**Création dynamique du menu de Cesium\_Opendata avec param.json**

**Le fichier param.json**

Le menu de gauche dans Cesium n’est pas codé directement dans le index.html : tous les paramètres nécessaires sont stockés dans un fichier de paramétrage param.json, qui est ensuite la par la classe param.js pour créer les éléments html nécessaires et les évènements associés au html.

1. **Couches au format Geojson**

On peut ajouter les couches de données au format geojson (soit disponibles en flux depuis Opendata Strasbourg ou alors stockées en local dans le dossier data du serveur).

Les cas suivants sont actuellement pris en compte pour que le param.js les affiche correctement dans Cesium sans aucune modification de code :

|  |  |  |  |
| --- | --- | --- | --- |
| Fonctionnalités | Temps réel (bouton synchroniser) | Animation des données | 2ème couche attributaire |
| Surface | Oui | Oui | Non |
| Ligne | Oui | Non | Non |
| Point | Oui | Oui | Oui |

* **Temps réel** : fonctionnel de manière générale pour tous les types de données
* **Animation** : fonctionnel uniquement avec la donnée Qualité de l’air ;

(Peut-être fonctionnel de manière générale si l’attribut du temps est au format AAAA-MM-JJ et stocké dans la colonne « date\_echeance » de la donnée, mais nécessitera certainement un travail javascript dans la classe globe)

Si la donnée temps est stockée dans data\_echeance et signifie que la donnée est valable pour la veille de la date indiquée (comme la qualité de l’air), définir une valable ‘choice’ quelconque. Si la donnée est valable pour le jour même, définir la variable ‘choice’ = ‘default’.

* **Couche** **attributaire** : Fonctionnel en créant une nouvelle fonction createTableau + choiceTableau dans la classe Tableau qui se base sur le fonctionnement de la fonction createTableauPiscine 🡪 la nouvelle fonction sera lue automatiquement dans la classe globe

Les différents cas font chacun appel à une fonction différente dans la classe globe.js :

|  |  |  |  |
| --- | --- | --- | --- |
| Fonctions utilisées (classe Globe) | Temps réel | Animation | Couche attributaire |
| Surface | loadGeoJson | loadTimeSurf |  |
| Ligne | loadGeoJson |  |  |
| Point | loadPoint | loadTimePoint | loadJsonAttribut |

Notes pour les attributs :

* Attention à toujours utiliser les doubles guillemets " "
* Tous les attributs qui définissent une couleur sont au format "#FFFFFF" (voir <https://www.webfx.com/web-design/color-picker/>)
* Les attributs qui définissent des nombres ou des booléens (true/false) se définissent sans guillemets
* La valeur de transparence alpha est définie sur ]0,1] 🡪 1 définit une surface complètement opaque, mais 0 ne va pas fonctionner pour mettre une surface transparente, définir plutôt 0.001
* Pour les données privées : on ajoute un attribut « private = true «  après le name\_titre de l’onglet qu’on souhaite rendre privé

**Attributs obligatoires pour toutes les couches :**

|  |  |
| --- | --- |
| type\_donnee | prend la valeur ‘surface’, ‘ligne’ ou ‘point’ |
| name | le nom qui va apparaitre dans le menu de l’application |
| id\_data | l’identifiant html unique pour la donnée |
| url\_data | le lien vers le geojson |
| choice | donne un nom aux entités dont le tableau n’est pas retravaillé 🡪doit être unique pour que le highlight soit bien géré |

**Attributs optionnels pour toutes les couches :**

|  |  |
| --- | --- |
| identifiant | Une chaine de caractère qui va protéger la donnée privée 🡪 obligatoire si l’onglet est privé |
| temps\_reel | prend la valeur oui si la donnée est en temps réel et n’est pas défini sinon |
| url\_info | le lien vers la page opendata qui donne les infos sur la donnée (si la donnée vient de l’opendata) |
| nom\_legende | à définir si la donnée doit afficher une légende (attention à mettre des \_ à la place des espaces) |
| couleur\_legende | obligatoire si nom\_legende est défini 🡪 les couleurs qui vont apparaître dans la légende pour chaque valeur |
| choiceTableau | permet d’appeler automatiquement la fonction définie dans la classe Tableau pour mettre en forme les attributs de la donnée à condition que  nom de la fonction = createTableau + 'choiceTableau' |

**Pour les données surfaciques « normales » (sans animation) :**

**Attributs obligatoires :**

|  |  |
| --- | --- |
| champ\_classif | l’attribut selon lequel les données doivent être classifiées |
| couleur\_classif | contient les valeurs que peut prendre le champ\_classif et les couleurs à associer à chaque valeur |
| alpha | transparence des surfaces (nombre) |

**Attributs optionnels :**

|  |  |
| --- | --- |
| name\_contour | donner un nom lorsqu’on clique sur les contours |
| couleur\_contour | la couleur du contour 🡪 à définir obligatoirement lorsqu’on veut que les contours apparaissent |
| taille\_contour | L’épaisseur du contour en pixels |
| couleur\_highlight | La couleur de la surbrillance dans laquelle la surface va s’afficher lorsqu’on clique dessus 🡪 à définir pour qu’une surbrillance apparaisse |
| alpha\_highlight | La transparence de la surbrillance |
| couleur\_border | La couleur du contour autour du cadre de la légende |

**Pour les données surfaciques temporelles :**

**Attributs obligatoires :**

|  |  |
| --- | --- |
| animation | prend la valeur oui |
| champ\_classif | l’attribut selon lequel les données doivent être classifiées |
| couleur\_classif | contient les valeurs que peut prendre le champ\_classif et les couleurs à associer à chaque valeur |
| alpha | transparence des surfaces (nombre) |
| start | la date de début (nombre de jours depuis le jour d’aujourd’hui, négatif si dans le passé) |
| end | la date de fin (idem) |

**Attribut optionnel :**

|  |  |
| --- | --- |
| name\_contour | donner un nom lorsqu’on clique sur les contours  🡪 Attention, attribut obligatoire si on souhaite que les contours apparaissent (ici toujours blanc et de largeur 3) |

**Pour les données linéaires :**

**Attributs obligatoires :**

|  |  |
| --- | --- |
| champ\_classif | l’attribut selon lequel les données doivent être classifiées |
| couleur\_classif | contient les valeurs que peut prendre le champ\_classif et les couleurs à associer à chaque valeur |
| alpha | transparence des lignes (nombre) |
| ligne\_2D | true si les lignes doivent être clamp sur le photomaillage, false si les lignes sont 3D |

**Attribut optionnel :**

|  |  |
| --- | --- |
| epaisseur | la largeur de la ligne en pixel (valeur par défaut : 4) |
| couleur\_highlight | La couleur de la surbrillance dans laquelle la ligne va s’afficher lorsqu’on clique dessus 🡪 à définir pour qu’une surbrillance apparaisse |
| alpha\_highlight | La transparence de la surbrillance |

**Pour les données ponctuelles « normales » (sans 2ème couche attributaire) :**

**Attributs obligatoires :**

|  |  |
| --- | --- |
| point\_3D | prend la valeur false si les points n’ont pas de composante Z, true sinon |
| cluster | prend la valeur true si les points doivent être clusterisés, false sinon |
| image | le lien vers le billboard à utiliser pour les entités // les valeurs que prend champ\_classif et les liens vers les billboards associés si la donnée doit être classifiée |

**Attributs optionnels :**

|  |  |
| --- | --- |
| couleur | la couleur à la fois de la ligne qu’on trace depuis le sol jusqu’au billboard surélevé (si les points sont en 2D) et de la puce pour le cluster si les entités doivent être clusterisées  🡪 à définir obligatoirement quand cluster = true ou quand point\_3D = false |
| champ\_classif | l’attribut selon lequel les données doivent être classifiées, non défini si pas de classification |
| couleur\_legende | Pour les ponctuels : les liens vers les billboards de la légende (même image mais dans le 1er cas c’est un lien relatif vers l’image, ici c’est une balise html img) |

Attention : pour les données ponctuelles 2D, les attributs de la donnée ne seront visibles qu’avec la création d’une nouvelle fonction createTableau et la définition de choiceTableau.

Pour les points 2D, on crée une nouvelle entité billboard à une hauteur au-dessus du photomaillage : c’est une donnée uniquement visuelle sans donnée attributaire : le lien entre les attributs de l’entité et les billboard se fait via la fonction createTableau.

Note : en cas de classification des données ponctuelles, tous les cas non pris en compte dans la définition de ‘image’ (càd toutes les valeurs de champ\_classif non précisées) seront obligatoirement représentées avec le billboard marker\_black

**Pour les données ponctuelles avec 2ème couche attributaire (exemple des piscines) :**

**Attributs obligatoires :**

|  |  |
| --- | --- |
| couche\_attributaire | prend la valeur oui |
| point\_3D | prend la valeur false si les points n’ont pas de composante Z, true sinon |
| image | le lien vers le billboard à utiliser pour les entités // les valeurs que prend champ\_classif et les liens vers les billboards associés si la donnée doit être classifiée |
| url\_attribut | le lien vers le 2ème json attributaire à intégrer |

(pas de cluster pour les entités avec 2ème couche attributaire)

**Attributs optionnels :**

|  |  |
| --- | --- |
| couleur | la couleur de la ligne qu’on trace depuis le sol jusqu’au billboard surélevé si les points sont en 2D 🡪 à définir obligatoirement quand point\_3D = false |
| champ\_classif | l’attribut selon lequel les données doivent être classifiées, non défini si pas de classification |
| couleur\_legende | Pour les ponctuels : les liens vers les billboards de la légende (même image mais dans le 1er cas c’est un lien relatif vers l’image, ici c’est une balise html img) |

**Pour les données ponctuelles avec animation :**

**Attributs obligatoires :**

|  |  |
| --- | --- |
| animation | prend la valeur oui |
| point\_3D | prend la valeur false si les points n’ont pas de composante Z, true sinon |
| image | le lien vers le billboard à utiliser pour les entités // les valeurs que prend champ\_classif et les liens vers les billboards associés si la donnée doit être classifiée |
| start | la date de début (nombre de jours depuis le jour d’aujourd’hui, négatif si dans le passé) |
| end | la date de fin (idem) |

**Attributs optionnels :**

|  |  |
| --- | --- |
| couleur | la couleur de la ligne qu’on trace depuis le sol jusqu’au billboard surélevé (si les points sont en 2D)  🡪 à définir obligatoirement quand point\_3D = false |
| champ\_classif | l’attribut selon lequel les données doivent être classifiées, non défini si pas de classification |
| couleur\_legende | Pour les ponctuels : les liens vers les billboards de la légende (même image mais dans le 1er cas c’est un lien relatif vers l’image, ici c’est une balise html img) |

1. **Couches au format 3DTiles**

Le format 3DTiles est utilisé pour visualiser des modèles 3D (exemple de l’école dans le quartier Danube). Pas de création de tableaux d’attributs, ceux-ci seront affichés automatiquement tels que l’export les a gardés.

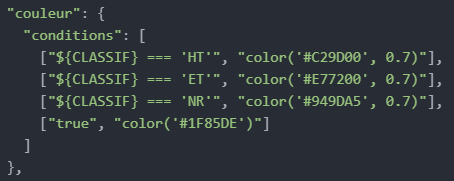
**Attributs obligatoires pour toutes les couches :**

|  |  |
| --- | --- |
| type\_donnee | prend la valeur ‘3dtiles’ |
| name | le nom qui va apparaitre dans le menu de l’application |
| id\_data | l’identifiant html unique pour la donnée |
| url\_data | le lien vers le tileset.json à l’intérieur du dossier dans lequel on a créé le 3dtiles |

**Attributs optionnels pour toutes les couches :**

|  |  |
| --- | --- |
| url\_info | le lien vers la page opendata qui donne les infos sur la donnée (si la donnée vient de l’opendata) |
| nom\_legende | à définir si la donnée doit afficher une légende |
| couleur\_legende | obligatoire si nom\_legende est défini 🡪 les couleurs qui vont apparaître dans la légende pour chaque valeur |
| couleur | A définir si on souhaite classifier la donnée 🡪 respecter la structure dans l’exemple du velum 3D |

Pour la couleur :



On définit à la fois le champ selon lequel on doit classifier, la valeur qu’il peut prendre, puis la couleur et la transparence. Pour l’exemple du velum, on classifie selon le champ classif, qui peut prendre trois valeurs.

La dernière ligne permet de définir une couleur par défaut pour tous les cas qui ne rentreraient pas dans la classification définie plus haut.

1. **Couches dessin geojson (exportés depuis Cesium)**

Les couches drawing.json téléchargées depuis Cesium peuvent être intégrées à nouveau dans le viewer avec le type de donnée « dessin ». Permet de retrouver le style dans lequel on avait dessiné chaque entité.

Exceptions qui confirment la règle :

* Les styles de lignes (pointillés, flèche) sont perdus et les lignes sont dessinées simplement
* Les lignes seront forcément clamp au photomaillage même si le dessin spécifiait que non
* Les points (billboard) perdent leurs étiquettes et seront dessinés avec un seul billboard interface.png

**Attributs obligatoires pour toutes les couches :**

|  |  |
| --- | --- |
| type\_donnee | prend la valeur ‘dessin’ |
| name | le nom qui va apparaitre dans le menu de l’application |
| id\_data | l’identifiant html unique pour la donnée |
| url\_data | le lien vers le json |

Pas d’attributs optionnels pour les dessins

**Comment personnaliser le tableau d’attributs dans Cesium**

Les fonctions qui créent les tableaux d’attributs sont stockées dans la classe TableauAttribut du fichier tableau.js. Comme expliqué précédemment, elles sont appelées dynamiquement dans la fonction globe grâce à l’attribut choiceTableau.

On peut alors lier une nouvelle fonction à une couche de donnée sans modifier la classe globe en respectant la condition suivante :

**Nom fonction dans tableau.js = createTableau + choiceTableau**

Attention à respecter les majuscules lors de la définition de l’attribut choiceTableau

Exemple : pour les limites de communes, on a défini dans le json l'attribut choiceTableau = Communes, la fonction doit donc s'appeler createTableauCommunes

Pour les paramètres de la nouvelle fonction, attention à respecter les attributs généralement utilisés dans la fonction load (car ceux-ci sont codés en dur dans la classe globe.js)

|  |  |
| --- | --- |
| Fonction qui va charger la couche de donnée | Paramètres à utiliser pour la fonction createTableau correspondante |
| loadGeoJson | (entity, dataSource) |
| loadPoint | (billboard, entity) |
| loadTime | (entity) |
| loadJsonAttribut | (entity, jsonAttribut, billboard, coordLabel, dataSource) |

Les paramètres et leurs rôles sont décrits dans tableau.js :

* entity est l’entité sur laquelle on cherche et on affiche les attributs
* dataSource est le GeoJson : il permet d’ajouter des éléments liés à la donnée tels que des textes (voir createTableauCommunes ou createTableauSections)
* pour loadPoint, on va lier les attributs du paramètre entity sur le nouvel élément billboard qu’on a créé

**Exemple :**

Pour un tableau d’attributs simple, copier-coller celui de Vitaboucle



On va choisir quels attributs afficher dans Cesium :

* On crée d’abord le tableau d’attributs (1ère ligne)
* On ajoute les lignes au tableau : la partie à gauche correspond à ce qu’on affiche dans la première colonne du tableau, et la partie droite dans quel attribut de l’entité on cherche la donnée à afficher
* On referme le tableau
* On ajoute les éventuels liens vers les informations de la donnée

On peut également mettre des conditions if sur les attributs à afficher, que ceux-ci ne s’affichent que si l’attribut est défini (voir createTableauPLU)

Il est possible d’afficher des photos, des images, ou des paragraphes de description dans le tableau d’attributs (voir createTableauPatrimoine ou createTableauPiscine) en créant les balises HTML correspondantes