

Configuration Accounting — guide des JSON

Objectif

Cette page sert de carte des fichiers de configuration du système Accounting / ByModule. Le point important est de distinguer trois familles : les **données statiques** utilisées par la simulation, les **settings** qui activent/configurent les modules, et les **definitions** qui décrivent surtout les champs disponibles dans les écrans d'édition/wizards.

1. Vue générale du chargement

flowchart LR

```
JSON[Resources/data_byModule/*.json] --> STATIC[StaticDataByModule.Initialize]
STATIC --> CATALOG[Catalogues statiques]
STATIC --> SETTINGS[ModulesSettings / PagesSettings]
STATIC --> DEFINITIONS[Definitions pour éditeur / wizard]

CATALOG --> GAME[Runtime Accounting]
SETTINGS --> GAME
DEFINITIONS --> EDITOR[Configuration des activités/modules]
EDITOR --> MODULECFG[Configuration sauvegardée du module]
MODULECFG --> GAME
```

Le point d'entrée Unity est `Assets/Src/Game/Data/ByModule/StaticDataByModule.cs`. Il charge les fichiers du dossier `Assets/GameSpecific/byModule/Resources/data_byModule` via `Resources.Load`.

2. Les JSON principaux

Fichier	Rôle	Quand le modifier
<code>modules.json</code>	Catalogue des modules disponibles : uid, nom, description, type de module, editor, icône/image, limites et métadonnées d'affichage.	Quand on ajoute un nouveau module ou qu'on change son identité dans le catalogue.
<code>modules_settings.json</code>	Valeurs/settings par défaut des modules : pages, préférences, bills, outils accounting, durée, options de validation, etc.	Quand on change le comportement par défaut d'un module existant ou sa configuration Accounting.
<code>modules_settings_definitions.json</code>	Décrit quels champs/préférences/bill definitions sont exposés pour chaque type de module dans les écrans de configuration.	Quand un nouveau paramètre doit devenir configurable dans le wizard/editor.
<code>module_pages.json</code>	Structure statique des pages et tableaux : pages, tables, rows, inputs, types d'inputs et outils comme journal général, grand livre, balance, états financiers, cash flow, etc.	Quand la structure visuelle ou fonctionnelle d'une page Accounting change.
<code>pages_settings.json</code>	Comportement d'une page : rangées/sections dynamiques, row editor, formules, navigation et autres options runtime.	Quand une page existante doit accepter un comportement dynamique différent.
<code>pages_settings_definitions.json</code>	Définitions des champs éditables de <code>pages_settings</code> .	Quand on ajoute un nouveau setting de page configurable.
<code>preferences_settings_definitions.json</code>	Définitions des préférences de module utilisables dans l'éditeur.	Quand une nouvelle préférence doit être sélectionnable/configurable.
<code>bills_settings_definitions.json</code>	Schéma des données configurables par type de bill : type, nombre de bills, dates, comptes, taxes, partner subtype, produit/service, règles spécifiques, etc.	Quand un nouveau type de bill ou un nouveau paramètre de génération doit être exposé à la configuration.
<code>bills_category_settings_definitions.json</code>	Schéma des options par catégorie de bills . Dans la version actuelle, ces definitions décrivent les options que l'éditeur peut présenter.	Quand une catégorie de bills reçoit une nouvelle option commune.
<code>accounts.json</code>	Plan comptable statique : comptes par défaut/by-sector/custom, numéro, nom, type, unique_identifier, ordre, balance par défaut et metadata cash-flow.	Quand on ajoute/modifie un compte ou son comportement comptable.
<code>account_types.json</code>	Définitions des types/catégories de comptes.	Quand la taxonomie des comptes change.
<code>taxes.json</code>	Taxes de vente et données associées chargées dans <code>SalesTaxes</code> .	Quand une taxe de vente ou sa configuration statique change.
<code>income_taxes.json</code>	Définitions générales des impôts sur le revenu.	Quand les types/règles générales d'impôt changent.

Fichier	Rôle	Quand le modifier
<code>employer_income_taxes.json</code>	Données de charges/taxes employeur.	Quand les règles employeur changent.
<code>income_tax_*_YEAR.json</code>	Tables fiscales annuelles fédérales/provinciales, chargées à la demande selon région et année.	Lors d'une mise à jour fiscale annuelle.
<code>income_taxes_codes.json</code>	Codes liés aux calculs/configurations d'impôt.	Quand la correspondance des codes fiscaux change.
<code>interests.json</code>	Définitions de taux/catégories d'intérêts.	Quand le calcul d'intérêts doit utiliser de nouvelles données statiques.
<code>immobilizations.json</code>	Catalogue/configuration des immobilisations.	Quand un type d'immobilisation ou ses données changent.
<code>contract_regularizations.json</code>	Données de régularisation des contrats.	Quand les règles de régularisation de contrats changent.
<code>regions.json</code>	Catalogue des régions supportées.	Quand une région est ajoutée/modifiée.
<code>regions_settings.json</code>	Comportements propres à une région : options fiscales/employeur/employé, congés et autres paramètres régionaux.	Quand une région doit avoir un comportement Accounting différent.
<code>act_settings.json</code>	Settings globaux d'activité utilisés par les modules.	Pour une configuration transversale à l'activité plutôt qu'à une page/module précis.
<code>constants.json</code>	Constantes globales ByModule utilisées par les systèmes runtime.	Quand un comportement dépend d'une constante partagée plutôt que d'un module particulier.
<code>dropdown_data.json</code>	Données de dropdown statiques utilisées dans les configurateurs.	Quand une liste de choix statique change.
<code>dynamic_dropdown_data_structure.json</code>	Structure des dropdowns dynamiques.	Quand une donnée sélectionnable dépend d'autres données/configurations.

3. Relation entre modules_settings et les definitions

flowchart TD

```
DEF[modules_settings_definitions.json] -->|liste des preferences|
```

```
PREF[preferences_settings_definitions.json]
```

```
DEF -->|liste des bills| BILLDEF[bills_settings_definitions.json]
```

```
BILLDEF --> WIZ[BillsDataWizardPageControl]
PREF --> WIZARD[Module settings wizard]

DEFAULT[modules_settings.json] --> MS[ModuleSettingsBase]
WIZ --> SAVED[Configuration du module/activité]
WIZARD --> SAVED
SAVED --> MS

MS --> PREFS[preferences]
MS --> BTD[bills_type_data]
MS --> BCS[bills_category_settings]
MS --> HIDDEN[bills_types_hidden_total]
```

À retenir : un fichier `*_definitions.json` ne représente pas forcément la valeur runtime. Il définit souvent *comment configurer* cette valeur. Les valeurs effectives d'un module finissent dans `ModuleSettingsBase`.

4. Configuration des bills dans un module

`ModuleSettingsBase` expose deux structures principales :

- `bills_type_data` → dictionnaire de `BillTypeData` / données détaillées de génération par bill;
- `bills_category_settings` → options communes par catégorie, par exemple le nombre de bills aléatoires à sélectionner dans une catégorie.

Le backend lit notamment `_moduleSettings.bills_category_settings` lors de la génération et utilise `bills_type_data` comme source réelle pour `BillFactory`.

bills_settings_definitions.json

Ce fichier est la grosse table de définition des champs disponibles pour les bills. On y retrouve notamment :

- `type`, `num_bills`, `min_num_entries`, `max_num_entries`;
- dates, fréquences, occurrences, délais et termes;
- comptes impactés : sales, expenses, payable, receivable, cash, loan, amortization, etc.;

- taxes : taxes de vente, déductions, taxes récupérables;
- relations : product/service, contract, immobilization, employee, supplier, investment;
- règles de création/édition et données spécifiques au type de bill.

5. Configuration des pages Accounting

flowchart LR

```

M[modules.json] --> UID[module uid]
UID --> MS[modules_settings.json]
MS --> ORDER[pages_order]
ORDER --> MP[module_pages.json]
MP --> STRUCT[Structure UI: page/table/input]
STRUCT --> PS[pages_settings.json]
PS --> BEHAVIOR[Rows/sections dynamiques, formules, navigation]

```

`module_pages.json` répond principalement à la question « **qu'est-ce qui existe dans la page ?** », tandis que `pages_settings.json` répond à « **comment cette page se comporte ?** ».

6. Plan comptable et calculs

flowchart LR

```

ACC[accounts.json] --> STATIC[StaticDataByModule.Accounts]
TYPES[account_types.json] --> STATIC
TAX[taxes / income_taxes / employer_income_taxes] --> STATIC
REG[regions + regions_settings] --> STATIC
STATIC --> BILL[Bills]
STATIC --> JOURNAL[General Journal]
STATIC --> LEDGER[General Ledger]
STATIC --> TB[Trial Balance]
STATIC --> FS[Financial Statements]

```

7. `regions_settings.json` : filtre régional critique

À ne pas oublier : ajouter une donnée dans son JSON principal ne suffit pas toujours à la rendre disponible dans le jeu. Pour plusieurs données Accounting, `regions_settings.json` agit comme un **filtre/routeur par région**.

Comptes

Pour les comptes, c'est particulièrement important. `UtilsByModule.GetRegionAccounts()` lit la région de l'activité, choisit un type via `global.accounts.type_by_regions`, puis prend uniquement les UIDs présents dans `global.accounts.data_by_type[type]`. Ensuite seulement, ces UIDs sont résolus dans `accounts.json`.

Donc un compte peut exister parfaitement dans `accounts.json` et **ne jamais apparaître** dans le jeu s'il n'est pas ajouté dans la liste appropriée de `regions_settings.json`. Après ce filtre régional, `GetUsedAccounts()` applique encore les `visibilityConditions` du compte et `preferences.hiddenAccounts`.

flowchart LR

```
A[accounts.json] --> ALL[Catalogue complet des comptes]
R[act_settings.region] --> RS[regions_settings.json]
RS --> LIST[Liste de comptes autorisés pour la région]
ALL --> FILTER[GetRegionAccounts]
LIST --> FILTER
FILTER --> VIS[visibilityConditions + hiddenAccounts]
VIS --> USED[Comptes réellement disponibles]
```

Ce que `regions_settings.json` pilote aussi

- **Taxes de vente** via `global.taxes` : par exemple GST/QST au Québec versus HST selon la province.
- **Retenues / impôts employés** via `global.income_taxes`.
- **Charges employeur** via `global.employer_income_taxes`.
- **Intérêts** via `global.interests`.
- **Colonnes de certaines pages/journaux** via `pages.<page_uid>.column_inputs_order`.
L'ordre et même les colonnes affichées peuvent changer selon la région.

- **Prefab de certaines pages/formulaires** via `pages.<page_uid>.prefab_reference`.
Exemple : le formulaire de paiement des taxes utilise un contrôle différent selon la province.

Pattern utilisé

La plupart de ces sections utilisent le même pattern :

```
{
  "type_by_regions": {
    "ca_on": "ca_on"
  },
  "data_by_type": {
    "default": [...],
    "ca_on": [...]
  }
}
```

Si la région courante n'est pas présente dans `type_by_regions`, le code retombe sur `default`.

Checklist quand on ajoute une donnée régionale

- Créer/modifier la donnée source : `accounts.json`, `taxes.json`, tables fiscales, etc.
- Vérifier `regions_settings.json` et ajouter l'UID dans chaque `data_by_type` où il doit être disponible.
- Si une nouvelle région/variante est nécessaire, ajouter le mapping dans `type_by_regions`.
- Pour une page régionale, vérifier aussi `column_inputs_order` ou `prefab_reference`.
- Tester au minimum la région `default` et une région explicitement mappée.

Exemple classique de bug : « j'ai ajouté le compte dans `accounts.json`, il charge sans erreur, mais il n'apparaît nulle part ». Le premier endroit à vérifier est `regions_settings.json → global.accounts`.

8. Règle pratique : quel fichier modifier ?

Je veux...	Commencer par...
Ajouter un nouveau module	<code>modules.json</code> + <code>modules_settings.json</code> + sa définition/settings/pages selon le besoin.
Changer la structure d'une page	<code>module_pages.json</code> .
Changer le comportement d'une page	<code>pages_settings.json</code> .
Ajouter une préférence configurable	<code>preferences_settings_definitions.json</code> puis consommation dans le code/settings.
Ajouter un paramètre de bill configurable	<code>bills_settings_definitions.json</code> puis le code serveur qui consomme le champ.
Changer combien/quels bills un module génère	Configuration effective <code>bills_type_data</code> / <code>bills_category_settings</code> du module.
Ajouter/modifier un compte	<code>accounts.json</code> .
Mettre à jour une table fiscale annuelle	<code>income_tax_*_YEAR.json</code> correspondant.
Changer une règle propre à une région	<code>regions_settings.json</code> .

Fichiers de code qui chargent cette configuration

- `Assets/Src/Game/Data/ByModule/StaticDataByModule.cs` — chargement central des JSON de `data_byModule`.
- `Assets/Src/Game/Data/ByModule/ModuleSettingsBase.cs` — désérialisation des settings runtime, incluant bills.
- `Assets/Src/UI/SimByModule/Data/ModuleSettingsDefinition.cs` — relie une définition de module aux définitions de préférences et de bills.
- `Assets/Src/UI/SimByModule/Data/BillsDataWizardPageControl.cs` — éditeur des données de bills.
- `Assets/Src/UI/SimByModule/Data/BillDataSetingDefinitionData.cs` — représentation Unity d'une définition de bill.
- `Assets/Src/UI/SimByModule/Data/BillCategorySettingDefinitionData.cs` — représentation Unity des options de catégorie.

Point important

Il n'existe pas dans cette branche de fichier statique `bills_settings.json` dans `Resources/data_byModule`. Les fichiers statiques présents sont `bills_settings_definitions.json` et `bills_category_settings_definitions.json`; les valeurs effectives des bills sont intégrées aux settings du module/activité sous `bills_type_data` et `bills_category_settings`.

Revision #2

Created 2026-09-17 21:23:49 UTC by Mathis

Updated 2026-09-17 21:48:19 UTC by Mathis