



New version of Custom Settings in Access – now even more dynamic, simpler, and smarter!

Simple. Fast. Extensible. Intuitive!

Introducing the new version (V2) of dynamic Custom Settings – a **smart solution** that shines with full IntelliSense support, intuitive operation, and flexible extensibility. All without additional coding effort!

What's new?

The revised solution now offers:

- ☒ **IntelliSense** in all settings calls
- ☒ Enum-based operation for maximum **clarity and accuracy**
- ☒ **Easy creation of custom settings** – without a single line of code!
- ☒ Read, Create, Change, and Delete – all integrated
- ☒ Frontend, backend, and user settings – directly supported
- ☒ **Individual user settings** – each user can save their own values
- ☒ New tables? No problem – can be integrated with minimal effort

Individual user settings

Want each **user** to save their **own preferences**? No problem. The framework supports user-based settings via the tblUser table, e.g.: last view opened, user color, default search filter. - Simply enter the user ID when calling up – done!

How does it work?

The system is based on tables and three powerful modules:

The tables

- **tblFrontend** Settings for the front end (client-specific)
- **tblBackend** Global settings (for all users)
- **tblUser** Individual user settings

modHelper_Settings

This module contains all functions for processing settings:

- **settingCreate:** Creates new entries
- **settingRead:** Reads settings securely depending on the data type
- **settingChange:** Changes existing settings in a type-safe manner
- **settingDelete:** Removes specific entries

modHelper_Enumeration

This module automatically handles enums, which provide IntelliSense. Here, new enum entries can be **added, renamed, and deleted** via code—even **dynamically at runtime**.

modHelper_SQL

A proven module from a previous article—provides secure, type-compliant SQL strings with `cSQL()`.

Data storage & type conversion

A special highlight of the system is the uniform storage of all values: All settings are stored internally as strings. When retrieved with **settingRead**, a type conversion to the originally defined data type takes place automatically. This data type information is also stored in the **SettingType** column and ensures that the return value is always correctly typed.

IntelliSense for settings?!

Yes! - Thanks to the use of enums (`EnumSettingName`, `EnumTable`, `EnumDataType`), IntelliSense support is directly available for every call, e.g., `settingRead(...)`, without the need for lookup or the risk of typos. You only need to specify the enum value – the rest is automatically assigned correctly.

Example: It's that simple:

Creating a new setting in the frontend table:

Here, the setting name must be passed as a string, and the `settingCreate` function then creates an entry as an enum. This allows IntelliSense to be used later.

```
settingCreate "Theme", dtString, "Dark", tblFrontend  
settingCreate "Items", dtInteger, 22, tblFrontend
```

Changing a setting in the frontend table:

```
settingChange Theme, dtString, "Light", tblFrontend  
settingChange Items, dtLong, 444444, tblFrontend
```

Reading a setting from the frontend table:

```
Debug.Print settingRead(Theme, tblFrontend)  
Debug.Print settingRead(Items, tblFrontend)
```

Deleting a setting in the frontend table:

```
settingDelete(Theme, tblFrontend)  
settingDelete(Items, tblFrontend)
```

User table:

```
settingCreate "Active", dtBoolean, True, tblUser, 1  
settingChange Active, dtBoolean, False, tblUser, 1  
Debug.Print settingRead(Active, tblUser, 1)  
settingDelete(Active, tblUser, 1)
```

+ Add new tables? - It's that easy:

1. Create a new table: e.g. tblInvoiceSettings (simply copy an existing table)
2. Add an enum entry to enumTable
3. Adjust the GetTableName() function

All functions (Create, Read, Change, Delete) work without any additional code. This means you can expand your system in a modular fashion without having to modify the core code.

Advantages


- ✓ Ready to use in no time
- ✓ No more hard-coded SQL
- ✓ IntelliSense prevents errors in names and types
- ✓ Settings depending on the area of application (front end, back end, user)

Are there any disadvantages?

- ♦ Important: If an enum is deleted but still used in the code (e.g., in **settingRead**, **settingChange**, or **settingDelete**), this will cause an error during compilation! However, this is a good safety mechanism, as it makes incorrect or outdated settings calls immediately visible and correctable.

Conclusion

This new version of Custom Settings with IntelliSense is a huge step toward professional, maintainable, and modern Access development. No cryptic SQLs. No errors due to names. Once integrated, you'll never want to do without it again!

Do you develop a lot with Access? **Then give it a try**—I look forward to your feedback or ideas for additional features! 



🚀 Neue Version der Custom Settings in Access – jetzt noch dynamischer, einfacher & intelligenter!

Einfach. Schnell. Erweiterbar. Intuitiv!

Ich präsentiere die neue Version (V2) der dynamischen Custom Settings – eine **smarte Lösung**, die mit voller IntelliSense-Unterstützung, intuitiver Bedienung und flexibler Erweiterbarkeit glänzt. Alles ohne zusätzlichen Code-Aufwand!

💡 Was ist neu?

Die überarbeitete Lösung bietet nun:

- ✅ **IntelliSense** in allen Settings-Aufrufen
- ✅ Enum-basierte Bedienung für höchste Klarheit & **Tippfehlerfreiheit**
- ✅ **Einfache Erstellung eigener Settings** – ohne eine einzige Codezeile!
- ✅ Read, Create, Change und Delete – alles integriert
- ✅ Frontend-, Backend- und User-Settings – direkt unterstützt
- ✅ **Individuelle User-Settings** – jeder User kann eigene Werte speichern
- ✅ Neue Tabellen? Kein Problem – mit minimalem Aufwand integrierbar

👤 Individuelle User-Einstellungen

Du möchtest, dass jeder **User eigene Voreinstellungen** speichert? Kein Problem. Das Framework unterstützt User-basierte Settings über die Tabelle tblUser, z. B.: letzte geöffnete Ansicht, Benutzerfarbe, Standard-Suchfilter. - Einfach beim Aufruf die UserID mitgeben – fertig!

🧠 Wie funktioniert's?

Die Grundlage des Systems bilden die Tabellen und drei mächtige Module:

Die Tabellen

- **tblFrontend** Einstellungen für das Frontend (Client-spezifisch)
- **tblBackend** Globale Einstellungen (für alle Nutzer)
- **tblUser** Individuelle User Settings

modHelper_Settings

Dieses Modul enthält alle Funktionen zur Verarbeitung von Einstellungen:

- **settingCreate:** Erstellt neue Einträge
- **settingRead:** Liest Settings je nach Datentyp sicher aus
- **settingChange:** Ändert vorhandene Einstellungen typensicher
- **settingDelete:** Entfernt gezielt Einträge

modHelper_Enumeration

Dieses Modul sorgt für den **automatischen Umgang mit Enums**, welche für IntelliSense sorgen. Hier können per Code neue Enum-Einträge **Hinzugefügt, Umbenannt sowie Gelöscht** werden – und das sogar **dynamisch zur Laufzeit**.

modHelper_SQL

Ein bewährtes Modul aus einem früheren Artikel – sorgt mit `cSQL()` für sichere, typkonforme SQL-Strings.

Datenspeicherung & Typumwandlung

Ein besonderes Highlight des Systems ist die einheitliche Speicherung aller Werte: Alle Einstellungen werden intern als **String** gespeichert. Beim Abrufen mit **settingRead** erfolgt automatisch eine Typkonvertierung in den ursprünglich definierten Datentyp. Diese Daten-Typ-Information wird in der Spalte **SettingType** mitgespeichert und sorgt dafür, dass der Rückgabewert immer korrekt typisiert ist.

IntelliSense für Settings?!

Ja! - Dank der Verwendung von Enums (`EnumSettingName`, `EnumTable`, `EnumDataType`) steht für jeden Aufruf z. B. `settingRead(...)` die IntelliSense-Unterstützung direkt zur Verfügung – ganz ohne Nachschlagen oder Risiko von Tippfehlern.

Du gibst nur den Enum-Wert an – der Rest wird automatisch korrekt zugeordnet.

Beispiel: So einfach ist's:

Erstellen eines neuen Settings in der Frontend Tabelle:

Hier muss die Settingbezeichnung als String übergeben werden, die Funktion `settingCreate` erstellt dann einen Eintrag als Enum. Damit später die Intelisense verwendet werden kann.

```
settingCreate "Theme", dtString, "Dark", tblFrontend  
settingCreate "Items", dtInteger, 22, tblFrontend
```

Ändern eines Settings in der Frontend Tabelle:

```
settingChange Theme, dtString, "Light", tblFrontend  
settingChange Items, dtLong, 444444, tblFrontend
```

Lesen eines Settings aus der Frontend Tabelle:

```
Debug.Print settingRead(Theme, tblFrontend)  
Debug.Print settingRead(Items, tblFrontend)
```

Löschen eines Settings in der Frontend Tabelle:

```
settingDelete(Theme, tblFrontend)  
settingDelete(Items, tblFrontend)
```

User Tabelle:

```
settingCreate "Active", dtBoolean, True, tblUser, 1  
settingChange Active, dtBoolean, False, tblUser, 1  
Debug.Print settingRead(Active, tblUser, 1)  
settingDelete(Active, tblUser, 1)
```

+ Neue Tabellen hinzufügen? - So einfach geht's:

1. Neue Tabelle anlegen: z. B. tblInvoiceSettings (Einfach eine vorhanden Tabelle kopieren)
2. Enum-Eintrag in enumTable hinzufügen
3. Funktion GetTableName() anpassen

Alle Funktionen (Create, Read, Change, Delete) funktionieren ohne weiteren Code. Du kannst dein System also modular erweitern, ganz ohne Eingriff in den Kerncode.

Vorteile

- ✓ Blitzschnell einsatzbereit
- ✓ Kein hartcodiertes SQL mehr
- ✓ IntelliSense verhindert Fehler bei Namen und Typen
- ✓ Settings je nach Anwendungsbereich (Frontend, Backend, User)

Gibt es Nachteile?

♦ **Wichtig:** Wird ein Enum gelöscht, aber im Code weiterhin verwendet (z. B. in `settingRead`, `settingChange` oder `settingDelete`), führt dies beim Kompilieren zu einem **Fehler!** Dies ist jedoch ein guter Sicherheitsmechanismus, da dadurch fehlerhafte oder veraltete Settings-Aufrufe direkt sichtbar und korrigierbar sind.

Fazit

Diese neue Version der Custom Settings mit IntelliSense ist ein riesiger Schritt in Richtung professionelle, wartbare und moderne Access-Entwicklung. Keine kryptischen SQLs. Keine Fehler durch Namen. Einmal integriert – nie wieder darauf verzichten!

Du entwickelst viel mit Access? **Dann probier's aus** – ich freu mich auf dein Feedback oder Ideen für weitere Funktionen! 