Lecture et écriture sur TimeTonic

Maintenant que nous pouvons communiquer avec l'API TimeTonic via le M5Stack, il faut savoir utiliser les principales requêtes permettant de lire, créer et modifier les données contenues dans un book (espace de travail). Voici les étapes principales, avec le détail des fonctions API à utiliser et des exemples de structure de requête.

---

1. Lire les données d’une smart table

Pour récupérer le contenu d’une smart table, il faut utiliser la requête **getTableValues**. Cette requête retourne les lignes (rows) et les valeurs des champs (fields) d’une table donnée.

Paramètres nécessaires :

- `req` : "getTableValues"
- `o_u` et `u_c` : identifiants utilisateur
- `sesskey` : clé de session API
- `b_c` : code du book (carnet)
- `catCode` : code de la table (catégorie)

Exemple de requête POST :

> ```
> req=getTableValues&o_u=USER_ID&u_c=USER_ID&sesskey=SESSION_KEY&b_c=BOOK_CODE&cat
> Code=TABLE_CODE
> ```

En version formatée pour le code C précédemment utilisé :

```
"req=getTableValues"
"&b_c=BOOK_CODE"
```

```
"&catCode=TABLE_CODE"
"&o_u=USER_ID"
"&u_c=USER_ID"
"&sesskey=SESSION_KEY"
```

La réponse contiendra un tableau JSON avec toutes les lignes et les valeurs des champs pour chaque ligne. Pour pouvoir exploiter ces données dans le code C, il faut utiliser un parseur, qui permet d'identifier et d'isoler chaque donnée lue.

---

2. Filtrage des données lues

Pour ne pas récupérer toutes les données de la table à chaque requête, on peut utiliser deux arguments dans celle-ci :

- `fieldIds` : permet de sélectionner les champs que l'on veut récupérer, sous forme d'un tableau JSON

Par exemple :

```
&fieldIds = [7457220, 7457221]
```

Où 7457220 et 7457221 sont les IDs des champs que je veux récupérer.

- `filterRowIds` : permet de sélectionner les lignes que l'on veut récupérer, ou bien au moyen d'une vue déjà existante dans la table, ou via une vue temporaire créée sous la forme d'un objet JSON.

Par exemple :

```
&filterRowIds={"applyViewFilters": 1181354}
```

Où 1181354 est l'ID de la vue que je souhaite utiliser. Pour récupérer cet ID, il faut **inspecter** la page lorsque vous êtes sur la vue qui vous intéresse, puis cliquer sur l'onglet "**network**". Dans les filtres en haut, sélectionner "**Fetch/XHR**" puis rafraichisser la page. Dans les fichiers qui se chargent alors, sélectionner **newDashboard?getUtilisateur** et défiler jusqu'à la section **Request Headers**. Dans celle-ci, trouvez la ligne **Referer**, elle contient en regard un lien terminant par /view/*Un numéro*. Ce numéro est l'ID de votre vue.

ATTENTION : le view ID peut maintenant être directement récupéré dans l'URL de la page : sélectionnez la vue souhaitée puis copiez le numéro après /view/ , il 'sagit du viewId.

Pour construire une vue avec un objet JSON, la manoeuvre est beaucoup plus compliquée : il faut remplacer l'ID de vue par un objet JSON avec une syntaxe particulière :

- `operator` : correspond à l'opérateur logique entre les différents filtres (voir les possibilitées dans la documentation TimeTonic)
- `id` : numéro arbitraire à donner à la vue (doit être unique), s'il s'agit d'une vue temporaire, utiliser "tmpId"
- `field_id` : ID du champ dans lequel on veut faire le filtrage
- predicate : opérateur logique du filtre (voir les possibilités dans la documentatio de TimeTonic)
- `operand` : corps du filtre (l'élément contre lequel seront comparées les valeurs)

Voici un exemple d'objet JSON à mettre après `"applyViewFilters":` :

```
{
  "filterGroup": {
    "operator": "and",
    "id": "tmpId",
    "filters": [
      {
        "field_id": 7457220,
        "json": {
          "predicate": "contains",
          "operand": "ABC"
        }
      },
      {
        "field_id": 7457221,
        "json": {
          "predicate": "is",
          "operand": "abc"
        }
      }
    ]
  }
}
```

NB : `"filterGroup"` (nom de l'objet JSON), `"filters"` (tableau JSON contenant les différents filtres) et `"json"`(objet JSON contenant les arguments de filtrage) sont des éléments statiques mais

nécessaires de la syntaxe. toute erreur dans la syntaxe entrainera une réponse valide mais ne contenant aucune donnée de la table interrogée (`"rowInfosLength": 0` dans la réponse).

---

3. Écrire des données dans une smart table

L'ajout ou la modification de données dans une smart table se fait avec la requête **createOrUpdateTableRow**. Pour modifier une ligne, il suffit d'indiquer l'identifiant de la ligne (`rowId=ROW_ID_EXISTANT` ou `rowId=tmpNEW_ROW`) et de fournir les nouveaux champs à modifier dans `fieldValues`.

Paramètres nécessaires :

- `req` : "createOrUpdateTableRow"
- `o_u` et `u_c` : identifiants utilisateur
- `sesskey` : clé de session API
- `b_c` : code du book (carnet)
- `catCode` : code de la table (catégorie)
- `fieldValues` : données à écrire. celles-ci doivent être sous la forme d'un objet JSON avec la syntaxe suivante :

```
{
"field ID": "Value",
"field ID 2": "Value 2",
"field ID 3": "Value 3"
}
```

Exemple de requête POST:

```
req=createOrUpdateTableRow&o_u=USER_ID&u_c=USER_ID&sesskey=SESSION_KEY&b_c=BOOK_CODE&catCode=TABLE_CODE&rowId=ROW_ID_EXISTANT&fieldValues={"FIELD_ID1":"Nouvelle Valeur"}
```

En version formatée pour le code C (bien noter les antislashs avant les guillemets contenus dans fieldValues : ils permettent de les considérer comme des caractères normaux sans interférer avec les balises de Strings):

```
"req=createOrUpdateTableRow"
"&o_u=USER_ID"
"&u_c=USER_ID"
"&sesskey=SESSION_KEY"
"&b_c=BOOK_CODE"
"&catCode=TABLE_CODE"
"&rowId=ROW_ID"
"&fieldValues={\"FIELD_ID\":\"NouvelleValeur\"}"
```

Seuls les champs à modifier doivent être précisés dans `fieldValues`.

---

4. Récupérer les identifiants nécessaires

Si les données que vous lisez/écrivez sont toujours au même endroit, vous pouvez manuellement regarder les IDs de chaque champ et les copier dans votre code. Cependant, si vous voulez écrire des données à des endroits dépendant de certains champs/valeurs, vous pouvez obtenir les IDs pertinents via les requêtes **getAllBooks** (pour les books) et **getBookTables** (pour les tables et champs). Avant de pouvoir lire ou écrire, il est indispensable de connaître :

- Le code du book (`b_c`)
- Le code de la table (`catCode`)
- Les identifiants des champs (`fieldID1`, `fieldID2`, etc.)
- L'identifiant de la ligne, si modification (`rowId`)

---

5. Résumé des étapes

1. **Authentification** : Obtenir un `sesskey` via l'API.

2. **Lister les tables** : Utiliser `getAllBooks` puis `getBookTables` pour identifier le book et la table cible si nécessaire.
3. **Lire des données** : Utiliser `getTableValues` pour obtenir les lignes et champs.
4. **Écrire des données** : Utiliser `createOrUpdateTableRow` avec `rowId=tmpNEW_ROW` si vous voulez créer une nouvelle ligne.

---

6. Bonnes pratiques

- Utiliser systématiquement la méthode POST pour transmettre les paramètres.
- Manipuler les objets JSON avec soin pour le paramètre `fieldValues`, et utiliser un parseur pour pouvoir utiliser les données lues.
- Toujours vérifier le champ `status` dans la réponse de l'API pour s'assurer du succès de l'opération (le code de succès 200 indique simplement que la requête est parvenue à l'API, pas nécessairement qu'elle a été correctement exécutée).

En résumé, la lecture et l'écriture dans une smart table TimeTonic via l'API sont simples, mais nécessites une grande attention aux détails : il faut bien identifier les bons paramètres (book, table, champs), utiliser les requêtes appropriées, et les structurer correctement afin de traiter ou d'envoyer les données correctement.

Revision #6
Created 24 June 2025 08:30:44 by Eyglier De Rooster Mathieu
Updated 21 July 2025 14:22:31 by Eyglier De Rooster Mathieu