur *Finish* .



Création d'un nouveau projet

Pour débuter des nouveaux développements avec le SDK de JMap 6.5, tel que le développement d'une nouvelle extension, il est conseillé de créer des projets distincts. Chaque nouveau projet

devra pointer vers les bibliothèques distribuées avec le SDK pour permettre la compilation. L'exemple suivant montre comment créer un nouveau projet JMap.

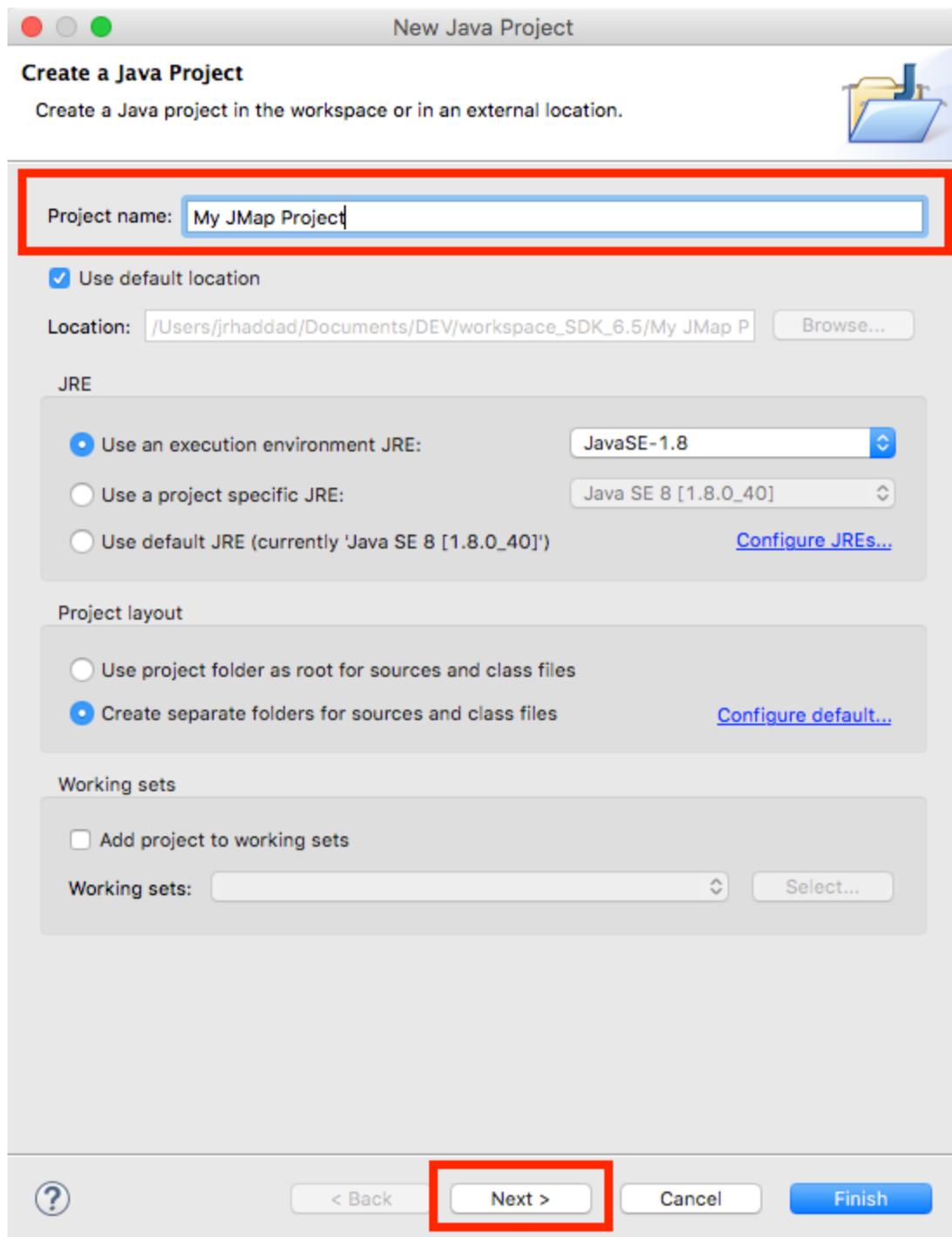
Étape 1

Dans le même *workspace Eclipse* contenant le projet pour le SDK de JMap 6.5, créez un nouveau projet Java.

Menu **File -> New -> Java Project**

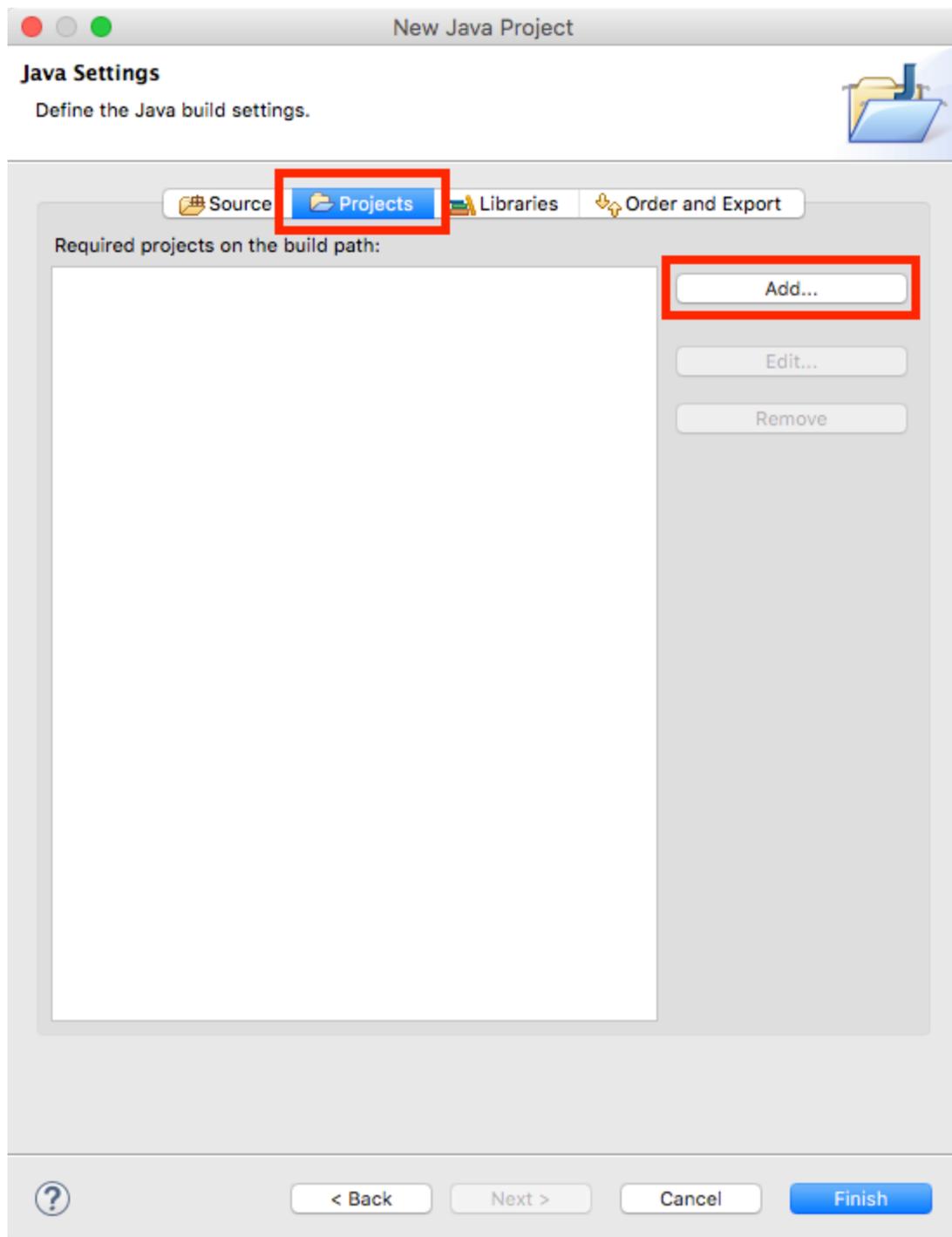
Étape 2

Donnez un nom à votre projet. Appuyez sur **Next** .



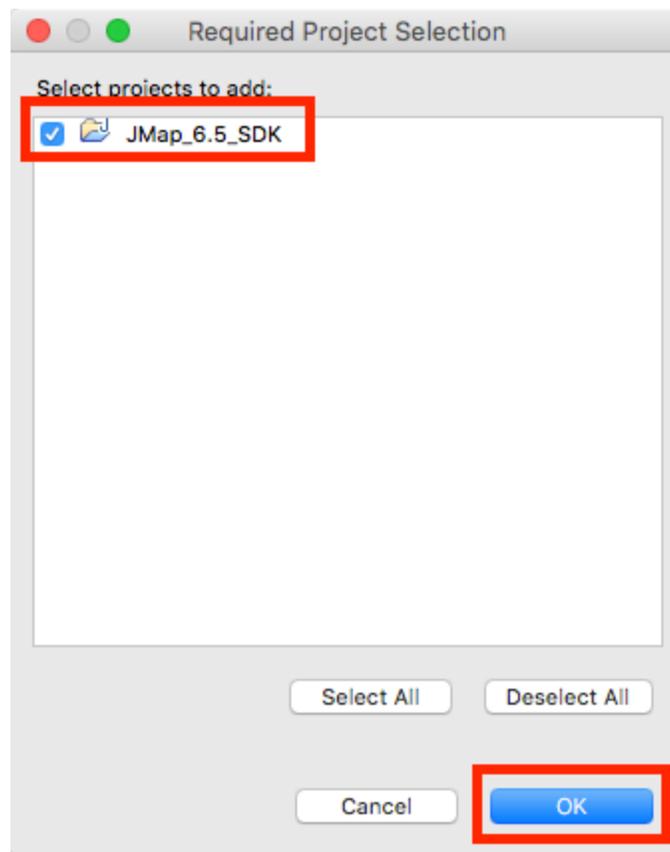
Étape 3

Sélectionnez l'onglet **Projects** . Appuyez sur **Add...** .



Étape 4

Sélectionnez le projet du SDK de JMap. Toutes les bibliothèques du SDK deviendront disponibles pour votre projet. Appuyez sur **OK**.



Étape 5

Appuyez sur *Finish* .

De nombreux exemples sont livrés avec le SDK de JMap 6.5. Tous les exemples sont dans le répertoire `JDK_HOME/examples` .

Le script Ant `build_examples.xml` permet de compiler et d'exécuter les différents exemples.

L'extension **Showcase** permet d'exécuter plusieurs petits exemples dans JMap Pro à l'aide d'une interface graphique unique.

L'extension **Hello World** démontre les bases d'une extension client de JMap Pro qui communique avec une extension de JMap Server.

L'exemple **Webextensions** démontre comment développer une action capable de répondre à des requêtes de JMap Web (voir Développement JMap Web).

Le diagramme de classes suivant présente une version simplifiée de l'architecture de JMap Pro.

Dans JMap, les éléments cartographiques sont des objets (points, lignes, polygones, textes, etc.) qui composent la carte. Les classes des éléments sont toutes dérivées de la classe abstraite *K2DElement*. Les éléments cartographiques sont organisés en couches qui sont affichées dans une application JMap à l'intérieur d'une vue (classe *View*). Chaque élément cartographique est associé à une géométrie (interface *Geometry*), possède un identifiant numérique et un certain nombre d'attributs. L'élément cartographique ne contient pas d'information géométrique directement. C'est plutôt la géométrie associée qui contient toutes les coordonnées géométriques. Les identifiants des éléments d'une même couche doivent être uniques. La méthode utilitaire *GeneratedUniqueId()* de la classe abstraite *K2DElement* permet de générer une séquence d'identifiants uniques pour les éléments.

Création d'éléments cartographiques

L'exemple suivant montre comment créer des éléments cartographiques :

```
// Create a point geometry
Point pointGeometry = new Point.Double(100., 100.);

// Create the element using the geometry, null attributes and auto generated id
K2DPoint point = new K2DPoint(pointGeometry, null, K2DElement.generateUniqueId());

// Create a line geometry
Point pointGeometry1 = new Point.Double(100., 100.);
Point pointGeometry2 = new Point.Double(200., 200.);
Line lineGeometry = new Line(pointGeometry1, pointGeometry2);

// Create some attributes
String attrib1 = "some value";
Integer attrib2 = new Integer(999);

// Create the element using the geometry, 2 attributes and auto generated id
K2DPolyline line = new K2DPolyline(lineGeometry, new Object[] {attrib1, attrib2},
                                   K2DElement.generateUniqueId());
```

Attributs des éléments

Les éléments cartographiques possèdent normalement des valeurs d'attributs. Ces valeurs sont les données descriptives des éléments. Tous les éléments appartenant à une même couche possèdent la même liste d'attributs (mêmes noms, mêmes types). Notez que les éléments ne contiennent que les valeurs des attributs (tableau d'objets) et non leur définition. La définition des ces attributs est gérée au niveau de la couche, à l'aide de la classe *Attribute*.

La liste des attributs des éléments d'une couche est déterminée par l'administrateur JMap au moment de la création de la couche (attributs liés). Les types des attributs sont définis à l'aide des constantes Java pour les types SQL (classe *java.sql.Types*).

Les attributs des éléments sont utilisés pour plusieurs fonctions telles que les infobulles, les étiquettes, les thématiques et le filtrage.

L'exemple suivant montre comment accéder aux valeurs des attributs des éléments :

```
K2DElement element = ...

Object[] attributeValues = element.getAttributes();
for (int i = 0; i < attributeValues.length; i++)
{
    System.out.println(i + " : " + attributeValues[i]);
}
```

L'exemple suivant montre comment accéder à la définition des attributs d'une couche.

```
VectorLayer layer= ...

Attribute[] attributes = layer.getAttributeMetaData();
for (int i = 0; i < attributes .length; i++)
{
    System.out.println(i + " name : " + attributes[i].getName());
    System.out.println(i + " title : " + attributes[i].getTitle());
    System.out.println(i + " type: " + attributes[i].getType());
}
```

Style des éléments

Quand les éléments se dessinent sur la carte, un objet de style est utilisé pour déterminer tous leurs aspects visuels. Une instance de la classe *Style* contient des propriétés telles que la couleur de la ligne, la couleur de remplissage, la police de caractères, la transparence, etc. Chaque couche possède un ou plusieurs styles. Le style utilisé pour afficher les éléments est déterminé par l'échelle de la carte et la présence de thématiques. Les thématiques déterminent le style des éléments affichés en fonction de la valeur des attributs de ces éléments.

L'exemple suivant montre comment obtenir le style utilisé par une couche à une échelle donnée. Cela ne tient pas compte de la présence de thématiques sur la couche.

```
K2DElement element = ...
VectorLayer layer = ...

Style style = layer.getStyle(view.getScaleFactor());
```

L'exemple suivant montre comment obtenir le style utilisé pour afficher l'élément spécifié à une échelle donnée. Le style retourné tient compte de la présence de thématiques sur la couche.

```
K2DElement element = ...
VectorLayer layer = ...

Style style = layer.getStyle(element, view.getScaleFactor());
```

Les propriétés d'un style peuvent être modifiées en appelant les méthodes de la classe `Style`. L'exemple suivant montre comment modifier les paramètres du style d'une couche de polygones à l'échelle courante de la carte.

```
VectorLayer layerOfPolygons = ...

Style style = layerOfPolygons.getStyle(view.getScaleFactor());

style.setBorderColor(Color.BLACK);
style.setFillColors(Color.BLUE);
style.setTransparency(.5f); // 50% transparent
```

Rectangle englobant à l'affichage

Le rectangle englobant à l'affichage d'un élément cartographique est défini comme le plus petit rectangle en coordonnées écran (DC) qui contient entièrement l'élément en tenant compte de son style. Le style d'un élément a un impact sur son rectangle englobant à l'affichage. Par exemple, une bordure de polygone épaisse augmente la taille du rectangle et la taille du symbole d'un point va déterminer la taille du rectangle englobant à l'affichage, etc.

The display bounds of a map element is the smallest rectangle in device coordinates (DC) that completely contains the element, taking its style into account. The style of an element influences the display bounds. For example, the thick border width of a polygon will make the display bounds bigger or the size of the symbol for a point will determine the display bounds for that point, etc.

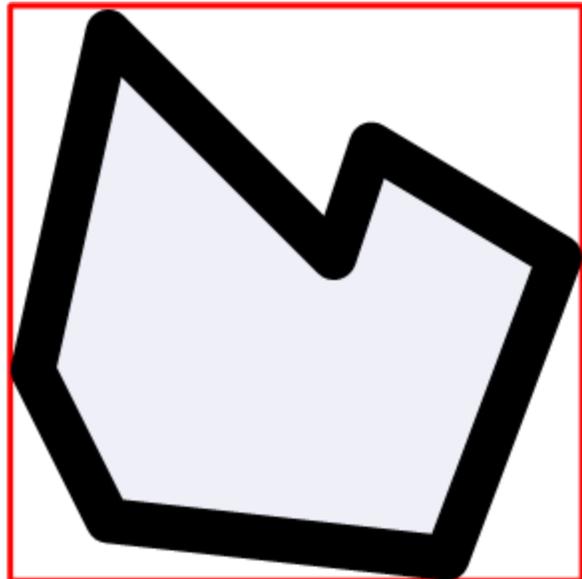
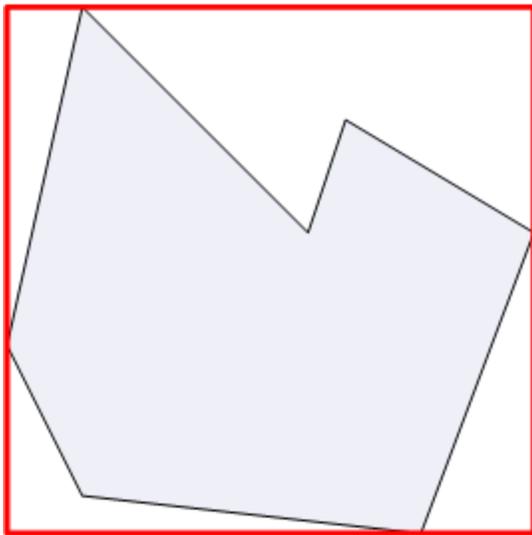
The following code example shows how to obtain the display bounds of a map element:

L'exemple de code suivant montre comment obtenir le rectangle englobant à l'affichage d'un élément cartographique.

```
K2DElement element = ...
View view = ...
VectorLayer layer = ...

Rectangle displayBounds = element.getDisplayBounds(view.getTransform(),
                                                    layer.getStyle(element, view.getScaleFactor()));
```

Notez que la méthode `getDisplayBounds` prend en paramètre la transformation de la vue (voir Systèmes de coordonnées) ainsi que le style de l'élément.



Rectangles englobants à l'affichage du même polygone avec des styles différents

Sélection

Les éléments cartographiques des couches vectorielles peuvent être sélectionnés. L'utilisateur possède plusieurs outils lui permettant de sélectionner des éléments des différentes couches vectorielles d'un projet JMap.

Il y a une propriété de la classe `K2DElement` qui s'appelle `selected` et qui indique si un élément est sélectionné ou non. Les éléments sélectionnés s'affichent avec un style différent, soit le style de sélection de la couche vectorielle.

Les couches vectorielles gèrent la liste de leurs éléments qui sont sélectionnés. L'API de la classe `VectorLayer` offre plusieurs méthodes relatives à la sélection d'éléments. Voir Couches et gestionnaire de couches pour plus d'information.

L'exemple de code source suivant montre comment sélectionner et désélectionner des éléments.

```
K2DElement element = ...
VectorLayer layer = ...

// add the element to the current selection
layer.addToSelection(element);

// test if element is selected, useless, just to demonstrate
boolean isSelected = element.isSelected();
```

```
// cycle through selection
Collection selection = layer.getSelection();
for (K2DElement element : selection)
    System.out.println(element);

// unselect element
layer.unselectElement(element);

// clear layer selection
layer.clearSelection();
```

Éléments stylés

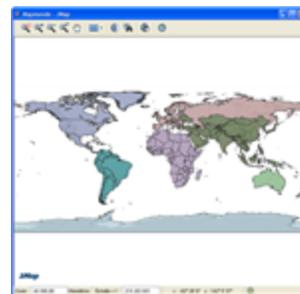
Les éléments stylés (classe *K2DStyledElement*) sont des éléments cartographiques spéciaux. Ils comportent leur propre style et ignorent le style de la couche qui les contient lorsqu'ils s'affichent. Ils sont utiles si vous souhaitez ajouter des éléments comportant leur propre style sur la carte, par programmation. Mis à part cette différence, les éléments stylés se comportent exactement comme les autres éléments cartographiques.

L'exemple de code suivant montre comment créer des éléments stylés.

```
K2DElement element = ...
Style style = ...

K2DStyledElement styledElem = new K2DStyledElement(element, style); // element will use t
```

Il existe 2 systèmes de coordonnées utilisés lors de la programmation avec JMap Pro. Il y a le système de coordonnées du monde (ou world coordinates - WC) et le système de coordonnées d'écran (ou *device coordinates* - DC). Le système WC est le système original des données tandis que le système DC est celui de l'écran servant à afficher la carte.



Système de coordonnées du monde (WC) VS système de coordonnées écran (DC)

WC

Toutes les coordonnées des géométries dans JMap sont en WC. Par exemple, si vos données utilise la projection MTM zone 8, les valeurs des coordonnées seront semblables à (300 000, 5 000 000). Si vos données ne sont pas projetées, elles sont donc en longitude et latitude et leur étendue est de -180 à 180 degrés est-ouest et de -90 à 90 degrés nord-sud.

DC

L'écran d'ordinateur servant à afficher la carte est divisé en pixels avec la coordonnées (0, 0) localisée en haut à gauche. C'est le système DC. Quand vous traitez des événements de souris (p.e. *MouseClicked*) dans une application JMap, vous traitez des coordonnées DC contenues dans l'événement et exprimés en pixels.

Transformation de coordonnées entre WC et DC

En programmant dans JMap, il est fréquent qu'on doive transformer des coordonnées entre WC et DC et ce, dans les 2 directions. Quand les éléments cartographiques d'une couche sont dessinés à l'écran, leurs coordonnées sont converties à la volée de WC vers DC afin d'allumer les bons pixels à l'écran. Par contre, quand un clic de souris survient sur la carte, la coordonnées DC du curseur de la souris est transformée en WC afin de sélectionner le bon élément sur la carte.

La classe *K2DTransform* contient une matrice de transformation affine utilisée pour convertir les données entre les systèmes DC et WC. Elle offre des méthodes de transformation dans les 2 directions. Chaque carte dans JMap (classe *View*) possède sa propre instance de transformation.

Principales méthodes de la classe *K2DTransform*

<i>transform(OrientedRectangle)</i>	Crée un nouveau rectangle orienté à partir du point WC spécifié et le transforme en DC.
<i>transform(Point)</i>	Crée un nouveau point à partir du point WC spécifié et le transforme en DC.
<i>transform(Rectangle)</i>	Crée un nouveau rectangle à partir de la rectangle WC spécifié et le transforme en DC.
<i>transformInv(OrientedRectangle)</i>	Crée un nouveau rectangle orienté à partir du point DC spécifié et le transforme en WC.
<i>transformInv(Point)</i>	Crée un nouveau point à partir du point DC spécifié et le transforme en WC.

transformInv(Rectangle)

Crée un nouveau rectangle à partir de la rectangle DC spécifié et le transforme en WC.

L'exemple de code source suivant montre comment transformer des coordonnées WC vers DC.

```
K2DTransform transform = ...
Point pointWC;
Point pointDC;

// Transform a point from WC to DC
pointWC = new Point.Double(100., 100.);
pointDC = transform.transform(pointWC);

// Transform a point from DC to WC
pointDC = new Point.Double(100., 100.);
pointWC = transform.transformInv(pointDC);
```

L'exemple de code source suivant montre comment transformer une coordonnées DC provenant d'un événement de souris vers WC. La transformation est obtenue de la vue.

```
View view = ...
K2DTransform transform = view.getTransform();
MouseEvent e = ...

Point pointWC = transform.transformInv(new Point.Double(e.getX(), e.getY()));
```

L'exemple de code source suivant montre comment transformer une coordonnées DC en utilisant l'événement de souris. Cette méthode simple ne peut être utilisée qu'à partir d'une classe d'outil dérivée de la classe *Tool*.

```
Point pointWC = Tool.toWCPoint(e);
```

Certaines précautions doivent être prises lors de la transformation (normale ou inverse) d'un rectangle, car si une rotation est appliquée sur la vue, le rectangle retourné pourrait être écrasé. Pour prévenir ce type de problème, il vaut mieux effectuer les calculs sur un rectangle orienté, initialisé à partir du rectangle à transformer.

```

View view = ...

final ViewState viewState = view.getViewState();

// Computes the display bounding box of the selection
final Rectangle displaySelectBounds = view.getLayerManager().getDisplaySelectedBounds(view)

if (displaySelectBounds != null)
{
    // Transforms the display bounding box into a WC extent.
    final OrientedRectangle selectBounds = viewState.getTransform().transformInv(
        new OrientedRectangle.Double(displaySelectBounds)
    );

    view.zoom(selectBounds);
    view.refresh()
}

```

La classe View

La vue (classe View) dans JMap est la composante graphique responsable de l'affichage de la carte. Les méthodes de cette classe permettent de contrôler la carte (zoom, déplacement, etc.), de demander son rafraîchissement, de renseigner sur l'échelle, etc.

Le tableau suivant présente les principales méthodes de la classe View.

Navigation	
<i>zoom(double)</i>	Effectue un zoom au centre de la carte en utilisant le facteur de grossissement spécifié. Un facteur de 2 produit une carte 2 fois plus rapprochée. Un facteur de .5 produit une carte 2 fois plus éloignée. Un rafraîchissement de la carte doit suivre.
<i>zoom(Rectangle)</i>	Effectue un recadrage de la carte autour du rectangle spécifié. Un rafraîchissement de la carte doit suivre.
<i>zoomExtent()</i>	Effectue un recadrage de la carte de manière à afficher l'étendue totale des données. Un rafraîchissement de la carte doit suivre.
<i>moveTo(double, double)</i>	Recentre la carte autour de la coordonnée spécifiée. L'échelle demeure inchangée. Un rafraîchissement de la carte doit suivre.

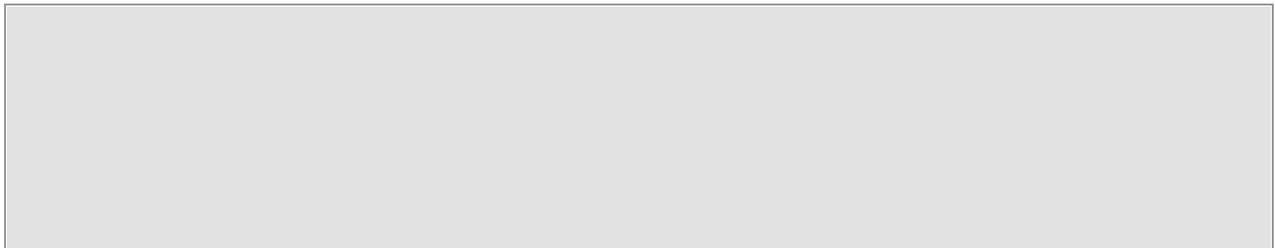
<i>moveTo(Point)</i>	Recentre la carte autour de la coordonnée spécifiée. L'échelle demeure inchangée. Un rafraîchissement de la carte doit suivre.
<i>pan(int, int)</i>	Déplace la carte en x et en y selon les valeurs spécifiées en coordonnées d'écran (DC). Un rafraîchissement de la carte doit suivre.
<i>setScale(double)</i>	Modifie l'échelle de la carte selon la valeur spécifiée. Un rafraîchissement de la carte doit suivre.
Affichage	
<i>refresh()</i>	Rafrâchit la carte. Après tout appel à des méthodes modifiant l'état de la carte, celle-ci doit être rafraîchie pour que les changements deviennent visibles.
<i>addMarker(Point, String, Style)</i>	Ajoute un marqueur (symbole indiquant une localisation) sur la carte à la coordonnée spécifiée et avec le message et l'apparence spécifiés.
<i>removeMarker(long)</i>	Enlève de la carte le marqueur spécifié.
<i>clearMarkers()</i>	Enlève tous les marqueurs présents sur la carte.
<i>getDCViewBounds()</i>	Retourne l'étendue de la carte en coordonnées d'écran (DC).
<i>getWCViewBounds()</i>	Retourne l'étendue de la carte en coordonnées terrain (WC).
<i>getTransform()</i>	Retourne la transformation de la vue. Cette transformation est utilisée pour la conversion des coordonnées entre les systèmes DC et WC.
<i>getViewOverlay()</i>	Retourne l' <i>overlay</i> de la vue. L' <i>overlay</i> est une couche spéciale servant à l'affichage de données volatiles, souvent utilisée pour réaliser des animations fluides, telles que le déplacement d'objets, du dessin, etc.
<i>getScaleFactor()</i>	Retourne le facteur d'échelle courant de la vue sous la forme du dénominateur (1 : dénominateur).
<i>getViewState()</i>	Retourne l'instance de la classe <i>ViewState</i> associée à cette vue. Voir plus bas pour plus d'informations sur ce sujet.

<code>getZoomLevel()</code>	Retourne la distance horizontale courante de la vue en coordonnées terrain (WC).
Autres	
<code>getLayerManager()</code>	Retourne l'instance du gestionnaire de couches (<code>LayerManager</code>) associé à la vue.
<code>getMapProjection()</code>	Retourne la projection cartographique (<code>Projection</code>) des données de la carte.
<code>getMapUnit()</code>	Retourne l'unité (<code>JMapUnit</code>) des données de la carte.
<code>removePopupMenuAction(Action)</code>	Enlève une action du menu contextuel de la vue.
<code>removePopupMenuItem(JMenuItem)</code>	Enlève un item du menu contextuel de la vue.
<code>addPopupMenuAction(Action)</code>	Ajoute une action au menu contextuel de la vue.
<code>addPopupMenuItem(JMenuItem)</code>	Ajoute un item au menu contextuel de la vue.
<code>addViewEventListener(ViewEventListener)</code>	Permet de s'enregistrer pour écouter les événements générés par la vue.
<code>removeViewEventListener(ViewEventListener)</code>	Permet de cesser d'écouter les événements générés par la vue.
<code>setCurrentTool(Tool)</code>	Remplace l'outil actif de la vue par l'outil spécifié en paramètre.

Les événements de la classe View

La vue dans JMap génère des événements dans plusieurs situations. Pour recevoir ces événements, vous devez enregistrer un *listener* sur la vue en questions. Notez que pour recevoir les événements générées par l'ensemble des vues ouvertes, il est préférable d'enregistrer un *listener* sur le gestionnaire de vues. Voir plus bas pour plus d'informations à ce sujet.

Pour recevoir les événements d'une vue, vous devez implémenter l'interface `ViewEventListener` et l'enregistrer auprès de la vue en utilisant la méthode `addViewEventListener()` tel que montré dans l'exemple suivant.



```

View view = ...

view.addViewEventListener(new ViewEventListener()
{
    @Override
    public void viewToolChangedOccurred(ViewToolChangedEvent e)
    {
    }

    ...
});

```

Notez qu'il existe aussi un adaptateur (*ViewAdapter*) qui simplifie le développement du *listener* .
Le tableau suivant présente les événements générés par la classe *View* .

Événements de la vue

viewChangedOccurred (ViewChangedEvent)	Lancé après un changement d'état de la vue suite à une opération de navigation (zoom, déplacement, etc.).
viewToolChangedOccurred (ViewToolChangedEvent)	Lancé quand l'outil actif sur la vue est remplacé par un autre outil.
viewRedrawOccurred (ViewRedrawEvent)	Lancé quand la vue se redessine, en tout ou en partie. Les éléments des couches sont redessinés.
viewRepaintOccurred (ViewRepaintEvent)	Lancé quand la vue est repeinte, en tout ou en partie. L'image de la carte est simplement rafraîchie, sans que les éléments des couches soient redessinés.
viewStaticElementChanged (ViewStaticElementEvent)	Lancé quand des éléments statiques (flèche du nord, échelle graphique, etc.) sont ajoutés ou enlevés de la vue.
viewPopupMenuShowing(ViewPopupMenuShowingEvent)	Lancé juste avant que le menu contextuel de la vue s'affiche. Permet de modifier le contenu du menu avant que ce dernier soit présenté à l'utilisateur.
viewReadyOccurred(ViewReadyEvent)	Lancé quand la vue devient prête à l'utilisation, c'est-à-dire après

	qu'elle soit affichée une première fois.
--	--

Pour plus d'information sur les événements en Java, consultez le tutoriel Java.

La classe ViewState

La classe *ViewState* contient certaines propriétés d'une vue qui définissent son état à un instant donné (échelle, étendue, transformation, etc.). Plusieurs méthodes dans L'API de JMap prennent en paramètre une instance de *ViewState*. Il est possible d'obtenir l'état d'une vue en appelant la méthode *getViewState()* de la classe *View*.

La classe ViewManager

Le gestionnaire de vues (classe *ViewManager*) est responsable de gérer une ou plusieurs vues qui sont présentes dans l'application. La classe possède des méthodes qui permettent d'effectuer des opérations sur l'ensemble des vues de l'application et pour connaître en tout temps, quelle vue est la vue active (celle qui a le focus).

Le tableau suivant présente les principales méthodes de la classe *ViewManager* .

Méthodes les plus utilisées de la classe <i>ViewManager</i>	
<code>addViewEventListener(ViewEventListener)</code>	Permet de s'enregistrer pour écouter les événements générés par la vue active.
<code>addViewManagerEventListener(ViewManagerEventListener)</code>	Permet de s'enregistrer pour écouter les événements générés par le gestionnaire de vues (voir plus bas).
<code>getActiveView()</code>	Retourne la vue active, c'est-à-dire la vue qui a le focus.
<code>getLayerManager()</code>	Retourne le gestionnaire de couches de la vue active.
<code>getView(String)</code>	Retourne la vue qui porte le nom spécifié en paramètre.
<code>getViews()</code>	Retourne la liste complète des vues présentes dans le gestionnaire de vues.
<code>refresh()</code>	Effectue un rafraîchissement sur l'ensemble des vues présentes dans le gestionnaire de vues.
<code>removeViewEventListener(ViewEventListener)</code>	Permet de cesser d'écouter les événements générés par la vue active du gestionnaire de vues.

<code>removeViewManagerEventListener(ViewManagerEventListener)</code>	Permet de cesser d'écouter les événements générés par le gestionnaire de vues.
<code>setCurrentTool(Tool)</code>	Remplace l'outil actif de la vue active par l'outil spécifié en paramètre.

Les événements de la classe `ViewManager`

Le gestionnaire de vues génère des événements associés à l'état des vues qui lui sont associées. Pour recevoir ces événements, vous devez implémenter l'interface `ViewManagerEventListener` et l'enregistrer auprès du gestionnaire de vues en utilisant la méthode `addViewManagerEventListener()` tel que montré dans l'exemple suivant.

```
View view = ...
ViewManager viewManager = view.getLayerManager();

viewManager.addViewManagerEventListener(new ViewManagerEventListener()
{
    @Override
    public void viewAdded(ViewAddedEvent e)
    {
        ...
    }
});
```

Notez qu'il existe aussi un adaptateur (`ViewManagerAdapter`) qui simplifie le développement du listener.

Le tableau suivant présente les événements générés par la classe `ViewManager`.

Événements du gestionnaire de vues	
<code>viewAdded(ViewAddedEvent)</code>	Lancé quand une vue est ajoutée au gestionnaire de vues
<code>viewActivated(ViewActivatedEvent)</code>	Lancé quand une vue a été activée, c'est-à-dire qu'elle a reçu le focus.
<code>viewDeactivated(ViewDeactivatedEvent)</code>	Lancé quand une vue a été désactivée, c'est-à-dire qu'elle a perdu le focus au profit d'une autre vue.
<code>viewRemoved(ViewRemovedEvent)</code>	Lancé quand une vue est enlevée du gestionnaire de vues.

Il est aussi possible d'enregistrer un *listener* auprès du gestionnaire de vues afin de recevoir les événements générés par l'ensemble des vues associées au gestionnaire de vues. Cela peut être plus pratique que d'enregistrer un *listener* sur chaque vue séparément. Pour ce faire, utilisez la méthode `addViewEventListener` de la classe `ViewManager`.

La classe ViewUtil

La classe `ViewUtil` offre des méthodes utiles pour exploiter la vue.

Le tableau suivant présente les méthodes de la classe `ViewUtil` les plus fréquemment utilisées.

Méthodes les plus utilisées de la classe <code>ViewUtil</code>	
<code>mailMap(View, JMapSvcConnection, int, MailMessage)</code>	Envoie une image de la carte par courriel aux destinataires définis dans l'objet de type <code>MailMessage</code> passé en paramètre.
<code>makeImage(View, int)</code>	Fabrique et retourne une image de la carte telle qu'affichée dans la vue. La largeur (en pixels) de l'image à fabriquer est passée en paramètre.
<code>saveAs(View)</code>	Enregistre une image de la carte sur le disque dur. Une fenêtre permettant de sélectionner l'emplacement du fichier est présentée à l'utilisateur.

Concepts

Les couches (classe `Layer` et ses dérivées) sont très importantes dans la programmation avec JMap. Elles contiennent et gèrent les données cartographiques qui s'affichent sur la carte (classe `View`). Les couches se divisent en 2 grandes familles: les couches vectorielles (classe `VectorLayer`) qui contiennent des données vectorielles et les couches matricielles (classe `RasterLayer`) qui contiennent des données matricielles (images).

Au chargement du projet, l'application cliente obtient la configuration des couches du serveur et crée les instances relatives à celles-ci dans le `LayerManager`. Une couche ne contient initialement pas de données et celles-ci seront chargées, en partie ou en totalité, après les rafraîchissements de la vue, en tenant compte du mode de chargement et des contraintes d'affichage (état de visibilité et seuils d'affichage).

Il existe 2 modes de chargement des couches dans JMap : par tuiles et par région. Les couches vectorielles supportent les 2 modes, tandis que les couches matricielles ne supportent que le mode par région.

Les couches tuilées divisent l'étendue de la couche en rangées et en colonnes, dont chaque cellule est une tuile de données (classes `TileSet` et `Tile`). Puisque les couches sont initialement

vides au démarrage de l'application, les tuiles de données seront chargées par l'application lors de leur premier affichage dans une vue. Une fois chargée dans une couche, une tuile restera en mémoire pour la durée de la session, si celle-ci n'expire pas prématurément, ou que le gestionnaire de mémoire ne décide de la supprimer.

Lorsqu'une tuile de données est demandée à JMap Server, toutes les géométries (interface *Geometry*) qui intersectent l'étendue de la tuile sont transférées sur le client. Une fois la tuile reçue, les géométries sont transformées en éléments (classe *K2DElement*), qui sont ensuite chargés dans le *TileSet* de la couche. Pour éviter qu'un élément soit dupliqué dans plusieurs tuiles de données, les éléments qui ne sont pas totalement inclus dans une tuile sont finalement déplacés dans la tuile univers de la couche.

Les couches chargées par région reposent sur la même structure (classes *TileSet* et *Tile*), mais ne disposent qu'une tuile unique ayant une étendue variable. Lorsqu'un changement est appliqué sur la matrice de transformation de la vue et qu'un rafraîchissement survient, les couches par région rechargent leur tuile unique avec les données qui intersectent l'étendue de la vue.

La classe *Layer* et ses dérivées

La classe *Layer*

La classe *Layer* est abstraite et ne peut donc pas être instanciée. Par contre, elle propose les méthodes de base de toutes les couches.

Méthodes les plus utilisées de la classe *Layer*

<i>addLayerEventListener (LayerEventListener)</i>	Ajoute un <i>listener</i> des événements générés par la couche.
<i>getElementType()</i>	Retourne le type d'éléments que contient la couche. Les types sont définis par les constantes de la classe <i>ElementTypes</i> .
<i>getId()</i>	Retourne l'identifiant numérique unique de la couche.
<i>getName()</i>	Retourne le nom de la couche.
<i>getStyleManager()</i>	Retourne l'instance du gestionnaire de styles (classe <i>StyleManager</i>) utilisé par la couche. Ce dernier gère l'ensemble des styles de la couche définissant l'apparence graphique des éléments sur la carte.
<i>getNextUserLayerId()</i>	Méthode statique qui génère un nouvel identifiant unique de couche utilisateur.

<i>invalidate()</i>	Invalide le cache d'une couche chargée par région afin de permettre le rafraîchissement de celle-ci lorsqu'aucun changement n'a été apporté à la matrice de transformation de la vue.
<i>isDrawable(double)</i>	Indique si la couche est affichée à l'échelle spécifiée, selon les seuils d'affichages définis.
<i>isSelectable()</i>	Indique si les objets de la couche sont sélectionnables ou non.
<i>isVisible()</i>	Indique si la couche est visible ou non.
<i>toLayerInfo()</i>	Retourne une structure de données sérialisable contenant toutes les informations de la couche.

La classe *VectorLayer*

La classe *VectorLayer* est dérivée de *Layer* et contient uniquement des données vectorielles. Elle possède plusieurs méthodes spécialisées pour les données vectorielles pour gérer des sélections d'éléments, effectuer des analyses spatiales, etc.

Les couches vectorielles possèdent un ensemble d'attributs qui sont communs à tous les éléments de la couche. Ces attributs fournissent les données descriptives des éléments de la couche.

Méthodes les plus utilisées de la classe *VectorLayer*

<i>addElement(K2DElement)</i>	Ajoute l'élément spécifié sur la couche.
<i>addElements(K2DElement[])</i>	Ajoute tous les éléments spécifiés sur la couche.
<i>addToSelection(K2DElement)</i>	Ajoute l'élément spécifié à la liste des éléments sélectionnés.
<i>addToSelection(Collection<K2DElement>)</i>	Ajoute une collection d'éléments à la liste des éléments sélectionnés.
<i>clearSelection()</i>	Vide la liste des objets sélectionnés sur cette couche.
<i>getAttributeMetaData()</i>	Retourne la liste des attributs (classe <i>Attribute</i>) de la couche.

<code>addDisplayFilter(DisplayFilter)</code>	Ajoute un nouveau filtre d'affichage (interface <code>DisplayFilter</code>) à la couche. Ce filtre permet de définir ce qui sera affiché et ce qui ne le sera pas.
<code>getElementAtPoint (Point, ViewState, boolean)</code>	Permet d'obtenir le premier élément de la couche détecté à la coordonnée spécifiée en WC dans la carte spécifiée (classe <code>ViewState</code>). Le dernier paramètre détermine si les éléments invisibles (à cause d'un filtre d'affichage, par exemple) doivent être considérés ou non.
<code>getElements()</code>	Retourne une collection contenant l'ensemble des éléments de la couche.
<code>getElementsAtPoint (Point WCCoord, ViewState viewScale, boolean onlyIfVisible)</code>	Permet d'obtenir l'ensemble des éléments de la couche détectés à la coordonnée spécifiée en WC dans la carte spécifiée (classe <code>ViewState</code>). Le dernier paramètre détermine si les éléments invisibles (à cause d'un filtre d'affichage, par exemple) doivent être considérés ou non.
<code>getElementsInSurface (Surface, ViewState, boolean)</code>	Permet d'obtenir l'ensemble des éléments de la couche qui intersectent la surface spécifiée en WC dans la carte spécifiée (classe <code>ViewState</code>). Le dernier paramètre détermine si les éléments invisibles (à cause d'un filtre d'affichage, par exemple) doivent être considérés ou non.
<code>getExtent()</code>	Retourne l'étendue totale des données de la couche sous la forme d'un rectangle.
<code>getSelectedElements()</code>	Retourne un tableau contenant l'ensemble des éléments qui son sélectionnés sur la couche.

La classe `Raster Layer`

La classe `RasterLayer` est dérivée de `Layer` et contient uniquement des données matricielles. Ses méthodes sont rarement utilisées par les développeurs d'applications JMap.

Méthodes les plus utilisées de la classe *RasterLayer*

<code>getExtent()</code>	Retourne l'étendue totale des données de la couche sous la forme d'un rectangle.
<code>getParameters()</code>	Retourne les paramètres matriciels (classe <i>RasterParameters</i>) utilisés par la couche matricielle (p.e. le format de l'image, la transparence, etc.).
<code>getRasterBands()</code>	Retourne un tableau des attributs des différentes bandes (classe <i>RasterBand</i>) contenues dans l'image.

Les événements de la classe *Layer*

Les couches dans JMap génèrent des événements dans plusieurs situations. Pour recevoir les événements générés par une couche, vous devez enregistrer un *listener* sur la couche en question. Notez que pour recevoir les événements générés par l'ensemble des couches du projet, il est préférable d'enregistrer un *listener* sur le gestionnaire de couches. Voir plus bas pour plus d'informations à ce sujet.

Pour recevoir les événements d'une couche, vous devez implémenter l'interface *LayerEventListener* et l'enregistrer auprès de la couche en utilisant la méthode `addLayerEventListener()` tel que montré dans l'exemple suivant:

```
Layer layer = ...

layer.addLayerEventListener(new LayerEventListener()
{
    @Override
    public void layerSelChangedEventOccurred(LayerSelChangedEvent e)
    {
        // TODO
    }
    ...
});
```

Notez qu'il existe aussi un adaptateur (*LayerAdapter*) qui simplifie le développement du *listener* . La super classe *LayerEvent* possède une méthode `getLayer()` et une méthode `getLayerManager()` permettant d'accéder respectivement à la couche ou au gestionnaire de couches ayant généré l'événement.

Le tableau suivant présente les événements générés par la classe *Layer* .

Événements de la couche les plus fréquemment utilisés

<code>layerElementsAddedEventOccurred(LayerElementsAddedEvent)</code>	Lancé après l'ajout de nouveaux éléments sur la couche. Les éléments ajoutés sont accessibles dans l'instance de l'événement.
---	---

<i>layerElementsChangedEventOccurred(LayerElementsChangedEvent)</i>	Lancé après la modification d'éléments de la couche. Les éléments modifiés sont accessibles dans l'instance de l'événement.
<i>layerElementsRemovedEventOccurred(LayerElementsRemovedEvent)</i>	Lancé après l'enlèvement d'éléments de la couche. Les éléments enlevés sont accessibles dans l'instance de l'événement.
<i>layerSelChangedEventOccurred(LayerSelChangedEvent)</i>	Lancé après une modification de la sélection sur la couche. Il peut s'agir d'éléments sélectionnés ou désélectionnés. Les éléments concernés sont accessibles dans l'instance de l'événement.
<i>layerStyleChangedEventOccurred(LayerStyleChangedEvent)</i>	Lancé après une modification du style de la couche.

La classe LayerManager

Chaque carte (classe *View*) dans JMap possède un gestionnaire de couches (classe *LayerManager*). Ce dernier est responsable de gérer l'ensemble des couches à afficher sur la carte, de même que leur ordre et leur organisation hiérarchique. De plus, le gestionnaire de couches permet d'écouter des événements sur l'ensemble des couches et d'effectuer certaines opérations sur celles-ci.

Méthodes les plus utilisées de la classe *LayerManager*

<i>addLayer(Layer)</i>	Ajoute la couche spécifiée à la position la plus élevée.
<i>addLayerEventListener(LayerEventListener)</i>	Ajoute un <i>listener</i> des événements générés par le gestionnaire de couches. Le gestionnaire de couches relaie aussi tous les événements générés par les couches qu'il gère.
<i>clearAllSelection()</i>	Efface les liste d'éléments sélectionnés sur chacune des couches.
<i>getAllLayers()</i>	Retourne l'ensemble ordonné des couches (couches utilisateur et couches normales). La couche à la position 0 est celle du dessous.
<i>getLayer(int)</i>	Retourne la couche ayant l'identifiant unique spécifié.
<i>getLayer(String)</i>	Retourne la couche ayant le nom unique spécifié.
<i>getLayerPos(Layer)</i>	Retourne la position de la couche spécifiée.

<i>getLayerTreeVisibility()</i>	Retourne une structure de données (classe <i>LayerVisibilitySet</i>) contenant l'état de la visibilité d'une couche. Cet état tient compte de l'état de la sélection des groupes dans la hiérarchie de couches et de la configuration de visibilité de celles-ci.
<i>getSelectedElements()</i>	Retourne l'ensemble des éléments sélectionnés sur l'ensemble des couches.
<i>getSelectedExtent()</i>	Retourne l'étendue de l'ensemble des éléments sélectionnés sur l'ensemble des couches.
<i>removeLayer(int)</i>	Enlève la couche située à la position spécifiée.

Les événements de la classe LayerManager

Le gestionnaire de couches dans JMap génère des événements. Pour recevoir les événements générés par un gestionnaire de couches, vous devez enregistrer un *listener* sur le gestionnaire de couches en question. Notez qu'un *listener* enregistré sur le gestionnaire de couches recevra les événements générés par l'ensemble des couches prises en charge par le gestionnaire.

Pour recevoir les événements d'un gestionnaire de couches, vous devez implémenter l'interface *LayerEventListener* et l'enregistrer auprès du gestionnaire de couches en utilisant la méthode *addLayerEventListener()*.

Événements du gestionnaire de couches

<i>layerPosChangedEventOccurred</i> (<i>LayerPosChangeEvent</i>)	Lancé après un changement de position d'une couche dans la liste des couches prises en charge par le gestionnaire de couches.
<i>layerRemovedEventOccurred</i> (<i>LayerRemovedEvent</i>)	Lancé après l'enlèvement d'une couche de la liste des couches prises en charge par le gestionnaire de couches.
<i>layerAddedEventOccurred</i> (<i>LayerAddedEvent</i>)	Lancé après l'ajout d'une couche dans la liste des couches prises en charge par le gestionnaire de couches.

Concepts

Les applications JMap Pro sont développées sur une base modulaire permettant l'ajout de fonctionnalités sans trop de difficultés. Les applications peuvent être décomposées en trois niveaux, dont deux sont extensibles afin de permettre la programmation de fonctionnalités supplémentaires.

Le premier niveau est le point d'entrée de l'application (classe `JMapApplicationLauncher`) qui prend en charge le type de l'application (applet, Java Web Start ou Java autonome) et instancie la classe d'application (instance de la classe `JMapApplication`) à utiliser. Le deuxième niveau est l'application JMap Pro, dont la classe abstraite `JMapApplication` est le moteur et fournit tous les services nécessaires pour son bon fonctionnement. Puisque la classe `JMapApplication` ne fournit pas l'interface graphique de l'application, il est alors nécessaire d'instancier une classe qui hérite de la classe `JMapApplication` et qui viendra instancier les composants graphiques de l'application, tels que la hiérarchie des couches et les barres de boutons. La classe `DockingClient`, incluse dans ce SDK, est un bon exemple d'application JMap Pro. Le troisième niveau concerne les extensions JMap. En effet, les applications JMap Pro permettent le développement et l'utilisation de classes d'extension (classe `JMapClientExtension`) pour ajouter de nouvelles fonctionnalités dans les applications. Le développement des extensions JMap Pro est expliqué dans la section suivante.

Communication avec JMap Server

Lors de l'initialisation de l'application, une connexion est établie avec JMap Server selon les paramètres d'application spécifiés. La communication avec le serveur est unidirectionnelle et permet d'échanger des messages à l'aide de requêtes et de réponses. Les détails de cette communication sont expliqués dans cette section.

Services de l'application

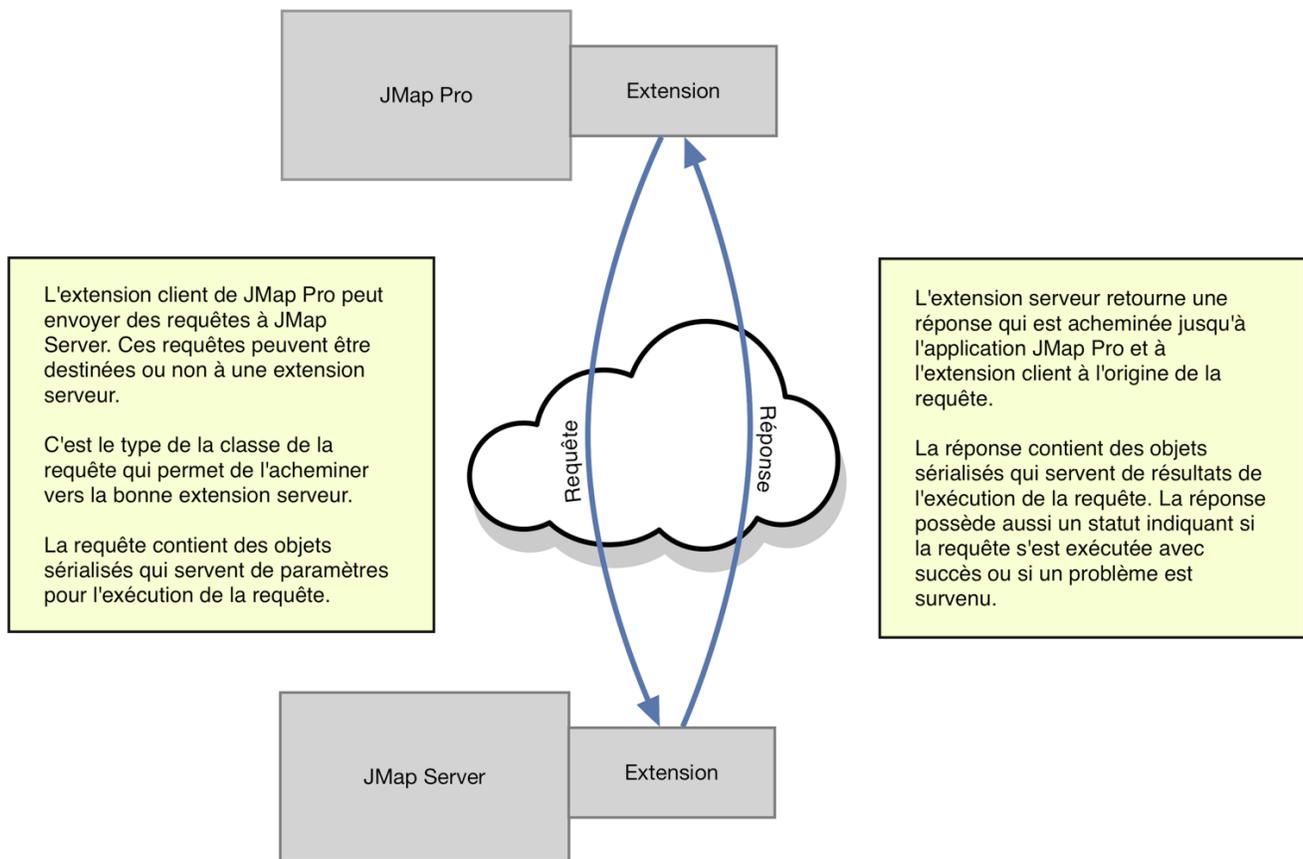
Voici les méthodes les plus utiles de la classe <code>JMapApplication</code>	
<code>addProjectListener(ProjectEventListener)</code>	Ajoute un écouteur de projet dans la liste des écouteurs de l'application.
<code>removeProjectListener(ProjectEventListener listener)</code>	Enlève un écouteur de projet de la liste des écouteurs de l'application.
<code>createNewView()</code>	Crée une nouvelle vue (classe <code>View</code>) initialisée avec le projet courant dans l'application.
<code>getCurrentProject()</code>	Retourne le projet actif (classe <code>Project</code>) dans l'application.
<code>getGuiService()</code>	Retourne l'instance de <code>JMapGuiService</code> à utiliser afin d'accéder et de manipuler les

	composants de l'interface graphique de l'application.
getLogger()	Retourne une instance de java.util.Logger permettant d'effectuer la journalisation dans l'application.
getClientExtension(String)	Retourne l'extension chargée (classe JMapClientExtension) pour le nom de classe spécifié.
getMessagingController()	Retourne le contrôleur de messagerie (classe JMapClientMessagingController) de l'application.
getEditionTransactionManager()	Retourne le contrôleur d'édition (classe EditionTransactionManager) de l'application.
getUserParameterController()	Retourne le contrôleur de paramètres utilisateurs (classe JMapUserParameterController).
getUserParameter(String)	Retourne le paramètre utilisateur (classe UserParameter) sauvegardé pour la clé spécifiée.
setUserParameter(UserParameter)	Définit un paramètre utilisateur à sauvegarder.

Contexte de l'application

Lors de l'initialisation de l'application, toutes les instances utiles qui définissent le contexte d'exécution de l'application sont stockés dans un singleton de la classe JMapApplicationContext.

Lors de l'ouverture de l'application JMap Pro, une connexion est établie vers le JMap Server associé. Cette connexion est utilisée pour acheminer les différentes requêtes systèmes, dont les requêtes de chargement de la configuration du projet et des données, ainsi que les requêtes d'extension.



La communication entre une application JMap Pro et JMap Server

La communication entre une application JMap Pro et JMap Server s'effectue par l'échange de requêtes et de réponses. Les requêtes et les réponses sont en fait des objets sérialisés que vous devez typiquement programmer et qui contiendront les propriétés nécessaires à l'exécution de la requête et au retour de l'information vers le client. Par exemple, pour une application de gestion des demandes des citoyens d'une ville, la requête et la réponse pourraient contenir les propriétés suivantes:

Requête	Réponse
nom du demandeur	statut de la sauvegarde de la demande dans la base de données
type de demande	identifiant unique généré à l'enregistrement
description	
coordonnées x et y de la localisation de la demande	

Exemple des propriétés d'une requête et d'une réponse

Pour plus d'information, consultez la section Programmation des requêtes d'extension.

Statut de la réponse

Chaque réponse possède un statut qui indique si la requête a été exécutée avec succès ou si un problème est survenu. La méthode `getStatus()` de la classe `JMapExtensionResponse` permet d'obtenir le statut de la réponse. Pour la liste des statuts possibles et leur description, référez-vous à la documentation de la classe `JMapExtensionResponse`.

Votre extension devrait toujours vérifier le statut d'une réponse avant de l'utiliser. Si le statut n'est pas égal à `JMAPSRV_STS_SUCCESS`, un traitement spécial doit être fait afin de gérer la situation d'erreur. Dans ce cas, la méthode `getMessage()` permet d'obtenir un message explicatif décrivant la cause de l'erreur.

Envoi des requêtes à JMap Server

La classe `JMapSrvConnection` est responsable de toutes les communications entre JMap client et JMap Server. Il est possible d'accéder à l'instance de la connexion en utilisant le contexte de l'application JMap tel que montré dans l'exemple suivant.

```
JMapSrvConnection jmapConn = JMapApplicationContext.getInstance().getConnection();
```

Les méthodes suivantes (héritées de la classe `JMapNetworkConnection`) sont utilisées pour l'envoi des requêtes à JMap Server et pour la réception des réponses.

Méthode	Description
<code>executeRequest()</code>	Envoie la requête passée en paramètre à JMap Server et retourne la réponse produite par JMap Server ou par une extension serveur. Cette méthode est bloquante. Cette méthode convient bien aux situations où il faut attendre la réponse du serveur avant de continuer.
<code>pushRequest()</code>	Envoie la requête passée en paramètre à JMap Server. Cette méthode est non bloquante. Lorsque la réponse sera reçue, la méthode <code>callback()</code> du client de la requête sera appelée. Cette méthode convient bien aux situations où il faut redonner le contrôle à l'utilisateur rapidement, et quand on peut traiter la réponse de manière asynchrone.

L'exemple de code suivant montre comment utiliser la méthode `executeRequest`.

```
MyRequest request = new MyRequest("This is a test", 555);
JMapSrvConnection jmapConn = JMapApplicationContext.getInstance().getConnection();
MyResponse response = (MyResponse)jmapConn.executeRequest(request); // Execution WILL block

if (response.getStatus() == MyResponse.JMAPSRV_STS_SUCCESS)
{
    // Do something useful with the response
}
else
{
    // Handle error here
}
```

L'exemple de code suivant montre comment utiliser la méthode `pushRequest`.

```
MyRequest request = new MyRequest("This is a test", 555);
request.setClient(new JMapRequestClient()
{
    // Will be called when the response is received from JMapServer
    @Override
    public void callback(JMapRequest request, JMapResponse response)
    {
        if (response.getStatus() == MyResponse.JMAPSRV_STS_SUCCESS)
        {
            // Do something useful with the response
        }
        else
        {
            // Handle error here
        }
    }
});
JMapSrvConnection jmapConn = JMapApplicationContext.getInstance().getConnection();
jmapConn.pushRequest(request); // Execution WILL NOT block here
```

Les tableaux suivants présentent les clés des composantes d'interface graphique de JMap Pro. Ces clés sont utilisées pour le contrôle de ces composantes par programmation, notamment dans les méthodes de la classe `JMapGuiService`.

Barres d'outils	
ZOOM_PAN	Barre d'outils contenant les boutons de zoom et de déplacement.
SELECTION	Barre d'outils contenant les boutons de sélection et de désélection.
ROTATION	Barre d'outils contenant les boutons de rotation et d'annulation de la rotation.
MEASURE_LABEL	Barre d'outils contenant les boutons des outils de mesure, d'effacement des mesures, d'étiquetage et d'effacement des étiquettes.
INFOS_SEARCH	Barre d'outils contenant les boutons des outils de rapport et de recherche.
PRINT_MAIL	Barre d'outils contenant les boutons des outils d'impression et d'envoi par courriel.

Menus	
PROJECT	Menu <i>Projets</i> contenant les items de chargement de projets, de gestion des contextes, de gestion des couches personnelles, etc.
VIEW	Menu <i>Affichage</i> contenant les items de contrôle de l'affichage des différentes composantes de l'interface graphique tels que le gestionnaire de couches, la vue d'ensemble, etc.
TOOLS	Menu <i>Outils</i> contenant les items de divers outils et extensions.
MAP	Menu <i>Cartes</i> contenant les items de de gestion des fenêtres de cartes.
HELP	Menu <i>Aide</i> contenant les items de rubriques d'aide.

Fenêtres	
LAYERS	Gestionnaire de couches cartographiques.
CONTEXTS	Fenêtre de gestion des contextes.
ELEMENTS_EXPLORER	Fenêtre contenant les explorateurs d'éléments des couches.
SELECTION_EXPLORER	Fenêtre contenant l'explorateur de sélection.

Fenêtres	
MESSAGES_VIEWER	Fenêtre de gestion des messages JMap.
PERSONALLAYERS	Fenêtre de gestion des couches personnelles.
QUERIES	Fenêtres des requêtes par attributs et spatiales.
GO_TO_COORDINATE	Fenêtre permettant de saisir une coordonnée à atteindre sur la carte.
GEOMETRY_INFO_PANEL	Fenêtre d'affichage des propriétés géométriques des éléments cartographiques.
OVERVIEW	Fenêtre de la vue d'ensemble.

Un outil JMap permet de fournir une interaction entre l'utilisateur et la carte au moyen de la souris. Par exemple, lors que l'utilisateur active l'outil de sélection et clique sur la carte pour sélectionner un élément, c'est la classe de l'outil de sélection qui effectue le travail. Cet outil est programmé pour effectuer une sélection à l'endroit cliqué. De même, lorsque l'utilisateur utilise l'outil de mesure de distance, c'est la classe de l'outil qui calcule et affiche la distance entre 2 points cliqués par l'utilisateur. Le développement d'outils JMap vous permet donc d'implémenter des actions personnalisées.

En général, un seul outil à la fois peut être en fonction. Pour qu'un outil devienne en fonction, il doit devenir l'outil actif d'une vue JMap. Pour se faire, il faut utiliser la méthode `setCurrentTool(Tool)` de la classe `View` ou de la classe `ViewManager`. Notez que le moyen utilisé pour rendre l'outil actif, tel qu'un bouton ou un item de menu, n'a aucun lien avec le fonctionnement de l'outil lui-même.

Lorsque l'utilisateur change l'outil actif (par exemple en appuyant sur un bouton), voici essentiellement le code qui est appelé.

```
JMapApplicationContext.getInstance().getViewManager().setCurrentTool(new MyTool());
```

Quand un outil est actif, il reçoit tous les événements de souris qui sont générés par la vue active. C'est la responsabilité de la classe de l'outil de traiter ces événements et de prendre les actions voulues.

Pour développer un nouvel outil, vous devez programmer une classe dérivée de la classe abstraite *Tool* et implémenter uniquement les méthodes dont vous avez besoin pour faire le travail de l'outil. Par exemple, si l'outil doit faire une action lorsque l'utilisateur effectue un clic de souris, vous devez implémenter la méthode *onToolClicked()* .

Méthodes de la classe <i>Tool</i>	
<i>init()</i>	Cette méthode est appelée lorsque l'outil devient l'outil actif d'une vue. Le code de cette méthode devrait servir, au besoin, à préparer le travail de l'outil. La vue en question est passée en paramètre à cette méthode.
<i>getCursor()</i>	Cette méthode est appelée par la vue quand l'outil devient actif afin de permettre à votre outil de fournir son propre curseur de souris. Ce curseur sera visible sur la vue tant que votre outil restera l'outil actif.
<i>onToolPressed()</i>	Cette méthode est appelée quand l'utilisateur enfonce l'un des boutons de la souris à l'intérieur de la vue.
<i>onToolReleased()</i>	Cette méthode est appelée quand l'utilisateur relâche l'un des boutons de la souris à l'intérieur de la vue.
<i>onToolClicked()</i>	Cette méthode est appelée après que l'utilisateur ait complété un clic de souris à l'intérieur de la vue.
<i>onToolMoved()</i>	Cette méthode est appelée à répétition lorsque l'utilisateur déplace la souris à l'intérieur de la vue.
<i>onToolDragged()</i>	Cette méthode est appelée à répétition lorsque l'utilisateur déplace la souris à l'intérieur de la vue, tout en maintenant un bouton enfoncé.
<i>terminate()</i>	Cette méthode est appelée lorsque l'outil devient inactif, c'est-à-dire lorsqu'un autre outil devient actif sur la vue. Le code de cette méthode pourrait servir, au besoin, à faire un travail de fermeture ou de libération de ressources.

L'exemple de code suivant montre un outil simple qui, lorsque l'utilisateur appuie et relâche le bouton gauche de la souris sur la carte, affiche les propriétés du premier élément trouvé à la coordonnée du curseur de la souris. Seules les couches vectorielles sont considérées et la recherche se fait de la couche la plus haute vers la couche la plus basse, dans l'ordre d'affichage des couches. Notez que les autres méthodes de la classe *Tool* ne sont pas implémentées puisqu'elles ne sont pas requises pour le fonctionnement de l'outil.

```
public class ElementInfoTool extends Tool
{
    public void onToolReleased(MouseEvent e)
    {
        super.onToolReleased(e);
    }
}
```

```
System.out.println("ElementInfoTool.onToolReleased");

// Obtain array of layers to cycle them in reverse order
final Layer aLayers[] = this.view.getLayerManager().getAllLayers();

// Get the layer visibility status from the layer manager (also includes the
// hierarchy visibility)
final LayerVisibilitySet layersVisibility =
this.view.getLayerManager().getLayerTreeVisibility();

// Transform mouse x,y coordinate to WC coordinate
// Point wcCoord = view.getTransform().transformInv(new Point.Double(e.getX(),
e.getY()));
final Point wcCoord = toWCPoint(e); // method inherited from class Tool

final ViewState viewState = this.view.getViewState();

// Cycle through every layer in reverse order (from top position to bottom
// position) looking for an element under the x,y position.
for (int i = aLayers.length - 1; i >= 0; i--)
{
    // Consider only vector layers
    if (!(aLayers[i] instanceof VectorLayer))
        continue;

    final VectorLayer vectorLayer = (VectorLayer) aLayers[i];

    // Layer must be visible, selectable and displayed at the current scale
    if (layersVisibility.isVisible(vectorLayer.getId()) && vectorLayer.isSelectable()
&& vectorLayer.isDrawable(viewState.getScale()))
    {
        final K2DElement elem = vectorLayer.getElementAtPoint(wcCoord, viewState, true);

        if (elem != null)
        {
            String message = "Selected element on layer " + vectorLayer.getName() + ":
\n\n"
                + "Class:" + elem.getClass().getName() + '\n'
                + "Id: " + elem.getId() + '\n'
                + "Geometry: " + elem.getGeometry().getClass() + '\n'
                + "Display bounds: " + elem.getDisplayBounds(viewState,
vectorLayer.getStyle(elem, viewState.getScale())) + '\n'
                + "Attributes:\n";

            final Object[] attribs = elem.getAttributes();
            final Attribute[] attribDefinitions = vectorLayer.getAttributeMetaData();

            for (int j = 0; j < attribs.length; j++)
            {
                message += " " + attribDefinitions[j].getName() + " : " + attribs[j] +
'\n';
            }

            JOptionPane.showMessageDialog(this.view, message);

            break;
        }
    }
}
```

```

    }
  }
}

```

Outils de dessin

Il est simple d'implémenter des outils qui dessinent sur la carte en dérivant simplement des classes des outils de dessin existantes de JMap.

En dérivant de ces classes, tout le fonctionnement de base est hérité. Les options suivantes sont disponibles:

- Modes persistant et volatile (détermine si des éléments sont créés);
- Sélection de la couche qui recevra les éléments dessinés;
- Paramètres du style des éléments dessinés (couleurs, types de trait, etc.);
- Affichage des dimensions sur la carte pendant le dessin.

Le tableau suivant présente la liste des classes des outils de dessin disponibles dans JMap. Toutes ces classes sont elles-mêmes dérivées de la classe *ToolDraw*.

Classes de dessin dérivées de ToolDraw	
<i>ToolDrawBox</i>	Dessine une boîte (rectangle) sur la carte.
<i>ToolDrawCircle</i>	Dessine un cercle sur la carte.
<i>ToolDrawLine</i>	Dessine une ligne simple sur la carte.
<i>ToolDrawMultiLine</i>	Dessine une multiligne sur la carte.
<i>ToolDrawPolygon</i>	Dessine un polygone sur la carte.

Le tableau suivant présente la liste des méthodes les plus couramment utilisées de la classe *ToolDraw*.

Méthodes de la classe ToolDraw	
<i>getStyleContainer()</i>	Retourne l'objet de type <i>StyleContainer</i> qui contient les styles des différents types d'éléments (polygones, lignes, etc.) qu'il est possible de dessiner. C'est la méthode à appeler pour modifier le style des éléments qui seront dessinés par l'outil.
<i>setDrawLayer(VectorLayer)</i>	Spécifie la couche qui va recevoir les éléments dessinés par l'outil, si les éléments sont persistants (voir méthode

	<i>setPersistant(boolean)</i> plus bas). Si aucune couche n'est spécifiée, une couche système est utilisée par défaut.
<i>setPersistant(boolean)</i>	Détermine si les éléments seront stockés sur une couche ou non (voir méthode <i>setDrawLayer(VectorLayer)</i> plus haut). S'ils ne sont pas stockés, les éléments disparaissent aussitôt l'opération de dessin terminée.

L'exemple de code suivant montre un outil qui dérive de *ToolDrawPolygon* et qui hérite de toutes les capacités de dessin de celle-ci.

```
public class CreatePolygonTool extends ToolDrawPolygon
{
    private final static String LAYER_NAME = "Zones";

    private VectorLayer targetLayer; // The layer that will hold the polygons

    public void init(View view)
    {
        super.init(view);

        // Make sure the layer exists. If not create it and register it in the
        // layer manager of the view.
        final LayerManager layerMgr = view.getLayerManager();

        this.targetLayer = (VectorLayer)layerMgr.getLayer(LAYER_NAME);

        if (this.targetLayer == null)
        {
            // The layer does not exist, create it
            this.targetLayer = new VectorLayer(Layer.getNextUserLayerId(), // avoid id conflict
                LAYER_NAME,
                ElementTypes.TYPE_POLYGON);

            // Tell JMap that information on this layer does not come from the
            // server
            this.targetLayer.setLocal(true);

            // Set the mouse-over text for the layer. It will be displayed
            // when the user stops the mouse pointer over an element.
            this.targetLayer.setMouseOverConfiguration(new MouseOverConfiguration("This is a zone"));

            // Add the layer to the layer manager list. All layers need to be
            // registered in the layer manager in order to be displayed in the
            // view. The layer is added to the top.
            layerMgr.addLayer(this.targetLayer);
        }

        // Tell the superclass that the resulting elements should be persisted
        // when completed. That means that they will be placed on a layer.
        // Otherwise, they would be deleted as soon as they are completed.
        setPersistent(true);

        // Tell the superclass to place the resulting elements on the target layer.
        // Otherwise, they would be placed on the default drawing layer
        // (id = LayerManager.LAYER_ID_USER_DRAWINGS).
        setDrawLayer(this.targetLayer);

        // Modify the style of the surface elements (polygons).
        final Style surfaceStyle = getStyleContainer().getSurfaceStyle();

        surfaceStyle.setFill(Color.GREEN);
        surfaceStyle.setTransparency(.50f);

        // To have the valid drawing style displayed in the layer bar, make the layer
        // default style identical to the drawing style.
        this.targetLayer.setStyle(surfaceStyle, 0.);
    }
}
```

```
}
```

Les extensions de JMap Pro sont des modules développés en langage Java qui se greffent à l'application JMap Pro afin de lui ajouter des nouvelles fonctionnalités. Les extensions sont spécifiées en paramètres des applications et elles sont initialisées lors du démarrage de ces dernières. Typiquement, les extensions s'intègrent dans l'interface graphique d'une application JMap Pro en insérant des boutons et des menus permettant d'activer les fonctions fournies par l'extension.

Pour développer et rendre disponible une extension aux utilisateurs, vous devez effectuer les 2 étapes suivantes:

1. Développer votre extension en créant une classe Java qui dérive de la classe `JMapClientExtension` ;
2. Déployer votre extension dans JMap Server afin qu'elle soit accessible par les applications JMap Pro.

Consultez les sections suivantes pour plus d'information.

Pour programmer des extensions de JMap Pro, il est important de respecter un ensemble de règles simples. La présente section décrit les règles de programmation.

La classe `JMapClientExtension`

La première étape pour le développement d'une extension de JMap Pro consiste à programmer une classe qui dérive de la classe abstraite `JMapClientExtension` . Cette classe comporte les 3 méthodes suivantes, appelées à différents moments du cycle de vie de l'extension:

Méthodes de la classe `JMapClientExtension`

`init(JMapApplicationContext,
Map<String, String>)`

Cette méthode est appelée lorsque l'extension est chargée par l'application JMap Pro. À ce moment, les interfaces graphiques ne sont pas encore initialisées. Vous pouvez mettre dans cette méthode tout code servant à préparer le fonctionnement de votre extension. Cela pourrait inclure le chargement d'un fichier de paramètres, les vérifications de dépendances, etc. La méthode reçoit en paramètre une instance de `JMapApplicationContext` . Cette classe permet d'accéder à l'ensemble des ressources de l'application JMap.

	La méthode reçoit aussi une collection (Map) de paramètres. Ces paramètres sont des informations utiles pour l'extension. Voir plus bas pour plus d'information.
<i>initGUI()</i>	Cette méthode est appelée par l'application JMap lorsque son interface graphique a été initialisée. Dans cette méthode, vous devez mettre le code qui initialisera l'interface graphique de votre extension. Cela pourrait inclure l'ajout de boutons ou de barres d'outils, de menus, de fenêtres, etc. Voir la section Intégration dans l'interface graphique pour plus d'information. Cette interface graphique permettra aux utilisateurs d'accéder aux fonctions offertes par votre extension.
<i>destroy()</i>	Cette méthode est appelée par l'application JMap juste avant que celle-ci se termine. Vous devez mettre dans cette méthode le code nécessaire pour terminer votre extension. Cela pourrait inclure du code servant à libérer des ressources, enregistrer des paramètres, etc.

Des paramètres sont passés à l'extension par la méthode *init()*, à l'aide d'une collection de type *Map<String, String>*. Il existe 2 paramètres qui sont prédéfinis dans JMap et correspondent à des constantes définies dans la classe *JMapClientExtension*.

- EXTENSION_PARAMETER_GUI_VISIBLE : Indique à l'extension si ses composantes d'interfaces graphiques (autres que les barres d'outils) doivent être visibles à l'ouverture de l'application.
- EXTENSION_PARAMETER_TOOLBAR_VISIBLE : Indique à l'extension si ses barres d'outils doivent être visibles à l'ouverture de l'application.

Ces paramètres contiennent les valeurs *true* ou *false* et sont renseignés par l'Administrateur JMap lors du déploiement d'une application JMap Pro. Votre extension doit respecter ces paramètres. Une extension peut aussi recevoir des paramètres qui lui sont propres et qui sont renseignés par l'administrateur JMap.

L'exemple de code suivant montre comment accéder aux paramètres depuis la méthode *init()*.

```
public void init(JMapApplicationContext appContext, Map<String, String> mapExtensionParam
{
    String param = mapExtensionParameters.get(JMapClientExtension.EXTENSION_PARAMETER_TOOLB
    boolean toolbarVisible = Boolean.parseBoolean(param);

    ...
}
```

La classe JMapApplicationContext

La classe *JMapApplicationContext* est très utile dans le développement des extensions car elle permet d'accéder à un ensemble de ressources générales de l'application. Cette classe est un *singleton*. On peut donc avoir accès facilement à l'instance de n'importe où en utilisant la méthode statique *JMapApplicationContext.getInstance()*.

Méthodes les plus utiles de la classe *JMapApplicationContext*

<i>getConnection()</i>	Retourne l'instance de la connexion (classe <i>JMapSrvConnection</i>) vers JMap Server.
<i>getFrame()</i>	Retourne l'instance de la fenêtre principale de l'application.
<i>getViewManager()</i>	Retourne l'instance du gestionnaire de vues (classe <i>ViewManager</i>) de l'application.
<i>getApplication()</i>	Retourne l'instance de l'application JMap (classe <i>JMapApplication</i>).
<i>getJMapHome()</i>	Retourne le chemin vers le dossier principal de JMap sur l'ordinateur de l'utilisateur. C'est à cet endroit que JMap écrit ses données.

La classe JMapApplication

La classe *JMapApplication* représente l'application JMap. Elle fournit des méthodes qui donnent accès à des ressources de l'application (interface graphique, journaux d'événements, etc.). Elle fournit aussi des méthodes qui permettent d'effectuer des tâches de manière simple (créer une nouvelle carte, fermer le projet, etc.).

Méthodes les plus utiles de la classe *JMapApplication*

<i>createNewView()</i>	Crée une nouvelle vue cartographique qui s'affiche automatiquement dans l'application.
<i>getCurrentProject()</i>	Retourne l'instance du projet (classe <i>Project</i>) ouvert dans l'application.
<i>getGuiService()</i>	Retourne l'instance de la classe <i>JMapGuiService</i> de l'application. Voir la section Intégration dans l'interface graphique plus d'information.
<i>getLogger()</i>	Retourne l'instance de la classe <i>Logger</i> de l'application. Utilisée pour consigner des messages de votre extension dans les journaux de l'application JMap.

Comme développeur d'extensions pour JMap Pro, vous aurez probablement à développer des applications qui doivent effectuer des communications entre le côté client et JMap Server. Que ce soit pour communiquer avec votre propre extension côté serveur ou pour effectuer des requêtes générales à JMap Server, le principe de programmation reste le même.

Pour plus d'information sur la communication client-serveur de JMap, consultez la section Communication client-serveur.

Programmation de la requête

Pour créer votre classe de requête, vous devez respecter les 2 règles suivantes :

- la classe doit dériver de la classe *JMapExtensionRequest* ;
- toutes les propriétés de la classe doivent être sérialisables.

Vous êtes libres d'ajouter toute les propriétés et méthodes qui sont utiles pour l'exécution de votre requête. Ces propriétés et méthodes pourront être exploitées du côté serveur par votre extension.

L'exemple de code source suivant montre comment programmer une requête simple.

```
// This request is used to save a new citizen complaint
public class SaveComplaintExtensionRequest extends JMapExtensionRequest
{
    private String citizenName;
    private int requestType;

    public void setCitizenName(String citizenName)
    {
        this.citizenName = citizenName;
    }

    public String getCitizenName()
    {
        return this.citizenName;
    }

    public void setRequestType(int requestType)
    {
        this.requestType = requestType;
    }

    public int getRequestType()
    {
        return this.requestType;
    }
}
```

Programmation de la réponse

Pour créer votre classe de réponse, vous devez respecter les 2 règles suivantes :

- la classe doit dériver de la classe `JMapExtensionResponse` ;
- toutes les propriétés de la classe doivent être sérialisables.

Vous êtes libres d'ajouter toutes les propriétés et méthodes qui sont utiles pour retourner l'information au client relativement à l'exécution de la requête. Ces propriétés et méthodes pourront être exploitées du côté client par votre extension suite à l'exécution de la requête sur le serveur.

L'exemple de code source suivant montre comment programmer une réponse simple.

```
// This response is returned to client after a citizen complaint save request was executed
public class SaveComplaintExtensionResponse extends JMapExtensionResponse
{
    private long uniqueId;

    public void setUniqueId(long uniqueId)
    {
        this.uniqueId = uniqueId;
    }

    public long getUniqueId()
    {
        return this.uniqueId;
    }
}
```

En étant dérivée de `JMapExtensionResponse`, votre classe de réponse hérite de quelques propriétés utiles dont le statut de la réponse et un message explicatif en cas de problème.

Statut de la réponse

Chaque réponse possède un statut qui indique si la requête a été exécutée avec succès ou si un problème est survenu. La méthode `getStatus()` de la classe `JMapExtensionResponse` permet d'obtenir le statut de la réponse. Pour la liste des statuts possibles et leur description, référez-vous à la documentation de la classe `JMapExtensionResponse`.

Votre extension devrait toujours vérifier le statut d'une réponse avant de l'utiliser. Si le statut n'est pas égal à `JMAPSRV_STS_SUCCESS`, un traitement spécial doit être fait afin de gérer la situation d'erreur. Dans ce cas, la méthode `getMessage()` permet d'obtenir un message explicatif décrivant la cause de l'erreur.

Certaines extensions client peuvent être complètement invisibles pour l'utilisateur mais la majorité des extensions s'intègrent à l'interface graphique de l'application JMap Pro. Cette intégration peut prendre plusieurs formes. C'est dans la méthode `initGui()` que l'extension doit faire l'initialisation de son interface graphique.

La classe JMapGuiService

La classe *JMapGuiService* est utilisée pour accéder aux composantes de l'interface graphique (menus, barres d'outils, etc.) et pour ajouter ou enlever des composantes. Pour faciliter l'accès aux composantes graphiques, chaque composante possède une clé unique. Cette clé doit être fournie lorsque une composante est ajoutée ou enlevée. Pour la liste complète des clés disponibles, consultez la section Liste des composantes d'interface graphique. La classe *JMapGuiFactory* décrite plus bas fournit des méthodes pour créer des composantes de l'interface graphique.

Les principales méthodes de la classe *JMapGuiService* sont:

Méthodes les plus utiles de la classe <i>JMapGuiService</i>	
<i>addMenuItem()</i>	Permet d'ajouter un item à un menu existant de l'application en spécifiant la clé du menu.
<i>getMenu()</i>	Permet d'obtenir un menu en utilisant sa clé. Il est ensuite possible d'utiliser l'objet retourné pour modifier le menu.
<i>addToolBar()</i>	Permet d'ajouter une barre d'outils à une position spécifique dans l'interface de l'application.
<i>getToolBar()</i>	Permet d'obtenir une barre d'outils en utilisant sa clé. Il est ensuite possible d'utiliser l'objet retourné pour modifier la barre d'outils.
<i>addDockableComponent()</i>	Permet d'ajouter une fenêtre dockable à une position spécifique dans l'interface de l'application.
<i>removeDockableComponent()</i>	Permet d'enlever une fenêtre dockable en utilisant sa clé.
<i>getGuiFactory()</i>	Retourne l'instance de <i>JMapGuiFactory</i> . Cette classe permet de créer des composantes d'interface graphique telles que des boutons et des barres d'outils.

La classe JMapGuiFactory

La classe *JMapGuiFactory* est utilisée pour créer les boutons, menus et barres d'outils compatibles avec les applications JMap. Les méthodes *createButton()*, *createMenu()*, *createToolBar()*, etc. sont appelées pour créer les différentes composantes correspondantes.

Exemples d'intégration

Ajout d'un bouton sur une barre d'outils existante

L'exemple de code suivant montre comment ajouter un bouton sur une barre d'outils existante. La barre d'outils est retrouvée par sa clé. Pour la liste complète des clés disponibles, consultez la section Liste des composants d'interface graphique de JMap Pro. Notez que l'action associée au bouton n'est pas montrée dans l'exemple.

```
// This button will be in a group thus only one button can be pressed among this group.
AbstractButton button = guiFactory.createToggleButton(new ButtonAction(),
    appContext.getApplication().getGuiService().getMainButtonGroup());

// Add the button on the Zoom and Pan toolbar.
appContext.getApplication().getGuiService().getToolBar("ZOOM_PAN").add(button);
```

Ajout d'une barre d'outils

L'exemple de code suivant montre comment créer une barre d'outils à l'aide de la classe *JMapGuiFactory*. Notez que dans cet exemple la classe *JMapApplicationActionManager* est utilisée pour accéder aux actions des boutons. Cette classe permet de gérer des instances uniques des actions. Ensuite, 2 boutons sont créés et ajoutés à la barre. Finalement, la barre d'outils est ajoutée à la verticale, du côté droit de l'application. Remarquez le *HashMap* de propriétés qui est utilisé pour placer la barre d'outils. Ce principe est utilisé avec tous les composants pour préciser leurs caractéristiques.

```
// Create the new toolbar
final ToolBar toolbar = guiFactory.createToolBar("SHOWCASE", "JMap example showcase");

// Create and add buttons on toolbar
toolbar.add(
    guiFactory.createToggleButton(applicationActionManager.getAction(CreatePolygonAction)
    );

toolbar.add(
    guiFactory.createButton(applicationActionManager.getAction(ShowDockingPanelAction.class)
    );

// Adds the new toolbar vertically to the right side of the application frame.
HashMap<String, Object> properties = new HashMap<String, Object>();
properties.put("CONTEXT_INIT_SIDE", new Integer(SwingConstants.EAST)); // Dock to the east
jmapGuiService.addToolBar(toolbar, properties);
```

Ajout d'une fenêtre

La méthode *addDockableComponent()* permet d'ajouter des composants de type fenêtre dans l'interface. Elle prend en paramètre un *HashMap* de paires clé-valeur pour définir les paramètres d'affichage de la nouvelle fenêtre. L'ensemble des clés et valeurs possibles est défini dans la documentation de la méthode.

```
// Create a panel with the needed components
final JPanel panel = new JPanel();
...

// Set the properties of the dockable panel.
HashMap<String, Object> properties = new HashMap<String, Object>();
properties.put("KEY", COMPONENT_KEY); // Will be used to reference the panel
properties.put("TITLE", "JMap 6.5 SDK showcase."); // Will be displayed on the title bar
properties.put("CONTEXT_INIT_SIDE", SwingConstants.EAST); // Tells JMap to dock the panel

appContext.getApplication().getGuiService().addDockableComponent(panel, properties);
```

Ajout d'un menu

L'exemple de code suivant montre comment créer et ajouter un menu à l'interface.

```
// Create menu and add items to it
JMenu menu = new JMenu("SDK");

JMenuItem item1 = new JMenuItem("Item 1");
JMenuItem item2 = new JMenuItem("Item 2");

this.menu.add(item1);
this.menu.add(item2);

// Add menu to menu bar at position 4
appContext.getApplication().getGuiService().getMenuBar().add(menu, 4);
```

Ajout d'un item de menu à un menu existant

L'exemple de code suivant montre comment ajouter des items aux menus existants de l'application. Remarquez qu'une clé est utilisée pour retrouver les menus existants. Pour la liste complète des clés disponibles, consultez la section Liste des composantes d'interface graphique.

```
// Add 2 menu items to existing menus TOOLS and HELP
JMenuItem item3 = new JMenuItem("Item 3");
JMenuItem item4 = new JMenuItem("Item 4");

appContext.getApplication().getGuiService().addMenuItem("TOOLS", item3, false);
appContext.getApplication().getGuiService().addMenuItem("HELP", item4, false);
```

Ajout d'un item au menu contextuel de la carte

L'exemple de code suivant montre comment ajouter une section d'items au menu contextuel de la carte. Ce dernier s'affiche quand l'utilisateur clique avec le bouton de droite sur la carte. De plus,

l'item de menu n'est actif que si l'utilisateur a cliqué sur au moins un éléments d'une couche de la carte. Dans le cas contraire, l'item de menu sera désactivé.

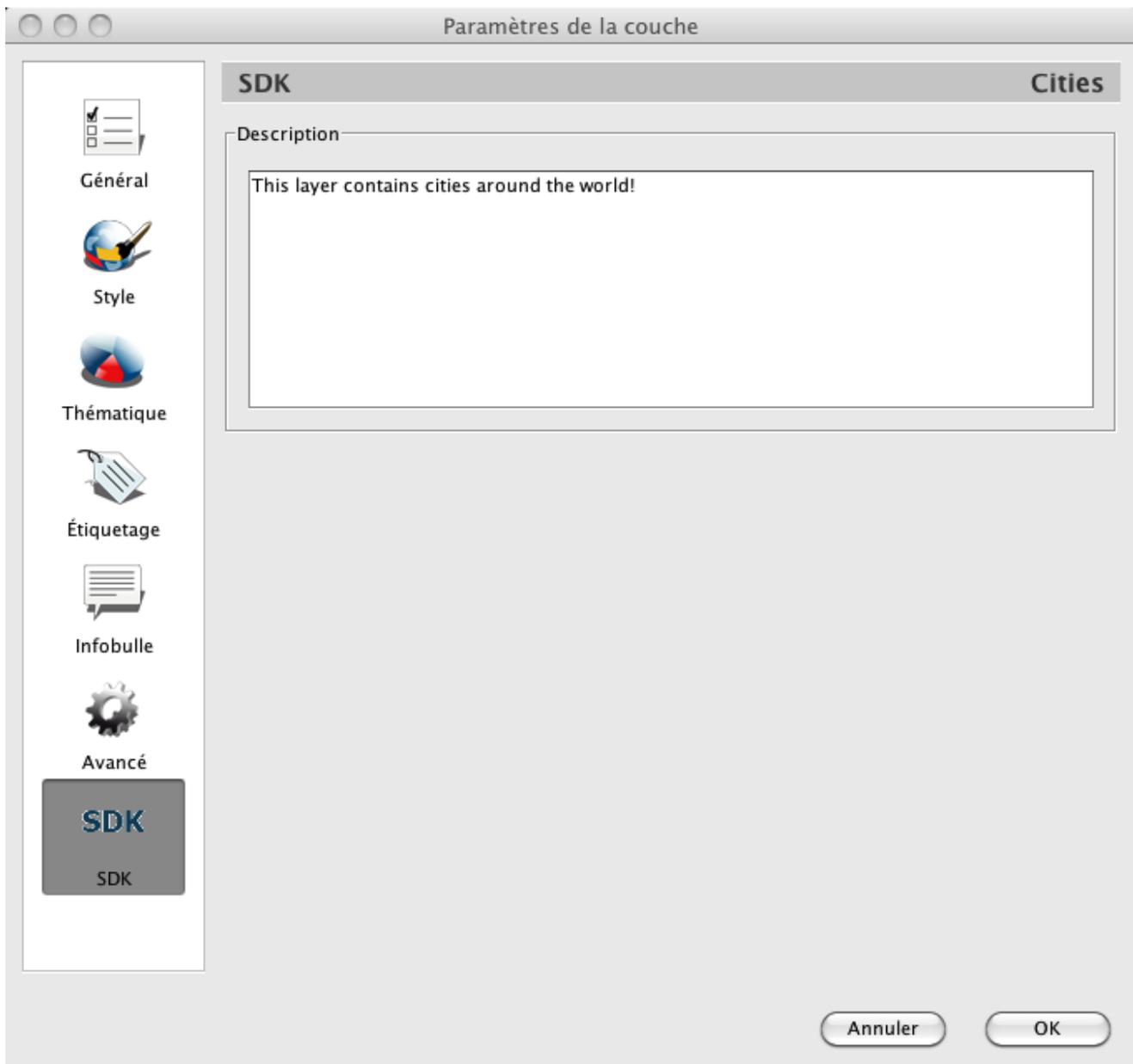
```
final ViewMenuAction menuAction = new ViewMenuAction();

// Obtain the currently active view
View view = JMapApplicationContext.getInstance().getViewManager().getActiveView();
if (view != null)
{
    // Add a separator and a menu item
    view.addPopupMenuSeparator();
    view.addPopupMenuAction(menuAction);
}

// Register an event listener with the View Manager so that we are notified when the view
JMapApplicationContext.getInstance().getViewManager().addViewEventListener(new ViewAdapte
{
    @Override
    public void viewPopupMenuShowing(ViewPopupMenuShowingEvent e)
    {
        // Enable map menu item only if the user clicked on some element
        K2DElement[] elements = e.getView().getLayerManager().getElementsAtPoint(e.getWcCoord
        menuAction.setEnabled(elements.length > 0);
    }
});
```

Ajout d'une section à la fenêtre de paramètres des couches

Il est possible d'ajouter des sections supplémentaires dans la fenêtre des paramètres d'une couche, tel que montré dans l'image ci-dessous. Pour ce faire, vous devez implémenter votre propre classe dérivée de la classe abstraite *AbstractLayerSettingsCustomPage* et implémenter ses 3 méthodes abstraites. Vous devez fournir un titre et une icône pour votre section en les passant au constructeur de la super classe. Ceux-ci seront affichés dans la fenêtre des paramètres.



Section supplémentaire ajoutée pour une couche

Méthodes abstraites de la classe *AbstractLayerSettingsCustomPage*

<i>getContentPanel()</i>	Dans cette méthode, vous devez fabriquer et retourner l'instance de <i>JPanel</i> qui affiche l'interface graphique des paramètres de votre section. C'est cette interface graphique qui s'affichera.
<i>validateSettings()</i>	Cette méthode est appelée lorsque l'utilisateur appuie sur OK pour confirmer les changements et fermer la fenêtre. Votre méthode doit retourner vrai seulement si

	les paramètres entrés sont valides. Dans le cas contraire, la fenêtre refusera de se fermer.
<code>applySettings()</code>	Cette méthode est appelée lorsque l'utilisateur appuie sur OK pour confirmer les changements et fermer la fenêtre, et après l'appel de la méthode <code>validateSettings()</code> . Vous devez appliquer les changements de paramètres sur la couche.

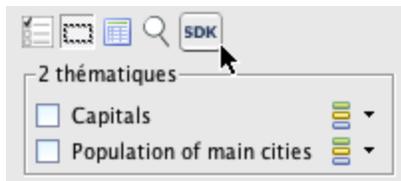
La méthode `getLayer()` de la classe `AbstractLayerSettingsCustomPage` vous permet de connaître à quelle couche les paramètres s'appliquent. Pour enregistrer votre section de paramètres, vous devez utiliser la méthode `addLayerSettingsCustomPage` de la classe `ShowLayerSettingsAction` tel que montré dans l'exemple de code source suivant.

```
// Obtain a reference to the ShowLayerSettingsAction instance
ShowLayerSettingsAction showLayerSettingsAction = (ShowLayerSettingsAction)appContext.get
.get

// Add the custom panel for the Cities layer
showLayerSettingsAction.addLayerSettingsCustomPage(SDKLayerSettingsPanel.class.getName())
```

Ajout d'un bouton personnalisé sur le gestionnaire de couches

Vous pouvez ajouter des boutons associés à une couche dans le gestionnaire de couches. Ces boutons peuvent servir à déclencher des actions personnalisées spécifiques à une couche. Les boutons s'ajoutent de manières différentes dans la section hiérarchique et dans la section liste.



Exemple de bouton personnalisé ajouté à une couche

L'exemple de code source suivant montre comment ajouter un bouton pour la couche `Cities` .

```
// Obtain layer instance from layer manager
Layer layerCities = appContext.getViewManager().getLayerManager().getLayer("Cities");

JMapGuiService guiService = appContext.getApplication().getGuiService();
JMapGuiFactory guiFactory = guiService.getGuiFactory();

AbstractButton abstractButton = guiFactory.createButton(new ButtonAction());

// Add the button to the 'More options' of the LayerTreeBar (hierarchy) for the layer Cities
guiService.getLayerTreeBar().addCustomButton(abstractButton, layerCities.getId());
```

```
// Add the button to the LayerPanel of the LayerBar (list) for the layer Cities
LayerPanel layerPanel = guiService.getLayerBar().getLayerPanel(layerCities.getId());
Button button = new Button(new ButtonAction(), false, new Dimension(18, 18));
layerPanel.addCustomButton(button);
```

Pour être déployée à l'intérieur des applications JMap Pro, les extensions client doivent respecter certaines règles. Si ces règles sont bien respectées, l'extension apparaît dans la section de déploiement de JMap Admin.

1 - Regrouper les classes de l'extension dans une archive (JAR)

Toutes les classes et ressources (images, etc.) de l'extension doivent être contenues dans un fichier d'archive unique de type JAR. Utiliser un nom de fichier significatif et unique car ce même nom devra être utilisé aux étapes suivantes.

2 - Inclure un fichier manifest

L'archive de l'extension doit inclure un fichier *manifest.mf* avec les entrées suivantes:

Variable	Description
extension_class	Identifie la classe principale de l'extension. Il s'agit de la classe dérivée de la classe abstraite <i>JMapClientExtension</i> .
extension_name	Spécifie le nom de l'extension. Ce nom apparaît entre autres dans JMap Admin lors du déploiement des applications.
extension_version	Spécifie le numéro de version de l'extension. Cette information apparaît dans JMap Admin lors du déploiement des applications. Le numéro de version est utilisé seulement pour faciliter la gestion des extensions.

Voici un exemple de contenu d'un fichier manifest:

```
extension_class: jmap.extensions.edition.EditionExtension
extension_name: Edition
extension_version: 1.0.0049
```

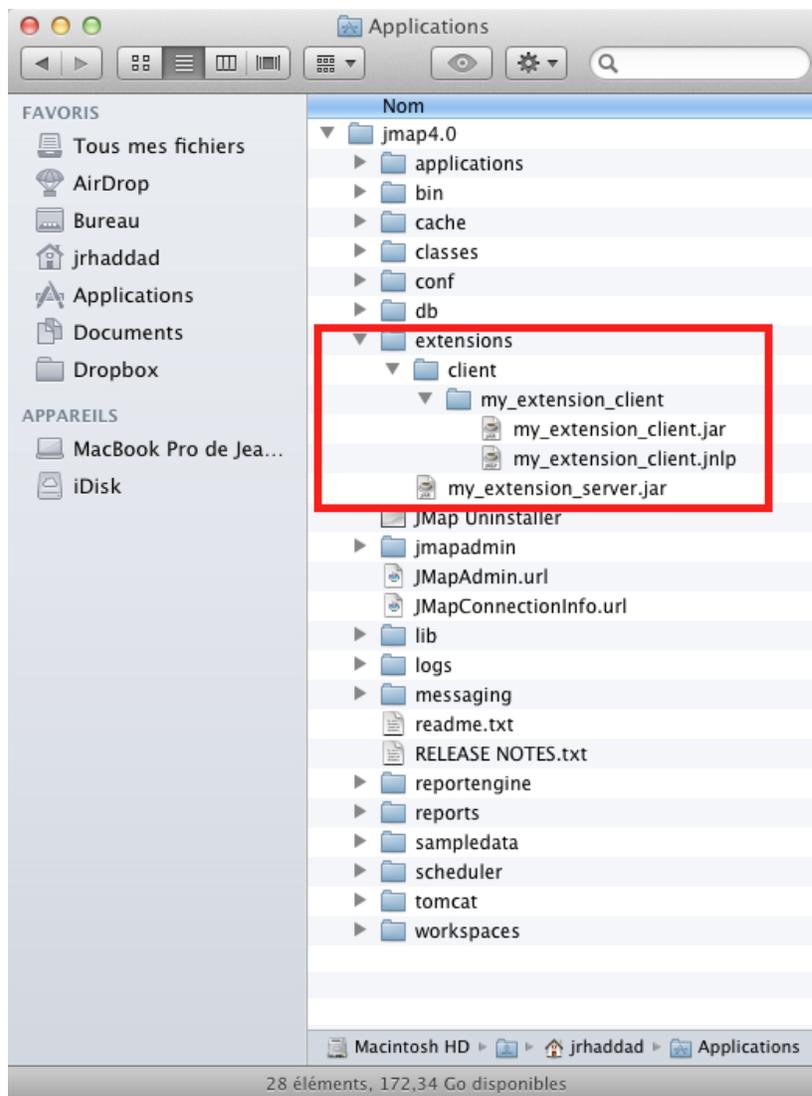
3 - Fournir un fichier JNLP

Le fichier JNLP est obligatoire. Il décrit la librairie déployée. Le fichier doit porter le même nom que le fichier JAR de l'extension (sauf pour l'extension .jnlp). L'exemple qui suit montre en caractères gras les parties qui doivent être modifiées.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- JNLP File for Extension libraries -->
<jnlp
  spec="1.0+"
  codebase="http://$JMAPSERVER_HOST$: $JMAPSERVER_WEBPORT$ $PATH$/edition_client"
  href="edition_client.jnlp">
  <information>
    <title>Edition Extension</title>
    <vendor>K2 Geospatial</vendor>
    <description>Edition Extension</description>
    <description kind="short">Edition Extension</description>
  </information>
  <security>
    <all-permissions/>
  </security>
  <resources>
    <jar href="edition_client.jar"/>
  </resources>
  <component-desc/>
</jnlp>
```

4 - Placer les fichiers dans le bon répertoire

Tous les fichiers qui composent l'extension (fichier JAR, fichier JNLP, autres fichiers) doivent être placés dans un répertoire spécialement créé pour l'extension, à l'intérieur du répertoire destiné aux extensions client (*JMAP_HOME/extensions/client*). **Le nom du répertoire de l'extension doit obligatoirement être identique au nom du JAR de l'extension.** L'image suivante montre l'organisation des fichiers et répertoires sur Windows pour l'extension *Edition* de K2 Geospatial.



La signature d'une librairie Java permet de certifier que la librairie provient d'une source identifiable et que son contenu n'a pas été altéré. L'utilisateur qui exécute le code de la librairie en question peut faire confiance en l'authenticité de la librairie. Ceci est encore plus important lorsque la librairie requiert des accès au système pouvant pauser des problèmes de sécurité tels que des accès aux données, des accès réseau, etc.

La signature des libraires Java se fait à l'aide de l'outil *jarsigner* qui est inclus avec le JDK de Java. Pour ce faire, vous devez posséder un certificat de signature de code Java émis pour votre organisation. Ces certificats peuvent être achetés d'une compagnie de sécurité reconnue telle que *Thawte* (<http://www.thawte.com/>) ou *Verisign* (<http://www.verisign.com/>). Une fois que vous possédez votre certificat, vous devez l'importer dans un *keystore* à l'aide de l'outil *keytool* qui est aussi inclus avec le JDK de Java. Notez que l'outil *keytool* peut aussi produire des certificats de développement.

Ces certificats ne sont pas reconnus comme provenant de sources fiables et vont engendrer des messages d'avertissements présentés à l'utilisateur. Par contre ils permettent de développer et de tester vos développements à l'interne. Pour plus d'information sur le sujet, consultez la documentation des outils de sécurité du JDK de java (<http://docs.oracle.com/javase/8/docs/technotes/tools/index.html>).

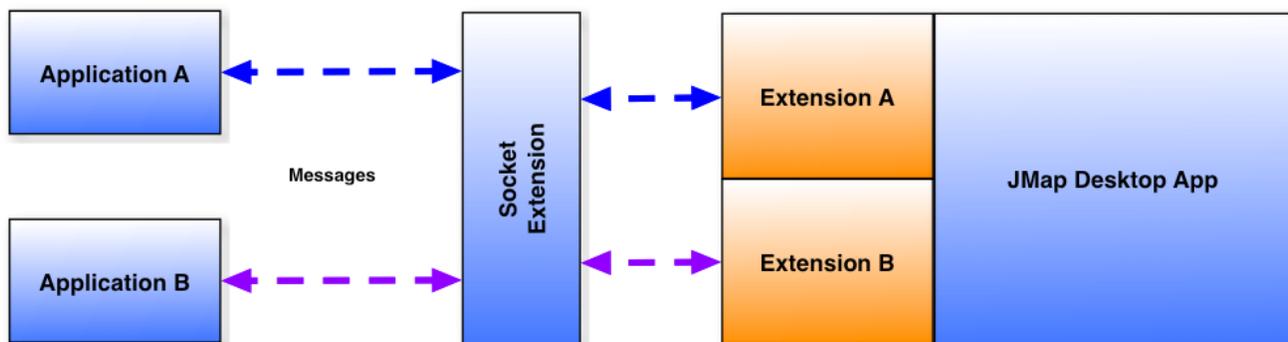
Les bibliothèques des extensions de JMap Pro doivent obligatoirement être signées afin de pouvoir être déployées correctement dans une application JMap Pro. Si vous utilisez le *Générateur d'extensions* fourni avec le SDK de JMap, les scripts *Ant* générés prennent en charge la signature de votre extension. Donc, lorsque vous exécutez ces scripts pour compiler et fabriquer la bibliothèque de votre extension, celle-ci est signée automatiquement. Cette signature se fait en utilisant le certificat de développement livré avec le SDK. Si vous voulez utiliser votre propre certificat, vous devez remplacer le fichier `JDK_HOME/tools/extensionbuilder/RES/keystore.jks` par votre propre *keystore* dans le quel vous aurez importé votre certificat.

Il existe plusieurs façons de permettre la communication entre une application JMap Pro et une autre application client, dite application externe. L'application externe est une application exécutée localement sur le même ordinateur que JMap Pro. On ne parle pas ici d'applications web (HTML, javascript), mais bien d'applications client. L'extension JMap Socket offre une façon simple et efficace de faire cette communication.

Extension Socket pour JMap

L'extension Socket est une extension générique qui permet une communication bidirectionnelle par socket entre des applications externes et les extensions de JMap Pro. Cette extension ne traite pas directement les messages. Son rôle est limité à assurer les communications. Voici un scénario décrivant l'utilisation typique de cette extension :

Une application externe envoie vers JMap l'identifiant d'un élément et celui-ci est automatiquement localisé et sélectionné sur la carte. À l'inverse, l'utilisateur clique sur un élément de la carte et JMap envoie l'identifiant de l'élément vers l'application externe qui affiche un formulaire des propriétés de l'élément.



Description du fonctionnement général

L'extension Socket offre un moyen standard de communication permettant aux extensions de JMap de communiquer avec des applications externes. Les applications externes sont des applications exécutées localement, sur le même ordinateur que JMap Pro. Il peut s'agir d'applications écrites dans n'importe quel langage de programmation (Java, C++, Windev, C#, VB.NET, etc.). Notez que les applications web (HTML, javascript, etc.) ne sont pas supportées par cette méthode d'intégration. D'autres méthodes conviennent mieux aux applications web (voir Communication Java - Javascript).

Pour communiquer, les applications externes et les extensions JMap s'envoient des messages qui sont en réalité des tableaux d'octets. Le contenu de ces tableaux d'octets est interprété de part et d'autre, l'extension Socket n'interprétant pas leur contenu.

C'est toujours l'application externe qui se connecte vers JMap en premier, en ouvrant un socket vers le port configuré dans l'extension Socket (fichier socket.properties). Par la suite, c'est encore l'application externe qui effectue la première communication. Cette première communication est spéciale et s'appelle le handshake. Lors du handshake, l'application externe indique son identifiant unique (une chaîne de caractères identifiant l'application).

L'extension de JMap qui désire communiquer avec l'application externe doit de son côté s'enregistrer auprès de l'extension Socket. Elle doit fournir le même identifiant que l'application externe. C'est cet identifiant qui permet de faire le lien entre les deux.

Par la suite, l'application externe tout comme l'extension JMap peuvent s'envoyer des messages à tout moment de part et d'autre. Chaque partie (l'application externe ou l'extension) est responsable d'interpréter les messages et de prendre les actions voulues.

Optionnellement, l'extension Socket peut démarrer l'application externe si celle-ci ne répond pas. Une fois démarrée, l'application externe pourra initier les communications, tel que décrit plus haut.

Programmation du côté de l'application externe

Les exemples de code qui suivent sont en langage Java. Une application externe programmée dans un autre langage devrait implémenter la même logique, dans le langage de l'application.

1. Ouverture d'un socket en utilisant l'adresse 127.0.0.1 et le port 12351):

```
Socket socket = new Socket("127.0.0.1", 12351);
```

2. Envoi du message de handshake

Le message de handshake sera reçu et interprété par l'extension Socket uniquement. Remarquez que le nombre d'octets du message doit précéder le contenu du message. Le contenu du message est envoyé sous la forme d'un tableau d'octets. L'encodage des caractères (charset) de ce message est défini dans la configuration de l'extension Socket (fichier socket.properties).

```
DataOutputStream dos = new ByteArrayOutputStream(socket.getOutputStream());

String handshake = "handshake:MyIdentifiant";
dos.writeInt(handshake.getBytes().length);
dos.write(handshake.getBytes());
dos.flush();
```

3. Envoi des messages subséquents

Les messages subséquents sont envoyés de la même manière que le message de handshake. Par contre, ils seront reçus et interprétés par les extensions enregistrées avec le même identifiant. Les extensions qui reçoivent les messages doivent donc être en mesure de comprendre le contenu des messages et d'effectuer les actions correspondantes. L'encodage des caractères du message est libre et doit être convenu de l'extension et de l'application externe.

```
DataOutputStream dos = new ByteArrayOutputStream(socket.getOutputStream());

String message = "locate :id=333";
dos.writeInt(message.getBytes().length);
dos.write(message.getBytes());
dos.flush();
```

4. Réception des messages en provenance de JMap

La réception des messages se fait par le même socket que l'envoi. La longueur du message est lue en premier, et ensuite le tableau d'octets du message. Notez que la réception devrait se faire de manière asynchrone, dans une thread à part.

```
DataInputStream dis = new DataInputStream(socket.getInputStream());

int length = dis.readInt();
```

```
if (length > 0)
{
    byte[] messageBytes = new byte[length];
    dis.readFully(messageBytes);
}
```

Programmation du côté de l'extension JMap

L'exemple suivant montre comment enregistrer une extension JMap auprès de l'extension Socket afin d'être averti en cas de connexion, de déconnexion et de réception de messages en provenance de l'application externe. Ce code pourrait être placé dans la méthode `init()` de l'extension JMap.

```
SocketClientExtension.register(new SocketClient()
{
    @Override
    public void socketMessageReceived(byte[] bytes)
    {
        String data = new String(bytes /*, Charset.forName("ISO-8859-5")*/);

        // Interpret content of data...
    }

    @Override
    public void socketConnectionOpened()
    {
    }

    @Override
    public void socketConnectionClosed()
    {
    }

    @Override
    public String getIdentifier()
    {
        return "MyIdentifier";
    }

    @Override
    public String getExecutable()
    {
        // Optional - return null if not needed
        return "c:/external_app.exe";
    }
});
```

L'exemple suivant montre comment envoyer un message vers l'application externe à partir de l'extension JMap.

```
String message = "openform: id =99";  
byte[] bytes = message.getBytes();  
SocketClientManager.getInstance().sendMessage("MyIdentifiant" , bytes);
```

Les applets Java sont exécutées dans l'environnement d'un navigateur web. De ce fait, elles peuvent interagir avec le code javascript présent dans la page web qui contient l'applet et ce, dans les 2 directions. Cette communication permet de réaliser des interfaces html pour contrôler la carte et des intégrations simples avec d'autres applications exécutées dans l'environnement du navigateur web. Cela ne s'applique pas aux applications JMap Pro déployées avec la méthode JavaWebStart (en dehors d'un navigateur web) car il n'y a pas de page web impliquée dans ces cas.

Javascript vers Java

L'exemple de page web suivante est une page normale de démarrage d'une application JMap Pro de type applet à laquelle on a ajouté des fonctions en javascript (*pan* et *zoom*) et des hyperliens pour appeler ces fonctions. Lorsqu'elles sont appelées, les fonctions font appel à l'API de JMap pour contrôler la carte.

L'application JMap Pro possède une méthode `getApplicationContext()` permettant d'accéder à l'ensemble des composantes de l'application.

```

<%@page contentType="text/html;charset=ISO-8859-1"%>

<%
String username = request.getParameter("username");
String password = request.getParameter("password");
String parameters = null;

if (username != null && username.length() != 0)
{
parameters = "?username=" + username;
if (password != null)
parameters += "&password=" + password;
}
%>

<html>
<head>
<title>
aa
</title>
<script src="library/deployJava.js"></script>
<script src="library/jmap.js"></script>
</head>

<BODY BGCOLOR="#ffffff" topmargin="0" leftmargin="0" rightmargin="0" bottommargin="0">

<script>
function zoom(factor)
{
document.jmap.getApplicationContext().getViewManager().getActiveView().zoom(factor);
document.jmap.getApplicationContext().getViewManager().getActiveView().refresh();
}
function pan(x, y)
{
document.jmap.getApplicationContext().getViewManager().getActiveView().pan(x, y);
document.jmap.getApplicationContext().getViewManager().getActiveView().refresh();
}
</script>

<a href="javascript:zoom(2.);">Zoom in</a>
<a href="javascript:zoom(0.5);">Zoom out</a>
<a href="javascript:pan(0, 200);">Pan north</a>
<a href="javascript:pan(0, -200);">Pan south</a>
<a href="javascript:pan(200, 0);">Pan west</a>
<a href="javascript:pan(-200, 0);">Pan east</a>

<script>
var attributes =
{
name: "jmap",
codebase: "http://127.0.0.1:8080/aa",
code: "com.kheops.jmap.client.application.JMapApplicationLauncher",
archive: "dockingClient.jar,jmap_application.jar,jmap_client.jar,jmap_client_images.",
width: "100%",
height: "100%",
mayscript: true,
separate_jvm: true,
parameters: "-appclassname jmap.viewers.docking.AppDocking -project 'The World&

```

```
};

var parameters = {fontSize:16, jnlp_href:'dockingClient.jnlp'} ;

deployJava.runApplet(attributes, parameters, '1.6.0_10');
</script>

</body>
</html>
```

Java vers Javascript

À partir d'une applet JMap Pro, il est possible d'appeler des fonctions javascript présentes dans la page web qui contient l'applet. Cela permet une interaction entre l'application JMap Pro et son environnement html. Par exemple, il serait possible de sélectionner un élément sur la carte et d'afficher dans la page html des informations sur l'élément en question.

Pour activer cette communication java-javascript, il est nécessaire de spécifier le paramètre **mayscript: true** dans les attributs servant à démarrer l'applet, tel que montré dans l'exemple suivant.

```
<script>
var attributes =
{
  name: "jmap",
  codebase: "http://127.0.0.1:8080/aa",
  code: "com.kheops.jmap.client.application.JMapApplicationLauncher",
  archive: "dockingClient.jar,jmap_application.jar,jmap_client.jar,jmap_client_images.jar",
  width: "100%",
  height: "100%",
  mayscript: true,
  separate_jvm: true,
  parameters: "-appclassname jmap.viewers.docking.AppDocking -project &quot;The World&quot;";
};

var parameters = {fontSize:16, jnlp_href:'dockingClient.jnlp'} ;

deployJava.runApplet(attributes, parameters, '1.6.0_10');
</script>
```

Pour appeler une fonction javascript à partir de votre code Java (possiblement de votre extension JMap), vous devez utiliser l'API de JSObject, tel que démontré dans l'exemple suivant.

```
JSObject w = JSObject.getWindow((Applet)appContext.getRootPaneContainer());
w.eval("show_form(" + id + ");");
```

Dans cet exemple, la méthode `getWindow` reçoit l'instance de l'applet en paramètre. La méthode `getRootPaneContainer` de la classe `JMapApplicationContext` retourne le `container` de plus haut niveau de l'application, ce dernier étant l'instance de `Applet` lorsque l'application est exécutée à l'intérieur du navigateur Web.

La méthode `eval` prend en paramètres la fonction javascript à appeler avec ses paramètres. Dans cet exemple, un identifiant est passé à la fonction javascript.

L'application JMap Pro prend certains paramètres au démarrage. Ces paramètres permettent de spécifier l'adresse du serveur JMap, les ports de communication, le projet à ouvrir et de nombreuses autres options.

Les paramètres sont passés à l'application de différentes manières selon le mode de démarrage de celle-ci. En applet Java et en `JavaWebStart`, les paramètres sont passés dans le fichier `JNLP` de l'application. En application, les paramètres sont passés à la ligne de commande ou peuvent être spécifiés dans un script Ant.

L'exemple suivant montre les paramètres passés à la ligne de commande pour démarrer une application JMap Pro ouvrant le projet `The World` et chargeant l'extension `Showcase`.

```
-appclassname jmap.viewers.docking.AppDocking -server jmap3.k2geospatial.com -directpor
-project "The world" -extensions jmap.examples.showcase.extension.ShowCaseClientExtensi
```

Le tableau suivant décrit les différents paramètres :

Paramètres (* = requis)	
-appclassname *	Classe principale de l'application à exécuter. La seule valeur possible actuellement est <code>jmap.viewers.docking.AppDocking</code> .
-server *	Le nom ou l'adresse IP du serveur JMap auquel l'application doit se connecter.
-directport	Le port de communication IP pour les connexions direct avec JMap Server.
-httpport	Le port de communication IP pour les connexions par proxy HTTP avec JMap Server.

Paramètres (* = requis)	
-project	Le projet à ouvrir par défaut. Doit être entre guillemets si le nom comporte des espaces.
-language	La langue des interfaces graphiques de l'application. Les valeurs supportées sont <i>fr</i> , <i>en</i> , <i>es</i> , <i>pt</i> et <i>default</i> . La valeur <i>default</i> signifie que la langue utilisée sera la langue par défaut du système d'exploitation de l'utilisateur.
-country	Le pays, utilisé avec la langue, pour déterminer les formats d'affichage des dates et des nombres.
-username	Le nom d'utilisateur à utiliser pour l'ouverture de la session.
-password	Le mot de passe à utiliser pour l'ouverture de la session.
-sessionid	Indique le numéro de session pour se connecter à une session déjà ouverte sur le serveur JMap.
-autozoom	Indique à l'application JMap de localiser un emplacement ou un élément automatiquement à l'ouverture. La syntaxe est la suivante: -autozoom REGION;x;y;width;height OU -autozoom OBJECT;LayerName;attribute;value OU -autozoom OBJECT;LayerName;attribute;value;maxScale
-connection	Type de connexion à utiliser entre l'application et le serveur JMap. Les valeurs possibles sont : - direct : Ouvre une connexion directe vers JMap Server en utilisant le port direct. - proxy : Ouvre une connexion par proxy HTTP vers JMap Server en utilisant le port HTTP. - any : Tente une connexion directe. En cas d'échec, bascule en connexion par proxy HTTP.
-proxypath	Si la connexion est de type proxy HTTP, spécifie un chemin relatif pour le proxy HTTP.
-serverid	Si la connexion est de type proxy HTTP, spécifie vers quelle instance de JMap Server la connexion doit s'ouvrir, dans le

Paramètres (* = requis)	
	cas où plusieurs instances de JMapServer sont disponibles. Cela permet d'utiliser le proxy HTTP comme aiguilleur de requêtes. Les ID de serveurs sont configurées dans les fichiers <code>jmsconnections.xml</code> .
-showconnectionmoredetails	Détermine si la fenêtre de connexion doit montrer la liste des projets disponibles sur JMap Server. Valeurs possibles : <i>true</i> , <i>false</i>
-usediskcache	Détermine si le cache sur disque est activé ou non. Valeurs possibles : <i>true</i> , <i>false</i>
-diskcachepath	Si le cache sur disque est activé, détermine le dossier où les données en cache seront sauvegardées.
-diskcachesize	Si le cache sur disque est activé, détermine la taille maximale totale des données en cache. Des données s'effacent automatiquement si le cache atteint la taille limite. La valeur est exprimée en octets. Une valeur -1 indique une taille illimitée.
-usememorycache	Détermine si le cache en mémoire est activé ou non. Si le cache est activé, les données en mémoire sont gérées de la manière suivante: quand l'espace réservé devient plein, c'est à dire que le cache atteint la taille limite (paramètre <code>-maxmemory</code>), des données sont retirées automatiquement de la mémoire. La quantité de données retirées dépend du pourcentage spécifié (paramètre <code>-percentreleasememory</code>). Valeurs possibles : <i>true</i> , <i>false</i>
-maxmemory	Si le cache en mémoire est activé, détermine la taille maximale des données en mémoire. La valeur est exprimée en octets. La valeur par défaut est 33554432 (32 Mo).
-percentreleasememory	Détermine le pourcentage de mémoire à libérer lorsque que le cache mémoire devient plein. Le pourcentage est fonction de la taille totale du cache mémoire. La valeur est un entier entre 1 et 100.
-logos	Liste des logos à afficher sur la carte ainsi que leur position et leur transparence.

Paramètres (* = requis)	
	Exemple: <code>-logos "jmaplogo.gif?x=5&y=5&transparency=30.0&relativeTo=NE"</code>
-northarrow	Paramètres d'affichage d'une flèche du nord sur la carte, incluant le modèle, la position, la taille, etc. Exemple: <code>-northarrow Simple3D,0,50,5,5</code>
-displayscalebar	Détermine si l'échelle graphique doit être affichée sur la carte. Valeurs possibles : <i>true</i> , <i>false</i>
-extensions	La liste des extensions à initialiser au démarrage de l'application, séparées par des virgules. Exemple: <code>-extensions jmap.extensions.googlemap.client.GoogleMapsExtension,jmap.examples.showcase.extension.ShowCaseClientExtension</code>

Environnement de débogage à distance

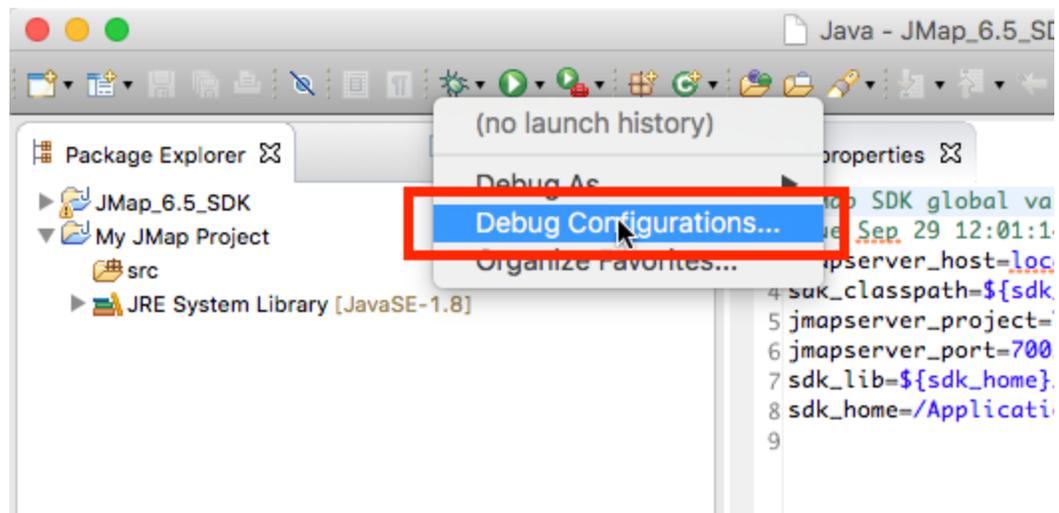
Il peut être utile de faire le débogage à distance des extensions de JMap Server. Une fois déployées, ces extensions sont exécutées directement dans le processus de JMap Server. Il devient donc impossible de les déboguer sans le faire à distance. L'environnement d'exécution de Java permet de faire le débogage à distance en ajoutant un paramètre spécial dans l'environnement de JMap Server. Vous pourrez par la suite vous y connecter à partir d'Eclipse et suivre l'exécution de vos extensions JMap, pas à pas.

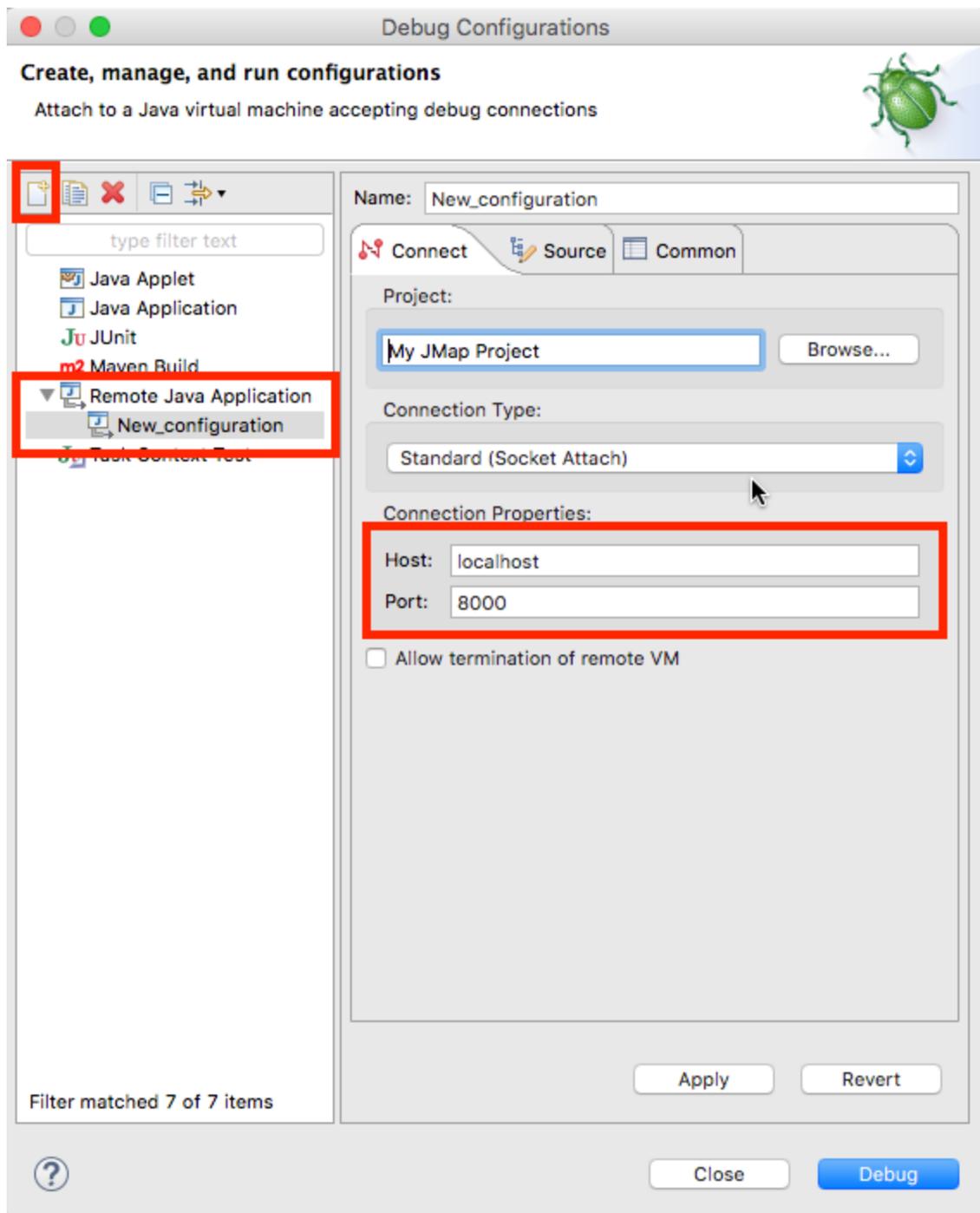
Pour activer le débogage à distance dans JMap Server, vous devez modifier le fichier `startjmapserver.vmoptions` qui se situe dans le répertoire `JMAP_HOME/bin`. Vous devez ajouter la ligne débutant par `-Xdebug`.

```
-Xmx768m
-XX:MaxPermSize=256m
-Djava.awt.headless=true
-Dfile.encoding=ISO-8859-1
-Xdebug
-Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=8000
```

Vous devez ensuite redémarrer JMap Server. Ce dernier sera alors en mode débogage et sera en attente de commandes en provenance d'Eclipse. En production, n'activez pas le mode de débogage car cela a pour effet de réduire de beaucoup les performances du système.

Dans Eclipse, vous devez créer une configuration de débogage à distance en prenant soin d'utiliser les mêmes paramètres que ceux fournis dans le fichier de configuration montré plus haut (ce sont les paramètres par défaut).





Les extensions serveur de JMap sont des modules développés en langage Java qui se greffent au serveur JMap afin de répondre à des nouveaux types de requêtes et d'effectuer des tâches côté

serveur. Les extensions serveur peuvent comporter des interfaces de configuration qui s'insèrent dans la section *Extensions* de JMap Admin. Souvent, une extensions serveur travaille de pair avec une extension client.

Pour développer et rendre disponible une extension serveur, vous devez effectuer les 2 étapes suivantes:

1. Développer votre extension en créant une classe Java qui implémente l'interface `JMapServerExtension`.
2. Facultativement, développer une interface de configuration JMap Admin pour votre extension.
3. Déployer votre extension dans JMap Server.

Consultez les sections suivantes pour plus d'information.

L'interface `JMapServerExtension`

La première étape pour le développement d'une extension serveur de JMap consiste à écrire une classe qui implémente l'interface `JMapServerExtension`. Cette interface comporte les 3 méthodes suivantes, appelées à différents moments du cycle de vie de l'extension:

Méthodes de l'interface <code>JMapServerExtension</code>	
<code>init()</code>	Cette méthode est appelée lorsque au démarrage de JMap Server lors du chargement initial des extensions serveur ou quand l'administrateur demande la réinitialisation de l'extension à partir de JMap Admin. Elle sert à initialiser l'extension. Vous pouvez mettre dans cette méthode tout code servant à préparer le fonctionnement de votre extension. Cela pourrait inclure le chargement d'un fichier de paramètres, les vérifications de dépendances, etc.).
<code>processRequest</code> (<code>JMapExtensionRequest</code>)	Cette méthode est appelée lorsque JMap Server reçoit une requête destinée à votre extension. Vous devez mettre dans cette méthode le code nécessaire au traitement de la requête. De plus, la méthode doit retourner une réponse résultant du traitement de la requête.
<code>destroy()</code>	Cette méthode est appelée à l'arrêt de JMap Server ou quand l'administrateur demande la réinitialisation de l'extension à partir de JMap Admin. Elle sert à exécuter du code nécessaire à la fermeture de l'extension. Cela pourrait inclure la fermeture de fichiers ou de connexions vers d'autres systèmes.

Pour effectuer son travail, votre extension peut faire appel aux services offerts par JMap Server. Cela inclut par exemples, les extractions de données spatiales, des accès aux bases de données relationnelles connectées à JMap Server, l'accès au système de journalisation (log files), etc.

Pour plus d'information sur les services offerts par JMap Server, consultez la section *Services de JMap Server* .

La classe JMapExtensionRequest

Vous devez implémenter une classe qui dérive de la classe *JMapExtensionRequest* . Cette classe a pour but de fournir toute l'information nécessaire à votre extension serveur pour qu'elle exécute son travail. Les requêtes sont typiquement initialisée du côté client et transmises à JMap Server pour traitement par votre extension. Vous pouvez inclure dans cette classe toutes les propriétés requises en vous assurant qu'elles soient toutes sérialisables.

Lors de l'initialisation de votre extension, le type de votre requête (le nom de classe complet incluant le paquetage) est associé à votre extension. De cette manière, lorsque JMap Server reçoit une requête de ce type, celle-ci est automatiquement aiguillée vers votre extension (méthode *processRequest*). Pour plus d'information sur la manière de faire le lien entre votre requête et votre extension, consultez la section *Déploiement des extensions serveur* .

Pour plus d'information sur la programmation des requêtes, consultez la section *Communication client-serveur* .

La classe JMapExtensionResponse

Vous devez aussi implémenter une classe qui dérive de la classe *JMapServerResponse* . Cette classe a pour but de fournir les informations résultant de l'exécution de votre requête. La méthode *processRequest* de votre extension doit retourner une instance de cette classe. Selon la nature de la requête, la réponse peut retourner une grande quantité d'information ou alors simplement un statut d'exécution de la requête (succès, échec, etc.). Vous pouvez inclure toutes les propriétés requises dans votre classe en vous assurant qu'elles soient toutes sérialisables.

Pour plus d'information sur la programmation des réponses, consultez la section *Communication client-serveur* .

La classe JMap Server

La classe *JMapServer* est la principale classe à partir de laquelle vous pouvez accéder aux différents services du serveur JMap. Cette classe est un singleton. Vous pouvez donc y avoir accès de n'importe où par la méthode statique *JMapServer.getInstance()* .

JMapHome

Le chemin du répertoire principal de JMap Server est accessible par la méthode statique `getJMapHome()` de la classe `JMapServer`. Cela peut être utile de connaître ce chemin pour lire ou écrire des données dans les sous-répertoires de JMap Server.

Journalisation

L'outil de journalisation de JMap Server permet d'enregistrer des événements dans les fichiers de journalisation. La classe de l'outil est `Logger` et c'est un *singleton*. On peut donc avoir accès à l'instance unique par la méthode statique `Logger.getInstance()`.

Les différentes versions de la méthode `log` sont utilisées pour enregistrer les messages, selon le type d'information à enregistrer.

Le tableau suivant présente les méthodes les plus utilisées de la classe `Logger`.

Méthodes de la classe <code>Logger</code>	
<code>log(int, String)</code>	Enregistre un message du niveau spécifié.
<code>log(int, String, String)</code>	Enregistre un message du niveau spécifié associé à l'utilisateur spécifié.
<code>log(int, String, Throwable)</code>	Enregistre un message du niveau spécifié ainsi que la trace de l'exception passée en paramètre.
<code>log(int, String, Throwable, String)</code>	Enregistre un message du niveau spécifié, associé à l'utilisateur spécifié, ainsi que la trace de l'exception passée en paramètre.
<code>setLogLevel(int)</code>	Modifie le niveau des messages qui seront enregistrés.

Les différents niveaux de messages disponibles sont définies par des constantes de la classe `Logger`.

- LEVEL_DEBUG
- LEVEL_INFO
- LEVEL_WARNING
- LEVEL_ERROR
- LEVEL_FATAL

L'exemple de code suivant montre comment enregistrer un message avec les différentes méthodes.



```

// Logs a message of level INFO
Logger.getInstance().log(Logger.LEVEL_INFO, "Extension ABC recieved a request for ..."

// Logs a message of level WARNING, tagged to user etardif
Logger.getInstance().log(Logger.LEVEL_WARNING, "Something occurred in Extension ABC ..

// Logs a message of level ERROR, and includes the exception stack trace
Exception e = ...;
Logger.getInstance().log(Logger.LEVEL_ERROR, "An unexpected error occurred in Extension

```

Bases de données

Les connexions aux bases de données relationnelles auxquelles JMap Server est connecté sont disponibles par programmation. Cela simplifie grandement les accès aux données car vous n'avez pas besoin d'ouvrir, de fermer et de gérer les connexions vers ces bases de données.

JMap Server gère les connexions aux bases de données dans des réserves de connexions (*connection pools*). Le principe de fonctionnement d'une réserve est le suivant. Lorsqu'une connexion est requise, elle est empruntée de la réserve. Elle est ensuite utilisée brièvement le temps d'exécuter une ou plusieurs requêtes. Finalement, et très important, la connexion est retournée dans la réserve et redevient disponible pour d'autres besoins. Les connexions ne sont donc jamais fermées.

Les méthodes `getDBConnPool(int)` et `getDBConnPool(String)` permettent d'obtenir une réserve de connexions (instance de la classe `DatabaseConnectionPool`) par son identifiant numérique ou par son nom.

Le tableau suivant montre les méthodes de la classe `DatabaseConnectionPool` les plus souvent utilisées.

Méthodes les plus utiles de la classe <code>DatabaseConnectionPool</code>	
<code>borrowConnection()</code>	Emprunte une connexion JDBC de la réserve. La connexion est alors réservée exclusivement.
<code>returnConnection(Connection)</code>	Retourne une connexion JDBC empruntée dans la réserve. Il est très important d'appeler cette méthode après l'utilisation d'une connexion.
<code>getStatus()</code>	Permet d'obtenir l'état de la réserve de connexions. Peut être appelée pour valider le bon fonctionnement de la réserve avant d'emprunter une connexion. Les états possibles sont définies par des constantes de la classe <code>ConnectionPoolInfo</code> (<code>CONNECTION_NOT_TESTED</code> , <code>CONNECTION_ERROR</code> ou <code>CONNECTION_OK</code>)

L'exemple de code source suivant montre comment utiliser une réserve de connexions vers des bases de données.

```

DatabaseConnectionPool pool = JMapServer.getInstance().getDBConnPool("parcels");
Connection conn = null;

try
{
    conn = pool.borrowConnection();

    // use connection to do queries.....

}
catch (SQLException e)
{
    e.printStackTrace();
}
finally
{
    // It is very important to return the connection to the pool.
    // Doing it in a finally clause is a good practice
    if (conn != null)
        pool.returnConnection(conn);
}

```

Extractions de données spatiales

Il est possible d'effectuer des extractions de données spatiales en utilisant le gestionnaire de données (classe *JMapServerDataManager*) de JMap Server. Le gestionnaire de données est accessible par la méthode *getDataManager()* de la classe *JMapServer* .

Le tableau suivant montre les méthodes les plus utilisées de la classe *JMapServerDataManager* .

Méthodes de la classe <i>JMapServerDataManager</i>	
<i>extractElements</i> (<i>JMapServerProject</i> , <i>JMapServerVectorLayer</i> , <i>QueryFilter[]</i> , <i>Attribute[]</i>)	Effectue l'extraction des données spatiales et de leurs attributs pour la couche spécifiée, appartenant au projet spécifié et qui passent les filtres passés en paramètres. Seuls les attributs qui sont indiqués dans le dernier paramètre sont inclus dans le résultat.
<i>extractElements</i> (<i>String</i>)	Effectue l'extraction des données spatiales et de leurs attributs selon la requête passée en paramètres. La syntaxe générale des requêtes est la suivante : <pre>select \$element from \$source { \$project { PROJECT_NAME } \$layer { LAYER_NAME } } where CONDITION</pre> Exemple : <pre>select \$element from \$source { \$project { The World } \$layer { Countries } } where COUNTRY = 'Peru'</pre>

<i>extractElements</i> (String, long[])	Effectue l'extraction des données spatiales et de leurs attributs selon la requête passée en paramètres et la liste d'identifiants spécifiée. Seuls les éléments dont l'identifiant est présent dans la liste sont retournés.
<i>extractElements</i> (String, long[], OrientedRectangle)	Effectue l'extraction des données spatiales et de leurs attributs selon la requête passée en paramètres, la liste d'identifiants ainsi que la région spécifiées. Seuls les éléments dont l'identifiant est présent dans la liste et qui intersectent la région sont retournés.

Pour faire l'extraction des données, vous pouvez aussi utiliser des filtres. Les filtres sont des objets qui permettent, selon différents critères, de contrôler quelles données doivent être extraites.

Les classes de filtres sont toutes dérivées de la classe abstraite *QueryFilter*. Le tableau suivant montre l'ensemble des types de filtres qui sont disponibles:

Types de filtres de requêtes	
<i>AttributeFilter</i>	Permet de définir une condition basée sur un attribut des données. La condition est définie par un attribut et par un ensemble de valeurs. Seules les données dont l'attribut en question a une valeur incluse dans l'ensemble de valeurs passent le filtre.
<i>GeometryTypes Filter</i>	Permet de définir une condition basée sur le type de géométries des données. La condition est définie par un u plusieurs types de géométries. Seules les données dont le type de géométrie correspond à ceux du filtre passent le filtre.
<i>SpatialQueryFilter</i>	Permet de définir une condition spatiale. Seules les données qui remplissent la condition passent le filtre. La condition est définie par une géométrie et une contrainte. Par exemple: toutes les géométries qui intersectent le polygone spécifié, ou toutes les géométries qui contiennent le point spécifié, etc.
<i>SQLQueryFilter</i>	Permet de définir une condition en langage SQL. C'est l'équivalent de la clause <i>where</i> d'une requête SQL. La clause <i>where</i> est interprétée par le système de base de données qui contient les données.

L'exemple de code source suivant montre comment extraire des données spatiales en utilisant un filtre spatial.

```
JMapServerProject serverProject = ...
JMapServerVectorLayer serverLayer = ...
Polygon region = ...

final JMapServerDataManager dataMgr = JMapServer.getInstance().getDataManager();

// Create a new spatial filter for all elements that intersect the polygon
final SpatialQueryFilter newFilter = new SpatialQueryFilter();
```

```
newFilter.setGeometry(region);
newFilter.setType(SpatialQueryFilter.SPATIAL_OP_INTERSECTS);

// Set the projection on the filter to indicate the coordinate system of the geometry
newFilter.setProjection(serverProject.getProject().getMapProjection());

// Do the extraction using the data manager. All attributes of the layer will be included
JMapGeoElement[] result = dataMgr.extractElements(serverProject,
                                                serverLayer,
                                                new QueryFilter[]{newFilter},
                                                serverLayer.getBoundAttributes() );
```

Envoi de courriels

La méthode statique `sendMail()` de la classe `MailService` permet l'envoi de courriels par JMap Server. Pour que l'envoi fonctionne, JMap Server doit être connecté à un serveur SMTP. Cette connexion peut être configurée lors de l'installation de JMap ou dans la section *Paramètres* de JMap Admin.

```
String to = "jo32@gmail.com;ann122@hotmail.com";
String from = "admin@123map.com";

try
{
    MailService.sendMail(MailService.toAddresses(to), "Map extraction completed", "The da
}
catch (AddressException e)
{
    e.printStackTrace();
}
catch (Exception e)
{
    e.printStackTrace();
}
```

Gestionnaire de sessions

Le gestionnaire de sessions (classe `JMapServerSessionManager`) de JMap Server est responsable de gérer les sessions actives dans le système. Il permet notamment d'accéder à l'utilisateur relié à une session en utilisant le numéro de session (*session id*). Ce numéro de session est accessible depuis chaque requête reçue par JMap Server, incluant les requêtes destinées aux extensions serveur. Le gestionnaire de session peut donc être utilisé pour connaître l'identité de l'utilisateur à l'origine d'une requête.

L'exemple de code qui suit montre comment accéder à l'utilisateur à l'origine d'une requête.

```

public JMapExtensionResponse processRequest(JMapExtensionRequest request)
{
    int sessionId = request.getSessionId();
    User user = JMapServer.getInstance().getSessionManager().getSessionUser(sessionId);
    System.out.println("##### Request originating from user: " + user.getName() + " (" + us
    ...
}

```

Gestionnaire d'utilisateurs

Le gestionnaire d'utilisateurs (interface *UserManager*) permet d'accéder à la liste des utilisateurs et des groupes utilisés par JMap Server pour le contrôle des accès. Il permet aussi d'accéder aux informations sur les utilisateurs. Si vous développez une extension serveur qui doit gérer sa propre liste de permissions, il est très utile d'accéder à la liste des utilisateurs utilisée par le système.

La méthode *getUserManager()* de la classe *JMapServer* retourne le gestionnaire d'utilisateurs (classe implémentant l'interface *UserManager*) en utilisation.

Méthodes les plus utiles de la classe *UserManager*

<i>getUser(String)</i>	Retourne l'utilisateur (instance de la classe <i>User</i>) dont le nom est spécifié en paramètre.
<i>getGroup(String)</i>	Retourne le groupe (instance de la classe <i>Group</i>) dont le nom est spécifié en paramètre.
<i>users()</i>	Retourne la liste des utilisateurs (instances de la classe <i>User</i>) utilisée par JMap Server.
<i>groups()</i>	Retourne la liste des groupes (instances de la classe <i>Group</i>) utilisée par JMap Server.

La classe *User* contient les informations relatives à un utilisateur telles que son nom complet, son adresse courriel, etc.

Workspaces

Les *workspaces* dans JMap sont des espaces réservés pour le stockage individuel des données des utilisateurs. Chaque *workspace* est en fait un sous-répertoire distinct du serveur JMap. C'est dans les *workspaces* que sont stockés les contextes créés par les utilisateurs. Comme programmeur, vous pouvez les utiliser pour y déposer des données.

La méthode `getWorkSpaceManager()` de la classe `JMapServer` permet d'accéder au gestionnaire de `workspaces` (classe `WorkSpaceManager`). Ce dernier fournit quelques méthodes utiles relatives aux `workspaces`. Par défaut, les `workspaces` sont localisés dans le répertoire `JMAP_HOME/workspaces`.

Le tableau suivant montre les méthodes les plus utiles de la classe `WorkSpaceManager`.

Méthodes les plus utiles de la classe <code>WorkSpaceManager</code>	
<code>getUserWSDirectory(String)</code>	Retourne le chemin complet vers le répertoire du <code>workspace</code> de l'utilisateur spécifié en paramètre.
<code>emptyUserWS(String)</code>	Efface tout le contenu du <code>workspace</code> de l'utilisateur spécifié en paramètre.
<code>deleteUserWS(String)</code>	Efface le répertoire du <code>workspace</code> de l'utilisateur spécifié en paramètre.

Pour être déployée dans JMap Server, les extensions serveur doivent respecter certaines règles. Si ces règles sont bien respectées, l'extension apparaît dans la section `Extensions` de JMap Admin.

1 - Regrouper les classes de l'extension dans une archive (JAR)

Toutes les classes de l'extension doivent être contenues dans un fichier d'archive unique de type JAR. Utiliser un nom de fichier significatif et unique.

2 - Inclure un fichier manifest

L'archive de l'extension doit inclure un fichier `manifest.mf` avec les entrées suivantes :

Variable	Description
extension_class	Identifie la classe principale de l'extension. Il s'agit de la classe qui implémente l'interface <code>JMapServerExtension</code> .
extension_request	Identifie la classe utilisée comme requête pour cette extension. C'est cette entrée qui permettra d'acheminer les requêtes jusqu'à votre extension. Si votre extension supporte plusieurs classes de requêtes, elles doivent toutes dériver de cette classe.
extension_response	Identifie la classe utilisée comme réponse pour cette extension. Si votre extension supporte plusieurs classes de réponses, elles doivent toutes dériver de cette classe.

extension_name	Spécifie le nom de l'extension. Ce nom apparaît dans JMap Admin dans la section <i>Extensions</i> .
extension_version	Spécifie le numéro de version de l'extension. Cette information apparaît dans JMap Admin dans la section <i>Extensions</i> . Le numéro de version est utilisé seulement pour faciliter la gestion des extensions.

Voici un exemple de de contenu d'un fichier manifest:

```
extension_class: jmap.extensions.tracking.server.TrackingExtension
extension_name: Tracking
extension_request: jmap.extensions.tracking.common.TrackingRequest
extension_response: jmap.extensions.tracking.common.TrackingResponse
extension_version: 5.0.0010
```

3 - Placer le fichier dans le bon répertoire

Le fichier JAR de l'extension doit être placé dans le répertoire des extensions serveur (*JMAP_HOME/extensions*).

4 - (Facultatif) Placer les fichiers de l'interface de configuration de l'extension dans le bon répertoire

Les fichiers qui composent l'interface de configuration (pages JSP) doivent être copiés dans le répertoire réservé à cette fin (*JMAP_HOME/jmapadmin/extensions*).

Chaque extension serveur peut comporter une interface de configuration intégrée à JMap Admin. En utilisant cette interface, les administrateurs JMap peuvent configurer les paramètres de fonctionnement de votre extension (p. ex. la sécurité, la connectivité aux bases de données, une sélection de couches, etc.). Notez que cette interface est tout à fait optionnelle.

Cette interface est composée d'une ou plusieurs pages JSP. La page JSP principale (celle qui est appelée en premier) doit absolument porter le même nom que la classe de l'extension. Par exemple, si le nom de la classe de l'extension serveur est

```
jmap.extensions.tracking.server.TrackingServerExtension
```

alors la page JSP principale doit être

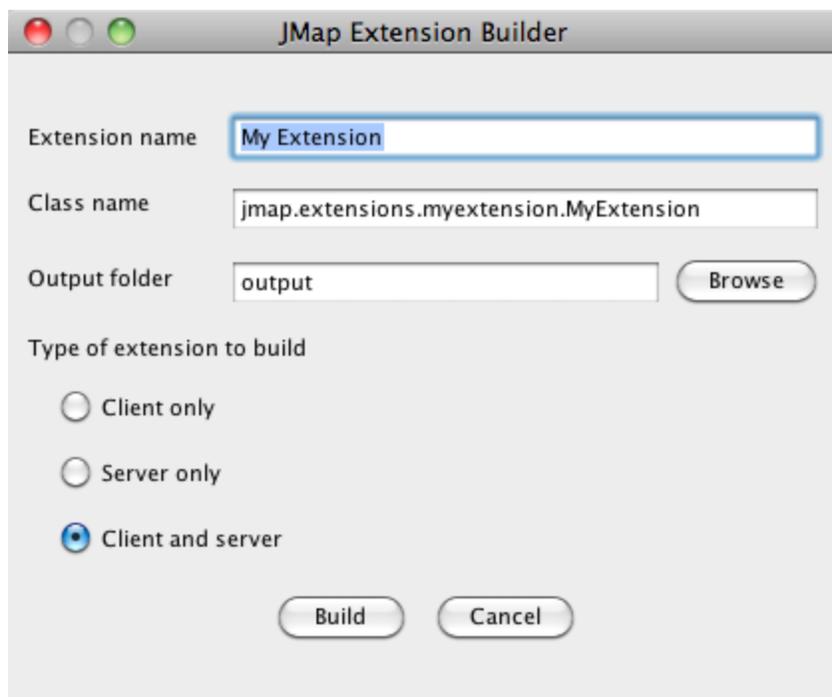
```
jmap.extensions.tracking.server.TrackingServerExtension.jsp
```

Pour plus d'information sur la programmation de ces pages JSP, veuillez vous référer aux exemples inclus dans le SDK.

La documentation de l'API Java de JMap 6.5 est disponible en ligne à l'adresse <http://dev.k2geospatial.com/jmap/javadoc/6.5/>.

Un rapport des différences entre l'API de JMap 6.0 et JMap 6.5 est disponible à l'adresse <http://dev.k2geospatial.com/jmap/javadoc/6.5/changes.html>.

Le générateur d'extensions de JMap est un outil qui permet de générer rapidement le code source pour des extensions client et serveur de JMap. Le code source généré est de base, et vous devez le compléter afin d'implémenter les fonctions de votre extension. Cet outil aide à économiser du temps en permettant aux développeurs de se concentrer sur l'objectif réel de leur développement plutôt que sur les détails techniques de programmation et de déploiement des extensions JMap.



L'outil possède une interface graphique mais il peut aussi être utilisé à la ligne de commandes, sans interface graphique. Les paramètres acceptés par le générateur d'extensions sont les suivants. Notez qu'ils sont tous optionnels et que des valeurs par défaut sont proposées pour les paramètres manquants.

Paramètre	Description
-name <i>extensionname</i>	Le nom de l'extension tel qu'il apparaîtra dans JMap Admin. Peut contenir des espaces. Ex.: Suivi de marchandises, Analyse de réseaux, etc.
-class <i>classname</i>	Le nom complet (avec paquetage) de la classe principale de l'extension. Les noms des autres classes nécessaires à l'extension seront dérivées à partir de ce nom. Ex.: jmap.extensions.Tracking
-dest <i>destfolder</i>	Le répertoire de destination pour les fichiers générés. Si le répertoire existe déjà, il sera effacé et recréé. La valeur par défaut est <i>output</i> .
-target <i>target</i>	Le type d'extension à générer. Les valeurs acceptées sont client , server ou both . La valeur par défaut est <i>both</i> . Des classes et paquetages différents sont créés selon l'option sélectionnée.
-gui <i>true false</i>	Détermine si l'interface graphique doit s'afficher ou non. Les valeurs possibles sont true et false . La valeur par défaut est <i>true</i> . Si l'interface graphique est utilisée, ses champs seront initialisés avec les valeurs fournies en paramètre.

La méthode la plus simple pour exécuter le générateur d'extensions est d'utiliser le script *Ant* fourni avec l'outil. Les paramètres passés à l'outil peuvent être modifiés à l'intérieur du script *Ant*.

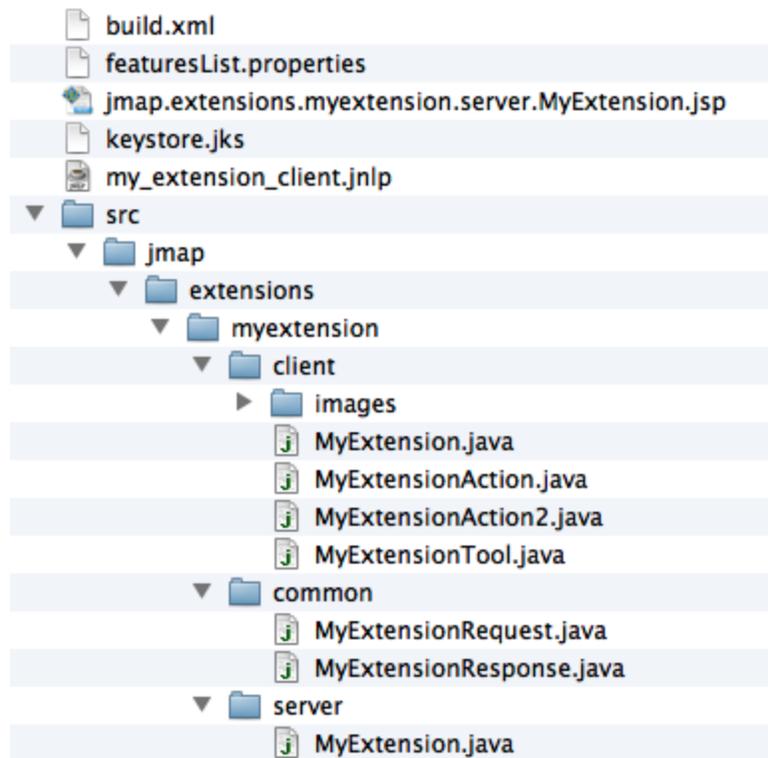
```
ant -f extensionbuilder.xml
```

Le générateur d'extensions peut aussi être exécuté à la ligne de commandes. (Sur Mac ou Linux, remplacer les " ; " par des " : " dans le *classpath* .)

```
java -classpath extensionbuilder.jar;../lib/kheops_util.jar jmap.sdk.tools.extensionbuilder.ExtensionBuilder -name extensionname -class classname -dest destfolder -target target -gui true|false
```

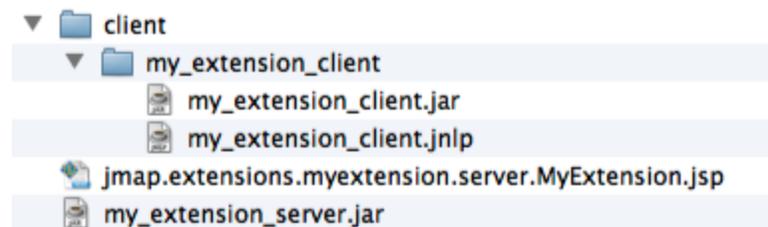
Après l'exécution de l'outil, le répertoire de destination contient les sources de l'extension générée ainsi que quelques autres fichiers. Si une extension client a été demandée (options *client* ou *both*), il y a un fichier JNLP, un fichier *keystore.jks* (pour la signature de votre extension client) et un fichier *featureslist.properties*. Si une extension serveur a été demandée (options *server* ou *both*), il y a un fichier JSP (interface de configuration de l'extension serveur). Un script *Ant* (*build.xml*) est aussi créé dans le répertoire. Ce script peut être utilisé pour compiler l'extension et générer les archives JAR pour le client et le serveur.

L'image suivante montre la structure des fichiers générés.



La structure des fichiers générés par le générateur d'extensions

Pour exécuter le script *Ant*, il suffit de lancer la commande *ant* à partir du répertoire de destination. Après l'exécution, le répertoire *dist* contient tous les fichiers à déployer sur JMap Server. L'image suivante montre la structure des fichiers générés.



La structures des fichiers générés par l'exécution du script Ant de l'extension

Développement JMap Web (6.5)

Cette section explique comment développer pour JMap Web et comment intégrer JMap Web au sein de d'autres applications web.

JMap Web est une application web qui fait utilisation de technologies tels que HTML5, JavaScript, CSS et JSON. L'application JMap Web est construite au dessus de bibliothèques JavaScript de tiers incluant OpenLayers, jQuery et plusieurs autres.

Brève vue d'ensemble du design de JMap Web

JMap Web fournit une application Web capable d'interagir avec JMap Serveur. L'ergonomie de son interface utilisateur est principalement adaptée pour les navigateurs web d'ordinateurs portables et d'ordinateurs de bureau.

Les éléments principaux d'une application JMap Web sont les suivants:

- Le client: Peut être décrit comme une application web à une seule page (index.jsp). Le client consiste en la page telle qu'elle sera rendue dans le navigateur des utilisateurs.
- Un Web Map Service: Donne accès aux tuiles de la carte.
- Le «Dispatcher AJAX»: Un service web auquel des requêtes HTTP sont envoyées. Le «Dispatcher AJAX» achemine ensuite les requêtes au «Action Handler» approprié. Ceci sera présenté davantage dans la section Envoyer des requêtes au serveur et actions personnalisées de ce document.

Bibliothèques JavaScript de tiers utilisées

JMap Web fait utilisation de plusieurs bibliothèques JavaScript de tiers. Vos extensions peuvent tirer parti de celles-ci sans aucune configuration additionnelle. Le tableau suivant fournit une brève explication de l'utilisation de ces bibliothèques par JMap Web.

Bibliothèque	Description
DataTables (v1.9.4)	Plugin jQuery qui facilite la création de tableaux HTML puissants pour la représentation de données. Ceci est principalement utilisé pour afficher les résultats de recherches.
fancybox (v2.15)	Affiche du contenu dans une interface de style «lightbox». JMap Web utilise fancyBox pour afficher des documents de taille-réelle inclus dans le contenu d'une infobulle.

Google Maps (v3)	Nécessaire pour afficher les couches de base «Roadmap», «Terrain», «Satellite» et «Hybride» de Google Maps.
jQuery (v1.9.1)	jQuery est une bibliothèque JavaScript rapide, petite, et riche en fonctionnalités. Elle simplifie la navigation du document HTML, la manipulation d'éléments, la gestion des événements, l'animation, et les requêtes AJAX. Son API est facile à utiliser et fonctionne sur une multitude de navigateurs.
jQuery-ui (v1.11.4)	Bibliothèque d'éléments d'interface utilisateur nécessitant jQuery. Présentement, JMap Web utilise une version personnalisée qui n'inclut que la composante «autocomplete».
moment.js (v2.10.6)	Bibliothèque utilisée pour lire, valider, manipuler et afficher des dates en JavaScript.
malihu-custom-scrollbar-plugin (v3.0.7)	Ajuste l'apparence des barres de défilement dans les infobulles. Sa fonction principale est de standardiser l'apparence des barres de défilement dans les différents navigateurs web.
OpenLayers (v2.13.1)	Ceci est la bibliothèque la plus importante de JMap Web. OpenLayers permet l'affichage de la carte dans JMap Web et gère également la majorité des interactions avec celle-ci.
Proj4js (v1.1.0)	<p>Proj4js est une bibliothèque qui permet de traduire des coordonnées géographiques d'un système de projection cartographique à l'autre. Proj4js est utilisé par OpenLayers, mais n'est pas fourni avec celle-ci.</p> <p>Afin de pouvoir accomplir des transformations pour des projections nommées (ex: "EPSG:3857"), les projections doivent être définies dans Proj4js. Seulement un petit nombre de définitions de projections est disponible par défaut.</p> <p>JMap Web inclut un fichier de définition de projection Proj4js pour chacune des projections supportées par JMap Serveur. Les définitions de projections seront chargées en mémoire au fur et à mesure qu'elles seront requises.</p>
Twitter Bootstrap (3.1.1)	«Framework front-end» pour créer des interfaces utilisateur.
bootstrap-datetimepicker (v4.15.35)	Outil pour sélectionner une date/heure avec Twitter Bootstrap.
bootstrap-multiselect (v0.9.10)	Outil pour sélectionner des choix multiples avec Twitter Bootstrap.

Bibliothèques JavaScript de JMap et leur architecture

En plus des bibliothèques décrites précédemment, JMap Web inclut également ses propres bibliothèques basées sur le type de base «Class» d'OpenLayers. Ceci permet à JMap Web de bénéficier des paradigmes de la programmation orientée objet dans un contexte JavaScript (qui favorise généralement une approche plutôt prototypique et fonctionnelle.).

Une bibliothèque JavaScript JMap consiste essentiellement en un regroupement de fichiers JavaScript, de feuilles de styles en cascade (CSS) et de ressources (images, sons, etc.).

Les bibliothèques JavaScript de JMap ont été conçues pour une utilisation dans d'autres modèles d'applications JMap de la famille «JMap Web/JMap Mobile». En divisant le code parmi plusieurs bibliothèques, la réutilisation du code est possible dans divers environnements tels JMap Web et JMap Mobile. Les bibliothèques JavaScript JMap définissent des comportements uniques pour des contextes spécifiques.

JMap Web utilise les bibliothèques JavaScript JMap suivantes:

- core
- desktop_ui

Les classes modèles réutilisables de JMap sont généralement incluses dans core. Cette bibliothèque est donc fréquemment utilisée et est présente dans d'autres modèles d'applications JMap de la famille «JMap Web/JMap Mobile».

La bibliothèque core est écrite en JavaScript ES5. En plus de classes modèles JMap, elle comporte des contrôles OpenLayers réutilisables, des classes couches OpenLayers additionnelles, des «wrappers» de fonctionnalités HTML5 et plus. La bibliothèque core est la bibliothèque de plus bas niveau. Elle s'occupe de tâches générales telles que l'initialisation de la carte. Tout autre bibliothèque est ensuite chargée au-dessus comme si elle était une extension.

Quant à elle, la bibliothèque desktop_ui contient des classes qui se concentrent plus sur l'affichage et la gestion des éléments visuels. Elle fait utilisation des bibliothèques DataTables, jQuery et Twitter Bootstrap. Tout élément visuel de l'interface de JMap Web qui n'est pas géré par OpenLayers provient de desktop_ui.

Depuis la version 6.5, le modèle d'application JMap Web inclut un API public permettant au développeurs de facilement s'intégrer aux fonctionnalités de JMap.

Une documentation JSDoc générée pour cet API JavaScript est disponible en ligne sur <http://dev.k2geospatial.com/jmap/web/api/6.5/>

Avant d'aborder la création d'une extension JMap Web, cette section du document va expliquer le processus de démarrage de JMap Web.

Note : Cette section fait référence au modèle d'application JMap Web tel qu'il est inclus au moment de l'installation de JMap Serveur. Ce processus, brièvement décrit ici, va être différent si vous choisissez d'ajouter un déploiement JMap Web au sein de votre propre application web existante. Pour plus de détails, consultez la section Intégrer JMap Web dans votre propre application de ce document.

Les fichiers du modèle d'application JMap Web sont disponibles sous le répertoire `$JMAP_HOME$/applications/templates/html/web`.

index.jsp

Ceci est le document web qui sera servi par JMap Serveur lorsque l'application sera demandée par l'utilisateur. Ouvrez ce fichier dans un éditeur de texte. Comme vous pouvez le remarquer, `jQuery` et `jmap.min.js` sont chargés dans la balise `head` du document.

Plus bas, la carte est initialisée lorsque le document sera prêt à être manipulé.

```
$(document).ready(function() {
  var options = {
    jmapUrl: '$APPLICATION_PROTOCOL$://$APPLICATION_HOST$: $APPLICATION_WEBPORT$ $PATH$',
    mapConfig: {
      // Configuration properties
    },
    onMapInit: function() {
      console.log('Map was initialized.');
    }
  };

  var resizeUi = function() {
    if (JMap.app && JMap.app.map)
    {
      $(JMap.app.map.div).height($(window).height()).width($(window).width());
      JMap.app.map.updateSize();
      JMap.app.popupManager.updateDrawnPopups();
    }
  };

  $(window).on('resize', resizeUi);

  // Object auto zoom handling...
}

// Region auto zoom handling...

$('#map').height($(window).height()).width($(window).width());
JMap.initialize(document.getElementById('map'), options);
});
```

JMap.initialize(map, options)

La fonction `JMap.initialize` est définie dans le fichier `jmap.min.js`. Elle charge toutes les dépendances nécessaires et va ensuite initialiser tous les objets requis par l'application. Parmi ces objets, `JMap.app` et `JMap.app.map` qui demeureront globalement disponibles.

L'objet `JMap.app` est une instance de la classe `Application` de la bibliothèque `core`. L'objet `JMap.app.map` est une instance de la classe `OpenLayers.Map`.

Paramètres	Description
map	Requis {String} L'élément ou l'identifiant d'un élément existant sur la page qui va contenir la carte.
options	<p>Requis {Object} Doit au minimum fournir une propriété <code>jmapUrl</code> {String}. Supporte les clefs de propriétés suivantes:</p> <ul style="list-style-type: none"> • <code>jmapUrl</code>: Requis. {String} L'adresse de votre déploiement JMap Web. • <code>mapConfig</code>: {Object} Options additionnelles assistant la configuration de la carte. • <code>onMapInit</code>: {Function} <i>Handler</i> qui sera appelé une fois que le processus d'initialisation de la carte est complété. Pratique afin d'ajouter de nouveaux contrôles OpenLayers une fois que la bibliothèque OpenLayers sera chargée.

L'objet `mapConfig` peut inclure les clefs de propriétés suivantes afin de personnaliser l'application. La valeur par défaut est identifiée en gras.

Propriétés du mapConfig	Description
Options de la carte	
displayUnits	null 'm' 'km' 'ft' 'mi' 'in' '' {String} L'unité de la carte. Par défaut sera la valeur du projet JMap.
initialZoom	null {Array} Coordonnées géographiques qui décrivent l'étendue: [gauche, bas, droite, haut]
mapUnits	null 'm' 'km' 'ft' 'mi' 'in' '' {String} L'unité de la carte. Par défaut sera la valeur du projet JMap.
maximumExtent	null {Array} Coordonnées géographiques qui décrivent l'étendue: [gauche, bas, droite, haut]
projection	null {Object} Si fourni, l'objet doit inclure une propriété String code qui correspond à un code EPSG.
Couches additionnelles	

addShowPositionLayer	true false
Services au démarrage	
activateGeolocationServiceOnLaunch	true false
loadGoogleMapsApiOnLaunch	true false - Sera automatiquement remplacé à true si le déploiement contient au moins une couche Google ou si au moins un parmi ceux-ci est vrai : addGoogleDirections, addGoogleGeocoding, addGoogleStreetView.
Options de la carte et de la navigation	
addGeolocateButton	true false
addInitialViewButton	true false
addMapOverview	true false
addMousePosition	true false
addPanControls	true false
addScaleBar	true false
addZoomInButton	true false
addZoomOutButton	true false
isMapOverviewMaximized	true false
Outils JMap	
addInfoReportTool	true false
addMeasureAreaTool	true false
addMeasureDistanceTool	true false
addMeasureCircularAreaTool	true false
addMouseOverTool	true false
addRedLiningTool	true false
Fonctionnalités d'édition JMap	

addEditionTools	true false
addEditionCreateElementTools	true false - La propriété addEditionTools doit également être true pour que ceci prenne effet.
addEditionSelectElementTool	true false - La propriété addEditionTools doit également être true pour que ceci prenne effet.
addEditionShowElementFormButton	true false - La propriété addEditionTools doit également être true pour que ceci prenne effet.
Fonctionnalités de sélection JMap	
addSelectionTools	true false
addCircleSelectionTool	true false - La propriété addSelectionTools doit également être true pour que ceci prenne effet.
addLineSelectionTool	true false - La propriété addSelectionTools doit également être true pour que ceci prenne effet.
addPointSelectionTool	true false - La propriété addSelectionTools doit également être true pour que ceci prenne effet.
addRectangleSelectionTool	true false - La propriété addSelectionTools doit également être true pour que ceci prenne effet.
addShapeSelectionTool	true false - La propriété addSelectionTools doit également être true pour que ceci prenne effet.
Autres fonctionnalités	
addFullScreenButton	true false
addLayersMainMenuitem	true false
addLogo	true false
addLogoutAsideMenuLink	true false
addParametersAsideMenuLink	true false
addPrintButton	true false
addSearchMainMenuitem	true false
Fonctionnalités de tiers	

addGoogleDirections	true false
addGoogleGeocoding	true false
addGoogleStreetView	true false

Après le processus d'initialisation

Une fois que la carte est initialisée avec succès, vous pouvez l'accéder à l'aide de JavaScript en utilisant la variable globale `JMap.app.map`.

La manipulation de la carte sera possible en accédant à la variable globale `JMap.app.map`. Cette variable est l'instance de `OpenLayers.Map` en utilisation. Vous pouvez également interagir avec elle en utilisant l'API d'OpenLayers.

Paramètres d'URL

JMap Web supporte les paramètres d'URL suivants:

```
autozoom {String}
```

Ce paramètre permet de spécifier une étendue sur laquelle la vue de la carte sera initialisée. Deux types d'`autoZoom` peuvent être utilisés, `region` ou `object`. Le type est déterminé par le premier argument '`type`'.

Types d' AutoZoom	Description
Region	<p>L'utilisateur précise la zone visible en décrivant un rectangle...</p> <p>Syntaxe: "type;x;y;width;height"</p> <ul style="list-style-type: none"> • type: {String} Le type de la requête. Dans ce cas, doit être "region". • x: {Number} La valeur X de la coordonnée inférieure gauche du rectangle. • y: {Number} La valeur Y de la coordonnée inférieure gauche du rectangle. • width: {Number} La largeur du rectangle. • height: {Number} La hauteur du rectangle. <p>Exemple: ?autozoom=region;9;39;20;20</p>
Object	<p>L'utilisateur précise un objet sur lequel il veut faire le zoom...</p> <p>Syntaxe: "type;layerName;field;value;maxScale"</p>

- `type: {String}` Le type de la requête. Dans ce cas, doit être "object".
- `layerName: {String}` Le nom de la couche qui contient l'objet sur lequel on veut faire le zoom.
- `field: {String}` Le champs qui contient la valeur de l'objet sur lequel on veut faire le zoom.
- `value: {Number | String}` La valeur spécifique de l'objet sur lequel on veut faire le zoom. Si le champs est une chaîne de caractères, utilisez ' '. Voir l'exemple plus bas.
- `maxScale: {Number, optionnel}` L'échelle maximale que la carte doit respecter lorsqu'elle affiche les résultats.

Exemple d'une valeur numérique: ?

```
autozoom=object;citiesLayer;city_id;1032
```

Exemple d'une valeur chaîne de caractères: ?

```
autozoom=object;citiesLayer;city_name;'montreal';15
```

Cette section donne un aperçu de la façon de développer des extensions pour JMap Web.

Les extensions sont modulaires et offrent généralement des fonctionnalités adaptés pour des tâches spécifiques. Par défaut, JMap Web offre un ensemble de fonctions de base. Les extensions JMap Web permettent la personnalisation et l'ajout de nouvelles fonctions aux déploiements basés sur le modèle d'application JMap Web.

Tout comme les extensions de JMap Pro, les administrateurs JMap peuvent choisir quelles extensions ils/elles veulent déployer lors de l'assistant de déploiement d'applications. Les extensions choisies seront chargées au moment du processus d'initialisation de JMap Web.

Similaire aux bibliothèques JavaScript de JMap, une extension JMap Web consiste en un regroupement de fichiers JavaScript, feuilles de styles (CSS) et ressources (images, sons, etc.).

Vous pouvez utiliser l'outil **JMap Web extension builder** afin de rapidement créer les fichiers nécessaires à une extension Web. Ouvrez une fenêtre de ligne de commande et naviguez au répertoire `tools/webextensionbuilder` du SDK de JMap 6.5 installé sur votre ordinateur.

Utilisation du générateur d'extensions JMap Web

Vous trouverez le fichier `webextensionbuilder.xml` . Ouvrez-le dans un éditeur de texte. Modifiez-le de manière à spécifier une liste d'arguments qui vous permettront de créer votre extension. Les arguments suivants sont exigés :

Argument	Description
fullname	{String} Le nom complet lisible de l'extension. C'est ce que les administrateurs et les utilisateurs JMap verront. Un «shortname» sera dérivé depuis cette valeur. Le «shortname» sera utilisé entre autre comme le nom de plusieurs fichiers.
namespace	{String} Le nom du «namespace» JavaScript (une variable <i>Object</i> globale) de votre extension. Si vous planifiez déployer plusieurs de vos extensions simultanément, vous pouvez spécifier un «namespace» qui contient <u>au plus</u> un point «.» de manière à séparer les éléments de votre «namespace». Exemple: MonEntreprise.HelloWorld.
version	{String} Le numéro de version de votre extension. Aucun format spécifique est imposé.
dest	L'endroit sur le disque où votre extension sera créée.

Exemple d'un fichier `webextensionbuilder.xml` modifié qui sera utilisé pour créer votre extension :

```
<project name="JMap 6.5 SDK - Web Extension Builder" basedir="." default="run">

  <!-- set global properties for this build -->
  <property file="../../sdk.properties"/>

  <target name="run">
    <java fork="true" classname="jmap.sdk.tools.webextensionbuilder.WebExtensionBuilder"
      classpath="webextensionbuilder.jar;${sdk_classpath}">
      <!-- Use default parameter values. Uncomment following line to use other parameters
      <arg line="-fullname 'Web SDK Documentation Example' -namespace Example -version 1.0
    </java>
  </target>
</project>
```

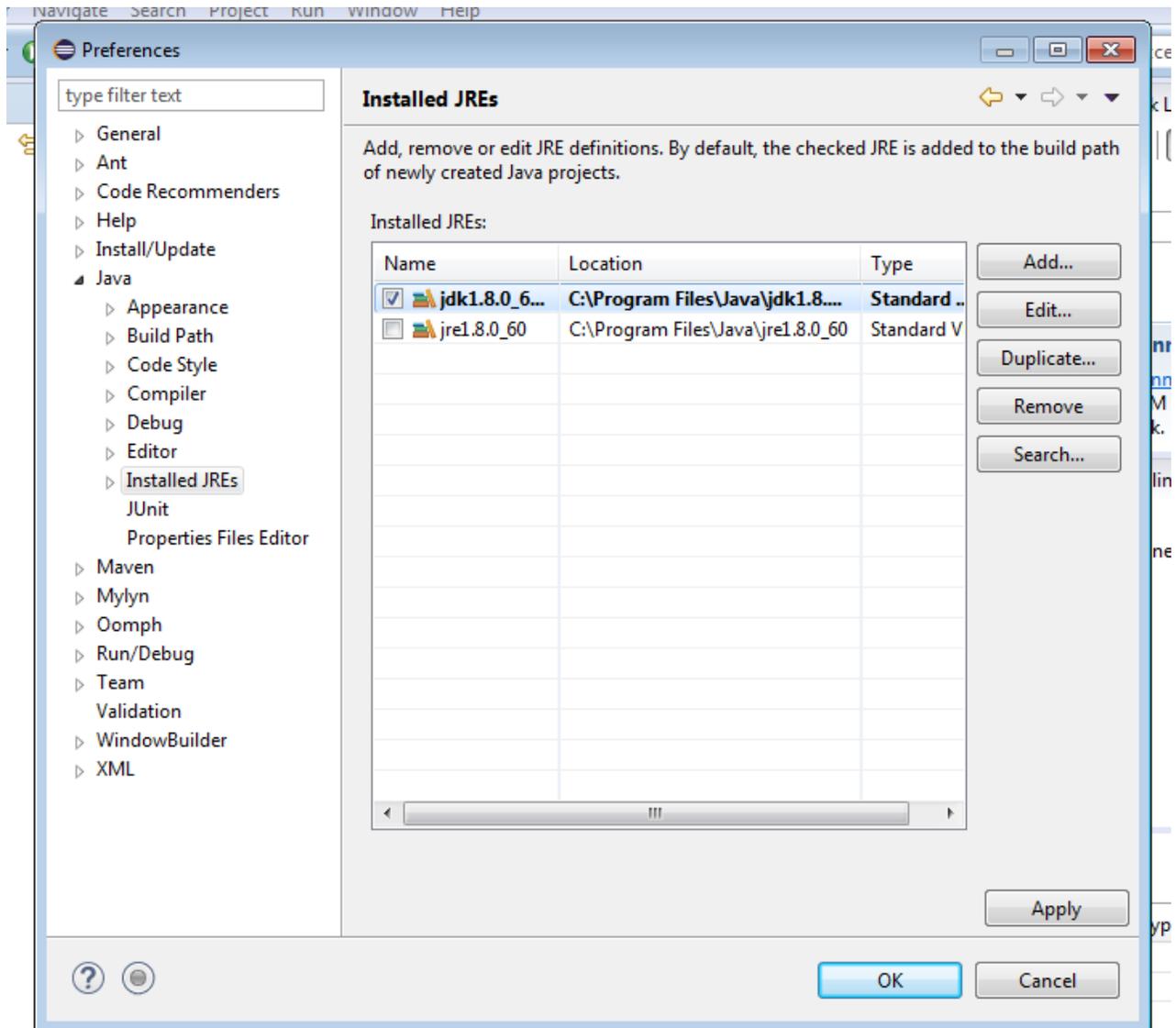
Création de l'extension à l'aide de ant avec la ligne de commande

Depuis le terminal, invoquez le script ant :

```
ant -f webextensionbuilder.xml
```

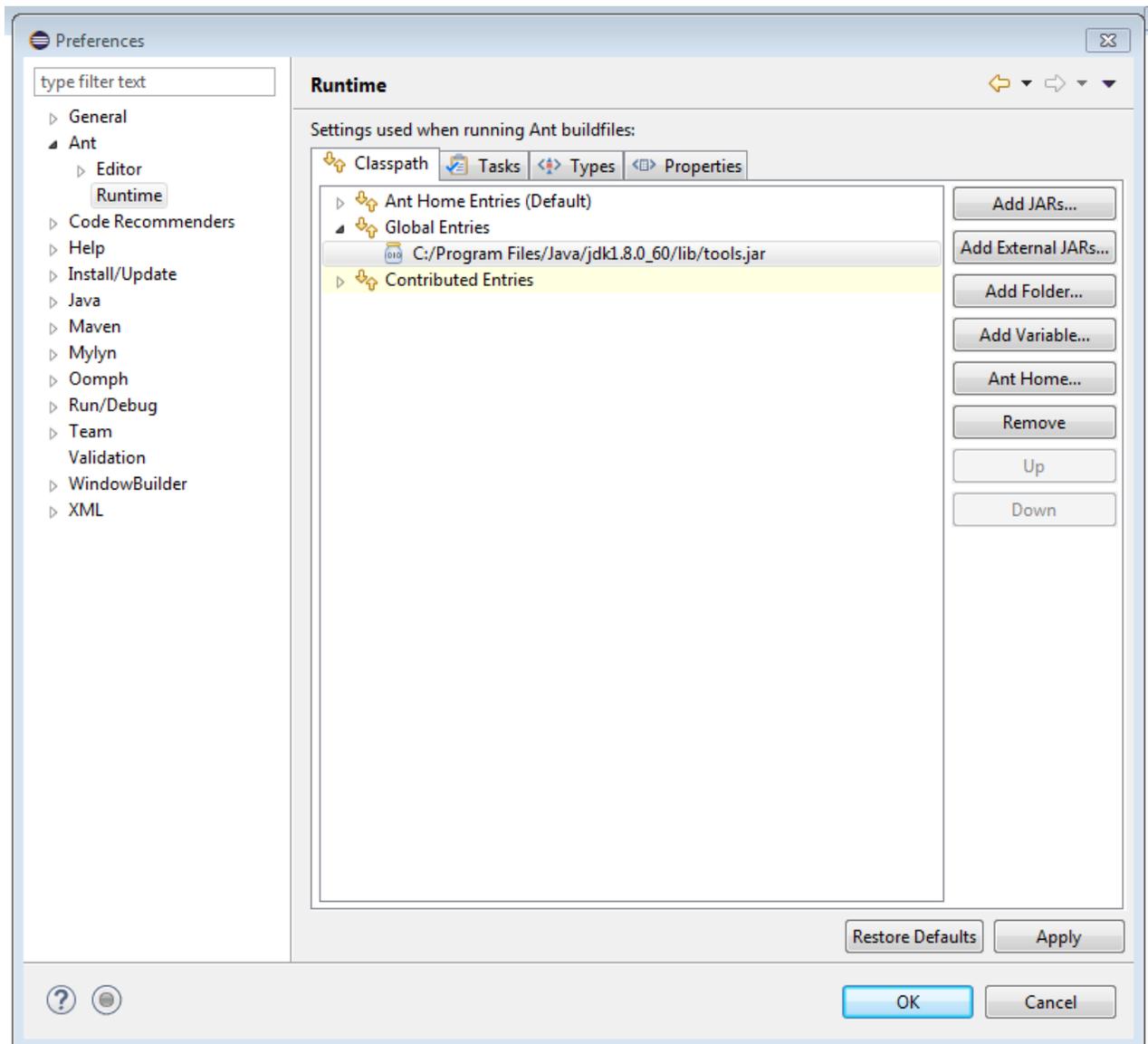
Création de l'extension à l'aide de ant sous Eclipse

Pour créer une extension JMap Web à l'aide d'Eclipse, confirmer d'abord qu'un Java Developer Kit (JDK) est installé sur votre poste et est configuré comme étant le JRE par défaut d'Eclipse.



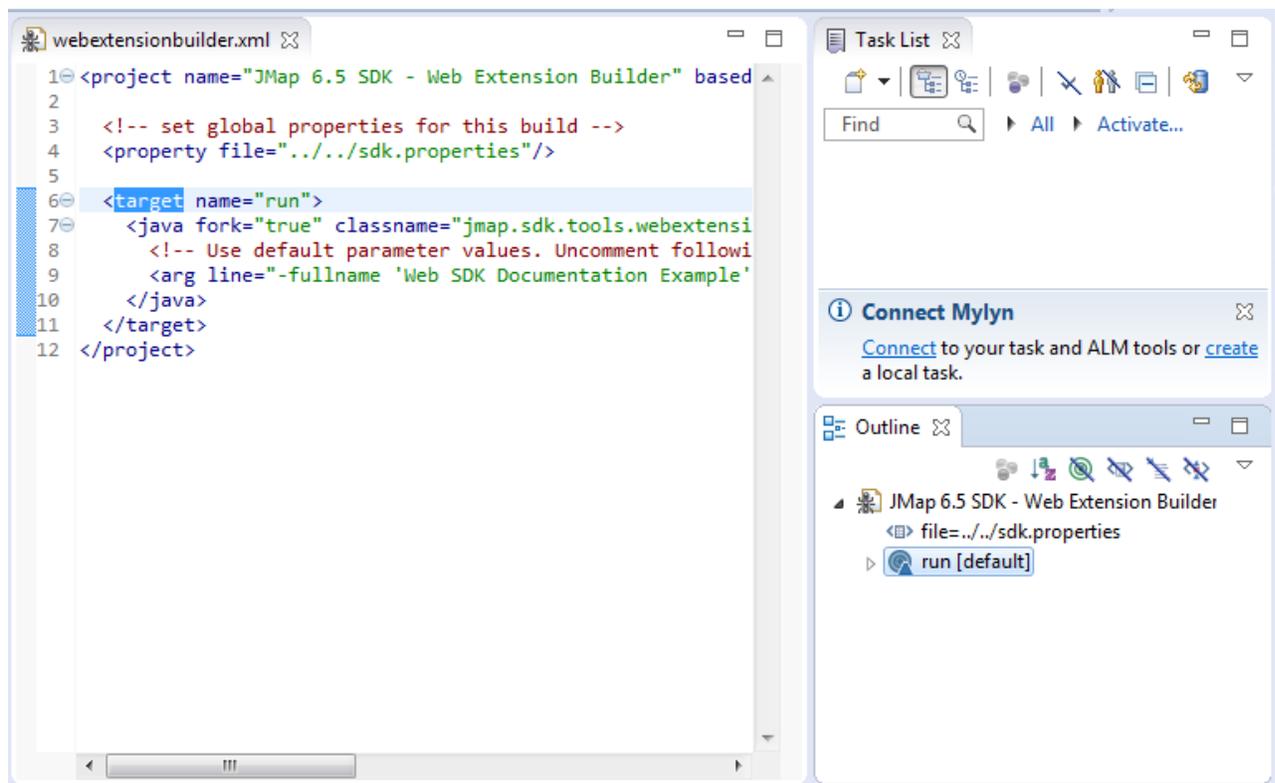
Les JREs d'Eclipse

Vous devriez également confirmer que le fichier `tools.jar` de votre JDK soit ajouté au *Ant Runtime Configuration* d'Eclipse:



Le Ant Runtime Configuration d'Eclipse. Ajoutez le `JDK_HOME/lib/tools.jar` à la section *Global Entries* de la *class path*.

Exécutez la tâche de build en ouvrant le fichier `webextensionbuilder.xml` dans Eclipse. Exécutez la cible «run».



Éxécutez la cible «run»

Comprendre les fichiers de l'extension

Après l'exécution du script ant, vous allez posséder le minimum de code d'une extension à l'emplacement spécifié.

output

```
+ - - web_sdk_documentation_example
+ - - actions
|   + - - build.xml
|   + - - readme.markdown
|   + - - src
|     + - - Example
|       + - - SampleAction.java
+ - - assets
|   + - - css
|     | + - - web_sdk_documentation_example.css
|     + - - js
|       + - - web_sdk_documentation_example.js
+ - - extension.json
```

7 directories, 6 files

Par défaut, ainsi que pour des fins de démonstration, une action du AJAX Dispatcher nommée `web_sdk_documentation_example_sample_action` est incluse. Pour plus de détails sur les actions du AJAX Dispatcher, référez-vous à la section Envoyer des requêtes au serveur et actions personnalisées.

Les fichiers CSS et JavaScript de votre extension seront chargés durant le processus d'initialisation de JMap Web.

Tout autre ressource CSS et JavaScript devra être chargée par programmation depuis le fichier JavaScript généré (dans ce cas-ci, le fichier `websdkdocumentation_example.js`).

Puisque les fichiers de votre extension seront chargés au moment de l'exécution, il est important de ne pas renommer ou déplacer les fichiers CSS et JavaScript générés. Le nom de ces fichiers devra toujours correspondre au «shortname» défini dans le fichier `extension.json`.

Le code source généré du fichier JavaScript comporte des commentaires qui décrivent les propriétés et fonctions **requis** qui forment votre extension JMap Web. Il est recommandé de faire la lecture de ce fichier afin que vous familiarisiez avec son contenu.

Parmi les fichiers générés se trouve `extension.json` . Ce fichier va servir de manifest pour JMap Serveur. Il doit contenir du JSON valide. Ce fichier ne doit pas être détruit, déplacé ou renommé.

Interaction avec OpenLayers

Tel que mentionné précédemment, une variable globale `JMap.app` est définie durant l'initialisation. Une des responsabilités de cet objet est de maintenir une référence à la carte de votre application.

`JMap.app.map` est l'instance de la classe `OpenLayers.Map` de votre application. Les bibliothèques JavaScript de JMap font utilisation de l'API d'OpenLayers afin de pouvoir accomplir une multitude de tâches. Des tâches telles que la manipulation des couches, des opérations sur des coordonnées/étendues en plus de la gestion des contrôles et des infobulles de la carte.

OpenLayers offre une vaste galerie d'exemples portant sur le développement. Il est recommandé que vous utilisiez cette ressource pendant votre développement afin de vous familiariser avec leur API. La galerie ainsi que la documentation de l'API sont des ressources indispensables que vous devriez consulter fréquemment.

Exemples

Cette portion de la documentation présente des exemples de quelques tâches qui peuvent être accomplies dans une extension JMap Web.

Ajouter un bouton

L'API d'OpenLayers comporte une classe `OpenLayers.Control.Button`. Elle permet de facilement ajouter un bouton à l'interface de JMap Web. Ce bouton sera ensuite capable de lancer des événements lorsqu'il sera cliqué. Il peut également être stylisé avec la feuille de styles de votre extension.

Faisant parti de l'interface de JMap Web, le panneau de contrôle principal présente une collection de `OpenLayers.Control.Button` . En fait, ce panneau est une instance de `OpenLayers.Control.Panel`.

Cet exemple démontre comment vous pouvez créer votre propre bouton et l'ajouter au panneau de contrôle principal.

Ajoutez le bout de code suivant à la méthode *init* de votre extension. Faites une mise-à-jour de votre déploiement par la suite.

```
/*
 * Returns an array of OpenLayers.Control instances that match the criteria.
 */
var panels = JMap.app.map.getControlsBy('displayClass', 'jmapMainPanel');

/*
 * By default, there's only one instance of OpenLayers.Control.Pane that has
 * its displayClass set to 'jmapMainPanel'.
 *
 * Therefore, you can directly access the first item in
 * the array. This is for demonstration purposes only.
 */
var myPanel = panels[0];

var myButton = new OpenLayers.Control.Button({
  /*
   * CSS class names applied to your button's <div> will be prefixed with this
   * string.
   */
  displayClass: 'myButtonClass',
  /*
   * The function that will be launched once your button clicks.
   */
  trigger: function(event) {
    alert('The button was clicked');
  }
});

/*
 * Add the newly created button to the map.
 */
myPanel.addControls([myButton]);
```

Un nouveau bouton devrait être visible. Il est représenté par le nœud *DOMElement* suivant:

```
<div class="myButtonClassItemInactive olButton"></div>
```

Au moment de l'initialisation du bouton, il est possible de définir un type correspondant soit à *OpenLayers.Control.TYPE_BUTTON* (défaut), *OpenLayers.Control.TYPE_TOGGLE* (bascule entre un état actif/inactif à chaque clic) ou *OpenLayers.Control.TYPE_TOOL* (Seulement un seul bouton parmi le *OpenLayers.Control.Panel* peut être activé à tout moment).

Afin d'encourager une apparence cohérente parmi les boutons du panneau de contrôle principal, les styles suivants seront appliqués:

```
.jmap .jmapMainPanel > .olButton {
  display: inline-block;
  height: 40px;
  width: 40px;

  cursor: pointer;

  background-color: #636D6F;
  background-image: url("../images/Sprite_40x40.png");
  background-repeat: no-repeat;
  box-shadow: 0px 0px 3px rgba(19, 18, 18, 0.26);

  moz-box-shadow: 0px 0px 3px rgba(19, 18, 18, 0.26);
  webkit-box-shadow: 0px 0px 3px rgba(19, 18, 18, 0.26);
}
```

Il est recommandé que vous utilisiez ces styles comme point de départ, cependant vous pouvez définir vos propre styles via CSS:

```
.myButtonClassItemInactive {
  background-color: #F00 !important;
}

/*
 * Only applicable if you specified a `OpenLayers.Control.TYPE_TOGGLE`
 * button type.
 */
.myButtonClassItemActive {
  background-color: #0F0 !important;
}
```

Vous n'avez pas à spécifier une propriété *trigger* si vous avez spécifié un type de bouton `OpenLayers.Control.TYPE_TOGGLE`. Vous devriez alors enregistrer un écouteur sur le bouton.

```
myButton.events.register('activate', this, function(event) {
  alert('On!');
});

myButton.events.register('deactivate', this, function(event) {
  alert('Off!');
});
```

La gestion des outils

Note: La connaissance du type de base `OpenLayers.Class` est fortement recommandée avant de continuer avec cette documentation.

Plusieurs classes ont été développées de façon à supporter des types d'interactions variées lorsqu'un événement tactile/souris est commis. Les voici:

JMap.Html5Core.Control.TouchClickControl : Cette classe hérite de *OpenLayers.Control*. Une seule instance est créée et ajoutée à la carte durant le processus d'initialisation. La propriété *handler* de ce contrôle peut détecter deux types d'événements: *click* et *holdreleased*. Vos outils peuvent implémenter des fonctions vous permettant de réagir à ces deux types d'événements.

JMap.Html5Core.Tool.ToolManager : Durant le processus d'initialisation, une seule instance de cette classe est produite et est rendue accessible depuis la variable *JMap.app.clickToolManager*. Cet objet fait la gestion de tous les outils enregistrés en plus de l'outil présentement activé. Le *currentTool* de cet objet sera le destinataire des événements *click* et *holdreleased* détectés par l'instance de *JMap.Html5Core.Control.TouchClickControl*.

JMap.Html5Core.Tool.Tool : Super classe depuis laquelle tous les outils doivent hériter. Vos écouteurs d'événements *click* et *holdreleased* doivent être définis dans votre sous classe ou instance de *Tool*.

Le code suivant crée un outil:

```
var sampleTool = new JMap.Html5Core.Tool.Tool({
  name: 'Sample Tool',
  clickHandler: function(event) {
    var coord = JMap.app.map.getLonLatFromPixel(event.xy);
    alert('Clicked at lon: ' + coord.lon + ', lat: ' + coord.lat);
  },
  holdReleasedHandler: function(event) {
    alert(`holdreleased` event performed.`);
  },
});
```

Après son initialisation, vous devez l'enregistrer auprès du *JMap.app.clickToolManager* puis l'activer.

```
JMap.app.clickToolManager.addTool(sampleTool);
JMap.app.clickToolManager.setCurrentTool(sampleTool);
```

Si un outil était déjà activé au moment où *setCurrentTool()* est appelé, cet outil sera désactivé.

Note: Les outils de la bibliothèque core de JMap sont inclus dans cette documentation afin de vous aider à créer vos propres outils.

Travailler avec des couches vectorielles

OpenLayers offre plusieurs classes permettant la visualisation de plusieurs types de données. La classe *OpenLayers.Layer.Vector* est particulièrement utile afin d'afficher des éléments de données vectorielles créés par programmation.

Le code suivant crée et ajoute une couche vectorielle à une application initialisée. En utilisant votre outil personnalisé, vous serez capable d'ajouter des points lors que des événements clic seront commis.

```
var vectorLayer = new OpenLayers.Layer.Vector('Vector Layer Tool Example', {});
JMap.app.map.addLayer(vectorLayer);

var vectorLayerTool = new JMap.Html5Core.Tool.Tool({
  name: 'Vector Layer Tool',
  clickHandler: function(event) {
    var coord = JMap.app.map.getLonLatFromPixel(event.xy),
        point = new OpenLayers.Geometry.Point(coord.lon, coord.lat),
        feature = new OpenLayers.Feature.Vector(point);

    vectorLayer.addFeatures([feature]);
  },
  holdReleasedHandler: function(event) {
    if (typeof vectorLayer === 'object' && vectorLayer instanceof OpenLayers.Layer.Vector)
      vectorLayer.removeAllFeatures();
  },
});
```

De façon similaire, tel que décrit dans l'exemple précédent, le code suivant va enregistrer votre outil auprès du `JMap.app.clickToolManager` puis l'activer.

```
JMap.app.clickToolManager.addTool(vectorLayerTool);
JMap.app.clickToolManager.setCurrentTool(vectorLayerTool);
```

Modifier des données vectorielles

Présentement, JMap Web n'offre pas d'API permettant d'éditer des données vectorielles. Cependant, vous pouvez utiliser l'API d'OpenLayers afin de créer, modifier et détruire vos propres éléments associés à vos propres couches vectorielles qui furent créées par programmation.

Voici quelques liens expliquant les fonctionnalités d'édition d'OpenLayers:

- [Drawing Simple Vector Features Example](#)
- [OpenLayers Draw Feature Example](#)
- [Drag Feature Example](#)
- [OpenLayers Select Feature Example](#)
- [OpenLayers Modify Feature Example](#)
- [Feature Events Example](#)

Travailler avec les styles vectoriels

Il est possible de définir l'apparence de vos éléments de couches personnelles. Ceci est généralement accompli en associant un `OpenLayers.StyleMap` à votre couche.

Un StyleMap peut contenir n'importe laquelle des propriétés décrites ici afin de modifier l'apparence des éléments de la couche.

- Feature Styles Example

Recommandations

Dans un environnement de production, il est recommandé que les fichiers de votre extension soient minifiés afin de limiter le nombre de requêtes HTTP au chargement. Ceci permet également de masquer votre code jusqu'à un certain niveau. Nous utilisons le Closure Compiler de Google et le recommandons.

Tel que mentionné précédemment, si votre extension possède des dépendances, elles devront être chargées par programmation depuis le fichier JavaScript généré de votre extension.

La documentation de l'API d'OpenLayers est générée en par l'analyse des commentaires du code source. Ceci signifie que certaines méthodes ou propriétés existantes peuvent ne pas être documentées. La navigation du code source d'OpenLayers peut vous aider si quelque chose dans la documentation n'est pas clair.

Méthodologie du développement d'extensions

Les développeurs peuvent souhaiter de travailler différemment lorsqu'ils développent des extensions JMap Web. Les deux méthodes suivantes sont généralement favorisées:

1. Le développeur travail dans le répertoire de l'extension générée. Il modifie les fichiers de l'extension, copie ses fichiers au répertoire `$JMAP_HOME$/extensions/web`, fait la mise-à-jour de son déploiement puis test les changement dans son application.
2. Le développeur travail directement dans les fichiers déployés de son application.

Les deux approches ont leur propres avantages et inconvénients.

La première technique, bien que plus lente, permet un processus plus sécuritaire et incrémental. Faire le suivi des changements du code est plus simple.

La seconde technique est plus rapide, mais demandera au développeur de copier ses changements vers sa copie de travail obtenue avec son logiciel de contrôle de version. Cette approche peut également mener à la perte de données si une mise-à-jour de l'application est faite avant que les changements ne soient reproduits dans le répertoire `$JMAP_HOME$/extensions/web`.

En plus d'exécuter du code dans le navigateur de l'utilisateur, une extension JMap Web peut également définir des actions qui peuvent tirer parti de l'API de JMap Serveur. Ces actions peuvent être enregistrées auprès du service AJAX Dispatcher de votre déploiement et être interpellées par des requêtes HTTP.

Une extension JMap Web peut contenir plusieurs actions. Vous pouvez également créer des classes additionnelles (qui ne sont pas des actions) qui peuvent être référencées depuis vos actions.

Créer une action

Créez une classe qui hérite de *AbstractHttpRequestAction* . Afin de suivre cet exemple, le nom de votre classe devrait être *MyFirstAction* .

```
package myextension;
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import jmap.http.AJAX.servlets.AbstractHttpRequestAction;

public class MyFirstAction extends AbstractHttpRequestAction
{
    @Override
    public void execute(HttpServletRequest request, HttpServletResponse response) throws IOException
    {
    }
}
```

Comme vous avez pu le remarquer, une action nécessite les classes *HttpServletRequest* et *HttpServletResponse* afin d'être compilée. Ajoutez le le JAR *servlet-api* de *Tomcat Server* à votre chemin de compilation (*build path*). Ce JAR peut être récupéré dans le répertoire *\$JMAP_HOME\$/tomcat/lib* .

Une action doit posséder une méthode *execute* . Cette méthode sera appelée lorsqu'une requête HTTP spécifiant votre action sera reçue par l'AJAX Dispatcher.

Pour les besoins de cet exemple, voici une action *Hello World!* typique.

```
package myextension;
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import jmap.http.AJAX.servlets.AbstractHttpRequestAction;

public class MyFirstAction extends AbstractHttpRequestAction
{
    @Override
    public void execute(HttpServletRequest request, HttpServletResponse response) throws IOException
    {
        final PrintWriter writer = response.getWriter();
        String name = request.getParameter("name");

        if (name != null && name.trim() != "")
            writer.print("Hello, " + name);
        else
            throw new IllegalArgumentException("Invalid argument. Must specify a `name` parameter");
    }
}
```

Produire un fichier JAR pour votre action

Afin d'être inclut dans une extension JMap Web, votre code serveur Java doit être contenu dans un seul fichier JAR.

En utilisant Ant

En utilisant l'outil `webextensionbuilder` du SDK de JMap 6.5 (décrit dans la section Programmer des extensions JMap Web) afin de produire le code de base de l'extension, une action `SampleAction` est fournie comme point de départ. Un fichier `build.xml` est également inclut. Vous pouvez utiliser ce fichier avec Ant afin de compiler et produire le fichier JAR de votre extension.

Copiez le fichier `MyFirstAction.java` au répertoire `actions/src` de votre extension. Puisque la classe `MyFirstAction` est définie à l'intérieur du package `myextension`, créez un répertoire `myextension` sous `actions/src` et collez-y votre fichier `MyFirstAction.java`. À ce stade, vous devriez avoir les items suivant dans votre répertoire actions:

```
actions/
+-- build.xml
+-- readme.markdown
+-- src
    +-- Example
    |   +-- SampleAction.java
    +-- myextension
        +-- MyFirstAction.java

3 directories, 4 files
```

En utilisant la ligne de commande/un terminal, naviguez vers le répertoire actions de votre extension et exécutez la commande suivante:

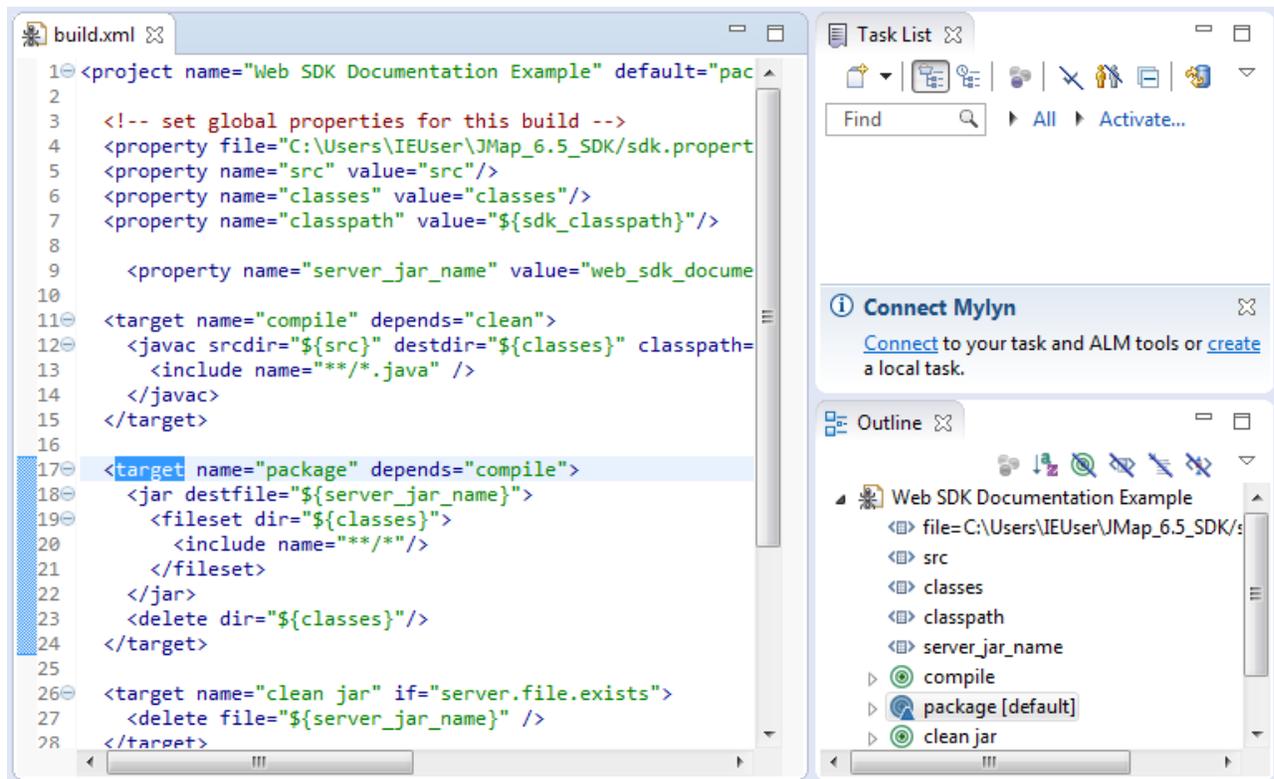
```
ant -f build.xml
```

Lorsque vous aurez votre fichier JAR, copiez le au répertoire *actions* de votre extension.

En utilisant Ant depuis Eclipse

Comme c'était le cas lors de l'utilisation de l'outil *webextensionbuilder*, vous pouvez utiliser Ant depuis Eclipse afin de produire le fichier jar serveur de votre extension. Ceci nécessite qu'un JDK soit configuré comme étant le JRE par défaut d'Eclipse et que le fichier *tools.jar* du JDK soit ajouté au classpath de la configuration du runtime de Ant. Ces étapes furent précédemment décrites ici.

1. Ouvrez le fichier *actions/build.xml* de votre extension dans Eclipse.
2. Exécutez la cible ant nommée «package».



Exécutez la cible «package».

Lorsque vous aurez votre fichier JAR, copiez le au répertoire actions de votre extension.

Identifier les actions de votre extension

Le fichier *extension.json* de votre extension informe JMap Serveur à propos de lui-même. Vous devez maintenant y indiquer que votre extension comporte des actions. Ouvrez ce fichier dans un éditeur de texte.

La propriété actions est un tableau qui peut contenir plusieurs littéraux objets. Chacun de ces objets décrivent une action.

Les littéraux objets du tableau actions doivent avoir les clefs de propriétés suivantes:

Clefs de propriétés des actions	Description
name	{String} Ceci est le nom de l'action tel qu'elle sera enregistrée auprès du AJAX Dispatcher. Cette valeur doit être unique au sein de votre déploiement JMap Web. Les requêtes HTTP doivent spécifier cette valeur au niveau du paramètre action de celles-ci. Ne fait pas nécessairement référence au nom du fichier source de votre classe.
classname	{String} Le nom complet de votre classe. Doit inclure les packages.
version	{String} Identifie une version. Principalement utile pour fins de débogage.

Cet extrait démontre comment représenter l'action MyFirstAction dans l'extension web_sdk_documentation_example qui fût crée plus tôt:

```
{
  "actions": [
    {
      "name": "hello",
      "className": "myextension.MyFirstAction",
      "version": "1.0.0"
    }
  ],
  "fullname": "Web SDK Documentation Example",
  "namespace": "Example",
  "shortname": "web_sdk_documentation_example",
  "version": "1.0"
}
```

Votre extension peut inclure autant d'actions que vous le voulez. Il suffit simplement de les identifier auprès du fichier *extension.json* de votre extension.

Après avoir modifié votre fichier *extension.json*, confirmez qu'il contient du JSON valide. JSONLint est un validateur JSON en ligne qui peut être utilisé à cette fin.

Appeler votre action depuis votre extension JMap Web

Tel que mentionné précédemment, la méthode *execute* de votre action sera appelée lorsque l'AJAX Dispatcher recevra une requête HTTP qui indique le nom de votre action au niveau de son paramètre action.

L'exemple suivant illustre comment vous pouvez envoyer une requête HTTP auprès de votre action en utilisant `jQuery.ajax()`. Pour tester ceci, vous pouvez soit inclure cet extrait dans la méthode `init` de votre extension ou bien le copier et le coller dans la console JavaScript de votre navigateur qui inspecte présentement votre déploiement.

```
$.ajax(JMap.app.ajaxDispatcher, {
  data: {
    "action": "hello", // The action name as defined in the extension.json file.
    "name": "Developer" // The `name` parameter as expected by the action.
  }
}).error(function(jqXHR, textStatus, errorThrown) {
  alert(textStatus);
}).success(function(data, textStatus, jqXHR) {
  alert(data);
});
```

Le modèle d'application JMap Web offre plusieurs actions par défaut afin de faciliter les interactions avec JMap Serveur. Cette section va en présenter quelques-unes et va démontrer comment vous pouvez les utiliser.

Note: La majorité des actions nécessitent des paramètres décrivant l'état de la carte. Pour la plus grande partie, ces informations peuvent être obtenues en utilisant l'API d'OpenLayers.

Tous les exemples de cette section font référence à un déploiement basé sur le projet *The World*. Le déploiement couvre la totalité de l'étendue du projet et contient les couches "Base" et "Cities". Référez-vous aux captures d'écran suivantes afin de créer un déploiement similaire.



Layers			
	Name	Type	Visible
Overlays	↕ Cities	Overlay	<input checked="" type="checkbox"/>  
Base layers	↕ Base	Base layer	<input checked="" type="checkbox"/>  

Couches configurées

Extents

+

-



x: 50.21, y: -100.18 Degrees

Maximum extent **Initial extent**

Maximum extent x	<input type="text" value="-180"/>	Degrees
Maximum extent y	<input type="text" value="-90.0111"/>	Degrees
Maximum extent width	<input type="text" value="360.0111"/>	Degrees
Maximum extent height	<input type="text" value="180"/>	Degrees

Propriétés géographiques, partie 1.

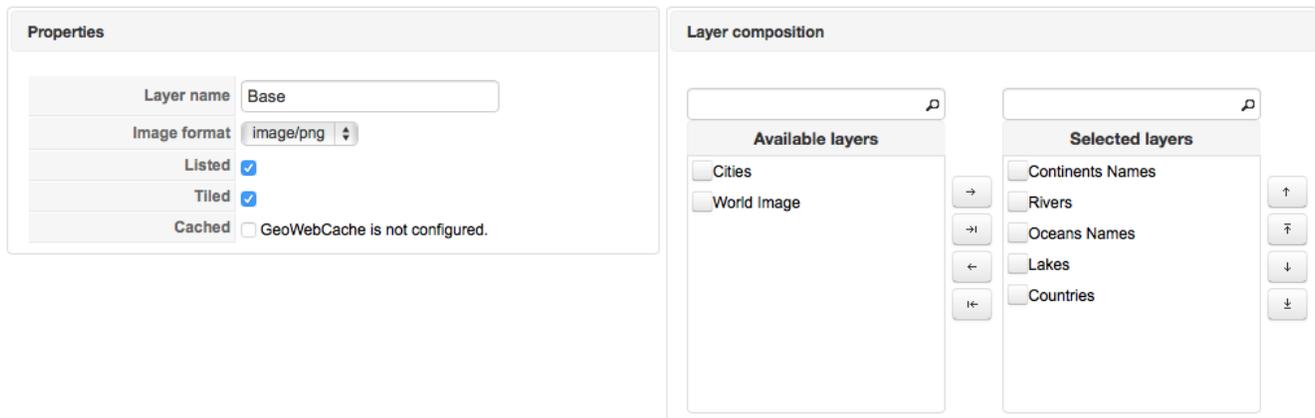
Maximum scale

Maximum scale	1 :	<input type="text" value="3000000.0"/>	Ex: 2000
Allow additional levels	<input type="checkbox"/>		

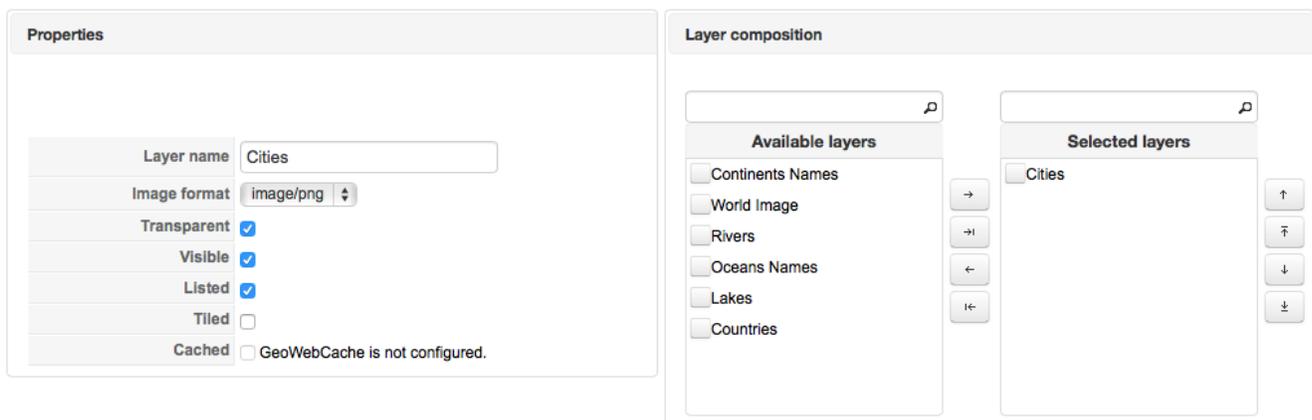
Levels

Level	Scale
1	1:558510621
2	1:279255310
3	1:139627655
4	1:69813827
5	1:34906913
6	1:17453456
7	1:8726728

Propriétés géographiques, partie 2.



Configuration de la couche Base



Configuration de la couche Cities

Action 'extractelemnts'

L'action *extractelemnts* retourne une liste d'éléments d'une couche JMap.

Le type de requête 'geometry'

La requête *geometry* fait une extraction des éléments qui intersectent **ou** sont contenus dans une géométrie fournie.

Paramètres

Le tableau suivant indique la liste des paramètres qui peuvent être acheminés à l'action *extractelemnts*.

Paramètres de extractelemnts	Description

request	{String} "geometry" requis Cette chaîne identifie le type de requête qui est acheminée à l'action. D'autres types de requêtes seront possiblement supportés par cette action dans le futur.
geometry	{String} required Une géométrie WKT qui définit la région pour laquelle le processus d'extraction d'éléments sera exécuté.
res	{Number} requis La résolution actuelle de la carte.
bbox	{String} requis L'étendue pour la quelle vous voulez faire l'extraction d'éléments dans un format bounding box (exemple: "-100,-50,100,50").
drill	{Boolean} défaut: false Indique si vous voulez continuer l'analyse de la pile de couches après qu'un premier élément correspondant soit trouvé.
layers	{String} requis Spécifie pour quelle(s) couche(s) vous voulez faire l'extraction d'éléments. Plusieurs couches peuvent être demandées à la fois. Dans ce cas, vous devez séparer les éléments de la liste en utilisant des virgules. Utilisez le paramètre drill de manière à obtenir les éléments pour plusieurs couches.

Exemple

Cet exemple fait une requête d'extraction d'éléments pour la couche «Cities» autour de l'écosse.

```
$.ajax(JMap.app.ajaxDispatcher, {data: {
  'action': 'extractelements',
  'request': 'geometry',
  'geometry': 'POLYGON((-6.378882 54.903806,-1.336158 54.903806,-1.336158 59.540037,-6.378882 59.540037,-6.378882 54.903806))',
  'res': JMap.app.map.getResolution(),
  'bbox': JMap.app.map.getExtent().toBBOX(),
  'layers': 'Cities'
}}).success(function(data, textStatus, jqXHR) {
  console.log(data);
});
```

Réponse

L'identifiant, la géométrie (sous la forme d'une chaîne WKT), les valeurs d'attributs, l'étendue et le point centré pour chacun des éléments sont retournés. Les attributs liés ainsi que leur type SQL sont également inclus. L'index des valeurs d'attributs de l'élément correspond à l'index des

attributs configurés sur la couche dans JMap Admin. Si des éléments de plusieurs couches correspondent aux critères de la requête, un tableau de ces objets sera retourné.

```
{
  "layerId": 7,
  "elementAttributes": [
    {
      "sqlType": 12,
      "name": "CITY"
    },
    {
      "sqlType": 12,
      "name": "COUNTRY"
    },
    {
      "sqlType": 12,
      "name": "CAP"
    },
    {
      "sqlType": 4,
      "name": "POP2000"
    }
  ],
  "elements": [
    {
      "centeredPoint": {
        "x": -1.3899999735879476,
        "y": 54.910001831054686
      },
      "bounds": {
        "x": -1.3899999735879476,
        "width": 0,
        "y": 54.910001831054686,
        "height": 0
      },
      "attributeValues": [
        "Sunderland",
        "UK - England and Wales",
        "0",
        181100
      ],
      "geometry": "POINT(-1.3899999735879476 54.910001831054686)",
      "id": 622
    },
    {
      "centeredPoint": {
        "x": -4.2699998496102864,
        "y": 55.87000091552734
      },
      "bounds": {
        "x": -4.2699998496102864,
        "width": 0,
        "y": 55.87000091552734,
        "height": 0
      },
      "attributeValues": [
        "Glasgow",
        "Scotland",
        "0",
        610700
      ],
      "geometry": "POINT(-4.2699998496102864 55.87000091552734)",
    }
  ]
}
```

```
    "id": 624
  },
  {
    "centeredPoint": {
      "x": -3.2200001357125814,
      "y": 55.949998931884764
    },
    "bounds": {
      "x": -3.2200001357125814,
      "width": 0,
      "y": 55.949998931884764,
      "height": 0
    },
    "attributeValues": [
      "Edinburgh",
      "UK - England and Wales",
      "0",
      382600
    ],
    "geometry": "POINT(-3.2200001357125814 55.949998931884764)",
    "id": 625
  },
  {
    "centeredPoint": {
      "x": -2.9999998686837728,
      "y": 56.469999389648436
    },
    "bounds": {
      "x": -2.9999998686837728,
      "width": 0,
      "y": 56.469999389648436,
      "height": 0
    },
    "attributeValues": [
      "Dundee",
      "Scotland",
      "0",
      148900
    ],
    "geometry": "POINT(-2.9999998686837728 56.469999389648436)",
    "id": 627
  },
  {
    "centeredPoint": {
      "x": -2.1000000117349202,
      "y": 57.14999969482422
    },
    "bounds": {
      "x": -2.1000000117349202,
      "width": 0,
      "y": 57.14999969482422,
      "height": 0
    },
    "attributeValues": [
      "Aberdeen",
      "Scotland",
      "0",
      188500
    ],
  },
```

```
    "geometry": "POINT(-2.1000000117349202 57.14999969482422)",
    "id": 628
  },
  {
    "centeredPoint": {
      "x": -1.6000000117349202,
      "y": 54.99999816894531
    },
    "bounds": {
      "x": -1.6000000117349202,
      "width": 0,
      "y": 54.99999816894531,
      "height": 0
    },
    "attributeValues": [
      "Newcastle upon Tyne",
      "UK - England and Wales",
      "0",
      980000
    ],
    "geometry": "POINT(-1.6000000117349202 54.99999816894531)",
    "id": 1899
  }
]
}
```

Action 'layerinfo'

L'action *layerinfo* offre des détails au sujet des couche du projet associé au déploiement courant.

Le type de requête 'geteditablelayers'

Le type de requête *geteditablelayers* offre des détails au sujet des couches sur lesquelles l'utilisateur courant peut faire des tâches d'édition.

Paramètres

Cette action ne nécessite aucun paramètre. À la place, elle utilise l'information de la session courante afin d'effectuer cette opération.

Exemple

```
$.ajax(JMap.app.ajaxDispatcher, {data: {
  'action': 'layerinfo',
  'request': 'geteditablelayers'
}}).success(function(data, textStatus, jqXHR) {
  console.log(data);
});
```

Réponse

La réponse du serveur comprend un tableau d'objets (*editableLayers*). Chacun des objets du tableau contient les clefs de propriétés suivantes:

Clefs de propriétés d'une réponse geteditables	Description
fields	{Array} Tableau d'objets qui exposent le nom des attributs liés ainsi que leur type de donnée SQL.
forms	{Array} Les formulaires configurés de la couche.
id	{Number} L'identifiant de la couche.
idFieldName	{String} Le nom de l'attribut de la couche qui sert d'identifiant.
offset	{Object} Un objet qui contient le décalage actuel de la couche.
permissions	<p>{Number} Un entier entre 0 et 15 (inclusif) qui décrit les permissions envers la couche de l'utilisateur courant. Les masques de bits suivant signifient différentes permissions.</p> <ul style="list-style-type: none"> • 0x0000: Aucune. • 0x0001: Peut ajouter des éléments à la couche. • 0x0010: Peut modifier les éléments de la couche. • 0x0100: Peut détruire les éléments de la couche. • 0x1000: Peut modifier les valeurs d'attributs des éléments de la couche.

Les formulaires sont retournés sous la forme d'un tableau d'objets. Ils ont la structure suivante:

Clefs de propriétés des formulaires de la réponse	Description
id	{Number} L'identifiant unique du formulaire.
json	{String} Une représentation JSON d'une configuration de formulaire.
name	{String} Le nom du formulaire tel que défini dans JMap Admin.
permissions	{Number} Un entier entre 0 et 7 (inclusif) qui décrit les permissions envers le formulaire de l'utilisateur courant. Les permissions d'un formulaire ne concerne

	<p>que les formulaires externes. Les permissions d'un formulaire d'attributs de couche sont définis dans les permissions de la couche.</p> <ul style="list-style-type: none">• 0x000: Aucune.• 0x001: Peut ajouter des données dans un formulaire externe.• 0x010: Peut modifier des données dans un formulaire externe.• 0x100: Peut supprimer des données dans un formulaire externe.
type	<p>{String} Identifie le type de formulaire. Un parmi les suivants:</p> <ul style="list-style-type: none">• LAYER_ATTRIBUTES_FORM• LAYER_ATTRIBUTES_SUB_FORM• EXTERNAL_ATTRIBUTES_FORM• EXTERNAL_ATTRIBUTES_SUB_FORM
uidAttributeName	<p>{String} Ne concerne que les formulaires externes. Correspond au nom de l'attribut de la couche qui sera utilisé comme clef étrangère afin d'établir le lien parent-enfant.</p>

Exemple

Ceci est un exemple d'une réponse *geteditablelayers*. Il est possible que vous ayez des résultats différents. Dans tous les cas, la réponse devrait refléter vos permissions envers les couches et les formulaires.

```
{
  "editableLayers": [
    {
      "offset": {
        "x": 9.5367431640625e-7,
        "y": -3.186320304870577
      },
      "permissions": 0,
      "id": 4,
      "fields": [
        {
          "name": "COUNTRY",
          "serverDataType": 12
        },
        {
          "name": "CONTINENT",
          "serverDataType": 12
        },
        {
          "name": "POP_1994",
          "serverDataType": 8
        },
        {
          "name": "POP_GRW_RT",
          "serverDataType": 8
        },
        {
          "name": "POP_0_14",
          "serverDataType": 8
        },
        {
          "name": "POP_15_64",
          "serverDataType": 8
        },
        {
          "name": "POP_65PLUS",
          "serverDataType": 8
        },
        {
          "name": "POP_AREA",
          "serverDataType": 8
        }
      ],
      "idFieldName": "JMAP_ID",
      "forms": []
    },
    {
      "offset": {
        "x": 0.56195068359375,
        "y": 13.687942504882812
      },
      "permissions": 0,
      "id": 6,
      "fields": [
        {
          "name": "LAKE_NAME",
          "serverDataType": 12
        }
      ]
    }
  ]
}
```

```
        "name": "VOLUME_CKM",
        "serverDataType": 8
    },
    {
        "name": "AREA_SKM",
        "serverDataType": 8
    }
],
"idFieldName": "JMAP_ID",
"forms": []
},
{
    "offset": {
        "x": 20.934576233500025,
        "y": 10.050437668683657
    },
    "permissions": 0,
    "id": 5,
    "fields": [],
    "idFieldName": "JMAP_ID",
    "forms": []
},
{
    "offset": {
        "x": 16.215000694478334,
        "y": 16.463705277985326
    },
    "permissions": 0,
    "id": 3,
    "fields": [
        {
            "name": "HYD_NAME",
            "serverDataType": 12
        },
        {
            "name": "LENGTH_KM",
            "serverDataType": 4
        }
    ],
    "idFieldName": "JMAP_ID",
    "forms": []
},
{
    "offset": {
        "x": 31.452202349999993,
        "y": -14.421794805000005
    },
    "permissions": 0,
    "id": 2,
    "fields": [
        {
            "name": "CONTINENTNAME",
            "serverDataType": 12
        }
    ],
    "idFieldName": "JMAP_ID",
    "forms": []
},
{
```

```

    "offset": {
      "x": 1.6169597031546061,
      "y": 8.079999999999998
    },
    "permissions": 15,
    "id": 7,
    "fields": [
      {
        "name": "CITY",
        "serverDataType": 12
      },
      {
        "name": "COUNTRY",
        "serverDataType": 12
      },
      {
        "name": "CAP",
        "serverDataType": 12
      },
      {
        "name": "POP2000",
        "serverDataType": 4
      }
    ],
    "idFieldName": "JMAP_ID",
    "forms": [
      {
        "permissions": 0,
        "name": "Form",
        "json": "{\"formSections\": [{\"name\": \"Section 1\", \"nbRows\": 3, \"field",
        "id": 1,
        "type": "LAYER_ATTRIBUTES_FORM",
        "uidAttributeName": null
      }
    ]
  }
}

```

Action 'loadformdata'

L'action *loadformdata* offre les données des formulaires externes pour un élément d'une couche demandée.

Paramètres

loadformdata attend les paramètres suivants:

Paramètres de loadformdata	Description
data	{String} requis

Chaîne JSON qui représente un littéral objet JavaScript composé des propriétés suivantes:

- `elementId`: {Number} L'identifiant unique de l'élément pour lequel on fait la demande de données.
- `formId`: {Number} L'identifiant du formulaire externe.
- `layerId`: {Number} L'identifiant de la couche sur laquelle l'élément existe/le formulaire est configuré.
- `listFields`: {Array[Strings]} Une liste de noms de champs du formulaire pour lesquels vous voulez des valeurs. Un tableau vide va retourner les données pour tous les champs du formulaire.
- `mapValues`: Littéral objet JavaScript qui contient les valeurs des champs du formulaire d'attributs de la couche.

Exemple

Exemple de requête `loadformdata` . Cet exemple fait référence à un autre projet JMap sur lequel des formulaires externes sont configurés.

```
var data = {
  elementId: 6,
  formId: 2,
  layerId: 1,
  listFields: [],
  "mapValues": {
    "MOBILE_JMAP_ID": 6,
    "MOBILE_JMAP_GEOMETRY": "POINT(-8189010.0 5701129.5)",
    "AUTHOR": "jrhaddad",
    "CREATION_TIME": 1415767972000,
    "MODIFICATION_TIME": 1418400517000,
    "ABR_CODE": "342",
    "ABR_NAME": "ES34F",
    "ABR_LOC_TYPE": "Arrêt bus",
    "ABR_WHEEL_CHAIR": "Non accessible",
    "ABR_DATE_INSP": null,
    "ABR_STATUS": null
  }
};

$.ajax(JMap.app.ajaxDispatcher, {data: {
  'action': 'loadformdata',
  'data': JSON.stringify(data)
}}).success(function(data, textStatus, jqXHR) {
  console.log(data);
});
```

Réponse

Le serveur répond à l'aide d'un tableau à deux dimensions de littéraux objets JavaScript représentant les valeurs des champs du formulaire. Chaque item dans le tableau `rows` correspond

à une rangée de données. Plusieurs rangées seront retournées lorsque l'on demande des données pour un formulaire de type `EXTERNAL_ATTRIBUTES_SUB_FORM`.

Le nom du champs du formulaire, sa valeur et le type SQL de la donnée seront retournés.

```
{
  "rows": [
    [
      {
        "name": "insp_abribus.id_inspection",
        "value": 5,
        "type": 4
      },
      {
        "name": "insp_abribus.id_abribus",
        "value": 6,
        "type": 4
      },
      {
        "name": "insp_abribus.date_inspection",
        "value": 1415854800000,
        "type": 93
      },
      {
        "name": "insp_abribus.etat",
        "value": "Bon état",
        "type": 12
      },
      {
        "name": "insp_abribus.observations",
        "value": "Tout est ok",
        "type": 12
      }
    ]
  ]
}
```

Afin de déployer une extension avec votre déploiement JMap Web, vous devez d'abord installer l'extension sur votre serveur JMap. Vous installez une extension JMap Web en copiant son répertoire parent (nommé selon le shortname de l'extension) vers le répertoire `$JMAP_HOME$/extensions/web`.

Durant le processus de déploiement d'une application JMap Web, il est maintenant possible de choisir les extensions Web que vous souhaitez déployer. Sélectionnez votre nouvelle extension «Web SDK Documentation Example» et complétez l'assistant.

Application deployment wizard

Cancel

Previous

Create

Identification > Template > Path > Options > Extensions

 Web SDK Documentation Example (version 1.0)*Déployer l'extension d'exemple*

Le design de JMap Web permet d'intégrer des déploiements JMap Web dans vos propres applications. Cette section décrit les étapes nécessaires afin d'activer l'intégration.

Copier les bibliothèques JMap Web et dépendances vers votre serveur web

Vos applications JMap Web déployées, tel qu'elles sont servies par l'instance Tomcat de JMap, contiennent les bibliothèques et services web nécessaires pour l'intégration dans votre application. Les fichiers de votre déploiement se situent dans le répertoire `$JMAP_HOME$/applications/deployed`.

Parmi ces fichiers se trouve le répertoire `jmap`. Il contient toutes les ressources nécessaires pour intégrer votre déploiement JMap. **Ce répertoire ainsi que son contenu doit être copié vers votre serveur web et doit être accessible depuis les pages web de votre application.**

Les dépendances de JMap Web seront chargés durant le processus d'initialisation avant l'affichage de la carte. Pour plus de détails au sujet du processus d'initialisation, référez-vous à la section Le processus de démarrage de JMap Web de ce document.

- JQuery est requis durant le processus d'initialisation et doit être présent dans le document de votre application.
- L'ajout de feuilles de styles des dépendances de JMap Web va modifier les styles de votre document. Présentement, il n'y a malheureusement pas de façon d'éviter ceci.

Autoriser le partage de ressources d'origines croisées

Afin de permettre l'intégration de vos applications, quelques modifications au fichier `web.xml` de votre déploiement seront requises.

- La modification du fichier `web.xml` de l'application nécessitera un téléchargement/chargement de votre déploiement. Ceci peut être fait depuis la section déploiement de JMap Admin.
- Toutes modifications faites aux fichiers du déploiement seront perdues si vous en faites la mise-à-jour.

1. Activer l'authentification HTTP pour le filtre `JMapLoginFilter_index`.

```
<filter>
  <filter-name>JMapLoginFilter_index</filter-name>
  <!-- ... -->
  <init-param>
    <param-name>httpauthentication</param-name>
    <param-value>>true</param-value>
  </init-param>
</filter>
```

2. Instancier et configurer le filtre *CORS* . Ajoutez la déclaration suivante avant le premier «mapping» du *ByPassFilter* .

```
<filter>
  <filter-name>CorsFilter</filter-name>
  <filter-class>org.apache.catalina.filters.CorsFilter</filter-class>
  <init-param>
    <param-name>cors.allowed.headers</param-name>
    <param-value>Content-Type,X-Requested-With,accept,Origin,Access-Control-Request-Method</param-value>
  </init-param>
  <init-param>
    <param-name>cors.exposed.headers</param-name>
    <param-value>Access-Control-Allow-Origin,Access-Control-Allow-Credentials</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>CorsFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Intégrer une carte

Une fois que le répertoire *jmap* sera copié vers votre serveur web, il faudra ajouter les références aux scripts suivants dans la balise `<head>` du document dans lequel vous souhaitez intégrer votre déploiement.

```
<script type="text/JavaScript" src="jmap/vendor/jquery-1.9.1.min.js"></script>
<script type="text/JavaScript" src="jmap/jmap.min.js"></script>
```

- Il est possible que vous deviez modifier les chemins relatifs mentionnés plus haut.
- L'inclusion de *jQuery* n'est pas nécessaire si vous l'utilisez déjà. *JMap Web* nécessite une version minimale de 1.9.1 afin d'assurer la plus part des fonctionnalités.

L'initialisation de la carte requiert une balise `div` vide dans laquelle la carte sera créée.

```
<div id="map" style="height: 600px; width: 600px;"></div>
```

Tip: Si vous ne voyez pas la carte, assurez-vous que votre `div` possède une taille valide.

Ajoutez le JavaScript suivant pour déclencher l'initialisation de la carte.

```
<script type="text/JavaScript">
  $(document).ready(function() {
    var options = {
      // The URL of your deployed application. This must be an address that your users may
      jmapUrl: 'http://192.168.0.80:8080/web_deployment'
    };

    JMap.initialize(document.getElementById('map'), options);
  });
</script>
```

La fonction *JMap.initialize* qui est appelée plus haut est la même que celle qui fût détaillée ici. Vous pouvez modifier l'application intégrée en précisant ces mêmes propriétés de configuration dans un littéral objet *mapConfig* qui sera fournis à l'argument options.

embed_example.html

Un fichier *embed_example.html* est inclus dans le répertoire de votre application déployée. Ce fichier procure un exemple de d'application JMap Web intégrée. Vous pouvez ouvrir ce fichier dans un éditeur de texte afin de voir comment l'intégration est faite.

Vous pouvez également copier ce fichier **ainsi que le répertoire *jmap*** vers un serveur web séparé afin de pouvoir confirmer que le partage des ressources d'origines croisées est bien activé.

Gérer l'authentification

Si vous avez activé l'accès contrôlé sur votre déploiement JMap Web, une authentification auprès de JMap sera nécessaire avant de pouvoir initialiser la carte.

Le code suivant est fourni à titre d'exemple, mais n'est pas une façon recommandée de s'authentifier auprès de JMap Serveur.

```
<script type="text/JavaScript">
  $(document).ready(function() {
    $.ajaxSetup({
      xhrFields: {
        withCredentials: true
      }
    });

    $.ajax('http://192.168.0.80:8080/web_deployment/ajaxdispatch', {
      data: {
        action: 'ping'
      },
      beforeSend: function(xhr) {
        // Replace 'USERNAME:PASSWORD' with a valid string value.
        xhr.setRequestHeader('Authorization', 'Basic ' + btoa('USERNAME:PASSWORD'));
      },
    }).done(function(data, textStatus, jqXHR) {
      var options = {
        jmapUrl: 'http://192.168.0.80:8080/web_deployment'
        mapConfig: {},
        onMapInit: function() {
          console.log('Map was initialized.');
        }
      };

      JMap.initialize(document.getElementById('map'), options);
    });
  });
</script>
```

Développement JMap Web (7.0 Preview)

ATTENTION: Cette rubrique discute de la version «Preview» courante de JMap Web 7. Ce document est sujet à changer d'ici à la sortie officielle de JMap 7.

Cette section explique comment développer pour JMap Web et comment intégrer JMap Web au sein de d'autres applications web.

JMap Web est une application web qui fait utilisation de technologies tels que HTML5, JavaScript, CSS et JSON. L'application JMap Web est construite au dessus de bibliothèques JavaScript de tiers incluant OpenLayers, jQuery et plusieurs autres.

Brève vue d'ensemble du design de JMap Web

JMap Web fournit une application Web capable d'interagir avec JMap Serveur. L'ergonomie de son interface utilisateur est principalement adaptée pour les navigateurs web d'ordinateurs portables et d'ordinateurs de bureau.

Les éléments principaux d'une application JMap Web sont les suivants:

- Le client: Peut être décrit comme une application web à une seule page (index.jsp). Le client consiste en la page telle qu'elle sera rendue dans le navigateur des utilisateurs.
- Un Web Map Service: Donne accès aux tuiles de la carte.
- Le «Dispatcher AJAX»: Un service web auquel des requêtes HTTP sont envoyées. Le «Dispatcher AJAX» achemine ensuite les requêtes au «Action Handler» approprié. Ceci sera présenté davantage dans la section Envoyer des requêtes au serveur et actions personnalisées de ce document.

Bibliothèques JavaScript de tiers utilisées

JMap Web fait utilisation de plusieurs bibliothèques JavaScript de tiers. Vos extensions peuvent tirer parti de celles-ci sans aucune configuration additionnelle. Le tableau suivant fournit une brève explication de l'utilisation de ces bibliothèques par JMap Web.

Bibliothèque	Description
DataTables (v1.9.4)	Plugin jQuery qui facilite la création de tableaux HTML puissants pour la représentation de données. Ceci est principalement utilisé pour afficher les résultats de recherches.
fancybox (v2.15)	Affiche du contenu dans une interface de style «lightbox». JMap Web utilise fancyBox pour afficher des documents de taille-réelle

	inclus dans le contenu d'une infobulle.
Google Maps (v3)	Nécessaire pour afficher les couches de base «Roadmap», «Terrain», «Satellite» et «Hybride» de Google Maps.
jQuery (v2.1.4)	jQuery est une bibliothèque JavaScript rapide, petite, et riche en fonctionnalités. Elle simplifie la navigation du document HTML, la manipulation d'éléments, la gestion des événements, l'animation, et les requêtes AJAX. Son API est facile à utiliser et fonctionne sur une multitude de navigateurs.
jQuery-ui (v1.11.4)	Bibliothèque d'éléments d'interface utilisateur nécessitant jQuery. Présentement, JMap Web utilise une version personnalisée qui n'inclut que la composante «autocomplete».
jQuery UI Touch Punch (v0.2.3)	Bibliothèque qui adresse certains soucis de compatibilité de jQuery-ui sur interfaces tactiles (mobile, tablette).
moment.js (v2.10.6)	Bibliothèque utilisée pour lire, valider, manipuler et afficher des dates en JavaScript.
moment-jdateformatparser (v1.0.0)	Plugin de la bibliothèque moment.js permettant d'interpréter les formats de date Java (<code>java.text.SimpleDateFormat</code>).
malihu-custom-scrollbar-plugin (v3.0.7)	Ajuste l'apparence des barres de défilement dans les infobulles. Sa fonction principale est de standardiser l'apparence des barres de défilement dans les différents navigateurs web.
OpenLayers (v3.15.1)	Ceci est la bibliothèque la plus importante de JMap Web. OpenLayers permet l'affichage de la carte dans JMap Web et gère également la majorité des interactions avec celle-ci.
Ol3-Google-Maps (v0.4)	Cette bibliothèque de Mapgears permet l'utilisation des couches Google Maps avec OpenLayers 3.
Proj4js (v2.3.12)	<p>Proj4js est une bibliothèque qui permet de traduire des coordonnées géographiques d'un système de projection cartographique à l'autre. Proj4js est utilisé par OpenLayers, mais n'est pas fourni avec celle-ci.</p> <p>Afin de pouvoir accomplir des transformations pour des projections nommées (ex: "EPSG:3857"), les projections doivent être définies dans Proj4js. Seulement un petit nombre de définitions de projections est disponible par défaut.</p> <p>JMap Web inclut un fichier de définition de projection Proj4js pour chacune des projections supportées par JMap Serveur. Les définitions de projections seront chargées en mémoire au fur et à mesure qu'elles seront requises.</p>

slideReveal (v1.0.1)	Bibliothèque facilitant le menu principal de JMap Web.
switchery (v0.8.1)	Bibliothèque qui permet de produire des boutons radio similaires à ceux d'iOS 7.
Twitter Bootstrap (3.3.6)	«Framework front-end» pour créer des interfaces utilisateur.
bootstrap-datetimepicker (v4.15.35)	Outil pour sélectionner une date/heure avec Twitter Bootstrap.
bootstrap-multiselect (v0.9.10)	Outil pour sélectionner des choix multiples avec Twitter Bootstrap.

Depuis la version 6.5, le modèle d'application JMap Web inclut un API public permettant au développeurs de facilement s'intégrer aux fonctionnalités de JMap.

Une documentation JSDoc générée pour cet API JavaScript est disponible en ligne sur <http://dev.k2geospatial.com/jmap/web/api/7.0/>

Avant d'aborder la création d'une extension JMap Web, cette section du document va expliquer le processus de démarrage de JMap Web.

Note : Cette section fait référence au modèle d'application JMap Web 7 «Preview 3» tel qu'il est téléchargeable depuis notre site internet. Ce processus, brièvement décrit ici, va être différent si vous choisissez d'ajouter un déploiement JMap Web au sein de votre propre application web existante. Pour plus de détails, consultez la section Intégrer JMap Web dans votre propre application de ce document.

Les fichiers du modèle d'application JMap Web sont disponibles sous le répertoire `$JMAP_HOME$/applications/templates/html/web`.

index.jsp

Ceci est le document web qui sera servi par JMap Serveur lorsque l'application sera demandée par l'utilisateur. Ouvrez ce fichier dans un éditeur de texte. Comme vous pouvez le remarquer, *jQuery*, *OpenLayers* et *jmap-web.min.js* sont chargés dans la balise *head* du document.

Plus bas, la carte est initialisée lorsque le document sera prêt à être manipulé.

```

$(document).ready(function() {
  var options = {
    jmapUrl: '$APPLICATION_PROTOCOL$://$APPLICATION_HOST$: $APPLICATION_WEBPORT$$PATH$',
    language: '$LANGUAGE_CODE$',
    mapConfig: {
      // Configuration properties
    },
    onMapInit: function() {
      JMap.logMsg('Map was initialized.');
```

// resizeUi function...

// Object auto zoom handling...

```

  }
};

// Region auto zoom handling...

$('#map').height($(window).height()).width($(window).width());
JMap.initialize(document.getElementById('map'), options);
});

```

JMap.initialize(map, options)

La fonction *JMap.initialize* est définie dans le fichier *jmap-web.min.js*. Elle charge toutes les dépendances nécessaires et va ensuite initialiser tous les objets requis par l'application. Parmi ces objets, *JMap.app* et *JMap.app.map* qui demeureront globalement disponibles.

L'objet *JMap.app* est une instance de la classe *Application* de la bibliothèque *core*. L'objet *JMap.app.map* est une instance de la classe *ol.Map*.

Paramètres	Description
map	Requis {String} L'élément ou l'identifiant d'un élément existant sur la page qui va contenir la carte.
options	<p>Requis {Object} Doit au minimum fournir une propriété <i>jmapUrl</i> {String}. Supporte les clefs de propriétés suivantes:</p> <ul style="list-style-type: none"> • <i>jmapUrl</i>: Requis. {String} L'adresse de votre déploiement JMap Web. • <i>language</i>: {String} Langue de l'application. Valeurs supportées: "en", "es", "fr" et "pt". • <i>mapConfig</i>: {Object} Options additionnelles assistant la configuration de la carte. • <i>onMapInit</i>: {Function} <i>Handler</i> qui sera appelé une fois que le processus d'initialisation de la carte est complété. Pratique afin d'ajouter de nouveaux contrôles <i>OpenLayers</i> une fois que la bibliothèque <i>OpenLayers</i> sera chargée.

L'objet mapConfig peut inclure les clefs de propriétés suivantes afin de personnaliser l'application. La valeur par défaut est identifiée en gras.

Propriétés du mapConfig	Description
Options de la carte	
initialExtent	null {Array} Coordonnées géographiques qui décrivent l'étendue: [minX, minY, maxX, maxY]
maximumExtent	null {Array} Coordonnées géographiques qui décrivent l'étendue: [minX, minY, maxX, maxY]
projection	null {Object} Si fourni, l'objet doit inclure une propriété String code qui correspond à un code EPSG.
Couches additionnelles	
addShowPositionLayer	true false
Services au démarrage	
activateGeolocationServiceOnLaunch	true false
loadGoogleMapsApiOnLaunch	true false - Sera automatiquement remplacé à true si le déploiement contient au moins une couche Google ou si au moins un parmi ceux-ci est vrai : addGoogleDirections, addGoogleGeocoding, addGoogleStreetView.
Options de la carte et de la navigation	
addGeolocateButton	true false
addInitialViewButton	true false
addMapOverview	true false
addMousePosition	true false
addScaleBar	true false
addZoomButtons	true false
isMapOverviewMaximized	true false
Outils JMap	

addInfoReportTool	true false
addMeasureTools	true false
addMeasureAreaTool	true false
addMeasureDistanceTool	true false
addMeasureCircularAreaTool	true false
addMouseOverTool	true false
addRedLiningTool	true false
Fonctions d'édition JMap	
addEditionTools	true false
addEditionCreateElementTools	true false - La propriété addEditionTools doit également être true pour que ceci prenne effet.
addEditionModifyElementTool	true false - La propriété addEditionTools doit également être true pour que ceci prenne effet.
addEditionShowElementFormButton	true false - La propriété addEditionTools doit également être true pour que ceci prenne effet.
Fonctions de sélection JMap	
addSelectionTools	true false
addCircleSelectionTool	true false - La propriété addSelectionTools doit également être true pour que ceci prenne effet.
addLineSelectionTool	true false - La propriété addSelectionTools doit également être true pour que ceci prenne effet.
addPointSelectionTool	true false - La propriété addSelectionTools doit également être true pour que ceci prenne effet.
addRectangleSelectionTool	true false - La propriété addSelectionTools doit également être true pour que ceci prenne effet.
addShapeSelectionTool	true false - La propriété addSelectionTools doit également être true pour que ceci prenne effet.
Autres fonctions	

addFullScreenButton	true false
addLayersMainMenuitem	true false
addLogo	true false
addLogoutAsideMenuLink	true false
addPrintButton	true false
addSearchMainMenuitem	true false
Fonctionnalités de tiers	
addGoogleDirections	true false
addGoogleGeocoding	true false
addGoogleStreetView	true false

Après le processus d'initialisation

Une fois que la carte est initialisée avec succès, vous pouvez l'accéder à l'aide de JavaScript en utilisant la variable globale `JMap.app.map`.

La manipulation de la carte sera possible en accédant à la variable globale `JMap.app.map`. Cette variable est l'instance d'*ol.Map* en utilisation. Vous pouvez également interagir avec elle en utilisant l'API d'OpenLayers.

Paramètres d'URL

JMap Web supporte les paramètres d'URL suivants:

```
autozoom {String}
```

Ce paramètre permet de spécifier une étendue sur laquelle la vue de la carte sera initialisée. Deux types d'*autoZoom* peuvent être utilisés, *region* ou *object*. Le type est déterminé par le premier argument '*type*'.

Types d' <i>AutoZoom</i>	Description
Region	L'utilisateur précise la zone visible en décrivant un rectangle... Syntaxe: "type;x;y;width;height" <ul style="list-style-type: none"> type: {String} Le type de la requête. Dans ce cas, doit être "region".

	<ul style="list-style-type: none"> • x: {Number} La valeur X de la coordonnée inférieure gauche du rectangle. • y: {Number} La valeur Y de la coordonnée inférieure gauche du rectangle. • width: {Number} La largeur du rectangle. • height: {Number} La hauteur du rectangle. <p>Exemple: ?autozoom=region;9;39;20;20</p>
Object	<p>L'utilisateur précise un objet sur lequel il veut faire le zoom...</p> <p>Syntaxe: "type;layerName;field;value;maxScale"</p> <ul style="list-style-type: none"> • type: {String} Le type de la requête. Dans ce cas, doit être "object". • layerName: {String} Le nom de la couche qui contient l'objet sur lequel on veut faire le zoom. • field: {String} Le champs qui contient la valeur de l'objet sur lequel on veut faire le zoom. • value: {Number String} La valeur spécifique de l'objet sur lequel on veut faire le zoom. Si le champs est une chaîne de caractères, utilisez ' '. Voir l'exemple plus bas. • maxScale: {Number, optionnel} L'échelle maximale que la carte doit respecter lorsqu'elle affiche les résultats. <p>Exemple d'une valeur numérique: ? autozoom=object;citiesLayer;city_id;1032</p> <p>Exemple d'une valeur chaîne de caractères: ? autozoom=object;citiesLayer;city_name;'montreal';15</p>

Cette section donne un aperçu de la façon de développer des extensions pour JMap Web.

Les extensions sont modulaires et offrent généralement des fonctionnalités adaptés pour des tâches spécifiques. Par défaut, JMap Web offre un ensemble de fonctions de base. Les extensions JMap

Web permettent la personnalisation et l'ajout de nouvelles fonctions aux déploiements basés sur le modèle d'application JMap Web.

Tout comme les extensions de JMap Pro, les administrateurs JMap peuvent choisir quelles extensions ils/elles veulent déployer lors de l'assistant de déploiement d'applications. Les extensions choisies seront chargées au moment du processus d'initialisation de JMap Web.

Une extension JMap Web consiste en un regroupement de fichiers JavaScript, feuilles de styles (CSS) et ressources (images, sons, etc.).

Vous pouvez utiliser l'outil **JMap Web extension builder** afin de rapidement créer les fichiers nécessaires à une extension Web. Ouvrez une fenêtre de ligne de commande et naviguez au répertoire `tools/webextensionbuilder` du SDK de JMap 6.5 installé sur votre ordinateur.

Utilisation du générateur d'extensions JMap Web

Vous trouverez le fichier `webextensionbuilder.xml`. Ouvrez-le dans un éditeur de texte. Modifiez-le de manière à spécifier une liste d'arguments qui vous permettront de créer votre extension. Les arguments suivants sont exigés:

Argument	Description
fullname	{String} Le nom complet lisible de l'extension. C'est ce que les administrateurs et les utilisateurs JMap verront. Un «shortname» sera dérivé depuis cette valeur. Le «shortname» sera utilisé entre autre comme le nom de plusieurs fichiers.
namespace	{String} Le nom du «namespace» JavaScript (une variable <i>Object</i> globale) de votre extension. Si vous planifiez déployer plusieurs de vos extensions simultanément, vous pouvez spécifier un «namespace» qui contient <u>au plus</u> un point «.» de manière à séparer les éléments de votre «namespace». Exemple: MonEntreprise>HelloWorld.
version	{String} Le numéro de version de votre extension. Aucun format spécifique est imposé.
dest	L'endroit sur le disque où votre extension sera créée.

Exemple d'un fichier `webextensionbuilder.xml` modifié qui sera utilisé pour créer votre extension:

```
<project name="JMap 6.5 SDK - Web Extension Builder" basedir="." default="run">
  <!-- set global properties for this build -->
  <property file="../../sdk.properties"/>

  <target name="run">
    <java fork="true" classname="jmap.sdk.tools.webextensionbuilder.WebExtensionBuilder"
      classpath="webextensionbuilder.jar;${sdk_classpath}">
      <!-- Use default parameter values. Uncomment following line to use other parameters
      <arg line="-fullname 'Web SDK Documentation Example' -namespace Example -version 1.0"
    </java>
  </target>
</project>
```

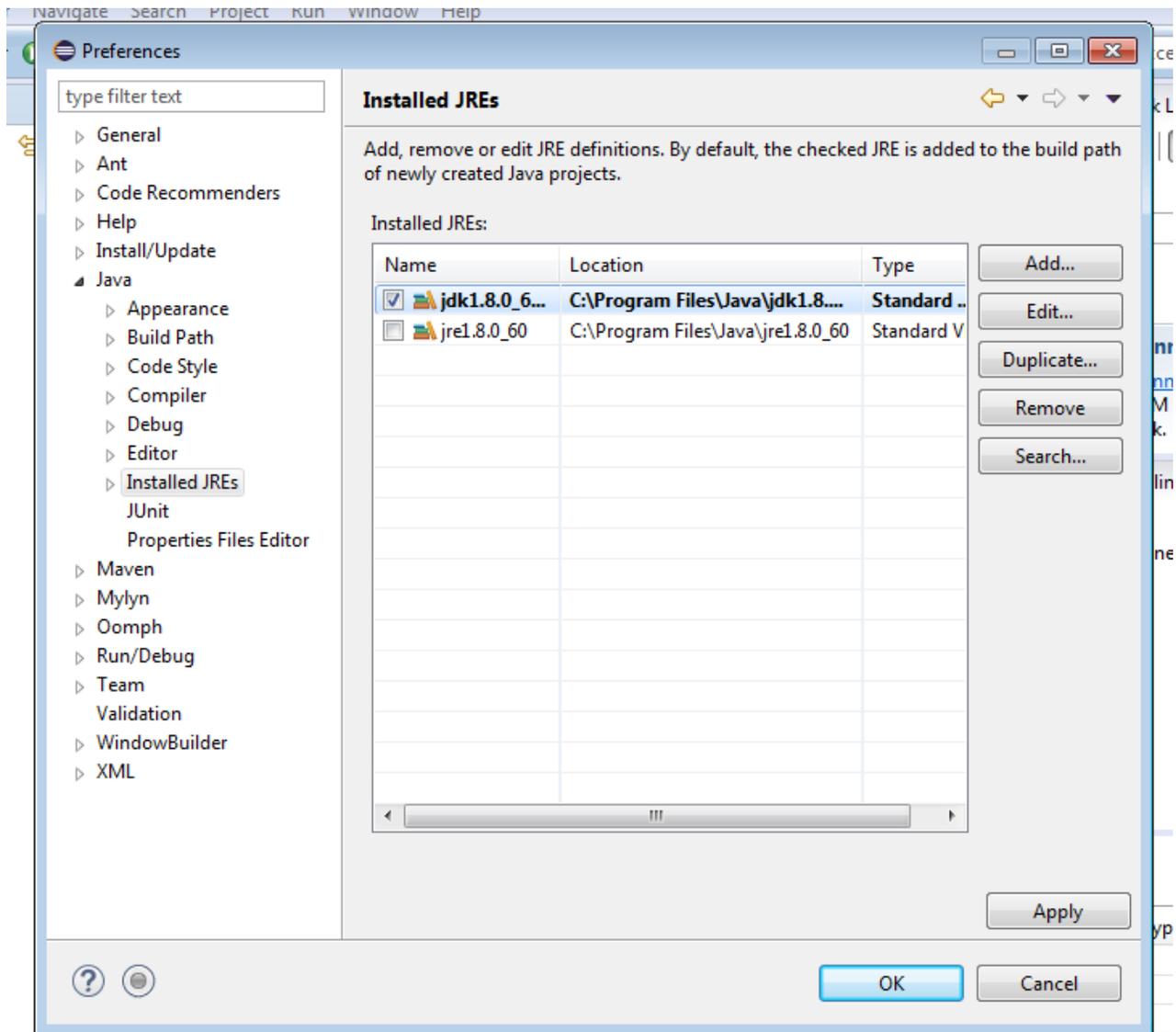
Création de l'extension à l'aide de ant avec la ligne de commande

Depuis le terminal, invoquez le script ant:

```
ant -f webextensionbuilder.xml
```

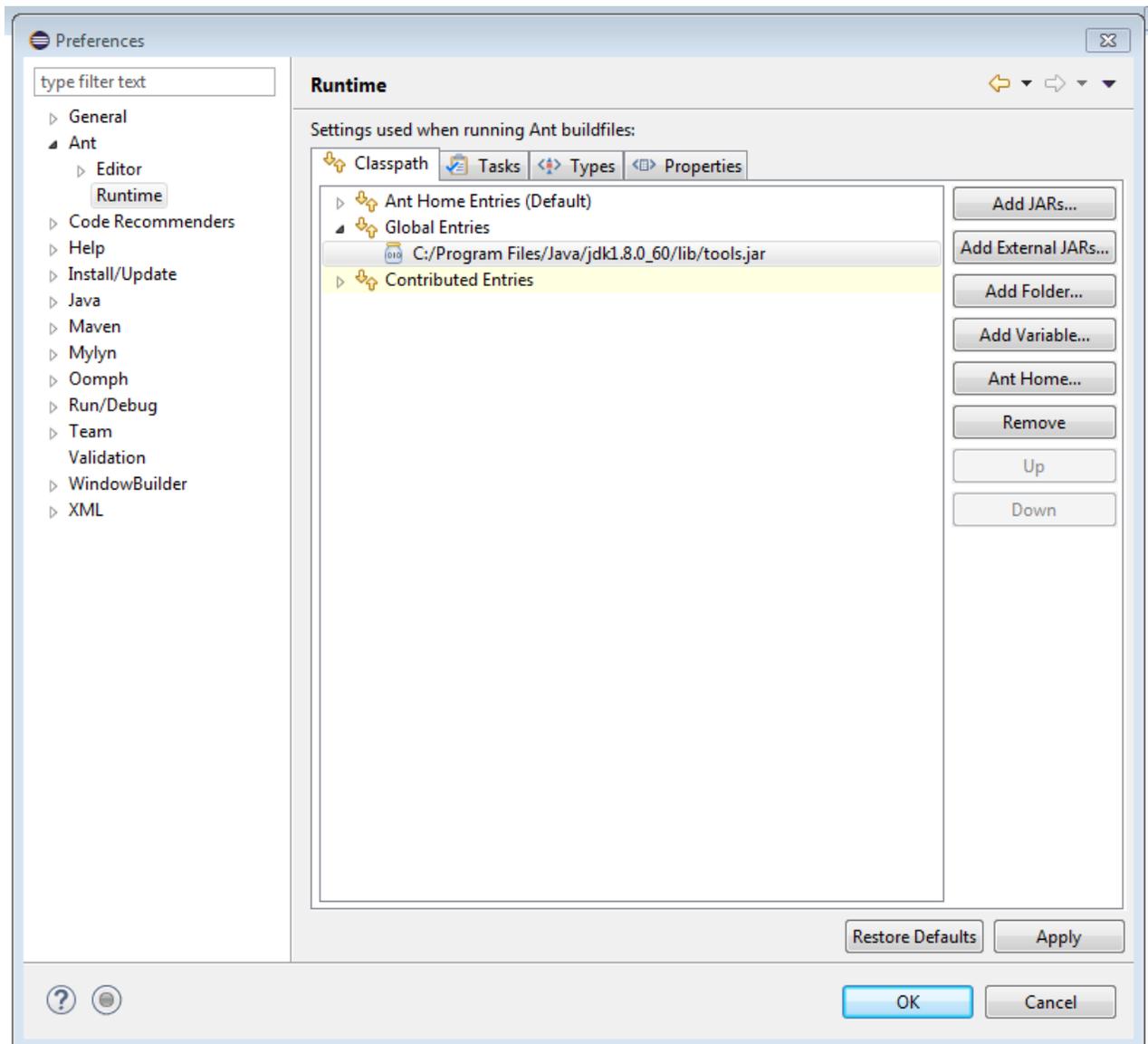
Création de l'extension à l'aide de ant sous Eclipse

Pour créer une extension JMap Web à l'aide d'Eclipse, confirmer d'abord qu'un Java Developer Kit (JDK) est installé sur votre poste et est configuré comme étant le JRE par défaut d'Eclipse.



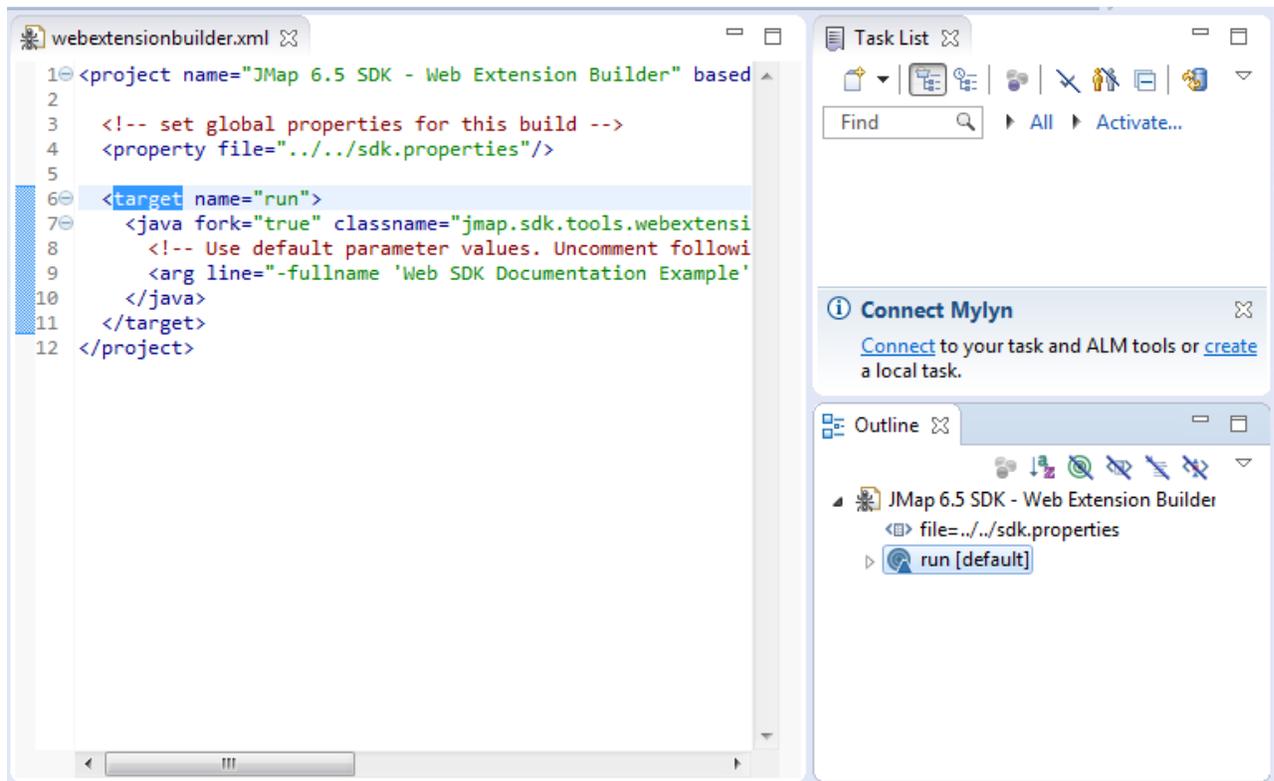
Les JREs d'Eclipse

Vous devriez également confirmer que le fichier `tools.jar` de votre JDK soit ajouté au *Ant Runtime Configuration* d'Eclipse:



Le Ant Runtime Configuration d'Eclipse. Ajoutez le `JDK_HOME/lib/tools.jar` à la section *Global Entries* de la *class path*.

Exécutez la tâche de build en ouvrant le fichier `webextensionbuilder.xml` dans Eclipse. Exécutez la cible «run».



Éxécutez la cible «run»

Comprendre les fichiers de l'extension

Après l'exécution du script ant, vous allez posséder le minimum de code d'une extension à l'emplacement spécifié.

```

output
+-- web_sdk_documentation_example
|   +-- actions
|   |   +-- build.xml
|   |   +-- readme.markdown
|   |   +-- src
|   |       +-- Example
|   |           +-- SampleAction.java
+-- assets
|   +-- css
|   |   +-- web_sdk_documentation_example.css
|   +-- js
|       +-- web_sdk_documentation_example.js
+-- extension.json

```

7 directories, 6 files

Par défaut, ainsi que pour des fins de démonstrations, une action du AJAX Dispatcher nommée `web_sdk_documentation_example_sample_action` est incluse. Pour plus de détails sur les actions du AJAX Dispatcher, référez-vous à la section Envoyer des requêtes au serveur et actions personnalisées.

Les fichiers CSS et JavaScript de votre extension seront chargés durant le processus d'initialisation de JMap Web.

Tout autre ressource CSS et JavaScript devra être chargée par programmation depuis le fichier JavaScript généré (dans ce cas-ci, le fichier `websdkdocumentation_example.js`).

Puisque les fichiers de votre extension seront chargés au moment de l'exécution, il est important de ne pas renommer ou déplacer les fichiers CSS et JavaScript générés. Le nom de ces fichiers devra toujours correspondre au «shortname» défini dans le fichier `extension.json`.

Le code source généré du fichier JavaScript comporte des commentaires qui décrivent les propriétés et fonctions **requises** qui forment votre extension JMap Web. Il est recommandé de faire la lecture de ce fichier afin que vous familiarisiez avec son contenu.

Parmi les fichiers générés se trouve `extension.json` . Ce fichier va servir de manifest pour JMap Serveur. Il doit contenir du JSON valide. Ce fichier ne doit pas être détruit, déplacé ou renommé.

Interaction avec OpenLayers

Tel que mentionné précédemment, une variable globale `JMap.app` est définie durant l'initialisation. Une des responsabilités de cet objet est de maintenir une référence à la carte de votre application.

`JMap.app.map` est l'instance de la classe `OpenLayers.Map` de votre application. Les bibliothèques JavaScript de JMap font utilisation de l'API d'OpenLayers afin de pouvoir accomplir une multitude de tâches. Des tâches telles que la manipulation des couches, des opérations sur des coordonnées/étendues en plus de la gestion des contrôles et des infobulles de la carte.

OpenLayers offre une vaste galerie d'exemples portant sur le développement. Il est recommandé que vous utilisiez cette ressource pendant votre développement afin de vous familiariser avec leur API. La galerie ainsi que la documentation de l'API sont des ressources indispensables que vous devriez consulter fréquemment.

Exemples

Cette portion de la documentation présente des exemples de quelques tâches qui peuvent être accomplies dans une extension JMap Web.

La gestion des outils

Deux classes ont été développées de façon à supporter des types d'interactions variées lorsqu'un événement tactile/souris est commis. Les voici:

`JMap.Html5Core.Tool.ToolManager` : Durant le processus d'initialisation, une seule instance de cette classe est produite et est rendue accessible depuis la variable `JMap.app.clickToolManager` . Cet objet fait la gestion de tous les outils enregistrés en plus de l'outil présentement activé.

`JMap.Html5Core.Tool.Tool` : Super classe depuis laquelle tous les outils doivent hériter. Vos écouteurs d'événements seront définis dans une interaction OpenLayers.

Le code suivant crée un outil:

```
SampleTool = function(options) {  
  
    this.hasInteraction = true;  
};
```

```
this.name = 'SampleTool';

this.buttonDiv = document.createElement('div');

$(this.buttonDiv).attr({
  'id': 'SampleToolId',
  'class': 'SampleToolClass CustomButtons'
})
.click($.proxy(function(event) {
  JMap.app.clickToolManager.setCurrentTool(this);
  event.stopImmediatePropagation();
}, this));

$('#JMapStandardToolBar').append(this.buttonDiv);

goog.base(this, options);
};
goog.inherits(SampleTool, JMap.Html5Core.Tool.Tool);

SampleTool.prototype.initializeInteraction = function() {
  this.interaction = new ol.interaction.Pointer({
    handleDownEvent: function(event) {
      var coord = JMap.app.map.getCoordinateFromPixel(event.pixel);
      alert('Clicked at lon: ' + coord[0] + ', lat: ' + coord[1]);
    }
  });
};

SampleTool.prototype.off = function() {
  this.interaction.setActive(false);
  goog.base(this, 'off');
};

SampleTool.prototype.on = function() {
  this.interaction.setActive(true);
  goog.base(this, 'on');
};

var sampleTool = new SampleTool();
```

Après son initialisation, vous devez l'enregistrer auprès du *JMap.app.clickToolManager* puis l'activer.

```
JMap.app.clickToolManager.addTool(sampleTool);
JMap.app.clickToolManager.setCurrentTool(sampleTool);
```

Si un outil était déjà activé au moment où `setCurrentTool()` est appelé, cet outil sera désactivé.

Travailler avec des couches vectorielles

OpenLayers offre plusieurs classes permettant la visualisation de plusieurs types de données. Les classes *ol.layer.Vector* et *ol.source.Vector* sont particulièrement utiles afin d'afficher des éléments de données vectorielles créés par programmation.

Le code suivant crée et ajoute une couche vectorielle à une application initialisée. En utilisant votre outil personnalisé, vous serez capable d'ajouter des points lors que des événements clic seront commis.

```
VectorLayerTool = function(options){

  this.hasInteraction = true;

  this.layerSource = null;

  this.name = 'VectorLayerTool';

  this.vectorLayer = null;

  this.buttonDiv = document.createElement('div');

  $(this.buttonDiv).attr({
    'id': 'VectorLayerToolId',
    'class': 'VectorLayerToolClass CustomButtons'
  })
  .click($.proxy(function(event) {
    JMap.app.clickToolManager.setCurrentTool(this);
    event.stopImmediatePropagation();
  }, this));

  $('#JMapStandardToolBar').append(this.buttonDiv);

  goog.base(this, options);
};
goog.inherits(VectorLayerTool, JMap.Html5Core.Tool.Tool);

VectorLayerTool.prototype.initializeInteraction = function() {
  this.layerSource = new ol.source.Vector({
    wrapX: false
  });

  this.vectorLayer = new ol.layer.Vector({
    source: this.layerSource,
  });

  this.interaction = new ol.interaction.Draw({
    id: 'VectorLayerToolInteraction',
    source: this.layerSource,
    style: new ol.style.Style(),
    type: 'Point'
  });
};

VectorLayerTool.prototype.off = function()
{
  JMap.app.map.removeLayer(this.vectorLayer);
  this.interaction.setActive(false);
  goog.base(this, 'off');
};

VectorLayerTool.prototype.on = function()
{
  JMap.app.map.addLayer(this.vectorLayer);
  this.interaction.setActive(true);
  goog.base(this, 'on');
};

var vectorLayerTool = new VectorLayerTool();
```

De façon similaire, tel que décrit dans l'exemple précédent, le code suivant va enregistrer votre outil auprès du `JMap.app.clickToolManager` puis l'activer.

```
JMap.app.clickToolManager.addTool(vectorLayerTool);  
JMap.app.clickToolManager.setCurrentTool(vectorLayerTool);
```

Modifier des données vectorielles

Présentement, JMap Web n'offre pas d'API permettant d'éditer des données vectorielles. Cependant, vous pouvez utiliser l'API d'OpenLayers afin de créer, modifier et détruire vos propres éléments associés à vos propres couches vectorielles qui furent créées par programmation.

Voici quelques liens expliquant les fonctionnalités d'édition d'OpenLayers:

- Draw Features
- Custom Interactions
- Box Selection
- Draw and Modify Features

Travailler avec les styles vectoriels

L'API de styles d'OpenLayers 3 est beaucoup plus poussé que celui de la version précédente. Il existe plusieurs façons de définir un style, soit:

- Une fonction `ol.style.GeometryFunction()`
- Une fonction `ol.style.StyleFunction()`
- Une ou plusieurs instances de `ol.style.Style`. Si plusieurs, les regrouper dans un *Array* (exemple).

Recommandations

Dans un environnement de production, il est recommandé que les fichiers de votre extension soient minifiés afin de limiter le nombre de requêtes HTTP au chargement. Ceci permet également de masquer votre code jusqu'à un certain niveau. Nous utilisons le Closure Compiler de Google et le recommandons.

Tel que mentionné précédemment, si votre extension possède des dépendances, elles devront être chargées par programmation depuis le fichier JavaScript généré de votre extension.

La documentation de l'API d'OpenLayers est générée en par l'analyse des commentaires du code source. Ceci signifie que certaines méthodes ou propriétés existantes peuvent ne pas être documentées. La navigation du code source d'OpenLayers peut vous aider si quelque chose dans la documentation n'est pas clair.

Méthodologie du développement d'extensions

Les développeurs peuvent souhaiter de travailler différemment lorsqu'ils développent des extensions JMap Web. Les deux méthodes suivantes sont généralement favorisées:

1. Le développeur travaille dans le répertoire de l'extension générée. Il modifie les fichiers de l'extension, copie ses fichiers au répertoire `$JMAP_HOME$/extensions/web`, fait la mise-à-jour de son déploiement puis teste les changements dans son application.
2. Le développeur travaille directement dans les fichiers déployés de son application.

Les deux approches ont leurs propres avantages et inconvénients.

La première technique, bien que plus lente, permet un processus plus sécuritaire et incrémental. Faire le suivi des changements du code est plus simple.

La seconde technique est plus rapide, mais demandera au développeur de copier ses changements vers sa copie de travail obtenue avec son logiciel de contrôle de version. Cette approche peut également mener à la perte de données si une mise-à-jour de l'application est faite avant que les changements ne soient reproduits dans le répertoire `$JMAP_HOME$/extensions/web`.

En plus d'exécuter du code dans le navigateur de l'utilisateur, une extension JMap Web peut également définir des actions qui peuvent tirer parti de l'API de JMap Serveur. Ces actions peuvent être enregistrées auprès du service AJAX Dispatcher de votre déploiement et être interpellées par des requêtes HTTP.

Une extension JMap Web peut contenir plusieurs actions. Vous pouvez également créer des classes additionnelles (qui ne sont pas des actions) qui peuvent être référencées depuis vos actions.

Créer une action

Créez une classe qui hérite de `AbstractHttpRequestAction`. Afin de suivre cet exemple, le nom de votre classe devrait être `MyFirstAction`.

```
package myextension;
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import jmap.http.AJAX.servlets.AbstractHttpRequestAction;

public class MyFirstAction extends AbstractHttpRequestAction
{
    @Override
    public void execute(HttpServletRequest request, HttpServletResponse response) throws IOException
    {
    }
}
```

Comme vous avez pu le remarquer, une action nécessite les classes *HttpServletRequest* et *HttpServletResponse* afin d'être compilée. Ajoutez le le JAR *servlet-api* de *Tomcat Server* à votre chemin de compilation (*build path*). Ce JAR peut être récupéré dans le répertoire `$JMAP_HOME$/tomcat/lib`.

Une action doit posséder une méthode *execute*. Cette méthode sera appelée lorsqu'une requête HTTP spécifiant votre action sera reçue par l'AJAX Dispatcher.

Pour les besoins de cet exemple, voici une action *Hello World!* typique.

```
package myextension;
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import jmap.http.AJAX.servlets.AbstractHttpRequestAction;

public class MyFirstAction extends AbstractHttpRequestAction
{
    @Override
    public void execute(HttpServletRequest request, HttpServletResponse response) throws IOException
    {
        final PrintWriter writer = response.getWriter();
        String name = request.getParameter("name");

        if (name != null && name.trim() != "")
            writer.print("Hello, " + name);
        else
            throw new IllegalArgumentException("Invalid argument. Must specify a `name` parameter");
    }
}
```

Produire un fichier JAR pour votre action

Afin d'être inclus dans une extension JMap Web, votre code serveur Java doit être contenu dans un seul fichier JAR.

En utilisant Ant

En utilisant l'outil `webextensionbuilder` du SDK de JMap 6.5 (décrit dans la section Programmer des extensions JMap Web) afin de produire le code de base de l'extension, une action `SampleAction` est fournie comme point de départ. Un fichier `build.xml` est également inclus. Vous pouvez utiliser ce fichier avec Ant afin de compiler et produire le fichier JAR de votre extension.

Copiez le fichier `MyFirstAction.java` au répertoire `actions/src` de votre extension. Puisque la classe `MyFirstAction` est définie à l'intérieur du package `myextension`, créez un répertoire `myextension` sous `actions/src` et collez-y votre fichier `MyFirstAction.java`. À ce stade, vous devriez avoir les items suivant dans votre répertoire actions:

```
actions/  
+-- build.xml  
+-- readme.markdown  
+-- src  
    +-- Example  
    |   +-- SampleAction.java  
    +-- myextension  
        +-- MyFirstAction.java
```

3 directories, 4 files

En utilisant la ligne de commande/un terminal, naviguez vers le répertoire actions de votre extension et exécutez la commande suivante:

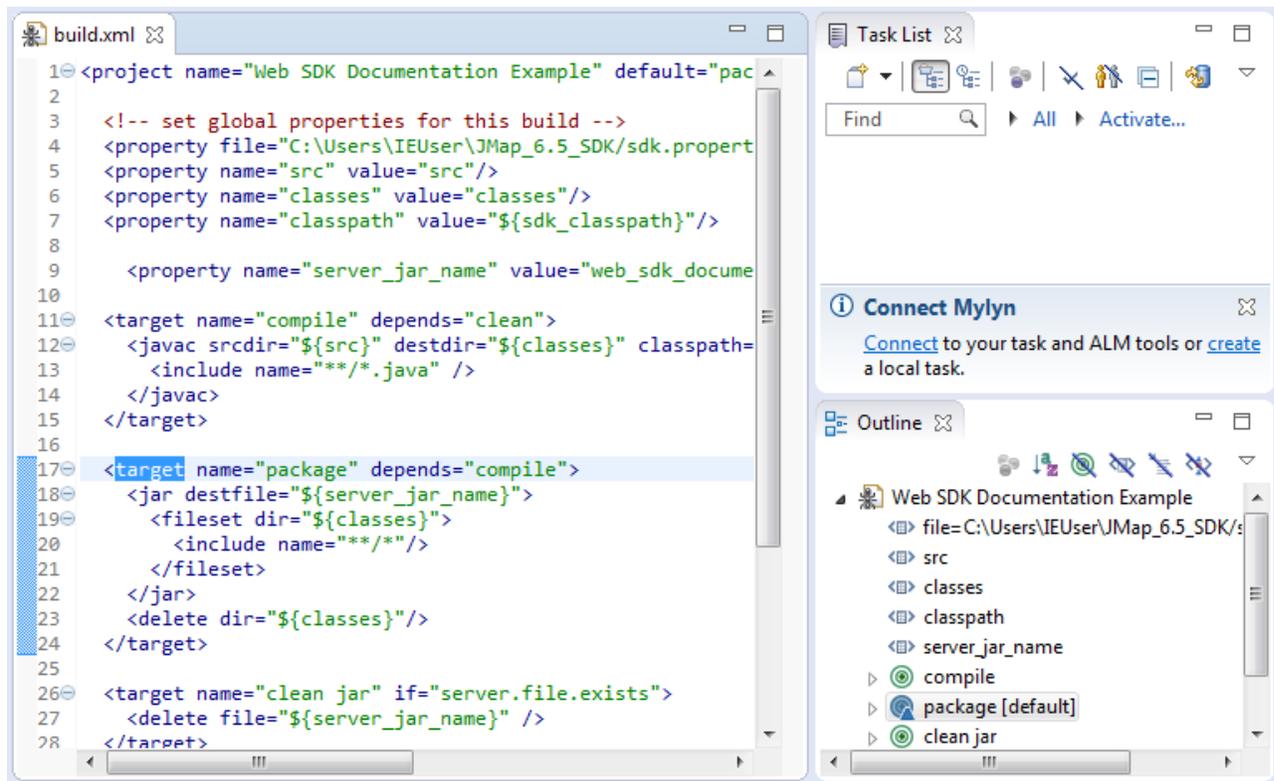
```
ant -f build.xml
```

Lorsque vous aurez votre fichier JAR, copiez le au répertoire `actions` de votre extension.

En utilisant Ant depuis Eclipse

Comme c'était le cas lors de l'utilisation de l'outil `webextensionbuilder`, vous pouvez utiliser Ant depuis Eclipse afin de produire le fichier jar serveur de votre extension. Ceci nécessite qu'un JDK soit configuré comme étant le JRE par défaut d'Eclipse et que le fichier `tools.jar` du JDK soit ajouté au classpath de la configuration du runtime de Ant. Ces étapes furent précédemment décrites ici.

1. Ouvrez le fichier `actions/build.xml` de votre extension dans Eclipse.
2. Exécutez la cible ant nommée «package».



Exécutez la cible «package».

Lorsque vous aurez votre fichier JAR, copiez le au répertoire actions de votre extension.

Identifier les actions de votre extension

Le fichier `extension.json` de votre extension informe JMap Serveur à propos de lui-même. Vous devez maintenant y indiquer que votre extension comporte des actions. Ouvrez ce fichier dans un éditeur de texte.

La propriété `actions` est un tableau qui peut contenir plusieurs littéraux objets. Chacun de ces objets décrivent une action.

Les littéraux objets du tableau `actions` doivent avoir les clefs de propriétés suivantes:

Clefs de propriétés des actions	Description
<code>name</code>	{String} Ceci est le nom de l'action tel qu'elle sera enregistrée auprès du AJAX Dispatcher. Cette valeur doit être unique au sein de votre déploiement JMap Web. Les requêtes HTTP doivent spécifier cette valeur au niveau du paramètre <code>action</code> de celles-ci. Ne fait pas nécessairement référence au nom du fichier source de votre classe.
<code>classname</code>	{String} Le nom complet de votre classe. Doit inclure les packages.

version	{String} Identifie une version. Principalement utile pour fins de débogage.
---------	---

Cet extrait démontre comment représenter l'action `MyFirstAction` dans l'extension `web_sdk_documentation_example` qui fût créé plus tôt:

```
{
  "actions": [
    {
      "name": "hello",
      "className": "myextension.MyFirstAction",
      "version": "1.0.0"
    }
  ],
  "fullname": "Web SDK Documentation Example",
  "namespace": "Example",
  "shortname": "web_sdk_documentation_example",
  "version": "1.0"
}
```

Votre extension peut inclure autant d'actions que vous le voulez. Il suffit simplement de les identifier auprès du fichier `extension.json` de votre extension.

Après avoir modifié votre fichier `extension.json`, confirmez qu'il contient du JSON valide. JSONLint est un validateur JSON en ligne qui peut être utilisé à cette fin.

Appeler votre action depuis votre extension JMap Web

Tel que mentionné précédemment, la méthode `execute` de votre action sera appelée lorsque l'AJAX Dispatcher recevra une requête HTTP qui indique le nom de votre action au niveau de son paramètre `action`.

L'exemple suivant illustre comment vous pouvez envoyer une requête HTTP auprès de votre action en utilisant `jQuery.ajax()`. Pour tester ceci, vous pouvez soit inclure cet extrait dans la méthode `init` de votre extension ou bien le copier et le coller dans la console JavaScript de votre navigateur qui inspecte présentement votre déploiement.

```
$.ajax(JMap.app.ajaxDispatcher, {
  data: {
    "action": "hello", // The action name as defined in the extension.json file.
    "name": "Developer" // The `name` parameter as expected by the action.
  }
}).error(function(jqXHR, textStatus, errorThrown) {
  alert(textStatus);
}).success(function(data, textStatus, jqXHR) {
  alert(data);
});
```

Le modèle d'application JMap Web offre plusieurs actions par défaut afin de faciliter les interactions avec JMap Serveur. Cette section va en présenter quelques-unes et va démontrer comment vous pouvez les utiliser.

Note: La majorité des actions nécessitent des paramètres décrivant l'état de la carte. Pour la plus grande partie, ces informations peuvent être obtenues en utilisant l'API d'OpenLayers.

Tous les exemples de cette section font référence à un déploiement basé sur le projet *The World* . Le déploiement couvre la totalité de l'étendue du projet et contient les couches "Base" et "Cities". Référez-vous aux captures d'écran suivantes afin de créer un déploiement similaire.

Layers

Add new layer ▾

	Name	Type	Visible
Overlays	↕ Cities	Overlay	<input checked="" type="checkbox"/>  
Base layers	↕ Base	Base layer	<input type="checkbox"/>  

Couches configurées

Extents



x: 50.21, y: -100.18 Degrees

Maximum extent Initial extent

Maximum extent x	<input type="text" value="-180"/>	Degrees
Maximum extent y	<input type="text" value="-90.0111"/>	Degrees
Maximum extent width	<input type="text" value="360.0111"/>	Degrees
Maximum extent height	<input type="text" value="180"/>	Degrees

Propriétés géographiques, partie 1.

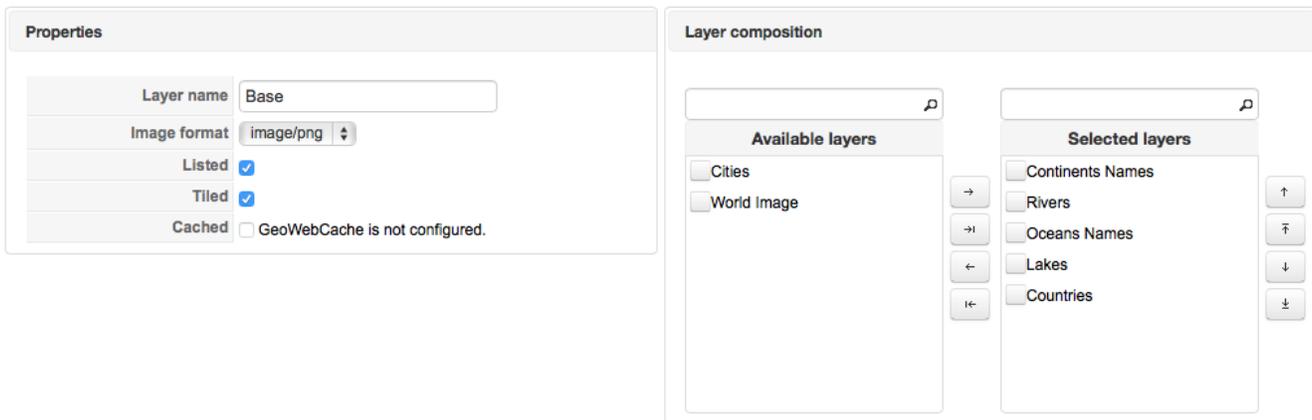
Maximum scale

Maximum scale	1 :	<input type="text" value="3000000.0"/>	Ex: 2000
Allow additional levels	<input type="checkbox"/>		

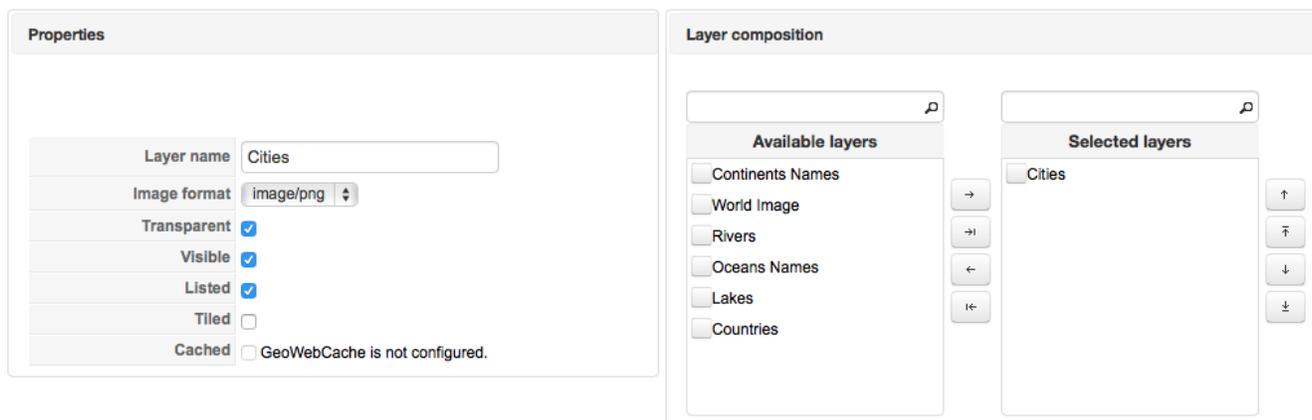
Levels

Level	Scale
1	1:558510621
2	1:279255310
3	1:139627655
4	1:69813827
5	1:34906913
6	1:17453456
7	1:8726728

Propriétés géographiques, partie 2.



Configuration de la couche Base



Configuration de la couche Cities

Action 'extractelemnts'

L'action *extractelemnts* retourne une liste d'éléments d'une couche JMap.

Le type de requête 'geometry'

La requête *geometry* fait une extraction des éléments qui intersectent **ou** sont contenus dans une géométrie fournie.

Paramètres

Le tableau suivant indique la liste des paramètres qui peuvent être acheminés à l'action *extractelemnts*.

Paramètres de extractelemnts	Description

request	{String} "geometry" requis Cette chaîne identifie le type de requête qui est acheminée à l'action. D'autres types de requêtes seront possiblement supportés par cette action dans le futur.
geometry	{String} requis Une géométrie WKT qui définit la région pour laquelle le processus d'extraction d'éléments sera exécuté.
res	{Number} requis La résolution actuelle de la carte. Cette information est accessible depuis la vue de la carte.
bbox	{String} requis L'étendue pour la quelle vous voulez faire l'extraction d'éléments dans un format bounding box (exemple: "-100,-50,100,50").
drill	{Boolean} défaut: false Indique si vous voulez continuer l'analyse de la pile de couches après qu'un premier élément correspondant soit trouvé.
layers	{String} requis Spécifie pour quelle(s) couche(s) vous voulez faire l'extraction d'éléments. Plusieurs couches peuvent être demandées à la fois. Dans ce cas, vous devez séparer les éléments de la liste en utilisant des virgules. Utilisez le paramètre drill de manière à obtenir les éléments pour plusieurs couches.
filters	{JSONArray} Spécifie une liste d'identifiants de filtres d'éléments à respecter lors de l'extraction.

Exemple

Cet exemple fait une requête d'extraction d'éléments pour la couche «Cities» autour de l'Écosse.

```
$.ajax(JMap.app.ajaxDispatcher, {data: {
  'action': 'extractelements',
  'request': 'geometry',
  'geometry': 'POLYGON((-6.378882 54.903806,-1.336158 54.903806,-1.336158 59.540037,-6.378882 59.540037,-6.378882 54.903806))',
  'res': JMap.app.map.getView().getResolution(),
  'bbox': JMap.app.map.getView().calculateExtent(JMap.app.map.getSize()).join(','),
  'layers': 'Cities'
}}).success(function(data, textStatus, jqXHR) {
  console.log(data);
});
```

Réponse

L'identifiant, la géométrie (sous la forme d'une chaîne WKT), les valeurs d'attributs, l'étendue et le point centré pour chacun des éléments sont retournés. Les attributs liés ainsi que leur type SQL sont également inclus. L'index des valeurs d'attributs de l'élément correspond à l'index des attributs configurés sur la couche dans JMap Admin. Si des éléments de plusieurs couches correspondent aux critères de la requête, un tableau de ces objets sera retourné.

```

{
  "layerId": 7,
  "elementAttributes": [
    {
      "sqlType": 12,
      "name": "CITY"
    },
    {
      "sqlType": 12,
      "name": "COUNTRY"
    },
    {
      "sqlType": 12,
      "name": "CAP"
    },
    {
      "sqlType": 4,
      "name": "POP2000"
    }
  ],
  "elements": [
    {
      "centeredPoint": {
        "x": -1.3899999735879476,
        "y": 54.910001831054686
      },
      "bounds": {
        "x": -1.3899999735879476,
        "width": 0,
        "y": 54.910001831054686,
        "height": 0
      },
      "attributeValues": [
        "Sunderland",
        "UK - England and Wales",
        "0",
        181100
      ],
      "geometry": "POINT(-1.3899999735879476 54.910001831054686)",
      "id": 622
    },
    {
      "centeredPoint": {
        "x": -4.2699998496102864,
        "y": 55.87000091552734
      },
      "bounds": {
        "x": -4.2699998496102864,
        "width": 0,
        "y": 55.87000091552734,
        "height": 0
      },
      "attributeValues": [
        "Glasgow",
        "Scotland",
        "0",
        610700
      ],
      "geometry": "POINT(-4.2699998496102864 55.87000091552734)",
    }
  ]
}

```

```
    "id": 624
  },
  {
    "centeredPoint": {
      "x": -3.2200001357125814,
      "y": 55.949998931884764
    },
    "bounds": {
      "x": -3.2200001357125814,
      "width": 0,
      "y": 55.949998931884764,
      "height": 0
    },
    "attributeValues": [
      "Edinburgh",
      "UK - England and Wales",
      "0",
      382600
    ],
    "geometry": "POINT(-3.2200001357125814 55.949998931884764)",
    "id": 625
  },
  {
    "centeredPoint": {
      "x": -2.9999998686837728,
      "y": 56.469999389648436
    },
    "bounds": {
      "x": -2.9999998686837728,
      "width": 0,
      "y": 56.469999389648436,
      "height": 0
    },
    "attributeValues": [
      "Dundee",
      "Scotland",
      "0",
      148900
    ],
    "geometry": "POINT(-2.9999998686837728 56.469999389648436)",
    "id": 627
  },
  {
    "centeredPoint": {
      "x": -2.1000000117349202,
      "y": 57.14999969482422
    },
    "bounds": {
      "x": -2.1000000117349202,
      "width": 0,
      "y": 57.14999969482422,
      "height": 0
    },
    "attributeValues": [
      "Aberdeen",
      "Scotland",
      "0",
      188500
    ],
  },
```

```

    "geometry": "POINT(-2.1000000117349202 57.14999969482422)",
    "id": 628
  },
  {
    "centeredPoint": {
      "x": -1.6000000117349202,
      "y": 54.99999816894531
    },
    "bounds": {
      "x": -1.6000000117349202,
      "width": 0,
      "y": 54.99999816894531,
      "height": 0
    },
    "attributeValues": [
      "Newcastle upon Tyne",
      "UK - England and Wales",
      "0",
      980000
    ],
    "geometry": "POINT(-1.6000000117349202 54.99999816894531)",
    "id": 1899
  }
]
}

```

Action 'layerinfo'

L'action *layerinfo* offre des détails au sujet des couche du projet associé au déploiement courant.

Le type de requête 'geteditablelayers'

Le type de requête *geteditablelayers* offre des détails au sujet des couches sur lesquelles l'utilisateur courant peut faire des tâches d'édition.

Paramètres

Cette action ne nécessite aucun paramètre. À la place, elle utilise l'information de la session courante afin d'effectuer cette opération.

Exemple

```

$.ajax(JMap.app.ajaxDispatcher, {data: {
  'action': 'layerinfo',
  'request': 'geteditablelayers'
}}).success(function(data, textStatus, jqXHR) {
  console.log(data);
});

```

Réponse

La réponse du serveur comprend un tableau d'objets (*editableLayers*). Chacun des objets du tableau contient les clefs de propriétés suivantes:

Clefs de propriétés d'une réponse geteditables	Description
fields	{Array} Tableau d'objets qui exposent le nom des attributs liés ainsi que leur type de donnée SQL.
forms	{Array} Les formulaires configurés de la couche.
id	{Number} L'identifiant de la couche.
idFieldName	{String} Le nom de l'attribut de la couche qui sert d'identifiant.
offset	{Object} Un objet qui contient le décalage actuel de la couche.
permissions	<p>{Number} Un entier entre 0 et 15 (inclusif) qui décrit les permissions envers la couche de l'utilisateur courant. Les masques de bits suivant signifient différentes permissions.</p> <ul style="list-style-type: none"> • 0x0000: Aucune. • 0x0001: Peut ajouter des éléments à la couche. • 0x0010: Peut modifier les éléments de la couche. • 0x0100: Peut détruire les éléments de la couche. • 0x1000: Peut modifier les valeurs d'attributs des éléments de la couche.

Les formulaires sont retournés sous la forme d'un tableau d'objets. Ils ont la structure suivante:

Clefs de propriétés des formulaires de la réponse	Description
id	{Number} L'identifiant unique du formulaire.
json	{String} Une représentation JSON d'une configuration de formulaire.
name	{String} Le nom du formulaire tel que défini dans JMap Admin.
permissions	{Number} Un entier entre 0 et 7 (inclusif) qui décrit les permissions envers le formulaire de l'utilisateur courant. Les permissions d'un formulaire ne concerne

	<p>que les formulaires externes. Les permissions d'un formulaire d'attributs de couche sont définis dans les permissions de la couche.</p> <ul style="list-style-type: none">• 0x000: Aucune.• 0x001: Peut ajouter des données dans un formulaire externe.• 0x010: Peut modifier des données dans un formulaire externe.• 0x100: Peut supprimer des données dans un formulaire externe.
type	<p>{String} Identifie le type de formulaire. Un parmi les suivants:</p> <ul style="list-style-type: none">• LAYER_ATTRIBUTES_FORM• LAYER_ATTRIBUTES_SUB_FORM• EXTERNAL_ATTRIBUTES_FORM• EXTERNAL_ATTRIBUTES_SUB_FORM
uidAttributeName	<p>{String} Ne concerne que les formulaires externes. Correspond au nom de l'attribut de la couche qui sera utilisé comme clef étrangère afin d'établir le lien parent-enfant.</p>

Exemple

Ceci est un exemple d'une réponse *geteditablelayers*. Il est possible que vous ayez des résultats différents. Dans tous les cas, la réponse devrait refléter vos permissions envers les couches et les formulaires.

```
{
  "editableLayers": [
    {
      "offset": {
        "x": 9.5367431640625e-7,
        "y": -3.186320304870577
      },
      "permissions": 0,
      "id": 4,
      "fields": [
        {
          "name": "COUNTRY",
          "serverDataType": 12
        },
        {
          "name": "CONTINENT",
          "serverDataType": 12
        },
        {
          "name": "POP_1994",
          "serverDataType": 8
        },
        {
          "name": "POP_GRW_RT",
          "serverDataType": 8
        },
        {
          "name": "POP_0_14",
          "serverDataType": 8
        },
        {
          "name": "POP_15_64",
          "serverDataType": 8
        },
        {
          "name": "POP_65PLUS",
          "serverDataType": 8
        },
        {
          "name": "POP_AREA",
          "serverDataType": 8
        }
      ],
      "idFieldName": "JMAP_ID",
      "forms": []
    },
    {
      "offset": {
        "x": 0.56195068359375,
        "y": 13.687942504882812
      },
      "permissions": 0,
      "id": 6,
      "fields": [
        {
          "name": "LAKE_NAME",
          "serverDataType": 12
        }
      ]
    }
  ]
}
```

```

        "name": "VOLUME_CKM",
        "serverDataType": 8
    },
    {
        "name": "AREA_SKM",
        "serverDataType": 8
    }
],
"idFieldName": "JMAP_ID",
"forms": []
},
{
    "offset": {
        "x": 20.934576233500025,
        "y": 10.050437668683657
    },
    "permissions": 0,
    "id": 5,
    "fields": [],
    "idFieldName": "JMAP_ID",
    "forms": []
},
{
    "offset": {
        "x": 16.215000694478334,
        "y": 16.463705277985326
    },
    "permissions": 0,
    "id": 3,
    "fields": [
        {
            "name": "HYD_NAME",
            "serverDataType": 12
        },
        {
            "name": "LENGTH_KM",
            "serverDataType": 4
        }
    ],
    "idFieldName": "JMAP_ID",
    "forms": []
},
{
    "offset": {
        "x": 31.452202349999993,
        "y": -14.421794805000005
    },
    "permissions": 0,
    "id": 2,
    "fields": [
        {
            "name": "CONTINENTNAME",
            "serverDataType": 12
        }
    ],
    "idFieldName": "JMAP_ID",
    "forms": []
},
{

```

```

    "offset": {
      "x": 1.6169597031546061,
      "y": 8.079999999999998
    },
    "permissions": 15,
    "id": 7,
    "fields": [
      {
        "name": "CITY",
        "serverDataType": 12
      },
      {
        "name": "COUNTRY",
        "serverDataType": 12
      },
      {
        "name": "CAP",
        "serverDataType": 12
      },
      {
        "name": "POP2000",
        "serverDataType": 4
      }
    ],
    "idFieldName": "JMAP_ID",
    "forms": [
      {
        "permissions": 0,
        "name": "Form",
        "json": "{\"formSections\": [{\"name\": \"Section 1\", \"nbRows\": 3, \"field\"",
        "id": 1,
        "type": "LAYER_ATTRIBUTES_FORM",
        "uidAttributeName": null
      }
    ]
  }
}

```

Action 'loadformdata'

L'action *loadformdata* offre les données des formulaires externes pour un élément d'une couche demandée.

Paramètres

loadformdata attend les paramètres suivants:

Paramètres de loadformdata	Description
data	{String} requis

Chaîne JSON qui représente un littéral objet JavaScript composé des propriétés suivantes:

- `elementId`: {Number} L'identifiant unique de l'élément pour lequel on fait la demande de données.
- `formId`: {Number} L'identifiant du formulaire externe.
- `layerId`: {Number} L'identifiant de la couche sur laquelle l'élément existe/le formulaire est configuré.
- `listFields`: {Array[Strings]} Une liste de noms de champs du formulaire pour lesquels vous voulez des valeurs. Un tableau vide va retourner les données pour tous les champs du formulaire.
- `mapValues`: Littéral objet JavaScript qui contient les valeurs des champs du formulaire d'attributs de la couche.

Exemple

Exemple de requête `loadformdata` . Cet exemple fait référence à un autre projet JMap sur lequel des formulaires externes sont configurés.

```
var data = {
  elementId: 6,
  formId: 2,
  layerId: 1,
  listFields: [],
  "mapValues": {
    "MOBILE_JMAP_ID": 6,
    "MOBILE_JMAP_GEOMETRY": "POINT(-8189010.0 5701129.5)",
    "AUTHOR": "jrhaddad",
    "CREATION_TIME": 1415767972000,
    "MODIFICATION_TIME": 1418400517000,
    "ABR_CODE": "342",
    "ABR_NAME": "ES34F",
    "ABR_LOC_TYPE": "Arrêt bus",
    "ABR_WHEEL_CHAIR": "Non accessible",
    "ABR_DATE_INSP": null,
    "ABR_STATUS": null
  }
};

$.ajax(JMap.app.ajaxDispatcher, {data: {
  'action': 'loadformdata',
  'data': JSON.stringify(data)
}}).success(function(data, textStatus, jqXHR) {
  console.log(data);
});
```

Réponse

Le serveur répond à l'aide d'un tableau à deux dimensions de littéraux objets JavaScript représentant les valeurs des champs du formulaire. Chaque item dans le tableau `rows` correspond

à une rangée de données. Plusieurs rangées seront retournées lorsque l'on demande des données pour un formulaire de type `EXTERNAL_ATTRIBUTES_SUB_FORM`.

Le nom du champs du formulaire, sa valeur et le type SQL de la donnée seront retournés.

```
{
  "rows": [
    [
      {
        "name": "insp_abribus.id_inspection",
        "value": 5,
        "type": 4
      },
      {
        "name": "insp_abribus.id_abribus",
        "value": 6,
        "type": 4
      },
      {
        "name": "insp_abribus.date_inspection",
        "value": 1415854800000,
        "type": 93
      },
      {
        "name": "insp_abribus.etat",
        "value": "Bon état",
        "type": 12
      },
      {
        "name": "insp_abribus.observations",
        "value": "Tout est ok",
        "type": 12
      }
    ]
  ]
}
```

Afin de déployer une extension avec votre déploiement JMap Web, vous devez d'abord installer l'extension sur votre serveur JMap. Vous installez une extension JMap Web en copiant son répertoire parent (nommé selon le shortname de l'extension) vers le répertoire `$JMAP_HOME$/extensions/web`.

Durant le processus de déploiement d'une application JMap Web, il est maintenant possible de choisir les extensions Web que vous souhaitez déployer. Sélectionnez votre nouvelle extension «Web SDK Documentation Example» et complétez l'assistant.

Application deployment wizard

Cancel

Previous

Create

Identification > Template > Path > Options > Extensions

 Web SDK Documentation Example (version 1.0)*Déployer l'extension d'exemple*

Le design de JMap Web permet d'intégrer des déploiements JMap Web dans vos propres applications. Cette section décrit les étapes nécessaires afin d'activer l'intégration.

Copier les bibliothèques JMap Web et dépendances vers votre serveur web

Vos applications JMap Web déployées, tel qu'elles sont servies par l'instance Tomcat de JMap, contiennent les bibliothèques et services web nécessaires pour l'intégration dans votre application. Les fichiers de votre déploiement se situent dans le répertoire `$JMAP_HOME$/applications/deployed`.

Parmi ces fichiers se trouve le répertoire `jmap`. Il contient toutes les ressources nécessaires pour intégrer votre déploiement JMap. **Ce répertoire ainsi que son contenu doit être copié vers votre serveur web et doit être accessible depuis les pages web de votre application.**

Les dépendances de JMap Web seront chargés durant le processus d'initialisation avant l'affichage de la carte. Pour plus de détails au sujet du processus d'initialisation, référez-vous à la section Le processus de démarrage de JMap Web de ce document.

- JQuery est requis durant le processus d'initialisation et doit être présent dans le document de votre application.
- L'ajout de feuilles de styles des dépendances de JMap Web va modifier les styles de votre document. Présentement, il n'y a malheureusement pas de façon d'éviter ceci.

Autoriser le partage de ressources d'origines croisées

Afin de permettre l'intégration de vos applications, quelques modifications au fichier `web.xml` de votre déploiement seront requises.

- La modification du fichier `web.xml` de l'application nécessitera un téléchargement/chargement de votre déploiement. Ceci peut être fait depuis la section déploiement de JMap Admin.
- Toutes modifications faites aux fichiers du déploiement seront perdues si vous en faites la mise-à-jour.

1. Activer l'authentification HTTP pour le filtre `JMapLoginFilter_index`.

```
<filter>
  <filter-name>JMapLoginFilter_index</filter-name>
  <!-- ... -->
  <init-param>
    <param-name>httpauthentication</param-name>
    <param-value>>true</param-value>
  </init-param>
</filter>
```

2. Instancier et configurer le filtre CORS . Ajoutez la déclaration suivante avant le premier «mapping» du `ByPassFilter` .

```
<filter>
  <filter-name>CorsFilter</filter-name>
  <filter-class>org.apache.catalina.filters.CorsFilter</filter-class>
  <init-param>
    <param-name>cors.allowed.headers</param-name>
    <param-value>Content-Type,X-Requested-With,accept,Origin,Access-Control-Request-Method</param-value>
  </init-param>
  <init-param>
    <param-name>cors.exposed.headers</param-name>
    <param-value>Access-Control-Allow-Origin,Access-Control-Allow-Credentials</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>CorsFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Intégrer une carte

Une fois que le répertoire `jmap` sera copié vers votre serveur web, il faudra ajouter les références aux ressources suivantes dans la balise `<head>` du document dans lequel vous souhaitez intégrer votre déploiement.

```
<script type="text/javascript" src="jmap/vendor/jquery-2.1.4.min.js"></script>
<link rel="stylesheet" href="jmap/vendor/OpenLayers-v3.15.1_master/ol.css" />
<script type="text/javascript" src="jmap/vendor/OpenLayers-v3.15.1_master/ol.js"></script>
<script type="text/javascript" src="jmap/jmap-web.min.js"></script>
```

- Il est possible que vous deviez modifier les chemins relatifs mentionnés plus haut.
- L'inclusion de `jQuery` n'est pas nécessaire si vous l'utilisez déjà. JMap Web nécessite une version minimale de 1.9.1 afin d'assurer la plus part des fonctionnalités.

L'initialisation de la carte requiert une balise `div` vide dans laquelle la carte sera créée.

```
<div id="map" style="height: 600px; width: 600px;"></div>
```

Tip: Si vous ne voyez pas la carte, assurez-vous que votre `div` possède une taille valide.

Ajoutez le JavaScript suivant pour déclencher l'initialisation de la carte.

```
<script type="text/JavaScript">
  $(document).ready(function() {
    var options = {
      // The URL of your deployed application. This must be an address that your users may
      jmapUrl: 'http://192.168.0.80:8080/web_deployment'
    };

    JMap.initialize(document.getElementById('map'), options);
  });
</script>
```

La fonction *JMap.initialize* qui est appelée plus haut est la même que celle qui fût détaillée ici. Vous pouvez modifier l'application intégrée en précisant ces mêmes propriétés de configuration dans un littéral objet *mapConfig* qui sera fournis à l'argument options.

embed_example.html

Un fichier *embed_example.html* est inclus dans le répertoire de votre application déployée. Ce fichier procure un exemple de d'application JMap Web intégrée. Vous pouvez ouvrir ce fichier dans un éditeur de texte afin de voir comment l'intégration est faite.

Vous pouvez également copier ce fichier **ainsi que le répertoire *jmap*** vers un serveur web séparé afin de pouvoir confirmer que le partage des ressources d'origines croisées est bien activé.

Gérer l'authentification

Si vous avez activé l'accès contrôlé sur votre déploiement JMap Web, une authentification auprès de JMap sera nécessaire avant de pouvoir initialiser la carte.

Le code suivant est fourni à titre d'exemple, mais n'est pas une façon recommandée de s'authentifier auprès de JMap Serveur.

```
<script type="text/JavaScript">
  $(document).ready(function() {
    $.ajaxSetup({
      xhrFields: {
        withCredentials: true
      }
    });

    $.ajax('http://192.168.0.80:8080/web_deployment/ajaxdispatch', {
      data: {
        action: 'ping'
      },
      beforeSend: function(xhr) {
        // Replace 'USERNAME:PASSWORD' with a valid string value.
        xhr.setRequestHeader('Authorization', 'Basic ' + btoa('USERNAME:PASSWORD'));
      },
    }).done(function(data, textStatus, jqXHR) {
      var options = {
        jmapUrl: 'http://192.168.0.80:8080/web_deployment'
        mapConfig: {},
        onMapInit: function() {
          console.log('Map was initialized.');
        }
      };

      JMap.initialize(document.getElementById('map'), options);
    });
  });
</script>
```

Nous joindre

Par téléphone

Vous pouvez communiquer avec nous durant les heures de bureau (8 h 30 à 16 h 30, du lundi au vendredi) au +1 514.285.1211.

Sur le Web

Visitez notre site Web au k2geospatial.com pour plus de renseignements sur nos produits ou pour obtenir du soutien technique.

Par courriel

Soutien technique : support@k2geospatial.com

Ventes : ventes@k2geospatial.com

Notre adresse

K2 Geospatial

740 rue Notre-Dame Ouest, bureau 1260

Montréal , Québec, Canada, H3C 3X6