

Umgebung

- [Mehr → Ordner und Outbar](#)
- [Definitionen importieren und exportieren](#)
- [Geschützte Datenbanken](#)
- [Technik, Dateien und Übersetzungen](#)

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`, `PropTypePropertiesGentable`, `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.