Definitionen bearbeiten

- Definitionen anlegen, speichern und löschen
- Reiter „Allgemein“
- Reiter „Spalten“
- Reiter „JSON“
- Prüfungen und Meldungen

Definitionen anlegen, speichern und löschen

Die Seitenleiste

Links stehen alle Definitionen, die der geöffnete Ordner zeigt, gruppiert nach **Tabellen** und **Views**. Jeder Eintrag zeigt unter dem Namen die SQL-Tabelle, die Verbindung und – bei einer selbst angelegten – den Zusatz *eigene*.

- **Filtern …** sucht in Name, Tabellenname und Verbindung.
- Die Auswahl daneben schränkt auf *nur Tabellen* oder *nur Views* ein.
- Ein ⚠ hinter dem Namen heißt: das gespeicherte JSON dieser Definition lässt sich nicht lesen. Beim Öffnen erscheint dann der Dialog *Definition nicht lesbar* mit dem Rohtext und der Fehlermeldung – dort reparieren und **Übernehmen**; geschrieben wird erst beim Speichern.
- Die Fußzeile zählt, wie viele Einträge der Filter übrig lässt.

Aktualisieren in der Kopfleiste holt die Übersicht neu vom Server.

Neu

`Neu` legt eine Definition zu einer **vorhandenen** Tabelle an. Gibt es die Tabelle noch nicht, ist SQL Tabelle anlegen der Schritt davor.

Feld	Bedeutung
Art	*Tabelle (lesen und schreiben)* oder *View (nur lesen)*. Bestimmt Name und Typ der Custom Property: `TS_Definition_*` oder `TS_ViewDefinition_*`
Name	der öffentliche Name; er steht in der Route und im Namen der Custom Property. Buchstaben, Ziffern und `_`, beginnend mit einem Buchstaben
Verbindung	DEXPRO-Datenbankverbindung aus der DEXPRO-Lösung
Tabelle	Tippen sucht in der Tabellenliste der Verbindung, die Treffer stehen darunter. Übernommen wird nur ein **gewählter** Eintrag, nicht der getippte Text

Ein Name, den es in dieser Art schon gibt, wird sofort abgewiesen – nicht erst beim Speichern, wo die Spalten schon angelegt wären.

Nach `Anlegen` steht die neue Definition im Reiter **Spalten**, noch ohne eine einzige Spalte. Der übliche nächste Griff ist dort **+ aus Tabelle …**.

Speichern

`Speichern` schreibt die Definition als Custom Property und leert den Cache des TableService. Die Statuszeile nennt den Namen und die Länge des geschriebenen JSON.

- **Umbenennen**: ein geänderter Name verschiebt die Custom Property.
- **Namenskonflikt**: gibt es den Namen schon, kommt die Rückfrage *Name schon vergeben*. `Überschreiben` ersetzt den Inhalt. Wer zurückkönnen will, exportiert vorher.
- **Hinweise des Servers**: beim Schreiben räumt das Portalskript auf (leere Spalteneinträge, `schema`/`catalog`) und meldet Ungereimtheiten, die es nicht abweist. Sie stehen danach im Meldungsblock über dem Formular.
- **Spur einer Änderung**: wird inhaltlich wirklich etwas geändert, bekommt die Definition die Kennung `isCustomChanged`. Sie befindet sich dann nicht mehr im Zustand der Auslieferung, und der Editor sagt das beim nächsten Öffnen. Ein Speichern ohne inhaltliche Änderung setzt die Kennung nicht.

Solange etwas ungespeichert ist, steht oben das Abzeichen **nicht gespeichert**. `Verwerfen` setzt auf den zuletzt gespeicherten Stand zurück. Jeder Weg, der die geöffnete Definition verlassen würde, fragt vorher nach.

Mehr ? Als Kopie speichern …

Schreibt den aktuellen Stand unter einem neuen Namen in eine **neue** Custom Property. Die geöffnete Definition bleibt, wie sie ist; danach ist die Kopie geöffnet.

Mehr ? Löschen

Löscht die Custom Property der Definition. Danach endet jeder Aufruf von `/<Art>/<Name>` mit **404**.

Die Definitionsdatei auf der Platte bleibt dabei unangetastet: ein erneuter Lauf von `DEXPRO__CreateCustomPropertiesFromAllTableServices` legt eine ausgelieferte Definition also wieder an. Für eine eigene gilt das nicht – sie ist weg.

Mehr ? Exportieren

Lädt die geöffnete Definition als Datei `TS_Definition_<Name>.json` bzw. `TS_ViewDefinition_<Name>.json` herunter. Siehe auch JSON importieren und exportieren.

Reiter „Allgemein“

Der Reiter beschreibt die Definition als Ganzes: woher ihre Daten kommen und wie die Abfrage aussieht, die der TableServiceRouter daraus baut.

Die Felder

Feld	Eigenschaft	Bedeutung
Name	`nameApi`	Der öffentliche Name. Er steht in der Route `/<Art>/<Name>` und im Namen der Custom Property. Ein Umbenennen verschiebt die Property. Pflichtfeld
Verbindung	`connectionId`	Die DEXPRO-Datenbankverbindung aus der `dbConn.json`. Angeboten werden die gemeldeten Verbindungen; über *andere …* lässt sich eine eintragen, die (noch) nicht gemeldet wird. Pflichtfeld
Tabelle	`nameSQL`	Die SQL-Tabelle oder View hinter der Definition. Ist die Tabellenliste der Verbindung geladen, schlägt das Feld beim Tippen vor. Pflichtfeld
Standardsortierung	`orderBy`	Eine oder mehrere Zeilen aus Spalte und Richtung (*aufsteigend* / *absteigend*). Zur Wahl stehen die **deklarierten** Spalten der Definition. Die Sortierung gilt, solange der Client nicht selbst sortiert; eine Sortierung des Clients gewinnt pro Spalte

Erweitert

Feld	Eigenschaft	Bedeutung
Zusätzliche WHERE-Bedingung	`whereRaw`	Wird an jede Abfrage dieser Definition angehängt, z. B. `Active=1`. **Wird nicht geprüft** – ein Syntaxfehler legt die Tabelle lahm
nur unterschiedliche Zeilen	`distinct`	Setzt `DISTINCT` in die Abfrage

Feld	Eigenschaft	Bedeutung
meta (JSON)	`meta`	Wird unverändert an den Client durchgereicht – reine Anzeigehinweise
Weitere Eigenschaften (JSON)	–	Alles, wofür es hier kein Feld gibt. Es bleibt genau so erhalten, wie es in der Definition steht

> Der Editor verliert nichts. Jede Eigenschaft, für die es kein eigenes Feld gibt – auch `cache` –, erscheint unter *Weitere Eigenschaften (JSON)* und wird unverändert zurückgeschrieben.

Tabellenliste laden

Solange die Tabellen der Verbindung nicht gelesen sind, macht das Feld **Tabelle** keine Vorschläge; der Hinweis am Feld sagt das. Die Liste holt der Editor beim Verbindungswechsel sowie über die Knöpfe `Tabellen…` an den Stellen, an denen eine Tabelle zu wählen ist. Views bleiben dabei nur dort draußen, wo sie nicht in Frage kommen (z. B. beim Ändern einer Tabelle).

Meldet die Verbindung keine Tabellen oder lässt sich die Liste nicht lesen, sagt der Editor das – der Name ist dann von Hand einzutragen.

Reiter „Spalten“

Die Spaltenliste ist die ganze Schnittstelle einer Definition: **eine Spalte, die hier nicht steht, wird nicht geliefert**, lässt sich nicht filtern, sortieren oder durchsuchen und wird beim Anlegen und Ändern abgewiesen – gleich, was in der SQL-Tabelle steht.

Links die Liste der deklarierten Spalten, rechts das Formular zur ausgewählten.

Die Leiste über der Liste

Knopf	Wirkung
+ Spalte	hängt eine leere Spalte vom Typ `text` an
+ aus Tabelle …	übernimmt Spalten aus dem gelesenen Tabellenaufbau (siehe unten)
↑ / ↓	verschiebt die ausgewählte Spalte; die Reihenfolge ist die der Anzeige
Kopie	legt eine Kopie mit dem Zusatz `_Kopie` direkt darunter an
Entfernen	nimmt die Spalte aus der Definition – die SQL-Tabelle bleibt unberührt

In der Liste steht hinter dem Namen der `meta.type`. Ein kennzeichnet den Primärschlüssel. Eine hervorgehobene Zeile heißt: diese Spalte gibt es in der SQL-Tabelle nicht – der Router lässt sie aus jeder Antwort weg.

Ein **leerer Eintrag** entsteht durch ein Loch in der JSON-Liste (`[ {…}, , {…} ]`). Der Router übergeht ihn, das Speichern entfernt ihn.

+ aus Tabelle …

Zeigt alle Spalten der SQL-Tabelle, die noch nicht deklariert sind, mit SQL-Typ, Schlüssel- und Pflichtkennzeichen. Angehakte werden übernommen, und der Editor schlägt dabei gleich etwas vor:

- Primärschlüssel → `meta.type = key`, nicht sichtbar
- Spalte `Licence` → `meta.type = licence`, nicht sichtbar
- `bit` → `checkbox`, `int` → `int`, `decimal` → `number`, `date` → `date`, `datetime` → `datetime`, alles andere → `text` mit Suchfeld

- Datums- und Zeitspalten mit Standardwert sowie hochgezählte Spalten werden beim Anlegen nicht änderbar
- `Principal` und `CompanyCode` bekommen die **Firmenliste** aus der Tabelle `principal` als Werteliste vorgeschlagen – bei `CompanyCode` mit dem Standardwert `ANY`. Liegt die Definition nicht selbst in `DEX_MasterData`, wird die Verbindung dorthin gleich mit eingetragen

Alle Vorschläge lassen sich danach ändern.

Das Formular einer Spalte

Ist der Tabellenaufbau gelesen, steht oben eine Zeile dazu, was die Datenbank über diese Spalte sagt: Typ und Länge, Primärschlüssel, hochgezählt, NULL oder NOT NULL, Standardwert und ob sie beim Anlegen Pflicht ist.

Feld	Eigenschaft	Bedeutung
Name	`name`	Der SQL-Spaltenname. Er ist der Schlüssel in den Zeilen und beim Schreiben. Pflichtfeld
Beschriftung	`label`	Fehlt sie, zeigt der Client den Namen
SQL-Typ *(nur View)*	`type`	Views werden nicht ausgelesen – der Typ kommt von hier. Ohne ihn meldet die API `string`
Typ	`meta.type`	`text`, `int`, `number`, `date`, `datetime`, `checkbox`, `list`, `multiplelist`, `combobox`, `comboboxText`, `popup`, `iconorlist`, `key`, `licence`, `hidden`. `key` und `licence` sind die technischen Spalten, `hidden` lässt die Spalte in der API, aber nicht im Formular
Sichtbar	`meta.show`	*nein* blendet die Spalte in der Liste aus
Standardwert	`meta.defaultValue`	Wird beim Anlegen vorbelegt. Als Text, Zahl, *ja*, *nein* oder *leer (null)*
Suchfeld	`meta.searchAutocomplete`	Nimmt die Spalte in die Suche der Autovervollständigung auf
Werteliste	–	Woher die Auswahl kommt (siehe unten)

Werteliste

Vier Wege, die einander ausschließen. Der Client wertet sie in dieser Reihenfolge aus: `possibleValues` vor `meta.listScript` vor `meta.listItems`.

Auswahl	Was passiert
keine	freies Feld
feste Liste	Die Einträge stehen als `meta.listItems` in der Definition selbst – der Weg, den der DEX-Client für eine feste Auswahl kennt. Je Eintrag `key` (gespeicherter Wert) und `label` (angezeigter Text). Einträge ohne beides lässt der Client weg
aus einem Skript	`meta.listScript` nennt ein Portalskript; der Client ruft es über `dex_callScript` auf und erwartet eine JSON-Zeichenkette mit `{ value, label }`. Darunter stehen die `meta.listScriptParams` als Name-Wert-Liste – DOCUMENTS übergibt sie als Text
aus anderer Tabelle	Nachgeschlagen in einer Tabelle, geliefert unter `/table/<Definition>/col/<Spalte>/possibleValues`
feste Liste in `possibleValues`	Erscheint nur, wenn eine Definition sie schon mitbringt. Die API liefert sie, der DEX-Client zeigt sie **nicht**. Der Knopf *In feste Liste übernehmen* macht `meta.listItems` daraus

Für *aus anderer Tabelle*:

Feld	Bedeutung
Verbindung der Liste	leer lassen, wenn die Tabelle in derselben Datenbank liegt wie die Definition
Tabelle der Liste	Pflichtfeld – fehlt sie, endet der Aufruf mit 404
Wertespalte	die Spalte, deren Werte angeboten werden. Pflichtfeld
Anzeige	Vorlage für den angezeigten Text, `{{Spaltenname}}` wird je Zeile ersetzt
Schema / Katalog *(erweitert)*	MySQL: `schema` = Datenbankname, `catalog` = `def`. MS SQL: `schema` = `dbo`, `catalog` = Datenbankname
WHERE-Bedingung der Liste *(erweitert)*	schränkt die Liste ein
Wertespalte darf der Client wählen *(erweitert)*	`valueColumnByClient`

Übersetzungsfilter

`meta.translationFilter` sagt, wo die Übersetzungen der Liste stehen: **Application**, **PropType** und **PropTypeSubCategory** bilden einen festen Filter auf die Übersetzungstabelle. Aneinandergehängt sind sie zugleich der Namensraum, unter dem der Client die geladene Liste behält – zwei Spalten mit demselben Filter teilen sie sich. Ohne Filter bleiben die Beschriftungen unübersetzt.

Der Filter gilt für alle drei Listenarten: feste Einträge, Einträge aus einem Skript und die Beschriftungen einer nachgeschlagenen Liste.

Erweitert

Feld	Eigenschaft	Bedeutung
Pflichtfeld	`mandatory`	Vorgabe entscheidet die Datenbank (NOT NULL, kein Standardwert, keine Identity)
Beim Anlegen änderbar	`editableOnInsert`	*nein* blendet das Feld schreibgeschützt ein; Pflichtspalten bleiben änderbar
Beim Ändern änderbar	`editableOnUpdate`	Primärschlüssel sind *nein*, solange hier nichts anderes steht
meta.label	`meta.label`	überschreibt die Beschriftung im DEX-Tabellenclient
Breite	`meta.layout`	Breite des Eingabefelds im Raster, 1 bis 12
Änderbar / Pflicht im Client / „ANY“ anbieten	`meta.editable`, `meta.mandatory`, `meta.addAny`	Verhalten im Client
Listenformat	`meta.listFormat`	z. B. `%value% (%label%)`
Unbekannten Wert zeigen	`meta.showValueNotFound`	zeigt einen gespeicherten Wert, den es in der Liste nicht mehr gibt
Technischen Wert zeigen	`meta.showTechnicalNameInList`	
Übersetzungsvorsatz	`meta.translationPrefix`	wird den nachgeschlagenen Schlüsseln vorangestellt; ohne Übersetzungsfilter wirkungslos
Liste immer neu laden	`meta.alwaysLoadList`	nötig, wenn zwei Spalten denselben Filter, aber verschiedene Werte haben
Weitere meta-Eigenschaften (JSON)	–	z. B. `listFilter`, `iconName`, `width` – bleiben erhalten
Weitere Eigenschaften der Spalte (JSON)	–	bleiben erhalten

Reiter „JSON“

Der Reiter zeigt **genau das**, was im Wert der Custom Property steht – das Modell, das die beiden anderen Reiter bearbeiten, ohne jede Umformung. Er ist der Weg für alles, wofür es kein Feld gibt, und der schnellste Blick auf eine Definition im Ganzen.

Knopf	Wirkung
Übernehmen	liest das JSON und lädt es zurück in das Formular. Ist es kein gültiges JSON oder kein Objekt, sagt die Statuszeile, woran es liegt, und nichts wird übernommen
Formatieren	rückt das JSON wieder ein
Kopieren	legt den Text in die Zwischenablage

Über dem Feld steht, wie lang das JSON ist und wie lang es sein darf (Vorgabe 512 000 Zeichen). Die Grenze ist die der Custom Property; das Backend speichert oberhalb einer gewissen Größe kompakt, was die Statuszeile nach dem Speichern vermerkt.

> **Übernehmen ist noch kein Speichern.** Der Text wandert in das Formular, die Definition gilt danach als geändert – geschrieben wird sie erst mit `Speichern`. Solange ein Text im Feld steht, der noch nicht übernommen wurde, weist der Meldungsblock darauf hin.

Prüfungen und Meldungen

Über dem Formular steht in den Reitern *Allgemein* und *Spalten* ein Meldungsblock. Er wird bei jeder Änderung neu gerechnet und unterscheidet zwei Stufen:

- **Fehler** (rot) – damit funktioniert die Definition nicht.
- **Hinweise** (gelb) – die Definition läuft, tut aber nicht, was jemand erwartet. Der Router reicht `meta` unverändert durch; falsch wird es erst im Client, und dort still. Deshalb steht es hier.

Gespeichert werden kann in beiden Fällen: der Editor weist nichts ab, was das Portalskript annimmt. Hinweise, die erst beim Schreiben entstehen (aufgeräumte leere Einträge, ergänztes `schema`/`catalog`), kommen nach dem Speichern dazu.

Fehler

Meldung	Bedeutung
Name fehlt / ist nicht zulässig	`nameApi` benennt die Custom Property und die Route. Buchstaben, Ziffern und `_`, beginnend mit einem Buchstaben
Tabelle fehlt	ohne `nameSQL` gibt es nichts abzufragen
Verbindung fehlt	ohne `connectionId` weiß der Router nicht, welche Datenbank
Spalte *n* hat keinen **Namen**	eine Spalte ohne `name` ist kein Feld
Sortierung *n* hat keine Spalte	leerer Eintrag in `orderBy`

Hinweise

Meldung	Bedeutung
Keine Spalten	die Spaltenliste ist die ganze Schnittstelle – ohne sie kann nichts gelesen oder geschrieben werden
Spalte ist doppelt	der Router nimmt die erste und übergeht die zweite
n leere Einträge in `columns`	Löcher aus einer Definitionsdatei; das Speichern entfernt sie
Spalte gibt es in der Tabelle nicht	sie fehlt in jeder Antwort

Meldung	Bedeutung
Der Primärschlüssel ist nicht deklariert	`keyColumns` bleibt leer, ein Client kann keine Zeile ändern oder löschen
Spalte ist NOT NULL ohne Standardwert, aber nicht deklariert	jedes Anlegen schlägt fehl
Sortierung nach einer nicht deklarierten Spalte	der Router lässt den Eintrag fallen
Der Werteliste fehlt die **Wertespalte**	sie wird ignoriert
Der Werteliste fehlt die **Tabelle**	ihr Aufruf endet mit 404
Die feste Liste steht in `possibleValues`	die API liefert sie, der DEX-Client benutzt sie nicht – unter *Werteliste* übernehmen
View-Spalte ohne **SQL-Typ**	die API meldet `string`
`schema` / `catalog` ist leer	dort gehört die Datenbank hin, in der die Tabelle liegt

Hinweise rund um Wertelisten

Die häufigste Ursache für eine leere Auswahl im Client sind zwei Wege, die sich gegenseitig ausschließen:

- **feste Einträge und eine Werteliste** – der Client nimmt die Werteliste, `meta.listItems` bleibt unbeachtet.
- **Skript und Werteliste** – der Client nimmt die Werteliste; das Skript läuft, sein Ergebnis bleibt liegen.
- **Skript und feste Einträge** – das Skript gewinnt.
- **feste Einträge, aber ein Typ, der keine Liste ist** – der Client wertet sie nur bei `list`, `multiplelist`, `combobox` und `comboboxText` aus. `popup` und `iconorlist` bekommen sie nicht.
- **Einträge ohne `key` oder `label`** lässt der Client weg, doppelte `key` überschreiben einander.
- **`meta.listScriptParams` ohne `meta.listScript`** – die Parameter werden nirgends übergeben. Ein verschachtelter Parameter kommt im Skript als `[object Object]` an, weil DOCUMENTS Parameter als Text übergibt.
- **`meta.translationPrefix` ohne Übersetzungsfilter** – der Vorsatz wird nirgends benutzt.