



Introduction

MS Access databases are known for their flexibility and customisability. However, the question often arises as to the best way to save user-defined settings. Common methods such as external INI files or the Windows registry have their advantages and disadvantages. In this article, I present an elegant and efficient solution for saving, retrieving, changing and deleting user-defined settings directly in Access tables. Forget complicated recordset queries! With these simple VBA functions, which even support IntelliSense, you can manage settings with just one line of code. The example presented here saves settings in the frontend and backend. However, it can also be easily extended to other tables. For example, a user settings table that is saved in the user profile.

Why this approach?

- Simplicity: Short, intuitive commands make work easier. No complex recordset operations required.
- IntelliSense support: Faster and error-free code entry.
- Efficiency: Fast reading and writing of settings.

What is required?

- tblSettingsFE, tblSettingsBE.
- modSettings
- clsSettings

Tables:

The respective tables must be created as follows:

	Feldname	Felddatentyp
tblSettingsFE	set_name	Kurzer Text
	set_value	Kurzer Text
	set_type	Zahl

Module modSettings:

```
' DECLARATIONS -----
Private Const settingsBE As String = 'tblSettingsBE' ' table for the backend settings
Private Const settingsFE As String = 'tblSettingsFE' ' table for the frontend settings
' -----

' ENUMERATIONS -----
'
' Put all setting names in here (intelisense)
Public Enum eSettings
    'Frontend Settings (1 - 99)
    f_BackendPath = 1
    f_Version = 2
    f_Programmer = 3
    ...
    'Backend Settings (100 - XXX)
    b_BackupPath = 100
    b_BackupDays = 101
    b_ProjectName = 102
    ...
End Enum
```

The settings names are entered in the eSettings enumeration. This is necessary for the Intelisense to work. It is not necessary to enter these names in the tables as well. This is done by the newSetting() function. The advantage of this is that you can create new settings from the VBE without having to enter them manually in the BE. Especially if you do not have access to the backend of a customer.

The setting values are saved as a string, and the field data type is also specified when creating newSetting(). When the setting is retrieved, the string is then converted to the correct type based on the field data type. The assigned number in the enumeration can be used to control which table the setting refers to. There is a help function that realises this. In this example, we use the numbers up to 99 for the frontend and everything above that for the backend. This must also be reflected in the help function. This would also be the approach for integrating further setting tables.

Create a new setting in the frontend:

```
newSetting f_BackendPath, 'D:\backend, eString
```

Change a setting in the frontend:

```
setSetting f_BackendPath, 'E:\backend
```

Read a setting from the backend:

```
Debug.Print getSetting(b_BackupPath)
```

Delete a setting from the backend:

```
delSetting(b_ProjectName)
```

A detailed description of the functions can be found as comments in the code.

<https://www.dropbox.com/scl/fo/z1t153511gnia1hy00kj3/ACm-9qxbEbmRHTqjqbaM-A?rlkey=lgonc4j03h9766sqwpeb8toqj&st=pjl1tx4b&dl=0>



Einleitung

MS Access-Datenbanken sind bekannt für ihre Flexibilität und Anpassungsfähigkeit. Doch oft stellt sich die Frage, wie man benutzerdefinierte Einstellungen am besten speichert. Klassische Methoden wie externe INI-Dateien oder die Windows-Registrierung haben ihre Vor- und Nachteile. In diesem Artikel stelle ich eine elegante und effiziente Lösung vor, um benutzerdefinierte Einstellungen direkt in Access-Tabellen zu speichern, abzurufen, zu ändern und zu löschen. Vergessen Sie umständliche Recordset-Abfragen! Mit diesen einfachen VBA-Funktionen, die sogar IntelliSense unterstützen, können Sie Einstellungen mit nur einer Zeile Code verwalten. Das hier vorgestellte Beispiel speichert jeweils Einstellungen im Frontend und Backend. Kann jedoch auch einfach auf weitere Tabellen erweitert werden. Zum Beispiel einer User-Settings-Tabelle die im Userprofil gespeichert ist.

Warum dieser Ansatz?

- Einfachheit: Kurze, intuitive Befehle erleichtern die Arbeit. Keine komplexen Recordset-Operationen erforderlich.
- IntelliSense-Unterstützung: Schnellere und fehlerfreie Codeeingabe.
- Effizienz: Schnelles Lesen und Schreiben von Einstellungen.

Was wird benötigt?

- tblSettingsFE, tblSettingsBE.
- modSettings
- clsSettings

Tabellen:

Die jeweiligen Tabellen müssen folgendermaßen erstellt werden:

	Feldname	Felddatentyp
	set_name	Kurzer Text
	set_value	Kurzer Text
	set_type	Zahl

Modul modSettings:

```
' DECLARATIONS -----
Private Const settingsBE As String = "tblSettingsBE" ' table for the backend settings
Private Const settingsFE As String = "tblSettingsFE" ' table for the frontend settings
' -----

' ENUMERATIONS -----
'
' Put all setting names in here (intelisense)
Public Enum eSettings
    'Frontend Settings (1 - 99)
    f_BackendPath = 1
    f_Version = 2
    f_Programmer = 3
    ...
    'Backend Settings (100 - XXX)
    b_BackupPath = 100
    b_BackupDays = 101
    b_ProjectName = 102
    ...
End Enum
```

Die Settings-Bezeichnungen werden in die Enumeration eSettings eingetragen. Dies ist notwendig, damit die Intelisense funktioniert. Es ist nicht notwendig diese Bezeichnungen ebenfalls in die Tabellen einzutragen. Dies wird durch die newSetting() Funktion gemacht. Vorteil, Ihr könnt so neue Einstellungen aus der VBE heraus erstellen ohne sie händisch im BE einzutragen. Vor allem wenn man kein Zugriff auf das Backend von einem Kunden hat.

Die Settingswerte werden als String gespeichert, Zudem wird der Felddatentyp beim erstellen newSetting() mit angegeben. Beim abrufen der Einstellung wird dann anhand des Felddatentypes der String in den richtigen Typ konvertiert.

Anhand der zugewiesenen Nummer in der Enumeration kann man steuern auf welche Tabelle sich die Einstellung bezieht. Hierfür gibt es eine Hilfsfunktion die das erkennt. In diesem Beispiel verwenden wir die Nummern bis 99 für das Frontend und alles darüber für das Backend. Dies muss sich auch in der Hilfsfunktion widerspiegeln. Hier wäre auch der Ansatz um weitere Settingstabellen einzubinden.

Eine neue Einstellung im Frontend erstellen:

```
newSetting f_BackendPath, "D:\backend, eString
```

Eine Einstellung im Frontend ändern:

```
setSetting f_BackendPath, "E:\backend
```

Eine Einstellung aus dem Backend auslesen:

```
Debug.Print getSetting(b_BackupPath)
```

Eine Einstellung aus dem Backend löschen:

```
delSetting(b_ProjectName)
```

Eine detaillierte Beschreibung der Funktionen findet ihr als Bemerkungen im Code.

<https://www.dropbox.com/scl/fo/z1t153511gnia1hy00kj3/ACm-9qxbEbmRHTqjqbaM-A?rlkey=lgonc4j03h9766sqwpeb8toqj&st=pjl1tx4b&dl=0>