

TableService-Konfiguration (Neu)

- [Einstieg](#)
 - [Überblick und Aufruf](#)
 - [Typischer Ablauf: von der Tabelle bis zum Ordner](#)
- [Definitionen bearbeiten](#)
 - [Definitionen anlegen, speichern und löschen](#)
 - [Reiter „Allgemein“](#)
 - [Reiter „Spalten“](#)
 - [Reiter „JSON“](#)
 - [Prüfungen und Meldungen](#)
- [SQL-Tabellen](#)
 - [SQL Tabelle anlegen](#)
 - [SQL Tabelle ändern](#)
 - [Mehr → Alle SQL-Tabellen exportieren / SQL-Tabellen importieren](#)
- [Squeeze](#)
 - [Mehr → Tabelle in Squeeze Stammdaten hinzufügen](#)
 - [Mehr → Squeeze-Stammdaten in Tabelle laden](#)
- [Umgebung](#)
 - [Mehr → Ordner und Outbar](#)
 - [Definitionen importieren und exportieren](#)
 - [Geschützte Datenbanken](#)
 - [Technik, Dateien und Übersetzungen](#)

Einstieg

Überblick und Aufruf

Der **TableService-Definitionseditor** ist die Oberfläche für die Custom Properties `TS_Definition_*` und `TS_ViewDefinition_*`. Aus ihnen liest der TableServiceRouter, welche Tabellen und Sichten er über die API anbietet, welche Spalten er liefert und welche davon geschrieben werden dürfen.

Aufruf

Im Bereich **Stammdaten** stehen dafür zwei Ordner:

Ordner	Was in der Seitenleiste steht
Stammdaten-Konfiguration	die Definitionen der Stammdaten-Datenbank
Stammdaten-Konfiguration (Custom)	nur die eigenen Definitionen – die hier angelegt und die aus dem <code>Custom</code> -Ordner des TableService importierten

Beide Ordner öffnen dieselbe Oberfläche und können dasselbe. Sie unterscheiden sich allein darin, **welche** Definitionen in der Seitenleiste zur Auswahl stehen; in *(Custom)* trägt jeder Eintrag den Zusatz *eigene*.

Welche Definitionen ein Ordner zeigt, ist am Ordner hinterlegt. Weitere Ordner – etwa für eine andere Datenbank – richtet die Administration ein; der Editor selbst legt einen solchen Ordner zu einer geöffneten Definition an, siehe [Ordner und Outbar](#).

Was ein Ordner anzeigt, ist dabei eine **Anzeigeentscheidung, keine Befugnis**. Was geschrieben werden darf, entscheidet in jedem Fall das Portalskript – siehe [Geschützte Datenbanken](#).

Aufbau der Oberfläche

- **Kopfleiste** – `Neu`, `SQL Tabelle anlegen`, `SQL Tabelle ändern`, `Aktualisieren` und das Menü `Mehr` mit den Werkzeugen für Squeeze, den Tabellenaufbau und JSON.
- **Seitenleiste** – alle sichtbaren Definitionen, nach Tabellen und Views gruppiert, mit Filterfeld und Art-Auswahl.
- **Hauptbereich** – die geöffnete Definition in drei Reitern: [Allgemein](#), [Spalten](#) und [JSON](#); darüber `Speichern`, `Verwerfen` und das Menü `Mehr` der Definition.
- **Statuszeile** ganz unten – die Rückmeldung zur letzten Aktion. Grün heißt erledigt, gelb ein Hinweis, rot ein Fehler.

Backend

Jeder lesende und schreibende Aufruf geht an das Portalskript

`DEXPRO__TableServiceDefinitionEditor`. Dort fällt auch die Entscheidung, was geschrieben werden darf – siehe [Geschützte Datenbanken](#).

Typischer Ablauf: von der Tabelle bis zum Ordner

Der Weg, den eine neue eigene Tabelle im Regelfall nimmt. Jeder Schritt hat eine eigene Seite mit den Einzelheiten.

1. **Tabelle anlegen** – [SQL Tabelle anlegen](#). Verbindung wählen, Namen vergeben, Spalten eintragen. `ID` und `Licence` kommen von selbst dazu. Mit dem Haken *Definition dazu anlegen* führt der Dialog gleich zu Schritt 2.
2. **Definition anlegen** – [Definitionen verwalten](#). Art, Name, Verbindung, Tabelle.
3. **Spalten deklarieren** – [Reiter „Spalten“](#). + **aus Tabelle ...** übernimmt die Spalten der SQL-Tabelle samt Vorschlägen für Typ, Schlüssel- und Mandantenspalte. Was hier nicht steht, liefert die API nicht.
4. **Feinschliff** – [Reiter „Allgemein“](#) für Sortierung und Filter, im Reiter *Spalten* die Beschriftungen und Wertelisten.
5. **Meldungen lesen** – [Prüfungen und Meldungen](#). Fehlt der Primärschlüssel oder eine NOT-NULL-Spalte, steht das dort, bevor es im Client auffällt.
6. **Speichern**. Danach ist die Definition über die API erreichbar.
7. **Ordner anlegen** – [Ordner und Outbar](#), damit die Tabelle im Client erscheint. Danach ab- und wieder anmelden.

Dazu kommen die Wege, die man später braucht:

- Eine Spalte fehlt → [SQL Tabelle ändern](#); der Dialog zieht die betroffenen Definitionen gleich nach.
- Der Stand soll zum Kunden → [Alle SQL-Tabellen exportieren / importieren](#) für den Tabellenaufbau und [Definitionen importieren und exportieren](#) für die Definitionen.
- Die Tabelle soll nach Squeeze → [Tabelle in Squeeze Stammdaten hinzufügen](#), oder umgekehrt [Squeeze-Stammdaten in Tabelle laden](#).

Definitionen bearbeiten

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, Tabellename 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

 legt eine Definition zu einer **vorhandenen** Tabelle an. Gibt es die Tabelle noch nicht, ist [SQL Tabelle anlegen](#) der Schritt davor.

Feld	Bedeutung
Art	<i>Tabelle (lesen und schreiben)</i> oder <i>View (nur lesen)</i> . Bestimmt Name und Typ der Custom Property: <code>TS_Definition_*</code> oder <code>TS_ViewDefinition_*</code>
Name	der öffentliche Name; er steht in der Route und im Namen der Custom Property. Buchstaben, Ziffern und <code>_</code> , 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	<code>nameApi</code>	Der öffentliche Name. Er steht in der Route <code>/<Art>/<Name></code> und im Namen der Custom Property. Ein Umbenennen verschiebt die Property. Pflichtfeld
Verbindung	<code>connectionId</code>	Die DEXPRO-Datenbankverbindung aus der <code>dbConn.json</code> . Angeboten werden die gemeldeten Verbindungen; über <i>andere ...</i> lässt sich eine eintragen, die (noch) nicht gemeldet wird. Pflichtfeld
Tabelle	<code>nameSQL</code>	Die SQL-Tabelle oder View hinter der Definition. Ist die Tabellenliste der Verbindung geladen, schlägt das Feld beim Tippen vor. Pflichtfeld
Standardsortierung	<code>orderBy</code>	Eine oder mehrere Zeilen aus Spalte und Richtung (<i>aufsteigend</i> / <i>absteigend</i>). 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	<code>whereRaw</code>	Wird an jede Abfrage dieser Definition angehängt, z. B. <code>Active=1</code> . Wird nicht geprüft – ein Syntaxfehler legt die Tabelle lahm
nur unterschiedliche Zeilen	<code>distinct</code>	Setzt <code>DISTINCT</code> 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 <code>text</code> 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 <code>_Kopie</code> 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 `key` 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	<code>name</code>	Der SQL-Spaltenname. Er ist der Schlüssel in den Zeilen und beim Schreiben. Pflichtfeld
Beschriftung	<code>label</code>	Fehlt sie, zeigt der Client den Namen
SQL-Typ (<i>nur View</i>)	<code>type</code>	Views werden nicht ausgelesen – der Typ kommt von hier. Ohne ihn meldet die API <code>string</code>
Typ	<code>meta.type</code>	<code>text</code> , <code>int</code> , <code>number</code> , <code>date</code> , <code>datetime</code> , <code>checkbox</code> , <code>list</code> , <code>multiplelist</code> , <code>combobox</code> , <code>comboboxText</code> , <code>popup</code> , <code>iconorlist</code> , <code>key</code> , <code>licence</code> , <code>hidden</code> . <code>key</code> und <code>licence</code> sind die technischen Spalten, <code>hidden</code> lässt die Spalte in der API, aber nicht im Formular
Sichtbar	<code>meta.show</code>	<i>nein</i> blendet die Spalte in der Liste aus
Standardwert	<code>meta.defaultValue</code>	Wird beim Anlegen vorbelegt. Als Text, Zahl, <i>ja</i> , <i>nein</i> oder <i>leer (null)</i>
Suchfeld	<code>meta.searchAutocomplete</code>	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 <code>meta.listItems</code> in der Definition selbst – der Weg, den der DEX-Client für eine feste Auswahl kennt. Je Eintrag <code>key</code> (gespeicherter Wert) und <code>label</code> (angezeigter Text). Einträge ohne beides lässt der Client weg
aus einem Skript	<code>meta.listScript</code> nennt ein Portalskript; der Client ruft es über <code>dex.callScript</code> auf und erwartet eine JSON-Zeichenkette mit <code>{ value, label }</code> . Darunter stehen die <code>meta.listScriptParams</code> als Name-Wert-Liste – DOCUMENTS übergibt sie als Text
aus anderer Tabelle	Nachgeschlagen in einer Tabelle, geliefert unter <code>/table/<Definition>/col/<Spalte>/possibleValues</code>
<i>feste Liste in</i> <code>possibleValues</code>	Erscheint nur, wenn eine Definition sie schon mitbringt. Die API liefert sie, der DEX-Client zeigt sie nicht . Der Knopf <i>In feste Liste übernehmen</i> macht <code>meta.listItems</code> 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, <code>{{Spaltenname}}</code> wird je Zeile ersetzt
Schema / Katalog (erweitert)	MySQL: <code>schema</code> = Datenbankname, <code>catalog</code> = <code>def</code> . MS SQL: <code>schema</code> = <code>dbo</code> , <code>catalog</code> = Datenbankname
WHERE-Bedingung der Liste (erweitert)	schränkt die Liste ein
Wertespalte darf der Client wählen (erweitert)	<code>valueColumnByClient</code>

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	<code>mandatory</code>	Vorgabe entscheidet die Datenbank (NOT NULL, kein Standardwert, keine Identity)
Beim Anlegen änderbar	<code>editableOnInsert</code>	<i>nein</i> blendet das Feld schreibgeschützt ein; Pflichtspalten bleiben änderbar
Beim Ändern änderbar	<code>editableOnUpdate</code>	Primärschlüssel sind <i>nein</i> , solange hier nichts anderes steht
meta.label	<code>meta.label</code>	überschreibt die Beschriftung im DEX-Tabellenclient
Breite	<code>meta.layout</code>	Breite des Eingabefelds im Raster, 1 bis 12
Änderbar / Pflicht im Client / „ANY“ anbieten	<code>meta.editable</code> , <code>meta.mandatory</code> , <code>meta.addAny</code>	Verhalten im Client
Listenformat	<code>meta.listFormat</code>	z. B. <code>%value% (%label%)</code>
Unbekannten Wert zeigen	<code>meta.showValueNotFound</code>	zeigt einen gespeicherten Wert, den es in der Liste nicht mehr gibt
Technischen Wert zeigen	<code>meta.showTechnicalNameInList</code>	
Übersetzungsvorsatz	<code>meta.translationPrefix</code>	wird den nachgeschlagenen Schlüsseln vorangestellt; ohne Übersetzungsfiler wirkungslos
Liste immer neu laden	<code>meta.alwaysLoadList</code>	nötig, wenn zwei Spalten denselben Filter, aber verschiedene Werte haben
Weitere meta-Eigenschaften (JSON)	-	z. B. <code>listFilter</code> , <code>iconName</code> , <code>width</code> - 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 Meldungsbereich 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	<code>nameApi</code> benennt die Custom Property und die Route. Buchstaben, Ziffern und <code>_</code> , beginnend mit einem Buchstaben
Tabelle fehlt	ohne <code>nameSQL</code> gibt es nichts abzufragen
Verbindung fehlt	ohne <code>connectionId</code> weiß der Router nicht, welche Datenbank
Spalte <i>n</i> hat keinen Namen	eine Spalte ohne <code>name</code> ist kein Feld
Sortierung <i>n</i> hat keine Spalte	leerer Eintrag in <code>orderBy</code>

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
<i>n</i> leere Einträge in <code>columns</code>	Löcher aus einer Definitionsdatei; das Speichern entfernt sie

Meldung	Bedeutung
Spalte gibt es in der Tabelle nicht	sie fehlt in jeder Antwort
Der Primärschlüssel ist nicht deklariert	<code>keyColumns</code> 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 <code>possibleValues</code>	die API liefert sie, der DEX-Client benutzt sie nicht – unter <i>Werteliste</i> übernehmen
View-Spalte ohne SQL-Typ	die API meldet <code>string</code>
<code>schema</code> / <code>catalog</code> 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 Übersetzungsfiler** – der Vorsatz wird nirgends benutzt.

SQL-Tabellen

SQL Tabelle anlegen

SQL Tabelle anlegen ist der Schritt **vor Neu**: die Tabelle gibt es noch nicht. Der Dialog fragt Verbindung, Tabellennamen und Spalten ab und lässt das Portalskript ein `CREATE TABLE` ausführen – auf MySQL oder MS SQL, je nachdem, was in der `dbConn.json` zu dieser Verbindung steht.

Wo die Tabelle entsteht

Feld	Bedeutung
Verbindung	Die DEXPRO-Datenbank, in der die Tabelle angelegt wird. Workflow, DOCUMENTS und Squeeze stehen nie zur Wahl – siehe Geschützte Datenbanken
Tabellennamen	Buchstaben, Ziffern und <code>_</code> , beginnend mit einem Buchstaben oder <code>_</code> , höchstens 63 Zeichen. Ein Name, den es in dieser Datenbank schon gibt, wird abgewiesen

ID und Licence sind immer dabei

Jede so angelegte Tabelle beginnt mit denselben zwei Spalten wie jede ausgelieferte DEX-Tabelle:

Spalte	Form
<code>ID</code>	<code>BIGINT</code> , Primärschlüssel, von der Datenbank hochgezählt
<code>Licence</code>	<code>varchar(50)</code> , <code>NOT NULL</code> , Standardwert ist der Mandant dieser Anmeldung

Beide zeigt der Dialog nur an. Anlegen, ändern oder weglassen kann sie niemand – das entscheidet das Portalskript.

Damit ist der Primärschlüssel erschöpft: eine weitere Schlüsselspalte bietet der Dialog nicht an, und ohne Schlüssel gibt es auch keinen Auto-Wert, denn eine hochzählende Spalte muss der Schlüssel sein. In den Spaltenzeilen stehen **PK** und **Auto** deshalb nur als Strich.

Die eigenen Spalten

Je Zeile: Name, Typ, Länge, Standardwert und der Haken **NULL**.

Typ	MS SQL / MySQL	Länge
Text	NVARCHAR(n) / VARCHAR(n)	Zeichenzahl, Vorgabe 255
Langtext	NVARCHAR(MAX) / LONGTEXT	-
Zeichen (feste Länge)	NCHAR(n) / CHAR(n)	Vorgabe 10
Ganzzahl	INT	-
Ganzzahl (groß)	BIGINT	-
Ganzzahl (klein)	SMALLINT	-
Dezimalzahl	DECIMAL(p, s)	Stellen, Nachkommastellen - z. B. 18,2
Gleitkommazahl	FLOAT / DOUBLE	-
Ja/Nein	BIT / BIT(1)	-
Datum	DATE	-
Datum und Zeit	DATETIME	-
Binärdaten	VARBINARY(MAX) / LONGBLOB	-

- **NULL** - ohne diesen Haken muss jede neue Zeile einen Wert mitbringen, oder die Spalte braucht einen Standardwert.
- **Standardwert** - bei Datum und Zeit steht `now` für die aktuelle Zeit (`GETDATE()` bzw. `CURRENT_TIMESTAMP`).
- Spaltennamen folgen derselben Regel wie der Tabellename. Doppelte Namen und die beiden festen Namen `ID` und `Licence` werden abgewiesen.

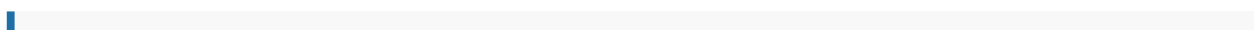
SQL anzeigen

`SQL anzeigen` holt das fertige Statement vom Portalskript und zeigt es in der Statuszeile - **ausgeführt ist dabei noch nichts**. Das Statement wird nicht in der Seite zusammengebaut, sondern im Backend aus geprüften Bausteinen; die Vorschau ist also genau das, was später läuft.

Anlegen

`Anlegen` führt das `CREATE TABLE` aus. Der Fortschrittskasten bleibt stehen, bis die Datenbank geantwortet hat.

Ist **Definition dazu anlegen** angehakt, steht anschließend gleich eine neue Definition offen - mit den Spalten der frisch angelegten Tabelle, noch ungespeichert. Der Name wird aus dem Tabellennamen abgeleitet und ist frei wählbar.



Fallen lassen kann der Editor eine Tabelle nicht. Ein `DROP TABLE` gibt es hier nicht – das bleibt Sache der Datenbank. Einzelne selbst angelegte Spalten lassen sich über [SQL Tabelle ändern](#) wieder löschen.

SQL Tabelle ändern

`SQL Tabelle ändern` ist der Schritt **danach**: die Tabelle steht, es fehlt eine Spalte. Der Dialog kann absichtlich nur den Teil von `ALTER TABLE`, bei dem nichts unabsichtlich verlorengeht.

Möglich	Nicht möglich
Spalte hinzufügen – immer mit NULL erlaubt, ohne Schlüssel, ohne Auto-Wert	Spalte umbenennen
Zeichenspalte verlängern – nur größer	Typ wechseln
Selbst angelegte Spalte löschen	Länge verkleinern, fremde Spalten löschen, Tabelle löschen

Die vorhandenen Spalten stehen deshalb als Text da, mit genau einem Eingabefeld dort, wo die Länge steht. Die Entscheidung darüber fällt im Portalskript, nicht im Formular.

Welche Tabelle geändert wird

Verbindung und Tabelle werden oben gewählt; Views stehen nicht zur Wahl, ihre Spalten kommen aus ihrer Abfrage. Sobald eine Tabelle feststeht, stehen ihre Spalten darunter, so wie die Datenbank sie meldet.

Spalte verlängern

Bei einer Zeichenspalte lässt sich die Länge **nur nach oben** ändern; der Tooltip nennt den aktuellen Wert und die Obergrenze des Typs. Was beim Verkürzen abgeschnitten würde, käme nicht zurück – deshalb gibt es diesen Weg nicht. Reicht die Obergrenze nicht, ist eine neue Spalte vom Typ *Langtext* der Weg.

Spalte löschen

Ein **x** hat nur eine Spalte, die **dieser Editor selbst angelegt** hat. Er merkt sich das in der Custom Property `TS_DdlAddedColumns`. Was aus der Auslieferung kommt oder von Hand entstanden ist, hat keinen Löschknopf und wird auch dann abgelehnt, wenn die Anfrage es verlangt.

Ein Klick markiert die Spalte nur. Gelöscht wird sie erst mit **Ändern** und einer Rückfrage – ein zweiter Klick auf das **x** nimmt die Markierung wieder weg.

Mit der Spalte fällt ihr Inhalt in jeder Zeile. Das lässt sich nicht zurückholen.

ID und **Licence** bleiben in jedem Fall, wie sie sind: eine Tabelle, in der sie anders aussehen, passt nicht mehr zu den anderen.

Änderungen in die Definitionen überführen

Die dritte Karte des Dialogs zieht die Definitionen nach – denn eine Spalte in der Datenbank ist erst zu gebrauchen, wenn eine Definition sie kennt.

Angeboten wird, was auf **genau diese Tabelle** zeigt (dieselbe Verbindung, derselbe Tabellenname), voreingestellt alles davon:

- Angehakte Definitionen bekommen die **neuen** Spalten eingetragen, **gelöschte** fallen dort heraus. Was schon dasteht, bleibt unangetastet.
- Die gerade **geöffnete** Definition wird dabei nicht im Hintergrund gespeichert: sie bekommt die Spalten im Arbeitsspeicher und gilt als geändert. So verliert niemand offene Änderungen dadurch, dass eine Tabelle erweitert wurde.
- Eine Definition, deren JSON nicht lesbar ist, wird nicht angefasst – sie muss erst von Hand geradegezogen werden.

Zeigt keine Definition auf die Tabelle, sagt der Dialog das: die Spalten entstehen dann nur in der Datenbank, und **Neu** macht daraus eine Definition.

SQL anzeigen und Ändern

Wie beim Anlegen zeigt **SQL anzeigen** das Statement, das später läuft, ohne etwas auszuführen.

Ändern führt es aus; bei markierten Löschungen heißt der Knopf **Ändern und löschen** und fragt vorher nach.

Ist nichts zu tun – keine neue Spalte, keine gewachsene Länge –, sagt der Dialog das, statt ein leeres Statement zu schicken.

Mehr ? Alle SQL-Tabellen exportieren / SQL-Tabellen importieren

Dieselben beiden Schritte wie *SQL Tabelle anlegen* und *SQL Tabelle ändern*, nur für eine **ganze Datenbank** statt für eine Tabelle. Gemeint ist der Umzug vom Entwicklungssystem zum Kunden: dort stehen zwanzig Tabellen, hier keine davon.

Was in der Datei steht

Die Datei beschreibt den **Aufbau** jeder Tabelle – Spalten, Typ, Länge, Stellen, NULL, Standardwert – und **keine einzige Zeile Daten**.

“ Das ist keine Sicherung. Wer Daten mitnehmen will, nimmt ein Werkzeug der Datenbank.

Sichten, Indizes, `UNIQUE`-Bedingungen, Fremdschlüssel und Trigger gehen nicht mit: was dieser Editor nicht anlegen kann, verspricht er auch nicht. Die Typen in der Datei sind die des Editors und nicht die einer bestimmten Datenbank – nur so lässt sich eine MySQL- in eine MS-SQL-Datenbank spielen. Ein `TINYINT` kommt deshalb als `SMALLINT` wieder an.

Exportieren

Angeboten werden die Datenbanken, in denen dieser Editor Tabellen anlegen darf. Die Workflow-Datenbank, das Repository und Squeeze stehen nicht dabei: ihr Aufbau gehört zum Produkt, kommt aus der Auslieferung und wäre in der Datei nichts als Länge – siehe [Geschützte Datenbanken](#).

Gelesen wird je Verbindung ein Aufruf; zusammengesetzt wird die Datei in der Seite. Eine Datenbank, die nicht antwortet, steht als Fehler in der Rückmeldung, die übrigen kommen trotzdem mit.

Importieren

Angeglichen wird in **eine Richtung**, und zwar genau in den drei Fällen, bei denen nichts verlorengeht:

Fall	Was geschieht
Tabelle fehlt	wird angelegt
Spalte fehlt	wird ergänzt
Zeichenspalte ist kürzer als in der Datei	wird verlängert

Gelöscht wird nichts. Eine Spalte, die in der Datei fehlt, aber in der Zieltabelle steht, bleibt stehen. Verkleinert wird nichts, und kein Typ wird umgedeutet: beides steht als Hinweis in der Planzeile und wird nicht ausgeführt.

Der Fall, um den es vor allem geht, ist der zusammengesetzte: eine Tabelle, die seit dem letzten Abgleich ein Feld dazubekommen hat **und** bei der eine Länge gewachsen ist, kommt in einem Lauf nach.

Schritt 1: Verbindungen zuordnen

Die Verbindungen aus der Datei werden denen dieser Installation zugeordnet, voreingestellt gleicher Name. Zur Wahl stehen die Datenbanken, in denen etwas entstehen darf. Eine Verbindung aus der Datei, zu der hier keine passt, bleibt auf *nicht importieren*.

Die Datenbanken des Produkts stehen auch hier nicht zur Wahl. Eine Zeile, die auf eine von ihnen zeigt, bleibt auf *nicht importieren* – siehe [Geschützte Datenbanken](#).

Schritt 2: „Das würde geschehen“

Prüfen rechnet den Plan; dieser Schritt **liest nur**. Danach steht Tabelle für Tabelle da, was entstünde, mit einem Haken je Zeile, der sich abwählen lässt, und mit Hinweisen zu allem, was übergangen wird (anderer Typ, kürzere Länge, unbekannter Typ, unbrauchbarer Spaltenname, Sicht statt Tabelle, geschützte Datenbank).

Jede neu angelegte Tabelle bekommt auch hier **ID** und **Licence** vorangestellt.

Erst **Importieren** schreibt – über dieselben Schritte wie *SQL Tabelle anlegen* und *SQL Tabelle ändern*, und damit unter denselben Grenzen.

Danach

Eine neue Spalte ist erst zu gebrauchen, wenn eine Definition sie kennt. Der Reiter [Spalten](#) nimmt sie über **+ aus Tabelle ...** in die passende Definition auf.

Squeeze

Mehr ? Tabelle in Squeeze Stammdaten hinzufügen

Dieser Dialog geht in die andere Richtung: nicht in die Datenbank, sondern **nach Squeeze**. Gewählt wird eine SQL-Tabelle – dieselbe Auswahl wie bei *SQL Tabelle ändern* –, und daraus entsteht über die Squeeze-API die passende Stammdatentabelle:

- ein `POST` auf `api/v2/masterData/tables` für die Tabelle, der mit ihrer Id antwortet,
- danach ein `POST` je Spalte auf `.../tables/<id>/columns`.

Die SQL-Tabelle selbst wird dabei nur gelesen.

Die Stammdatentabelle in Squeeze

Feld	Bedeutung
Name in Squeeze	Vorgeschlagen wird <code>Breeze_<Tabellenname></code> . Wer ihn ändert, behält seine Eingabe auch beim nächsten Tabellenwechsel
Beschreibung	Freier Text, den Squeeze zur Tabelle anzeigt. Bleibt das Feld leer, wird der Name der SQL-Tabelle eingetragen

Felder

Die Felder heißen in Squeeze genauso wie die Spalten der Tabelle – so kommt eine Übertragung zwischen beiden ohne Zuordnungstabelle aus.

- **ID geht nie mit.** Eine Squeeze-Stammdatentabelle zählt ihre Zeilen selbst und bringt eine eigene Id mit; eine zweite daneben bedeutete dasselbe und sagte etwas anderes, sobald beide Seiten schreiben.
- **Licence bleibt im Angebot** – welchem Mandanten eine Zeile gehört, weiß Squeeze von sich aus nicht.
- **Zeichenvorrat:** Squeeze nimmt höchstens 45 Zeichen aus Buchstaben, Ziffern, Unterstrich und Bindestrich. Eine Spalte mit Punkt, Leerzeichen oder Umlaut bekommt darum ein Feld für einen **Ersatznamen**; der Rückweg in die Tabelle bleibt über `sourceColumn` erhalten, das die SQL-Spalte behält. Kurze Namen wie `de` oder `en` sind in

Ordnung.

Übertragungsskript

Die frisch angelegte Stammdatentabelle ist **leer**. Der nächste Schritt steht deshalb gleich in der Rückmeldung – drei Haken:

Haken	Wirkung
Übertragungsskript erzeugen	legt ein Portalskript nach dem Vorbild von <code>Squeeze_UserExit_MasterData_TransferPrincipalsApi</code> an, mit einem <code>addColumnMap()</code> je Feld. Name: <code>Squeeze_UserExit_Export_<Tabelle></code> , Kategorie <code>Squeeze_UserExit</code> . Gibt es den Namen schon, kommt eine Rückfrage – der Quelltext könnte von Hand geändert sein
Stammdaten gleich übertragen	führt genau dieses Skript einmal aus. Dafür – und nur dafür – ist das erzeugte Skript <code>clientExecutable</code> , anders als die Vorlage
Täglich als Job ausführen	mit einer Uhrzeit daneben (<code>HH:MM</code>). Das Skript bekommt <code>RunAsJob</code> , den festen Zeitpunkt und die Uhrzeit; ohne den Haken läuft es nur, wenn es jemand startet

Die letzten beiden Haken hängen am ersten: was nicht angelegt wird, lässt sich weder ausführen noch einplanen. Zeitpunkt und Häufigkeit lassen sich danach im DOCUMENTS-Manager ändern.

Das `Breeze_` der Tabelle fällt im Skriptnamen weg, und „Export“ steht darin, weil es in diese Richtung geht: aus einer DEX-Tabelle **hinaus** nach Squeeze. Was von dort holt, heißt `Squeeze_UserExit_MasterData_*` und soll davon zu unterscheiden sein.

Wie viele Zeilen übertragen wurden, zählt das Skript nicht mit – das steht im Log unter `Logs/SqueezeImport`. Hat die SQL-Tabelle für diesen Mandanten keine Zeile, bleibt die Stammdatentabelle vorerst leer; das Skript überträgt beim nächsten Lauf, was dann da ist.

Die Gegenrichtung beschreibt [Squeeze-Stammdaten in Tabelle laden](#).

Mehr ? Squeeze-Stammdaten in Tabelle laden

Die Gegenrichtung zu [Tabelle in Squeeze Stammdaten hinzufügen](#): aus einer Squeeze-Stammdatentabelle in eine Tabelle einer DEX-Datenbank.

Erzeugt wird dafür ein Portalskript `Squeeze_UserExit_Import_<Zieltabelle>` nach dem Vorbild von `DEXPRO__UserExit_SqueezeMasterdata_FirmInformation` – mit `SqueezeImport`, einem `setTable()` für die Squeeze-Seite und einem `add()` je Feld.

Nur in eine Tabelle, die es gibt

Angelegt wird hier nichts und geändert auch nichts. Die Zieltabelle wird aus den vorhandenen gewählt, ihre Spalten werden gelesen, wie sie sind, und zugeordnet wird nur auf sie. Fehlt die Tabelle noch, ist [SQL Tabelle anlegen](#) der Schritt davor.

Feld	Bedeutung
Stammdatentabelle in Squeeze	die Tabelle, aus der gelesen wird. Ihre Felder stehen danach in der Zuordnung zur Wahl
Verbindung	die DEXPRO-Verbindung der Zieltabelle. Workflow, DOCUMENTS und Squeeze stehen nicht zur Wahl
Zieltabelle	die Tabelle, in die geschrieben wird

Zuordnung

Zugeordnet wird **spaltenweise von der Zieltabelle aus** und nicht von Squeeze her: was gefüllt werden *muss*, steht in der DEX-Tabelle. Eine Pflichtspalte ohne Wert lässt den ersten Lauf scheitern – der Dialog sagt es als Warnung, lehnt aber nicht ab, denn ein Standardwert kann es doch noch richten.

Je Zielspalte gibt es drei Möglichkeiten:

Auswahl	Bedeutung
ein Feld aus Squeeze	vorbelegt ist ein Squeeze-Feld gleichen Namens

Auswahl	Bedeutung
ein fester Wert	bei <code>Licence</code> vorbelegt mit dem Mandanten. <code>cLicence</code> geht dabei als Konstante in den Quelltext, alles andere als Text
- nicht füllen -	die Spalte bleibt leer

Die Spalte `ID` steht nur abgeblendet da: sie zählt sich selbst hoch.

Zieltabelle vor jedem Lauf leeren

“ **Die einzige Stelle im ganzen Editor, die Daten zerstört.** Mit diesem Haken beginnt jeder Lauf mit `TRUNCATE TABLE` auf der Zieltabelle. Was darin steht und nicht aus Squeeze kommt, ist danach weg.

So ist ein Stammdatenabgleich gemeint, und der Haken ist deshalb voreingestellt. Er lässt sich abwählen; das erzeugte Skript schreibt seinen Zustand ausdrücklich hin, statt sich auf den Vorgabewert der Bibliothek zu verlassen. Wer gleich ausführen lässt, bekommt vorher noch eine Rückfrage.

Anlegen, ausführen, einplanen

`Skript erzeugen` legt das Portalskript an; ein vorhandenes gleichen Namens wird nur nach Rückfrage überschrieben. Daneben stehen dieselben beiden Haken wie beim Export:

- **Stammdaten gleich laden** – führt das Skript einmal aus.
- **Täglich als Job ausführen** – mit Uhrzeit; ändern lässt sich das danach im DOCUMENTS-Manager.

Umgebung

Mehr ? Ordner und Outbar

Zu einer **gespeicherten** Definition legt der Editor den DOCUMENTS-Ordner an, über den die Tabelle im Client erscheint, und ordnet ihn einer Outbar zu. Bei einer noch nicht gespeicherten Definition bleibt der Punkt gesperrt: der Ordner zeigt auf den gespeicherten Namen.

Die Felder

Feld	Bedeutung
Ordnername	fest aus dem Namen der Definition abgeleitet: <code>Folder_Admin_ts_<Name></code> . Nicht änderbar
Bezeichnung	vorgeschlagen wird <code>pf:Folder_Admin_<Name></code> . Mit dem Vorsatz <code>pf:</code> ist es ein Schlüssel der Properties-Tabelle; ohne ihn zeigt DOCUMENTS die Bezeichnung unverändert an
Bezeichnung deutsch / englisch	erscheinen, sobald die Bezeichnung ein Schlüssel ist. Sie werden in die Properties-Tabelle geschrieben. Eine leere Sprache bleibt unangetastet und löscht nichts
Symbol	vorgeschlagen <code>entypo:database</code>
Ordnertyp	<i>öffentlicher Ordner</i> oder <i>nur Unterordner</i> . Die Tabelle zeigt in beiden Fällen das Gadget - <code>gadgetConfig</code> geht der Ordneransicht vor
Übergeordneter Ordner	voreingestellt <code>Admin_MasterData</code> . Er muss vorhanden sein
Outbar	voreingestellt <code>Administration_MasterData</code> ; daneben <i>keine Outbar</i> und <i>andere Outbar ...</i>
Zugriffsprofile des übergeordneten Ordners übernehmen	ohne Profil ist ein öffentlicher Ordner für alle Benutzer sichtbar

Die Eigenschaft `gadgetConfig`, die `Gadget_DEX_ApiCalls` mit der Tabelle aufruft, bleibt fest; `apiName` darin ist der Name der Definition. Vorhandene Eigenschaften des Ordners bleiben erhalten.

“ **Die Outbar muss es geben.** Zuordnen kann ein Portalskript, anlegen nicht – eine neue Outbar entsteht im DOCUMENTS-Manager. Eine Outbar, die funktioniert hat, merkt sich der Editor und bietet sie danach in der Auswahl an.

Ordner schon vorhanden

Gibt es den Ordner bereits, fragt der Dialog nach. **Aktualisieren** setzt Bezeichnung, Symbol und `gadgetConfig` neu und hängt ihn unter den gewählten übergeordneten Ordner; die übrigen Eigenschaften des Ordners und seine Rechte bleiben.

Properties erzeugen

Ist die Bezeichnung ein Schlüssel, steht sie im Ordnerbaum bis auf Weiteres als Schlüssel da: DOCUMENTS liest den Text aus den Properties-Dateien, und die entstehen erst aus der Tabelle. **Properties erzeugen** holt das nach und lädt den Principal neu – am besten gleich, vor dem Abmelden.

Abmelden

Ordner und Outbar erscheinen im Client erst, nachdem der Benutzer sich **ab- und wieder angemeldet** hat: der Ordnerbaum hängt an der Sitzung, ein Neuladen der Seite genügt nicht. **Jetzt abmelden** erledigt den ersten Schritt; ist der Client von hier aus nicht erreichbar, sagt der Dialog das. Bei ungespeicherten Änderungen kommt vorher die Warnung – der Ordner ist geschrieben, der bleibt, die Änderungen gingen verloren.

Definitionen importieren und exportieren

Drei Wege, Definitionen in den Editor hinein- und aus ihm herauszubekommen. Alle drei betreffen die **Definitionen**, nicht den Tabellenaufbau – der hat mit [Alle SQL-Tabellen exportieren / importieren](#) einen eigenen Weg.

Mehr ? Alle exportieren

Liest alle Definitionen und lädt sie als eine Datei `TableServiceDefinitions.json` herunter:

```
{ "exported": "...", "count": 12, "definitions": [ ... ] }
```

Das ist der Weg, um einen Stand mitzunehmen oder vor einer größeren Änderung zu sichern.

Einzeln geht dasselbe über **Mehr → Exportieren** bei geöffneter Definition; die Datei heißt dann `TS_Definition_<Name>.json` bzw. `TS_ViewDefinition_<Name>.json`.

Mehr ? JSON importieren ...

Nimmt beides an – eine Datei über den Dateiauswähler oder eingefügten Text:

Inhalt	Wirkung
eine einzelne Definition (<code>{ "nameApi": ... }</code> oder <code>{ kind, definition }</code>)	wird nur in den Editor geladen , als neue Definition und ungespeichert. Geschrieben wird erst mit <code>Speichern</code> . Die Art (<i>Tabelle / View</i>) steht im Dialog daneben, sofern das JSON sie nicht selbst mitbringt
eine Datei aus Alle exportieren (<code>{ definitions: [...] }</code>)	wird nach einer Rückfrage zurückgeschrieben : ein Aufruf je Eintrag, jeder mit <code>setOrAddCustomProperty</code> . Eine Definition gleichen Namens und gleicher Art wird dabei ersetzt

Beim Zurückschreiben eines Bündels werden vorhandene Definitionen ersetzt. Wer zurückkönnen will, exportiert vorher. Was nicht geschrieben werden konnte, steht danach einzeln in der Rückmeldung.

Aus Definitionsdateien importieren

Der Knopf steht in der Kopfleiste und ist **nur zu sehen, solange es noch keine einzige Definition gibt**. Er führt `DEXPRO__CreateCustomPropertiesFromAllTableServices` aus – denselben Lauf, der auch die Auslieferung bestückt: aus den Definitionsdateien unter `TableService/data/table-definitions` entstehen die Custom Properties `TS_Definition_*` / `TS_ViewDefinition_*`.

Einmal importiert, ist der Editor selbst der richtige Weg, etwas zu ändern. Ein erneuter Lauf überschreibe alles, was hier bearbeitet wurde – deshalb verschwindet der Knopf, sobald eine Definition vorhanden ist.

Nach dem Lauf setzt der Editor Suchfeld und Art-Filter zurück und holt die Übersicht frisch, damit das eben Importierte auch zu sehen ist.

Geschützte Datenbanken

Am Tabellenaufbau von drei Datenbanken ändert dieser Editor nichts. Sie stehen in keinem Dialog zur Wahl, der anlegt oder ändert, und das Portalskript weist eine Anfrage auch dann ab, wenn sie doch einmal darauf zeigt.

Datenbank	Warum
Workflow (<code>Dex_Workflow</code>)	In ihr liegt der Zustand jedes laufenden Prozesses, und ihren Aufbau liest und schreibt die Workflow-Engine selbst. Eine Spalte mehr wäre dort keine Tabellenänderung, sondern ein Eingriff in eine laufende Maschine
DOCUMENTS (<code>documents5</code> , <code>documents6</code> , <code>Dex_Documents</code>)	Seinen Aufbau bestimmt DOCUMENTS. Er kommt aus der Auslieferung, ein Update setzt ihn wieder, und was dazwischen von Hand entstanden ist, fällt heraus
Squeeze (<code>Dex_Squeeze</code>)	Der Aufbau gehört zum Produkt und kommt aus der Auslieferung: eine Spalte, die das nächste Update wieder wegnimmt, hilft niemandem

Lesen und ihre Definitionen bearbeiten bleibt bei allen dreien möglich – nur ihr Aufbau nicht. Eigene Tabellen gehören in eine eigene Datenbank.

Was das im Alltag heißt

- In den Dialogen *SQL Tabelle anlegen*, *SQL Tabelle ändern* und *Squeeze-Stammdaten in Tabelle laden* stehen diese drei Datenbanken gar nicht erst zur Wahl.
- Beim Export des Tabellenaufbaus werden sie nicht angeboten, beim Import bleiben Zeilen, die auf sie zeigen, auf *nicht importieren*.
- Ihre **Definitionen** lassen sich dagegen ganz normal öffnen, bearbeiten und speichern – der Editor ändert nur nichts an ihrem Tabellenaufbau.

Wo die Entscheidung fällt

Im **Portalskript**, bei jedem einzelnen Aufruf – nicht in der Seite. Der Editor ist `clientExecutable`, und ein Schalter aus der Anfrage wäre keine Befugnis, sondern eine Behauptung. Die Anfrage sagt nur, woher sie kommt.

Was die Oberfläche tut, ist deshalb Anzeige und nicht Sperre: sie bietet erst gar nicht an, was danach abgewiesen würde. Ein Plan, der erst beim Ausführen scheitert, ist schlechter als einer, der

die Zeile gar nicht erst anbietet.

Technik, Dateien und Übersetzungen

Diese Seite richtet sich an alle, die den Editor warten oder ausliefern.

Vier Dateien, die zusammengehören

Datei	Inhalt
<code>TableServiceDefinitionEditor.html</code>	die Hülle – kein sichtbarer Text, nur Struktur
<code>TableServiceDefinitionEditor.css</code>	Gestaltung
<code>TableServiceDefinitionEditor.js</code>	Verhalten
<code>TableServiceTranslations.json</code>	die Texte (de/en) und die Vorlage für <code>lang_properties</code>

Dazu das Backend: das Portalskript `DEXPRO__TableServiceDefinitionEditor` mit seinen Teilen für DDL und Schreibzugriffe.

Stil und Skript sind mit Absicht nicht inline

Der DOCUMENTS-Client liefert eine Content-Security-Policy der Form `style-src 'self' 'nonce-...'`. Sie blockiert ein inline `<style>`, ein inline `<script>` und **jedes** `style="..."`-Attribut, auch die zur Laufzeit erzeugten. Als eigene Dateien von derselben Herkunft greift `'self'`.

“ Ein inline `style="..."` irgendwo im JavaScript zerlegt das Layout im Client, ohne eine zweite Fehlermeldung. Deshalb schaltet der Editor Sichtbarkeit über CSS-Klassen.

`?v=` hochzählen

`TableServiceDefinitionEditor.css?v=<n>` und `...js?v=<n>` tragen den Stand der beiden Dateien. Der DOCUMENTS-Client behält Stil und Skript im Cache, und ein Neuladen der Seite holt sie nicht wieder. **Wer `.css` oder `.js` ändert, zählt die Zahl in beiden Zeilen hoch** – sonst läuft die Oberfläche beim nächsten Benutzer noch auf dem alten Stand.

Die `id="cssDocuments"` am Stylesheet wird im Skript gesucht: dort hängt der Editor die Signalfarbe des Mandanten als Regel an. Bitte nicht umbenennen.

Übersetzungen

Jeder sichtbare Text trägt im HTML nur seinen technischen Namen (`data-i18n`, `data-i18n-title`, `data-i18n-placeholder`). Eingesetzt wird er beim Start aus zwei Quellen, in dieser Reihenfolge:

1. **Die Übersetzungstabelle des Clients.** Gelesen wie überall in DEX über `DEX.translate(name, datei)`; gefüllt aus `lang_properties` (`Application Documents`, `PropType PropertiesGentable`, `PropTypeSubCategory TableService`) durch `DEXPRO__TableServiceExportTranslations`. **Was dort steht, gewinnt** – eine Übersetzung lässt sich also ändern, ohne eine Datei anzufassen.
2. **`TableServiceTranslations.json`** neben der Seite, als Rückfallebene und zugleich als Vorlage, aus der das Exportskript die Zeilen schreibt.

Fällt beides aus, bleibt der technische Name stehen – unschön, aber die Oberfläche bleibt bedienbar. Woher die Texte tatsächlich kommen, steht im Tooltip des Abzeichens rechts oben; das ist die erste Frage, wenn alles in einer Sprache erscheint.

Ein **neuer Text** gehört deshalb nie in die HTML-Datei, sondern in den Katalog – und von dort mit `DEXPRO__TableServiceExportTranslations` in die Tabelle.

Die Properties-Datei heißt `Gentable`, weil `getMessages()` nur Dateien im Tomcat-Kontext findet: `DEXPRO\Gentable` hängt in der `documents.xml` unter `/WEB-INF/classes`, und dort liegt `Gentable_<lang>.properties` mit unseren Texten unter `# SubCategory: TableService`.

Definitionsdateien bleiben unangetastet

Die Dateien unter `TableService/data/table-definitions` werden vom Editor **nicht** angefasst. Sie bleiben die Quelle der Auslieferung, und ein erneuter Lauf von `DEXPRO__CreateCustomPropertiesFromAllTableServices` überschreibt alles, was hier geändert wurde.

Für Dauerhaftes also **Exportieren** (siehe [Definitionen importieren und exportieren](#)) und das Ergebnis in die Auslieferung geben.