en_US.png ...english version of this page

# Principaux types d'items

Les grands types d'items sont définis dans les librairies ctlibstructure et ctlibstructureaddon. Entre crochets […] sont indiqués les noms correspondant aux classes dans l'interface graphique.

Tous les items de Computree héritent de la classe mère **CT_AbstractSingularItemDrawable** [Item]. A ce niveau les items ne sont définis que par deux attributs par défaut : un identifiant unique Computree (ID) et un nom (pour l'affichage, par défaut égale à l'ID). C'est la classe qu'on utilise quand on veut indiquer qu'on peut travailler avec tout type d'item (par exemple pour lire ses attributs).

De le classe **CT_AbstractSingularItemDrawable** [Item] héritent les classes abstraites suivantes :

- **CT_AbstractGeometricalItem** [Geometrical Item] : pour tous les items utilisant le système de coordonnées. Cette classe ajoute en attributs : une boite englobante (min et max en x, y et z), ainsi que le centre (x,y,z) de cette boite englobante.
- **CT_AbstractAttributes** [Attributes] : pour les items permettant de stocker des attributs de points, faces ou arêtes.

De **CT_AbstractGeometricalItem** [Geometrical Item] héritent les classes abstraites suivantes :

- **CT_AbstractItemDrawableWithPointClouds** [Item with points] : pour tous les items contenant des points (nuages de points).
- **CT_AbstractMeshModel** [Mesh model] : pour tous les items contenant des maillages.
- **CT_AbstractItemDrawableWithoutPointClouds** [Item without points] : pour tous les items ne contenant pas de points ni de maillage (formes géométriques, rasters, grilles 3D,…).

De **AbstractItemDrawableWithoutPointClouds** [Item without points] héritent les classes abstraites suivantes :

- **CT_AbstractShape** [3D shape] : pour les formes géométriques 3D (sphères, cylindres,…).
- **CT_AbstractShape2D** [2D shape] : pour les formes géométriques 2D (cercles, polygones,…). Une sous classe **CT_AbstractAreaShape2D** [2D area shape] permet de distinguer les objets ayant une surface (polygones vs lignes).
- **CT_AbstractProfile** [Profile] : pour les profiles à une dimension (histogramme).
- **CT_AbstractImage2D** [Raster] : pour les rasters.
- **CT_AbstractGrid3D** [3D grid] : pour les grilles voxels 3D.
- **CT_AbstractGrid4D** [4D grid] : pour les grilles voxels 4D (x, y, z et poids).

# Nuages de points : Items héritant de CT_AbstractItemDrawableWithPointClouds

- **CT_Scene** [Point scene] : Classe instanciable permettant de stocker un nuage de points. Cette classe fournit donne accès au nuage de points ainsi que sa boite englobante. Elle est optimisée pour les "gros nuages", en particulier ceux créés lors de chargement de fichiers (dans ce cas elle optimise l'espace mémoire, tous les points chargés en une fois étant contigus).

- **CT_PointCluster** [Point cluster] : Classe instanciable permettant de stocker un groupe de points. En plus des points et de la boite englobante, cette classe fournit un barycentre géométrique toujours à jour. Cette classe permet l'ajout des points un par un. Elle est plutôt adaptée à de de petits clusters créés progressivement.

# Formes géométriques 3D : Items héritant de CT_AbstractShape

- **CT_Circle** [3D circle] : Classe instanciable permettant de stocker un cercle en 3D. Le cercle est défini par un centre (x,y,z), un rayon et la direction (dx, dy, dz) orthogonale au plan le contenant. Optionnellement une erreur d'ajustement peut être fournie.
- **CT_Cylinder** [Cylinder] : Classe instanciable permettant de stocker un cylindre. Le cylindre est défini par un centre (x,y,z), un rayon, une longueur et une direction (dx, dy, dz). Optionnellement une erreur d'ajustement de l'axe, et une erreur d'ajustement du rayon peuvent être fournies.
- **CT_Ellipse** [3D ellipse] : Classe instanciable permettant de stocker une ellipse en 3D. L'ellipse est définie par un centre (x,y,z), un grand axe et un petit axe. Chaque axe est déterminé par 2 points (x,y,z). Optionnellement une erreur d'ajustement peut être fournie.

- **CT_Line** [3D line] : Classe instanciable permettant de stocker un segment en 3D. La ligne est définie par deux points (x,y,z) définissant ses extrémités.
- **CT_PlanarBSpline** [Planar B-spline] : Classe instanciable permettant de stocker une courbe B-Spline. La courbe B-Spline est définie par une série de points de contrôle (x,y,z) associés à des valeurs nodales, et par un degré n.
- **CT_Sphere** [Sphere] : Classe instanciable permettant de stocker une sphère. La sphère est définie par un centre (x,y,z) et un rayon.

Pour toutes les formes 3D, il faut d'abord créer un objet purement géométrique (ne contenant que ses paramètres géométriques) héritant de la classe CT_ShapeData, qui est ensuite passé à l'item en lui-même. Cela permet de faire les calculs sur les objets géométriques, et de ne conserver en items que les éléments d'intérêt (optimisation de la mémoire).
Liste des classes géométriques correspondants aux différentes classes d'items :

- **CT_Circle** : **CT_CircleData**
- **CT_Cylinder** : **CT_CylinderData**
- **CT_Ellipse** : **CT_EllipseData**
- **CT_Line** : **CT_LineData**
- **CT_PlanarBSpline** : **CT_PlanarBSplineData**
- **CT_Sphere** : **CT_SphereData**

# Formes géométriques 2D : Items héritant de CT_ AbstractShape2D

Les formes géométriques visent à permettre l'import de données vecteur issues de Système d'Information Géographique en particulier. Il sont représentés en 3D dans un plan (réglable par l'utilisateur).

- **CT_Box2D** [2D box] : Classe instanciable permettant de stocker une boite englobante en 2D, c'est à dire un rectangle aligné sur les axes (x,y). Elle est définie par un centre, une longueur et une largeur. Alternativement on peut la créer à partir d'un point (x,y) en bas à gauche et un point (x,y) en haut à droite.
- **CT_Circle2D** [2D circle] : Classe instanciable permettant de stocker un cercle en 2D. Le cercle est défini par un centre (x,y) et un rayon.
- **CT_Line2D** [2D line] : Classe instanciable permettant de stocker un segment en 2D. La ligne est définie par deux points (x,y) définissant ses extrémités.
- **CT_Point2D** [2D point] : Classe instanciable permettant de stocker un point en 2D. Le point est défini par un point (x,y).
- **CT_Polygon2D** [2D polygon] : Classe instanciable permettant de stocker un polygone en 2D. Le polygone est définit par une série de points (x,y).
- **CT_Polyline2D** [2D polyline] : Classe instanciable permettant de stocker une polyligne en 2D. La polyligne est définie par une série de points (x,y).

Pour toutes les formes 2D, il faut d'abord créer un objet purement géométrique (ne contenant que ses paramètres géométriques) héritant de la classe CT_Shape2DData, qui est ensuite passé à l'item en lui-même. Cela permet de faire les calculs sur les objets géométriques, et de ne conserver en items que les éléments d'intérêt (optimisation de la mémoire).
Liste des classes géométriques correspondants aux différentes classes d'items :

- **CT_Box2D** : **CT_Box2DData**
- **CT_Circle2D** : **CT_Circle2DData**
- **CT_Line2D** : **CT_Line2DData**
- **CT_Point2D** : **CT_Point2DData**
- **CT_Polygon2D** : **CT_Polygon2DData**
- **CT_Polyline2D** : **CT_Polyline2DData**

# Profils de données : Items héritant de CT_ AbstractProfile

- **CT_Profile<type>** [Profile] : Classe instanciable permettant de stocker un Profil. Un profil est en fait un histogramme, donc un tableau de valeurs séquentielles. La logique de cet item est de pouvoir représenter un profil de données en 3D (par exemple un profil de densité de points). L'item profil définit donc également une origine du profil (x,y,z), une direction (dx, dy, dz), ainsi qu'une résolution et une valeur utilisée pour représenter une donnée manquante. C'est un peu l'équivalent d'un raster mais en une dimension, et sans contrainte directionnelle.

# Rasters : Items héritant de CT_ AbstractImage2D

- **CT_Image2D<type>** [Raster] : Classe instanciable permettant de stocker un raster c'est à dire une image dans le plan (x,y) et aligné sur les axes x et y, contenant dans chaque pixel une valeur (altitude, composante couleur,...). Le raster est défini par les coordonnées de son origine (valeurs minimales pour x et y, en bas à gauche), ses dimensions en nombre de pixels le long des axes x et y, sa résolution spatiale (taille en m d'un pixel), la valeur utilisée pour coder une valeur manquante ainsi que son niveau z par défaut pour une représentation en 3D (modifiable par l'utilisateur interactivement). Cet item est a un template <type> permettant de choisir le type de données à stocker dans chaque pixel.

## Grilles 2D : Items héritant de CT_ AbstractGrid3D

- **CT_Grid3D<type>** [3D grid] : Classe instanciable permettant de stocker grille en trois dimensions, donc composée de voxels (pixels 3D). C'est l'équivalent 3D des rasters, alignée sur les axes x, y et z. Une grille 3D est définie par sont origine (x,y,z), ses dimensions selon les axes x, y et z, sa résolution spatiale, la valeur utilisée pour coder une valeur manquante. Cet item a un template <type> permettant de choisir le type de données à stocker dans chaque voxel.
- **CT_Grid3D_Sparse<type>** [3D sparse grid] : Identique à la classe CT_Grid3D<type>, mais elle définit en plus une valeur par défaut. Seuls les voxels ayant une valeur différente de la valeur par défaut son effectivement stockées en mémoire, permettant dans certains cas (valeur par défaut ultra-majoritaire) d'économiser beaucoup de mémoire.
- **CT_Grid3D_Points** [Point 3D Grid] : Classe instanciable proposant une Grille 3D virtuelle, stockant dans chaque voxel les index des points (x,y,z) qu'il contient. Cette grille a pour vocation de permettre des recherches de voisinages optimisées, ou tout simplement une indexation spatiale de points.

## Grilles 4D : Items héritant de CT_ AbstractGrid4D

- **CT_Grid4D<type>** [4D grid] : Classe abstraite intermédiaire dont héritent CT_Grid4D_Dense<type> et CT_Grid4D_Sparse<type>, pour permettre un usage indifférencié.
- **CT_Grid4D_Dense<type>** [4D dense grid] : Equivalent de CT_Grid3D<type> mais avec 4 dimensions.
- **CT_Grid4D_Sparse<type>** [4D sparse grid] : Equivalent de CT_Grid3D_Sparse<type> mais avec 4 dimensions.

## Maillages : Items héritant de CT_AbstractMeshModel

- **CT_MeshModel** [Mesh model] : Classe instanciable permettant de stocker un maillage triangulé 3D. Cette classe fera l'objet d'une page dédiée.
- **CT_OPFMeshModel** [OPF mesh model] : Version spécialisée de CT_MeshModel, adaptée à des échanges de données au format OPF.

## Autres items pouvant être représentés en 3D : Items héritant directement de CT_ AbstractItemDrawableWithoutPointClouds

- **CT_AffiliationID** [Affiliation ID] : Classe instanciable permettant de stocker un identifiant. Cet item un peu particulier, permet d'affilier des items en leur ajoutant un identifiant commun. Cet item garantit l'unicité des identifiants des objets de référence, ensuite attribués à d'autres items à mettre en lien.
- **CT_Beam** [Beam] : Classe instanciable permettant de stocker un "rayon". Il s'agit d'une demi droite, définie par une origine (x,y,z) et une direction (dx,dy,dz).
- **CT_ColorComposite** [Color Composite] : Classe instanciable permettant de combiner trois raster existants pour générer une représentation RVB composite (chaque raster étant mappé sur une des composantes colorées).
- **CT_PlotGridManager** [Plot grid manager] : Classe instanciable permettant de de gérer une maillage régulier. Utilisée par exemple pour générer des raster à partir de données Lidar aéroporté.
- **CT_PlotListInGrid** [Plot grid list] : Classe instanciable permettant de gérer une liste de placettes (ayant chacune une emprise). Utilisée en traitements de données Lidar aéroporté par exemple.
- **CT_ReferencePoint** [Reference point] : Classe instanciable permettant de stocker un point de référence 3D. Chaque point de référence est défini par une coordonnée (x,y,z). Optionnellement on peut définir un rayon d'influence (buffer).
- **CT_Scanner** [Scan position] : Classe instanciable permettant de stocker une position de scan Lidar terrestre. Chaque position est définie par une origine (x,y,z), la direction de la "verticale" du scanner (dx,dy,dz), hFov (plage azimutale de scan), vFov (plage zénithale de scan), hRes (résolution azimutale de scan), vRes (résolution zénithale de scan), initTheta (azimut de départ), initPhi (angle zénithal de départ), clockWise (sens de rotation). Cette classe fournit des méthodes pour générer des CT_Beam.
- **CT_ScanPath** [Scan Path] : Classe instanciable permettant de stocker des trajectoires. Une trajectoire est définie par une série

de positions (x,y,z) associées à des temps GPS. Cela permet de connaitre la position en tout instant de la période d'acquisition. Cette classe est en particulier utilisée pour calculer les directions de scans de points Lidar aéroporté à partir de la trajectoire de l'avion.
- **CT_ShootingPatternD** [Shooting pattern] : Classe instanciable permettant de définir un design de scan quelconque avec des classes héritées (généralisation de CT_Scanner) et de générer des CT_Shot (équivalent de CT_Beam). A terme CT_ShootingPatternD et CT_Shot se substitueront totalement à CT_Scanner et CT_Beam (définis pour un cas particulier correspondant aux Lidar terrestres les plus courants).
- **CT_Triangulation2D** [2D triangulation] : Classe instanciable permettant de stocker une triangulation de Voronoï en 2D. Chaque sommet de la triangulation peut stocker une valeur (comme par exemple une altitude dans le cas où la triangulation représente le modelé du terrain).

- **CT_LoopCounter** [Loop counter] : Classe instanciable permettant de gérer les boucles de scripts de Computree. Le compteur permet ainsi aux étapes de connaître le tour en cours, sont nom et le nombre de tours total.

# Items attributaires (sans représentation 3D) : Items héritant directement de CT_ AbstractSingularItemDrawable

- **CT_ItemAttributeList** [Item attribute list] : Classe instanciable n'ayant aucune données par défaut (item "vide"). Son but est de permettre le stockages d'attributs de types quelconques dans un même conteneur. Utilisé pour stocker les résultats des métriques.
- **CT_TransformationMatrix** [Transformation matrix] : Classe instanciable permettant de stocker une matrice de transformation 3D 4x4.

# Attributs de points, faces et arrêtes : Items héritant de CT_ AbstractAttributes

- **CT_PointsAttributesColor** [Pointcolor attributes] : Classe instanciable permettant de stocker des attributs "couleurs de points".
- **CT_PointsAttributesNormal** [Point normal attributes] : Classe instanciable permettant de stocker des attributs "normales de points" (dx, dy, dz).
- **CT_PointsAttributesScalarMaskT** [Point <type> attributes] : Classe instanciable permettant de stocker des attributs du <type> passé au template.
- **CT_PointsAttributesScalarTemplated<type>** [Point <type> attributes] : Classe instanciable permettant de stocker des attributs de points du <type> passé au template.
- **CT_StdPointsAttributesContainer** [Point attributes list] : Classe instanciable permettant de stocker

- **CT_FaceAttributesColor** [Face color attributes] : Classe instanciable permettant de stocker des attributs "couleurs de faces" (maillages).
- **CT_FaceAttributesNormal** [Face normal attributes] : Classe instanciable permettant de stocker des attributs "normales de faces" (dx, dy, dz).
- **CT_FaceAttributesScalarT<type>** [Face <type> attributes] : Classe instanciable permettant de stocker des attributs de faces du <type> passé au template.

- **CT_EdgeAttributesColor** [Edge color attributes] : Classe instanciable permettant de stocker des attributs "couleurs d'arrêtes" (maillages).
- **CT_EdgeAttributesNormal** [Edge normal attributes] : Classe instanciable permettant de stocker des attributs "normales d'arrêtes" (dx, dy, dz).
- **CT_EdgeAttributesScalarT<type>** [Edge <type> attributes] : Classe instanciable permettant de stocker des attributs d'arrêtes du <type> passé au template.

Ces classes vont évoluer ou disparaître lors de la refonte du système d'attributs en cours.

Retour à l'index