Importer

AID 054 DE

Autor FA, MW, KN. Version 10.8. Zuletzt geändert 08.01.2018

Version	Änderungen
10.8	Anpassung der Beispiele, IdQuery und IdColumn, BatchSize ist ab dem xRM 1709 zum Batching nicht mehr nötig
10.7	Anpassung der Formatierungen
10.6	DoIf erweitert
10.5	Ergänzen der möglichen Importeroptionen
10.4	Hinweis auf den sp_importExample hinzugefügt
10.3	Beispiele bei Eval und DoIf ergänzt, iInsertUpdate aktualisiert
10.2	iTimestamp aktualisiert
10.1	iMove aktualisiert (HTML2Text, RTF2Text)
10.0	iMove aktualisiert
2.2	Beispiele modifiziert und ergänzt Importerfunktionen aktualisiert
2.1	iCommRestriction hinzugefügt
2.0	Letzter Stand vor Übernahme in Versionierung

Inhaltsverzeichnis

1. Einführung

1.1. Der Importer

Folgende Anforderungen waren unter anderem ausschlaggebend für die Entwicklung eines Importmoduls für ADITO4:

- Beliebig große Resultsets importierbar (keine prinzipielle Beschränkung).
- Sinnvolle Fehlerbehandlung (ein Fehler bricht nicht den ganzen Import ab).
- Ausführliches Logging und Debugging (ins Log oder in eine Datei).
- Preview ohne Datenänderung (Simulationsmodus zum Testen).
- Gekapselt, klare Schnittstellen (wartbarer und kommentierter Code).
- Vom Projektmanager erweiterbar (Zusatznutzen ohne Zusatzkosten).

Der Importer besteht aus folgenden ADITO-Objekten:

- `IMPORTER`: Wrapper für die anderen Importer-Bibliotheken.
- `lib_importer`: Kernfunktionen des Importers der auch den Importer-Konstruktor zur Verfügung stellt.
- `lib_importerMappingFunctions`: Standard Mapping-Funktionen für den Importer die bereits mitgeliefert werden.
- `lib_importerCustomMappingFunctions`: Hier können eigene (projektspezifische) Mapping-Funktionen hinterlegt werden.

1.1.1. IMPORTERTEST

Der Prozess `IMPORTERTEST` zeigt alle verschiedenen Möglichkeiten (Mapping (Config) und Optionen), die der Importer bietet. Ein weiteres Beispiel zur Verwendung des Importers finden Sie im `sp_importExample`.

1.2. Ziele bei der Erstellung des Importers

Ein Ziel bei der Erstellung des neuen Importer-Moduls war die möglichst einfache Bedienung und Konfiguration durch Systembetreuer und Projektmanager. Während der Designphase stand auf dem Whiteboard folgender Code als Zieldefinition:

```
var imp = new Importer();
imp.process( democonfig );
```

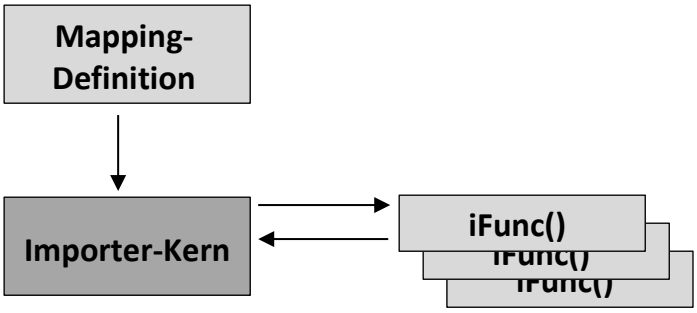
Komplexer sollte die Nutzung des Importermoduls (zusätzlich zur Definition des Importmappings) nach Möglichkeit nicht sein. Ein weiteres Ziel war die möglichst große Flexibilität bei der Definition des Importmappings und die Möglichkeit, den Import in einer Art „Simulationsmodus“ auszuführen, bevor tatsächlich Daten in die Datenbank übernommen werden.

Drittes Ziel war die mögliche Erweiterung des Importers für zukünftige Aufgaben durch die Betreuer eines ADITO4 Systems im Rahmen des vorhandenen Moduls. Diese Erweiterungen sollten nach Möglichkeit unter den einzelnen Systemen bzw. Betreuern austauschbar sein.

Der seit dem Beginn der Entwicklung erreichte Stand erfüllt die obigen Ziele und hat sich bei einigen Projekten bereits im Produktiveinsatz bewährt. Damit erhalten Sie ein flexibles und leicht anpassbares Werkzeug für die Übernahme von Daten aus Drittsystemen nach ADITO4.

1.3. Schematischer Aufbau

Die nachfolgende Grafik zeigt den schematischen Aufbau des Importers. Ein Importerkern wird über eine Mappingdefinition gesteuert, die für jede Zeile aus einem Eingabe-Resultset definiert, wie die Umsetzung von Eingabedaten zu Ausgabedaten erfolgen soll (das sog. mapping).



Über Importfunktionen (sogenannte `iFuncs`) ist das System modular erweiterbar und lässt sich so an verschiedene Bedürfnisse anpassen. Für die Standardfälle bei Datenübernahmen sind bereits alle notwendigen Funktionen vorhanden. Der Importerkern kann zur Laufzeit über verschiedene Optionen gesteuert werden (Logdatei, Simulationsmodus, etc.).

1.4. Definition des Mappings

1.4.1. Ziele für das Mapping

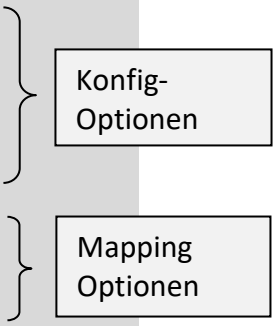
- Objektliterale, gekapselte Definition, keine Abhängigkeiten
- Pro Spalte Importset nur eine Zeile an Definition notwendig
- Selbstdokumentierendes Mapping ohne großen Aufwand
- Benutzbar für Insert und Update, zwischen Systemen übertragbar

1.4.2. Aufbau der Mappingkonfiguration

Die Erstellung einer Mappingdefinition geschieht in Form eines JavaScript-Objektliterals. Damit ist die Definition bereits ohne ausführliche Kommentierung relativ leicht lesbar und die komplette Definition kann in andere Systeme übernommen werden, indem dieser eine Codeblock übertragen wird. Das folgende Listing zeigt beispielhaft eine Mappingdefinition und dient an dieser Stelle nur der Verdeutlichung. Details dieser Definition werden später im Dokument erläutert. Die Mappingdefinition ist selbst nur eine Konfigurationsoption, besitzt aber eigene Konfigurationsmöglichkeiten.

```
var democonfig = {
 DataQuery : "select firmaid, name from firma
              order by firmaid ",
 AliasFrom : "Oracle",
 AliasTo: "AO_DATEN",
 ImportCommand: "insert",

 Mapping : [ [iNewID, { Target: "ORG.ORGID",
                        Key: true } ]
           , [iMove,  { Source: 1,
                        Target: "ORG.ORGNAME" } ] ]
 };
```



Dieses Konfigurationsobjekt wird der Importerinstanz dann als Parameter übergeben. Der eigentliche Start erfolgt durch den Aufruf der Methode `process()` des Importers. Der Importer selbst besitzt ebenfalls eigene Optionen, diese werden weiter unten beschrieben.

2. Arbeiten mit dem Importer

2.1. Anforderungen

2.1.1. Installation

Der Importer ist ein eigenständiger JDito-Prozess, den Sie als XML-Datei erhalten. Prüfen Sie im ADITO4 Designer, ob Sie den aktuellen Importer-Prozess vorliegen haben (Erkennbar, ob @@version vorhanden ist). Wurde Ihr Importer im Projektverlauf NICHT angepasst, so importieren Sie diesen Prozess mit dem ADITO4 Designer in Ihr System. Bitte denken Sie dabei daran, dass Objektnamen in ADITO4 systemweit eindeutig sein müssen. Sollte also bereits ein Objekt mit dem Namen `IMPORTER` in Ihrem System existieren, benennen Sie dieses bitte um.

2.1.2. Notwendige Kenntnisse

Wenn Sie mit dem Importer arbeiten und Daten übernehmen, dann reichen minimale JDito-Kenntnisse für die Erstellung der Mappingkonfiguration aus. Sie sollten allerdings die syntaktischen Anforderungen an JavaScript-Objektliterale kennen, um sicher eigene Mappings erstellen zu können. Falls Sie das Importermodul um eigene Importerfunktionen (`iFuncs`) erweitern wollen, sollten Sie über fundierte Erfahrung in der objektorientierten JavaScript- und JDito-Programmierung verfügen und das ADITO4 Datenbankmodell gut kennen.

Ein Objekt ist in JavaScript ein sogenannter „zusammengesetzter Datentyp“. Hier können Sie verschiedene benannte Eigenschaften in einer einzigen Variable vereinen. Die Eigenschaften in Objekten werden benannt, ein Objekt muss dabei immer vorher als solches definiert werden:

```
var o = new Object();
o.lunch = "Tom Yum";
o.stars = 5;
```

Alternativ können Sie das Objekt auch über die Array-Schreibweise ansprechen. Das Objekt wird mit geschweiften Klammern geöffnet:

```
var o = {
lunch: "Tom Yum",
stars: 5
};
```

2.2. Eingabedaten

Das Importermodul kann pro Durchlauf ein Set aus Eingabedaten importieren. Diese Eingabedaten liegen in Form einer Tabelle aus Zeilen und Spalten vor und können entweder aus einer Datenbankabfrage, einer CSV-Datei oder einem JavaScript-Array stammen. Details zu den möglichen Datenquellen finden Sie weiter unten. Pro Zeile dieser Eingabedaten wird ruft das Importer-Modul einmal den definierten Satz an Mappingfunktionen auf. Wenn Sie mehr als eine Datei oder mehr als eine Abfrage importieren müssen, dann wird für jedes Eingaberesultset eine Mappingkonfiguration angelegt und diese aufgerufen.

2.3. Ausgabedaten

Hauptziel des Importer-Moduls ist die einfache Übernahme von Daten nach ADITO4. Dennoch können Sie auch andere Tabellen damit befüllen. Jede Mappingfunktion kann ein Ergebnis liefern und dieses in einer Ausgabespalte ablegen. Der Ausgabepuffer besteht aus einem Objekt, dessen Eigenschaftsnamen den Tabellenspalten der Zieltabellen entsprechen. Damit der Spaltenname eindeutig ist, wird jedem Spaltenname der Name der zugehörigen Tabelle vorangestellt. Die Namen müssen denen der vorhandenen Datenbankobjekte entsprechen.

Die Importfunktionen benutzen für die Definition der Zielspalte den Parameter `Target` in der Mappingdefinition. Um beispielsweise die Spalte `Produktionsjahr` der Tabelle `Maschine` zu füllen, wird als Ziel `MASCHINE.PRODUKTIONSJAHR` angegeben. Wie Sie erkennen, werden Tabellen- und Spaltennamen wie in SQL durch einen Dezimalpunkt getrennt.

Beachten Sie bei der Angabe der Ausgabespalten die Groß- und Kleinschreibung der Spaltennamen!

2.4. Format der Mappingkonfiguration

Die Mappingdefinition besteht aus einem Objektliteral mit verschiedenen Eigenschaften, die in diesem Abschnitt erläutert werden.

2.4.1. Minimaldefinition (Beispiel)

```
var democonfig = {
 DataQuery : "select firmaid, name from firma where active = 'Y' ",
 AliasFrom : "Oracle",
 AliasTo: "AO_DATEN",
 ImportCommand: "insert",

 Mapping : [
       [iNewID, { Target: "ORG.ORGID",
                  Key: true }],
       [iMove,  { Source: 1,
                  Target: "ORG.ORGNAME" }]
      ]
 };
```

Die Zuordnung von Eingabe- zu Ausgabedaten wird über ein Array von Mappingfunktionen definiert. Diese werden in der Eigenschaft `Mapping` angegeben. Jeder Eintrag ist ein Array aus zwei Elementen, einer Mappingfunktion und einem Objektliteral mit der Angabe der Parametereigenschaften und ihren Werten. Das folgende Fragment zeigt ein Beispiel für eine solche Angabe.

```
 Mapping :  [ [iNewID, { Target: "ORG.ORGID",
                         Key: true  }]
            , [iMove,  { Source: 1,
                         Target: "ORG.ORGNAME" }] ]
```

Details zu den einzelnen Importfunktionen finden Sie weiter unten im Referenzteil dieses Dokuments. Die einzelnen Mappingfunktionen werden pro Datensatz aus dem Eingaberesultset nacheinander aufgerufen. Über die Positionierung der Funktionen können Sie also bei Bedarf die Verarbeitungsreihenfolge steuern.

2.5. Festlegen des Input-Resultsets (Konfig-Optionen)

Der Importer kann die für die Übernahme bestimmten Daten je Mappingkonfiguration aus einer Datenquelle übernehmen. Mögliche Datenquellen sind:

- SQL-Abfrage
- CSV-Datei
- JavaScript-Array
- XML-Datei oder XML-String

Abhängig von der gewählten Datenquelle wird das Resultset für die Eingabedaten über die folgenden Eigenschaften definiert, die nachfolgend noch genauer erläutert werden.

Eigenschaft	Beschreibung
DataQuery	Die SQL-Abfrage zur Erzeugung des Eingabe-Resultsets.
DataFile	Gibt den kompletten Pfad zu einer CSV-Datei an (nicht verwendbar mit DataQuery).
DataFunction	Verweist auf eine JavaScript-Funktion, die ein zweidimensionales Array mit den zu übernehmenden Daten liefert.

2.5.1. DataFile (Import aus einer CSV-Datei)

Verwenden Sie diese Angabe, um Daten aus einer CSV-Datei zu übernehmen. Der Inhalt der Eigenschaft DataFile ist der komplette Pfadname zur CSV-Datei. Geben Sie Pfadtrennzeichen immer mit einem Schrägstrich an (Unix-Konvention), dies funktioniert auch unter Windows (z.B. `c:/temp/import.csv`). Wenn Sie diese Datenquelle verwenden, dann müssen Sie noch die folgenden Optionen angeben, um das Format der Datei zu definieren.

`HeaderLines`	Anzahl der Kopfzeilen, die übersprungen werden sollen (z.B. `1`).
`RowSeparator`	Gibt den Zeilentrenner an (z.B. `"\n"`).
`ColumnSeparator`	Gibt den Spaltentrenner an (z.B. `";"`).
`StringDelimiter`	Gibt das Begrenzungszeichen für Zeichenketten an (z.B. `" ' "`).

Beispiel

```
var fileconfig = {
 AliasTo: "AO_DATEN",
 DataFile: "C:/Daten/CurrentProject/Kundenstamm-Streit.csv",
 HeaderLines: 1,
```

```
 RowSeparator: "\n",
 ColumnSeparator: ";",
 StringDelimiter: "",
```

2.5.2. XML (Import aus XML-Datei oder XML-String)

Mit dieser Angaben können XML-Dateien, die ein bestimmtes Format haben, ebenfalls wie eine Tabelle importiert werden. Das XML-Format definiert sich folgendermaßen:

```
<data>
 <row>
  <column><![CDATA[ADITO Software GmbH]]></column>
  <column><![CDATA[Gutenbergstr. 1]]></column>
  <column><![CDATA[84144]]></column>
  <column><![CDATA[Geisenhausen]]></column>
 </row>
</data>
```

Hierbei kann das Mapping wie bei einem Tabellenimport aufgebaut werden. Es kann hier auf die column-Tags mit Source zugegriffen werden. Die Werte der einzelnen Spalten, sollten in `CDATA` stehen, da es sonst vorkommen kann, dass der Prozess das XML-File und aber auch die Daten nicht vollständig oder korrekt lesen kann und es zu Fehlern führen kann.

Um diese Datenquelle verwenden zu können, müssen Sie noch folgende Optionen angeben, um das Format der Datei zu definieren.

Beispiel (XML von Datei laden):

```
var xmlconfig = {
 AliasTo: "AO_DATEN",
 DataFile: "C:/Daten/CurrentProject/Kunden.xml",
```

Beispiel (XML von String laden):

```
var xmlconfig = {
 AliasTo: "AO_DATEN",
 XMLObject: xmlString,
```

2.5.3. DataQuery (Import aus einer SQL-Abfrage)

Falls Sie Daten über eine SQL-Abfrage übernehmen, wird in der Eigenschaft `DataQuery` das SELECT-Statement für das Eingaberesultset angegeben. Damit die Daten auch bei größeren Tabellen übernommen werden können, wird die Abfrage in einzelne Etappen (*batches*) aufgeteilt. Das Batching wird über die Kernmethode `db.tablePage` vorgenommen. Um das Batching vornehmen zu können, muss in der DataQuery ein `order by` (auf eine eindeutige ID) enthalten sein.

`AliasFrom`	Aliasname für die Eingabedaten aus `DataQuery`.
`IdQuery`	Die SQL-Abfrage zur Erzeugung der Liste der einzulesenden Primärschlüsselwerte (nur verwendbar mit `DataQuery`). Werden `IdQuery` und `IdColumn` verwendet, wird das Batching nicht über die Kernmethode `db.tablePage` vorgenommen.

`IdColumn`	Die Spalte, die den Primärschlüssel für die Daten enthält. Werden `IdQuery` und `IdColumn` verwendet, wird das Batching nicht über die Kernmethode `db.tablePage` vorgenommen.

Beispiel

```
DataQuery: "select firmaid, name1, name2, strasse, web, tel, plz"
         + " from firma order by firmaid "
```

2.5.4. DataFunction (Import aus einem JavaScript-Array)

Als dritte Möglichkeit können Sie dem Importer ein selbst erzeugtes Array von Daten übergeben, die verarbeitet werden sollen. Dazu wird die Eigenschaft `DataFunction` mit einer Funktionsvariable oder einem Funktionsliteral belegt. Diese Funktion muss ein zweidimensionales Array zurückliefern (ein Array, bei dem jedes Element wiederum aus einem Array besteht).

Als Parameter bekommt diese Funktion vom Importermodul zwei Parameter übergeben. Der erste Parameter enthält die Nummer des Batches, der zweite die Anzahl Datensätze, die pro Batch geliefert werden sollen. Sobald keine Daten mehr zur Verfügung stehen, muss die Funktion den Wert `null` zurück liefern. Falls die Funktion alle Daten in einem Aufruf liefert, prüfen Sie einfach, ob die Nummer des aktuellen Batches höher ist als 1 und liefern dann `null` zurück, wie im folgenden Beispiel.

```
function myImportSet(pBatchNum, pBatchSize)
{
   if(pBatchNum > 1) return null;
   var result = ...  // code, der das array füllt
   return result;
}
```

Im Objektliteral mit der Mappingdefinition wird diese Funktion dann verwendet:

```
DataFunction: myImportSet,
```

Falls die Funktion kurz genug ist, um die Lesbarkeit nicht zu stören und Sie keine externen Abhängigkeiten in der Definition haben wollen, kann die Funktion auch direkt angegeben werden (auch als anonyme Funktion).

```
// beispiel
var arrayconfig = {
   DataFunction :
      function (pBatchNum, pBatchSize)
      {
         if(pBatchNum > 1)
            return null;
         else
            return [ ["0", "Zeile 0", 42, date.date() ]
                   , ["0", "Zeile 0", 42, date.date() ] ];
      } ,
   AliasTo: "AO_DATEN",
```

```
   ImportCommand: "insert",
   ...
```

2.5.5. ImportCommand (Importmodus)

Über die Eigenschaft `ImportCommand` wird definiert, ob die zu importierenden Daten als neue Datensätze per Insert hinzugefügt oder per Update aktualisiert werden sollen. Als mögliche Werte sind die Strings `insert`, `update` oder `insert+update` möglich.

Bei `insert` werden alle Datensätze in die ADITO4 Datenbank übernommen. Bei `update` wird nach den Schlüsselwerten gesucht und falls diese vorhanden sind, werden die Daten aktualisiert. Die Einstellung `insert+update` fügt nicht vorhandene Datensätze hinzu und aktualisiert Datensätze, deren ID bereits im System vorhanden ist. Hierbei wäre auch der optionale Parameter `Action` in den Mappingfunktionen zu beachten!

2.5.6. AliasTo (Zielalias für die ADITO4 Daten)

Diese Eigenschaft enthält den Namen des Alias, in dem die Tabellen von ADITO4 liegen.

2.6. Steuerung des Importers

Der Importer kann zur Laufzeit über verschiedene Optionen gesteuert werden. Damit ist beispielsweise die Ausgabe von Logmeldungen in eine Datei oder das Serverlog möglich, und es kann der Simulations- oder Vorschaumodus aktiviert werden und mehr.

Das folgende Listing zeigt den Einsatz einzelner Laufzeitoptionen für den Importer.

```
var imp = new Importer( importconfig );
imp.Log = "c:/tmp/importlog.txt";
imp.LogLevel = imp.LogLevels.Info;
imp.Preview = true;  //default ist FALSE
imp.Debug = true;    //default ist FALSE
imp.MaxRows = 10;    //Maximale Anzahl der Zeilen,
                       die verarbeitet wird
                     //Für Testzwecke
imp.process();
```

2.6.1. Liste der Importeroptionen

Die folgende Tabelle enthält die Liste der Optionen, mit denen der Importer zur Laufzeit gesteuert bzw. konfiguriert werden kann. Neben dem Optionsnamen ist der Eingabeparameter beschrieben.

Option	Bedeutung
Preview (true / false)	Schaltet den Vorschaumodus ein bzw. aus, bei dem auch einige Informationsmeldungen angezeigt werden, der Default ist `false`.
DebugCallback (function)	Falls definiert, wird diese Funktion vor dem Schreiben jedes übernommenen Datensatzes (und vor dem Generieren des Output-Buffers) ausgeführt. Unabhängig von Debug nutzbar.

Option	Bedeutung
ProgressCallback (function)	Falls definiert, wird diese Funktion nach dem Schreiben jedes Datensatzes ausgeführt. Unabhängig von Debug nutzbar. Erhält als Parameter die Information, ob der Datensatz übersprungen wurde, weil Probleme auftraten oder nicht (`true` oder `false`).
Log (string filename)	Definiert den Pfadnamen für die Protokolldatei. Falls nicht definiert oder auf einen leeren String gesetzt, erfolgt die Ausgabe in das Log des Servers. Wurde die Option enableLogBuffer gesetzt wird jede Logzeile zusätzlich in einem Array zwischengespeichert, dass auch nach dem Ausführen des Imports mit Hilfe von `getLogMessages` ausgelesen werden kann. Nicht empfohlen dies da sehr speicherintensiv ist. Mögliche Einstellungen: - `CONSOLE`: Loggt die Ausgabe in die Serverausgabe. Je nach `PREFERENCES` an alle entsprechenden Stellen wie z.B. Remote-Logger, File-Log, Telnet-Log, usw. - `LOGFILE`: Siehe „`CONSOLE`“. Aus Kompatibilitätsgründen noch enthalten. - `$global.logmsg`: Schreibt die Log-Ausgabe in eine globale Variable. - „“ (Leerstring): Es wird nichts geloggt.
ImportUser (string login)	Enthält den Text, der als `USER_NEW` bzw. `USER_EDIT` in die Tabellen von ADITO4 eingetragen werden soll.
LogLevel (enum)	Mögliche Einstellungen (in aufsteigender Reihenfolge): - `LogLevels.Minimal (0)` - `LogLevels.Error (1)` - `LogLevels.Warning (2)` - `LogLevels.Info (3)` - `LogLevels.Debug (4)` - `LogLevels.Preview (5)` `Default = LogLevels.Warning`
CompleteUpdate (true / false)	`CompleteUpdate` gibt an, ob geprüft werden soll, ob sich in den verschiedenen Spalten etwas geändert hat oder ob für die gesamten Spalten ein Update gemacht werden soll. Default = true

Option	Bedeutung
MaxRows (integer)	Gibt an, wieviele Datensätze maximal verarbeitet werden sollen.
LogCallback (function)	Diese Callback-Funktion erlaubt es, das komplette Logging zu übernehmen. (Wird für jeden Aufruf von `writeLog` aufgerufen.) Ist eine `function` angegeben werden die anderen Logmethoden übersprungen. Die Callback-Funktion erhält 2 Parameter: - Loglevel, das angefragt wird. Hier wird ein Wert aus „LogLevels“ (siehe oben) übergeben. - LogText, die Meldung die geloggt werden soll.
UseAOType (true / false)	Default ist `true`, wenn gesetzt wird die Spalte `AOTYPE` in der Tabelle `RELATION` verwendet, bei `false` wird der Relationstyp aus `PERSID` und `ORGID` selbst ermittelt. `Default = true`
UseUUID (true / false)	Default ist false, wenn gesetzt wird für die Ermittlung einer ID die JDito-Methode `util.getNewUUID()` verwendet, anderenfalls die Methode `util.getNewID().true` ist hierbei performanter als `false`, beachten Sie aber, dass in diesem Fall die ID-Spalten in der Datenbank vom Datentyp char(36) sein müssen.
TableCase (enum)	Definiert, wie die Tabellen in der Datenbank geschrieben sind. Mögliche Einstellungen (in aufsteigender Reihenfolge): - `Cases.Lower (0)` - `Cases.Upper (1)`
ColumnCase (enum)	Definiert, wie die Spalten in der Datenbank geschrieben sind. Mögliche Einstellungen (in aufsteigender Reihenfolge): - `Cases.Lower (0)` - `Cases.Upper (1)`
useAttributeCache	`true`: Anstatt jedesmal das Attribut mithilfe eines SQL-Statements aufzulösen (Wert `false`), werden alle Attribute einmal am Anfang geholt und im Speicher gehalten. Bei `iAttribute` wird `Autocreate` ignoriert.
Debug	Wenn true, werden Debugging-Meldungen ins Log ausgegeben. So können Fehler leichter gefunden werden . Default = `false`
insertArray	Interner Wert.
updateArray	Interner Wert.

Option	Bedeutung
Config	Interner Wert mit Importer-Konfiguration (Mapping usw.)
fileInputCharset	Gibt das Charset der Input-Datei an. Standard ist „UTF8“.
enableLogBuffer	Wenn `true` wird beim Loggen in jede Logzeile in einem Array zwischengespeichert, sodass am Ende mit Hilfe von `getLogMessages` alle Logmeldung geholt werden können. Nicht empfohlen da speicherintensiv.
InputRecord	Interner Wert.
OutputRecord	Interner Wert.
FuncBuffer	Ein globaler Zwischenspeicher für alle Mapping-Funktionen.
DataType	Interner Wert mit allen Datentypen aller Spalten.
KeyColumn	Interner Wert für Ermittlung der Update-Schlüssel.
skipEmptyValue	Wenn `true` (Standard), werden leere Input-Werte ignoriert (sinnvoll für den Anwendungsfall: Nur geänderte Datenbankfelder bekommen). Wenn `false` werden bei leeren Input-Werten diese auch leer geschrieben (sinnvoll für den Anwendungsfall: Komplette Zeile bekommen).
attributeCacheLoadedSuccessfully	Interner Wert, ob der Attribute-Cache aufgebaut werden konnte.
loadAttributeCache	Diese Funktion baut den Attributcache neu auf. Sie wird implizit in der process-Methode aufgerufen, wenn `useAttributeCache` auf `true` gesetzt ist.
recordCounts	Interner Wert .
showTimings	Methode um die Dauer der einzelnen Aktionen zu ermitteln. Wird implizit aufgerufen.
showCounts	Methode um die Anzahl geänderter Daten anzuzeigen. Wird implizit aufgerufen.
getNextBatch	Interne Methode.
createIDList	Interne Methode, `deprecated`.
writeLog	Methode zum Schreiben eines Log-Eintrages. Bei eigenen Mapping-Funktionen sollte immer diese Methode verwendet werden und kein `logging.log`.
getLogMessages	Gibt den Logbuffer aus, wenn dieser aktiviert wurde. Speicherintensiv.
isClientProcess	Methode, die angibt ob es sich um einen Client-oder um einen Server-Prozess handelt .

Option	Bedeutung
getFileContent	Interne Methode zum Laden von Inputdateien in einem Block (kein `BulkReader`).
dumpRecord	Interne Methode.
insertData	Interne Methode.
updateData	Interne Methode.
setDefaultAction	Interne Methode.
resolveSymbol	Methode zum Auflösen von Platzhaltern im Mapping. So kann bspw. bei `>>Value: "AA_{3}"<<` die Quellspalte 3 aufgelöst werden.
getDataTypes	Interne Methode.
setOutput	Setzt in Mapping-Funktionen den Ergebniswert, der in das Target geschrieben wird.
getOutput	Interne Methode.
generateKeyCondition	Liefert eine Update-Condition anhand der Key-Spalten.
getTableCase	Methode um eine Tabelle in uppercase/lowercase zu wandeln (je nach Einstellung). Muss für alle Datenbankaktionen verwendet werden.
getColumnCase	Methode um eine Spalte in uppercase/lowercase zu wandeln (je nach Einstellung). Muss für alle Datenbankaktionen verwendet werden.

2.6.2. getLogMessages()

Mit der Funktion `getLogMessages()` " ist es möglich, die `LogMessages` aus dem Importer zu erhalten, um sie später weiterverarbeiten zu können.

Beispiel

```
var logMessages = imp.getLogMessages();
```

Die Eigenschaft `enableLogBuffer = true` sollte aktiviert sein. Sie kann unter Umständen zu einer kritischen Performance führen.

2.6.3. Callback-Funktionen

Für das Debugging und die Überwachung des Importfortschritts können sogenannte Callback-Funktionen definiert werden. Diese werden dem Importer als Parameter in der Mappingkonfiguration übergeben.

Möchten Sie beispielsweise nach jedem importierten Datensatz eine Aktion ausführen oder einen Fortschritt anzeigen, dann weisen Sie der Eigenschaft `ProgressCallback` eine

Funktion zu. Diese Funktion erhält keine Parameter und wird nach jedem verarbeiteten Datensatz aufgerufen.

Analog dazu können Sie die Eigenschaft `DebugCallback` verwenden. Die hierfür definierte Funktion bekommt als Parameter eine Referenz auf den aktuellen Ausgabepuffer übergeben, so dass Sie dort eigene Diagnosemeldungen erzeugen können. Das folgende Codefragment zeigt den Einsatz dieser Eigenschaft.

```
function myprogressFunction() {
  log.log("Pling! Wieder eine Zeile importiert.");
}

// create new importer instance
var imp = new Importer( arrayconfig );

// set processing options
imp.Preview = false;
imp.Debug = true;
imp.ProgressCallback = myprogressFunction;

// process import configuration!
imp.process();
```

Der Parameter für die Debug-Callbackfunktion ist eine Referenz auf den Ausgabepuffer. Dabei handelt es sich um ein Objekt, dessen Eigenschaftsnamen den Spaltenbezeichnungen entsprechen. Weitere Details siehe den Abschnitt „Erweitern des Importers“ weiter unten.

Bedingt durch die Einschränkungen von JavaScript kann auf diesen Ausgabepuffer geschrieben werden. Bitte achten Sie darauf, auf den Puffer nur lesend zuzugreifen und die über diese Referenz erhaltenen Daten nicht zu verändern!

2.7. Beispiele

2.7.1. Import aus CSV-Datei

Das folgende Beispiel zeigt den Import aus einer CSV-Datei mit den folgenden Spalten:

```
MANDT;KUNNR;LAND1;NAME1;NAME2;ORT01;PSTLZ;REGIO;SORTL;STRAS;TELF1;MA
RKER
```

Hier der entsprechende Import in die Tabellen `ORG` und `RELATION`:

```
var fileconfig = {
  AliasTo: "AO_DATEN",
  DataFile: "C:/Daten/CurrentProject/Kundenstamm.csv",
  HeaderLines: 1,
  RowSeparator: "\n",
  ColumnSeparator: ";",
  StringDelimiter: "",
  ImportCommand: "insert",
```

```
 Mapping : [
   [iNewID, { Target: "RELATION.RELATIONID",
              Key: true }]
 , [iNewID, { Target: "ORG.ORGID",
              Key: true }]
 , [iJoin,  { Source: [3, 4],
              Delimiter: " ",
              Target: "ORG.ORGNAME" }]
 , [iMove,  { Source: 9,
              Target: "RELATION.ADDRESS" }]
 , [iMove,  { Source: 6,
              Target: "RELATION.ZIP" }]
 , [iMove,  { Source: 5,
              Target: "RELATION.CITY" }]
 , [iMove,  { Source: 2,
              Target: "RELATION.COUNTRY" }]
 , [iMove,  { Source: 1,
              Target: "ORG.CUSTOMERCODE" }]
 , [iMove,  { Value: "1",
              Target: "RELATION.AOTYPE" }]
 , [iMove,  { Value: "1",
              Target: "RELATION.ADDR_TYPE" }]
    ]
};
```

2.7.2. Update von Daten aus einer CSV-Datei

```
var fileupdater = {
   AliasTo: "AO_DATEN",
   DataFile: "C:/tmp/ctilog.csv",
   SkipRows: 1,
   RowSeparator: "\r\n",
   ColumnSeparator: ";",
   StringDelimiter: "'",
   ImportCommand: "insert+update",

   Mapping : [ [iNewID, { Target: "CTILOG.CTILOGID" } ]
             , [iMove,  { Source: 0,
                          Target: "CTILOG.ADDRESS" } ]
             , [iMove,  { Source: 1,
                          Target: "CTILOG.ANSWERMODE",
                          Key: true } ]
             , [iTimestamp, { Source: 2,
                              Target: "CTILOG.DATE_NEW",
                              Format: "yyyy-MM-dd HH:mm:ss" } ]
             ]
};
```

2.7.3. Komplexeres Beispiel mit Import aus einer Datenbank

```
var democonfig = {
   DataQuery : "select firmaid, name1, name2, strasse, region_neu,
                istdepotbank, web, tel, plz from firma order by
                firmaid ",
   AliasFrom : "Oracle",
   AliasTo: "AO_DATEN",
```

```
   ImportCommand: "insert",

   Mapping : [
   [iNewID,    { Target: "RELATION.RELATIONID",
                 Key: true } ]
 , [iNewID,    { Target: "ORG.ORGID",
                 Key: true } ]
 , [iJoin,     { Source: [1, 2],
                 Delimiter: "\n",
                 Target: "ORG.ORGNAME" } ]
 , [iMove,     { Source: 3,
                 Target: "RELATION.ADDRESS" } ]
 , [iKeyword,  { Source: 4,
                 Keytype: "bundesland",
                 Target: "RELATION.STATE" } ]
 , [iCopy,  { Source: "ORG.ORGID",
              Target: "RELATION.ORG_ID" } ]
 , [iMove,  { Value: "1",
              Target: "RELATION.AOTYPE" } ]
 , [iMove,  { Value: "IMPORTER",
              Target: "RELATION.USER_NEW" } ]
 , [iMove,  { Value: "IMPORTER",
              Target: "ORG.USER_NEW" } ]
 , [iMove,  { Value: "1",
              Target: "RELATION.ADDR_TYPE" } ]
 , [iSql,  {  Command: "select ort from firma where firmaid = {0}",
              Alias: "Oracle",
              Target: "RELATION.CITY" } ]
 , [iAttribute, { Source: 5,
                  Attribute: "betreuung|depotbank",
                  Type: 1,
                  Rowid: "{RELATION.RELATIONID}" } ]
 , [iWord,  { Source: 1,
              Regex: /\s+/,
              Target: "RELATION.BUILDINGNO",
             Index: 8 } ]
 , [iComm,  { Source: 6,
              Medium: 4,
              Rowid: "{RELATION.RELATIONID}" } ]
 , [iComm,  { Source: 7,
              Medium: 1,
              Rowid: "{RELATION.RELATIONID}",
              Standard: true } ]
 , [iInsert,  {  Table: "regtab",
                 Alias: "AO_DATEN",
                 Columns: [ {Name: "REGTABID", Key: true}
                          , {Name: "USER_NEW", Value: "IMPORTER"}
                          , {Name: "AOROLE_ID", Value: "42"}
                          , {Name: "REGTAB", Source: 8} ]
              } ]
    ]  };
```

3. Referenz

Dieser Teil des Dokuments beschreibt die Standard-Importfunktionen, die im Modul bereits enthalten sind und sofort genutzt werden können.

Die folgenden Abschnitte sind sehr wichtig und zum Verständnis des Importers notwendig!

3.1. Allgemeine Parameter für Konfig-Optionen

Jede Mappingfunktion benötigt im Normalfall Eingabedaten, die verarbeitet und in eine Ausgabespalte geschrieben werden. Die Ausgabespalte wird üblicherweise über den Parameter `Target` angegeben und enthält eine eindeutige Spaltenbezeichnung in der Form `TABELLE.SPALTE` als String.

Die Eingabedaten kommen entweder aus einer Spalte des Eingabe-Resultsets oder werden aus einer vorher bereits belegten Ausgabespalte wieder ausgelesen und wiederverwendet (beispielsweise, um die zuerst erzeugte `OrganisationsID` in einen Relationsdatensatz zu schreiben). Alternativ kann auch literaler Text als Eingabedaten verwendet werden (z.B. um als `USER_NEW` immer `System` zu erhalten). Für das Lesen der Eingabedaten stehen zwei Parameter zur Verfügung: `Source` und `Value`.

3.1.1. Source

`Source` enthält eine Ganzzahl und gibt den Spaltenindex im Eingaberesultset an. Dieser Index beginnt die Zählung bei 0! Die Angabe `Source: 4` Bezeichnet also die fünfte Spalte der Eingabedaten.

3.1.2. Value

`Value` ist mächtiger als `Source`, benötigt dafür aber auch mehr Verarbeitungszeit. Hier können Sie als String in Anführungszeichen literalen Text angeben oder Platzhalter benutzen, die in geschweifte Klammern eingeschlossen werden. Handelt es sich dabei um eine Ganzzahl, ist das Ergebnis identisch mit der Verwendung von `Source`. Handelt es sich dabei um eine Angabe in der Form `TABELLE.SPALTE`, dann wird der Inhalt der angegebenen Ausgabespalte ausgelesen und kann wieder verwendet werden. Dazu muss diese Spalte natürlich in einer früheren Mappingfunktion bereits gefüllt worden sein! Der große Vorteil von `Value` liegt aber darin, dass Sie diese drei Varianten miteinander mischen können. Folgende Angabe ist zulässig und fügt an die Spalte mit dem Index 3 jeweils den String „-DE“ an: `Value: "{3}-DE"` (aus `4711` wird dann `4711-DE`). Viele nötige Änderungen lassen sich bereits damit während des Importmappings durchführen. Benötigen Sie diese Funktionalität nicht, ist die Verwendung von `Source` die bessere Wahl, da diese Methode deutlich schneller ist.

3.1.3. DoIf

`DoIf` enthält eine Javascript-Bedingung oder eine Callback-Function. Aufgrund dieser wird entschieden, ob für diesen Datensatz eine Aktion ausgeführt wird oder nicht.

Beispiele

```
[iCommRestriction, {DoIf: "'{17}' != ''",
                    Medium: "{17}",
                    RelationID: "{var.NEWPERSRELID}"
                   }
]
```

```
 [iInsertUpdate, {Table: "OFFER",
                  Condition: "OFFER = '{0}' ",
                  ForceAction: "update",
                  DoIf: "'{0}' != '' ",
                  Columns: [
                  {Name: "STATUS", Value: 1}
                  ]
                  }
 ]
```

```
[iNewID, {Target: "PERS.PERSID",
          DoIf: "'{PERS.PERSID}' == ''"}
]
```

3.1.4. Eval

`Eval` gibt an, ob der Wert in `Source` oder `Value` als Javascript-Ausdruck verwendet wird. Das Ergebnis der Ausführung von Source wird als `Target` verwendet.

```
 [iMove, { Value: 'new AddrObject( "{11}", "{12}"
                                  ).formatAddress()',
           Target: "OFFER.ADDRESS",
           Eval: true
          }
 ]
```

3.1.5. Required

Bei jeder Mappingfunktion kann auch als Parameter `Required` angegeben werden. Mögliche Werte sind die beiden logischen Konstanten `true` bzw. `false`. Wird `true` angegeben und ist für die Mappingsfunktion über `Target` eine Zielspalte definiert, dann wird für den Import geprüft, ob diese Spalte gefüllt ist, d.h. einen Wert enthält, der nicht `NULL` und nicht der Leerstring ist. Nur dann erfolgt die Übernahme in die Zieldatenbank.

3.1.6. Action

Wenn das `ImportCommand` `"insert+update"` ist, kann bei jeder Mappingfunktion auch eine `Action` mit den möglichen Werten `"I"`, `"U"` oder `"I+U"` angegeben werden. Bei dem `ImportCommand` `"insert"` oder `"update"` ist die `Action` nicht notwendig und wird auch nicht berücksichtigt!
Mit dem Parameter kann in einer Mappingfunktion gesteuert werden, ob die Funktion z.B. nur für Update, nur für Insert oder für beides ausgeführt werden soll. `Action: "I+U"` prüft, ob der Datensatz bereits vorhanden ist und bei Erfolg wird dieser aktualisiert, andernfalls wird dieser neu angelegt.

Beispiel

```
ImportCommand: "insert+update"

Action: "I":  es wird immer nur ein Insert gemacht

Action: "U":  es wird immer nur ein Update gemacht (wenn es den
              Datensatz gibt)

Action: "I+U":  es wird geprüft, ob es den Datensatz gibt und wenn
                dieser vorhanden ist, ein Update gemacht,
                andernfalls ein Insert
```

3.1.7. DATE_NEW, USER_NEW, DATE_EDIT, USER_EDIT

Es ist nicht mehr nötig, die Spalten `DATE_NEW`, `USER_NEW`, `DATE_EDIT` und `USER_EDIT` mit in das Mapping aufzunehmen. Diese werden, je nach Art des Imports (bei `Insert` nur `DATE_NEW` und `USER_NEW` und bei Update nur `DATE_EDIT` und `USER_EDIT` (vorausgesetzt, es hat sich beim Update etwas geändert) automatisch gesetzt (wenn diese vier Spalten in der Tabelle vorhanden sind).

Bei Bedarf können diese Spalten jedoch auch im Mapping angegeben werden. Dann werden sie mit den Werten aus dem Mapping gefüllt.

3.2. Speichern von Funktionsergebnissen

Falls Sie Ergebnisse aus einer Mappingfunktion zwischenspeichern wollen, können Sie als Ausgabespalte (`Target`) eine Spalte angeben, die als Tabellenkennung `var` besitzt (z.B. `var.ADDR`). Alle Spalten, die der „Tabelle“ `var` angehören, werden nicht in die Datenbank geschrieben, sondern dienen nur als Variable während des Importlaufs. Daraus folgt, dass Sie nicht in eine Zieltabelle mit dem Namen `var` in der Datenbank importieren können (diese Beschränkung fällt aber kaum ins Gewicht, da dieser Name normalerweise nicht vergeben wird). Damit haben Sie eine Möglichkeit, Verarbeitungen über mehrere Importfunktionen hinweg zu realisieren, ohne dass der normale Einsatz des Importers unnötig komplex wird.

Das folgende Beispiel zeigt einen Ausschnitt aus einer Mappingkonfiguration eines realen Anwendungsfalls. In den zu übernehmenden Daten steht die Adresszeile aus Strasse und Hausnummer in einer Datenbankspalte mit dem Typ `CHAR(64)`. Diese Daten sollen in zwei Spalten `ADDRESS` und `BUILDINGNO` der Tabelle `RELATION` übernommen werden. Die Hausnummer wird dabei einfach durch Abschneiden des letzten, durch Leerzeichen getrennten Wortes extrahiert.

Zuerst wird über die Funktion `iMove` die Spalte mit der Adresszeile (die hier den Index 4 besitzt) von nachfolgenden Leerzeichen befreit und in der Zielspalte `var.ADDR` zwischengespeichert. Als zweite Funktion wird über `iWord` der linke Teilstring bis zum vorletzten Wort (die Straße) extrahiert und in `RELATION.ADDRESS` gespeichert. Als dritte Aktion schließlich wir über `iWord` das letzte Wort aus `var.ADDR` geholt und in `RELATION.BUILDINGNO` gespeichert.

```
    :
    :
, [iMove, { Source: 4,
            Target: "var.ADDR",
```

```
                Trim: "right" } ]

  , [iWord, { Value: "{var.ADDR}",
              Target: "RELATION.ADDRESS",
              Index: -2,
              Regex: " ",
              Substring: "left" } ]

  , [iWord, { Value: "{var.ADDR}",
              Target: "RELATION.BUILDINGNO",
              Index: -1,
              Regex: /\s+/ } ]
    :
    :
```

3.3. Fehlerbehandlung / Überspringen von Daten

Jede Importfunktion liefert im Normalfall als Funktionsergebnis den logischen Wert `true` zurück. Falls innerhalb der Funktion keine erfolgreiche Verarbeitung der Importdaten möglich ist, kann die Funktion den Wert `false` liefern. Dies bewirkt das Überspringen des aktuellen Datensatzes der Importdaten. Ebenso wird der aktuelle Importdatensatz übersprungen, wenn während des Ablaufs der Importfunktion ein Fehler (eine Exception) auftritt. Unterscheiden lassen sich die beiden Fälle in der Protokolldatei durch die verschiedenen Einträge.

Liefert eine Importfunktion `false` als Ergebnis, lautet der Protokolleintrag „`Import function <name> failed for row …`“, während bei einem Fehler die Meldung „`Exception in mapping function <name> for input row …`“ lautet.

Der Importer liefert nach dem Durchlauf eine Summenzeile in der Form „`Processed #### input rows. Skipped #### input rows`“ in der Protokollausgabe.

3.4. Importfunktionen

3.4.1. iDecode(pObject)

Beschreibung

Diese Funktion dient dem Ersetzen von Eingangswerten durch andere Werte. Dazu wird in einer Liste aus Werten, die abwechselnd die gesuchten und zu liefernden Wert enthält, nach einem Treffer gesucht. Dieser wird dann in die Ausgabespalte geschrieben. Die Funktion arbeitet analog zu der Oracle-Datenbankfunktion `DECODE`.

Parameter	
Value	Die Angabe der Eingangsdaten, kann Platzhalter für die Importspalten (`Format {#}`) oder die Ausgabespalten (`Format {tbl.col}`) enthalten.
Target	Die Definition der Zielspalte.

Parameter	
List	Gibt die Liste als Zeichenkette an, getrennt durch Semikola. Jeweils der erste Wert eines Paares wird gesucht und bei einer Übereinstimmung durch den zweiten ersetzt.
Action (opt.)	Die entsprechende `Action`, für die die Mapping-Funktion ausgeführt werden soll.

Beispiel

```
// falls in ORG:OCUNTRY "deutschland" steht, wird "DE" geliefert
   etc.
[iDecode, { Value: "{ORG.COUNTRY}",
            List: "frankreich;FR;deutschland;DE;italien;IT",
            Target: "RELATION.COUNTRYCODE",
            Action: "I" }]

[iDecode, { Value: "{4}",
            List: "0;N;1;Y",
            Target: "ORG.ACTIVE" }]
```

`DECODE` ist in Oracle das Äquivalent zu einer if-then-else-Anweisung.

3.4.2. iIgnore(pObject)

Beschreibung

Die einfachste Funktion, da sie keinerlei Aktion besitzt und nur der Dokumentation nicht übernommener Felder dient.

Parameter

Keine. Alle Parameter, die evtl. definiert sind, werden ignoriert.

Beispiel

```
[iIgnore, { Source: 8,
            Doc: "Dieser Parameter wird ignoriert" }]
```

3.4.3. iInsertUpdate(pObject)

Beschreibung

Erzeugt einen neuen oder aktualisiert einen vorhandenen Datensatz in der angegebenen Tabelle, der mit den Werten aus der Definition in der Eigenschaft `Columns` gefüllt wird.

Die Definition der Spaltenwerte erfolgt jeweils durch ein Objektliteral. In der Eigenschaft `Name` wird der Name der Spalte angegeben. Der Inhalt der Spalte kann über drei verschiedene Wege gefüllt werden.

- Die Angabe von `Source` und einer Quellspalte aus dem Input-Resultset.

- Die Angabe von `Value` und einem String, der auch Platzhalter für die Eingabespalten oder die Ausgabespalten enthalten kann.
- Die Angabe von `Key` mit dem Wert `true` für eine neue ID (bei Neuanlage).

Innerhalb der Definition der Spalten kann für jede Spalte optional der Parameter `Required` angegeben werden. Wird dieser mit dem Wert `true` belegt, dann wird vor der Ausführung geprüft, ob diese Spalte über einen Wert verfügt. Ist dies nicht der Fall, unterbleibt die Datenbankoperation.

Parameter	
Table	Der Name der Tabelle für den `INSERT`.
Columns	Ein Array mit den einzelnen Spaltendefinitionen. Die Groß-/Kleinschreibung dieser Spalten wird automatisch and den jeweiligen Alias angepasst. Jede Spalte besitzt folgende Parameter: • `Name`: Der Name der Spalte. • `Source`: Quellindex der Eingabedaten. • `Value`: Eingabedaten mit Platzhaltern. • `Required`: Falls `true`, erfolgt der Insert nur, wenn Spalte gefüllt. • `Key`: Erzeugt eine neue ID. Es kann jeweils nur entweder `Source`, `Value` oder `Key` verwendet werden!
Condition	Die Bedingung für das `UPDATE`, kann Platzhalter für die Eingabespalten oder die Ausgabespalten enthalten (z.B. `{4}` oder `{ORG.ORGID}`).
Action (opt.)	Die entsprechende Action, für die die Mapping-Funktion ausgeführt werden soll.

Beispiel

```
[iInsertUpdate, { Table: "RELATION",
                  Alias: "AO_DATEN",
                  Columns: (  {Name: "RELATIONID",
                               Source: 4,
                               Required: true },
                              {Name: "AOTYPE",
                               Value: "2" },
                              {Name: "PERS_ID",
                               Column: "PERS.PERSID" })   } ]
```

3.4.4. iJoin(pObject)

Beschreibung

Liest die bei `Source` als Array angegebenen Spalten und verbindet die einzelnen Werte mit dem String, der als `Delimiter` angegeben wurde. Das Ergebnis wird in der Ausgabespalte `Target` abgelegt.

Parameter	
Source	Der Index für die Quellspalte.
Value	Die Angabe der Eingangsdaten, kann Platzhalter für die Importspalten…
(Format {#})	… oder die Ausgabespalten (`Format {tbl.col}`) enthalten.
Target	Die Definition der Zielspalte.
Key (opt.)	Definiert die Zielspalte als Teil des Primärschlüssels.
Delimiter	Der String, der einzelne Einträge aus dem Array verbindet.
Action (opt.)	Die entsprechende Action, für die die Mapping-Funktion ausgeführt werden soll.

Beispiel

```
[iJoin, {Source: [3, 5],
         Delimiter: "\n",
         Target: "RELATION.ADDRESS"}]

[iJoin, {Value: ["{3}", "{5}"],
         Delimiter: "\n",
         Target: "RELATION.ADDRESS"}]
```

3.4.5. iMove(pObject)

Beschreibung

Liest die in `Source` angegebene Quellspalte und schreibt den Wert in die als `Target` angegebene Ausgabespalte. Alternativ kann der Inhalt einer externen Datei in die Zielspalte `Target` eingelesen werden. Als Dateiname wird dann der Inhalt von `Source` verwendet. Dazu muss die Eigenschaft `Blobfile` auf den Wert `true` gesetzt werden. Über den Parameter `Value` kann eine beliebige Zeichenkette zugewiesen werden, der Verweise auf die Eingabespalten oder die Ausgabespalten erlaubt.

Parameter	
Source	Der Index für die Quellspalte bzw. der Name der Blobdatei (inkl. Erw.).

Parameter	
Value	Ein JDito-Ausdruck, der zur Laufzeit berechnet wird. Dieser Ausdruck kann Verweise auf die Eingabespalten und die Ausgabespalten enthalten. Auf eine Eingabespalte wird mit `{#}` verwiesen, wobei das Zeichen `#` für den Index der Eingabespalte steht. Auf eine Ausgabespalte wird mit der Zeichenkette `{tbl.col}` verwiesen, wobei `tbl` für die Tabelle und `col` für die Spalte steht (wie in der Option `Target` angegeben).
Map	Gibt eine zuvor definierte Map (ein Key-Value-Paar) an, aus dem die Daten geholt werden. Wenn Map angegeben ist, muss auch Index angegeben werden. Klassisches Beispiel: Ländername (z.B. Deutschland) und Value (z.B. DE).
Index	Enthält den Key der in der Map gesucht werden soll. Funktioniert genau wie Value, ist allerdings zur besseren Lesbarkeit von `Value` getrennt. Wird nur ausgewertet, wenn Map angegeben wird.
Eval (opt.)	Gibt bei der gleichzeitigen Verwendung von `Value` an, ob der Inhalt von `Value` nach der Ersetzung der Platzhalter noch als JavaScript-Ausdruck ausgewertet werden soll oder nicht (mögliche Werte: `true`/`false`).
Target	Die Definition der Zielspalte.
Key (opt.)	Definiert die Zielspalte als Teil des Primärschlüssels.
Blobfile (opt.)	Definiert das Lesen einer externen Datei, wenn `true`.
Pathname (opt.)	Gibt den Pfadbestandteil im Filesystem an.
Trim (opt.)	Gibt an, ob der Datenwert vor dem Speichern in der Zielspalte noch um führende oder folgende Leerzeichen bereinigt werden soll. Mögliche Werte sind die Strings `left`, `right` oder `both`.
Action (opt.)	Die entsprechende `Action`, für die die Mapping-Funktion ausgeführt werden soll.
HTML2Text (opt.)	Gibt an, ob der HTML-Eingabewert zu plaintext konvertiert werden soll.
RTF2Text (opt.)	Gibt an, ob der RTF-Eingabewert zu plaintext konvertiert werden soll.

Beispiel

```
[iMove, { Source: 3,
          Target: "RELATION.ADDRESS" } ]
```

`iInsertUpdate` erzeugt Datensätze, `iMove` hingegen legt Daten in nur eine Spalte!

3.4.6. iNewID(pObject)

Beschreibung

Erzeugt mit Hilfe der JDito-Funktion `a.getNewID()` einen neuen Primärschlüsselwert und speichert diesen in der Ausgabespalte `Target`. Üblicherweise werden solche Spalten auch mit der Eigenschaft `Key` als Schlüsselspalten gekennzeichnet (siehe Beispiel).

Parameter	
Target	Die Definition der Zielspalte.
Key (opt.)	Definiert die Zielspalte als Teil des Primärschlüssels.
Action (opt.)	Die entsprechende `Action`, für die die Mapping-Funktion ausgeführt werden soll.

Beispiel

```
[iNewID, { Target: "ORG.ORGID",
           Action: "I",
           Key: true }]
```

Verwendung von `Action: "U"` oder `Action: "I+U"` führt hier zu keinen sinnvollen Ergebnissen!

3.4.7. iSql(pObject)

Beschreibung

Diese Funktion erlaubt die Ausführung von beliebigen SQL-Kommandos während des Imports. Diese können bei Bedarf auch Daten aus den Quellspalten benutzen. Sollte die Abfrage keinen skalaren Wert liefern, sondern ein komplettes Resultset, dann wird aus dem Resultset der Abfrage die erste Spalte der ersten Zeile als Ergebnis in die definierte Ausgabespalte geschrieben.

Parameter	
Target	Die Definition der Zielspalte.
Alias	Der Aliasname, über den das SQL-Kommando läuft.

Parameter	
Command	Das auszuführende SQL-Kommando. Um auf Werte aus den Quellspalten zuzugreifen, kann als Platzhalter der String `{n}` benutzt werden, wobei n den Index der Quellspalte angibt.
Action (opt.)	Die entsprechende `Action`, für die die Mapping-Funktion ausgeführt werden soll.

Beispiel

```
[iSql, { Target: "ORG.ORGCOUNTRY",
         Alias: "AO_DATEN",
         Command: "select name_de from countryinfo where iso2 =
                   '{1}'",
         Action: "I+U" }]
```

Es findet keine Prüfung auf Gültigkeit oder Sinnhaftigkeit des SQL-Statements statt! Bei Unachtsamkeit können hier auch schädliche SQL-Statements eingetragen werden!

3.4.8. iTimestamp(pObject)

Beschreibung

Wandelt einen Zeit- bzw. Datumsstring aus einer Eingabespalte in einen Datums-Longwert in ein von ADITO benötigtes Format um, das direkt in einer Timestamp-Spalte gespeichert werden kann. Standardwert für `iTimestamp` ist die Zeitzone `UTC`.

Parameter	
Source	Der Index der Quellspalte.
Target	Die Definition der Zielspalte (`tbl.col`).
Format	Gibt das Format an, in dem die Daten vorliegen. Wird nichts angegeben, nimmt der Importer "`yyyy-MM-dd HH:mm:ss`" an.
Timezone	Gibt die für die Konvertierung zu verwendende Zeitzone an. Die Namen der Zeitzonen entsprechen denen von ADITO (Java). Wird keine Zeitzone angegeben, benutzt der Importer die Zeitzone `UTC`.
Action (opt.)	Die entsprechende `Action`, für die die Mapping-Funktion ausgeführt werden soll.

Beispiel

```
[iTimestamp, { Source: 4,
               Target: "RELATION.DOB" ,
               Format: "dd.MM.yyyy HH:mm" } ]
```

3.4.9. iWord(pObject)

Beschreibung

Zerteilt den Eingabewert aus der Quellspalte `Source` mit Hilfe des Ausdrucks `Regex` in einzelne Wörter und schreibt dann das Wort Nummer `Index` in die Ausgabespalte `Target`. Alternativ kann auch der komplette Teilstring bis zum gesuchten Wort geliefert werden.

Parameter	
Source	Indexnummer der Eingabespalte.
Value	Ein String mit dem konstanten Wert.
Target	Die Definition der Zielspalte, die das Ergebnis der Funktion erhält.
Index	Die Angabe der Wortnummer. Negative Werte zählen die Wortnummer vom Ende des String her.
Substring	Gibt an, ob nicht das einzelne Wort, sondern der komplette Teilstring bis zum angegebenen Wort (inklusive) geliefert werden soll. Mögliche Werte sind die Strings `left` und `right`.
Separator	Optionale Angabe der Zeichenkette, mit der die einzelnen Worte bei der Angabe der Option `Substring` verbunden werden, der Default ist ein Leerzeichen.
Key (opt.)	Definiert die Zielspalte als Teil des Primärschlüssels.
Action (opt.)	Die entsprechende Aktion, für die die Mapping-Funktion ausgeführt werden soll.

Beispiel

```
[iWord, {Source: 3,
         Target: "PRODUKT.LAGER",
         Index: -1,
         Regex: /\s+/ }]

[iWord, {Value: "{5}",
         Target: "RELATION.ADDRESS",
         Index: -2,
         Regex: " ",
         Substring: "left" }]

[iWord, {Source: 3,
         Target: "RELATION.BUILDINGNO",
         Index: -1,
         Regex: /\s+/, Action: "I" }]
```

3.5. Funktionen für ADITO-Objekte

3.5.1. iAttribute(pObject)

Beschreibung

Legt ein Attribut aus dem Wert einer Eingabespalte an oder aktualisiert ein vorhandenes. Der Wert für `Attribute` ist der vollqualifizierte Attributname, also bei zweistufigen Attributen beide Namen, getrennt durch eine Pipe („|“).

Wird diese Funktion mit dem `ImportCommand` „`insert+update`“ verwendet, dann wird zuerst geprüft, ob es zu der `Rowid`, dem `Type` und dem `Attribute` bereits einen Eintrag gibt. Ist dieser vorhanden, wird der Eintrag aktualisiert, wenn nicht, ein neuer erzeugt. (Dabei wird auch die Action mitberücksichtigt!)

Mit den `ImportCommands` „`insert`“ oder „`update`“, wird das Attribut entweder nur angelegt oder nur aktualisiert.

Die Einstellung `importer.useAttributeCache = true` ermöglicht die Verwendung vom `Attributecache`.

Parameter	
Source	Indexnummer der Eingabespalte.
Value	Der String mit den Eingabedaten, kann Platzhalter für die Eingabespalten oder die Ausgabespalten enthalten (alternativ zu `Source`).
Attribute	Attributname, bei zweistufigen Attributen werden die Namen durch eine Pipe getrennt z.B. `Betreuung|Innendienst`
Rowid	Die Ausgabespalte mit dem Primärschlüssel, zu dem das Attribut angelegt wird.
Type	Der Objekttyp (`FrameID`), für den das Attribut definiert wurde.
ListEncoding	Falls mit `true` angegeben, wird der Eingabewert (aus `Source` oder `Value`) als String-kodiertes Array aus einer Liste aufgefasst. Damit wird das Attribut unter Umständen mehr als einmal angelegt.
Action (opt.)	Die entsprechende `Action`, für die die Mapping-Funktion ausgeführt werden soll.
Autocreate (opt.)	Ist diese Funktion auf `true` gesetzt, werden Keywords, die nicht vorhanden sind, angelegt (gilt nur für Combobox-Attributen). Wird nicht automatisch in den Attributcache geladen.

Beispiel

```
[ iAttribute, { Source: 2,
                Attribute: "Betreuung.Zielgruppe",
                Rowid: "RELATION.RELATIONID",
                Type: 1 } ]
```

Details zu unserem Datenbankmodell und weitere Informationen zum Aufbau des ADITO4 Referenzsystems finden Sie im AID 008 – DE – Das ADITO Referenz-Datenmodell.

3.5.2. iComm(pObject)

Beschreibung

Erzeugt oder aktualisiert einen Kommunikationseintrag in der Tabelle `COMM` aus dem Wert einer Eingabespalte. Ein Update erfolgt nur dann, wenn sich die Kommunikationsadressen unterscheiden. Der Wert für die Eigenschaft `Medium` muss als direkter Wert des Keyvalues angegeben werden.

Ein Import bzw. Eintrag in die Kommunikationstabelle erfolgt nur dann, wenn für die Spalte mit der Adresse auch ein Wert vorhanden ist, also in `Source` bzw. `Value` ein Wert vorhanden war.

Es wird in der Tabelle `COMM` die Spalte `ADDR`, sowie `SEARCHADDR` aktualisiert, falls sich etwas geändert hat. Dabei wird anhand der `Rowid` und des Mediums geprüft, ob hier bereits ein Eintrag vorhanden ist (sollten mehrere Einträge zur `Rowid` und Medium in der Tabelle vorhanden sein, werden alle mit dem neuen Wert überschrieben!!).

`iComm` ist eine Sonderform von `iInsertUpdate`!

Parameter	
Source	Der Index der Eingabespalte (alternativ zu `Value`).
Value	Der String mit den Eingabedaten, kann Platzhalter für die Eingabespalten oder die Ausgabespalten enthalten (alternativ zu `Source`).
Medium	Der Wert für die `MEDIUM_ID` (aus den Schlüsselwörtern).
Rowid	Der Wert mit der entsprechenden Relations-ID (default ist der Inhalt der Ausgabespalte von `RELATION.RELATIONID`). Kann Platzhalter für die Eingabespalten oder die Ausgabespalten enthalten (z.B. `{TEMP.COL2}` oder `{4}`).
OrgID (opt.)	Falls angegeben, wird die Relations-ID über die `OrgID` gesucht (kann auch mit `PersID` kombiniert werden für Typ3-

Parameter	
	Relationen). Kann Platzhalter für die Eingabespalten oder die Ausgabespalten enthalten.
PersID (opt.)	Falls angegeben, wird die Relations-ID über die `PersID` gesucht (kann auch mit `OrgID` kombiniert werden für Typ3-Relationen). Kann Platzhalter für die Eingabespalten oder die Ausgabespalten enthalten.
RelationID (opt.)	Falls angegeben, wird dieser Wert für die Relations-ID verwendet (kann nicht mit `OrgID` und/oder `PersID` kombiniert werden). Kann Platzhalter für die Eingabespalten oder die Ausgabespalten enthalten.
Standard (opt.)	Gibt an, ob es sich dabei um das Standardkommunikationsmedium handelt.
Action (opt.)	Die entsprechende Aktion, für die die Mapping-Funktion ausgeführt werden soll.

Beispiel

```
[iComm, {RelationID: "{4}",
         Value: "{8}",
         Standard: true,
         Medium: 12 }]
```

3.5.3. iCommRestriction(pObject)

Beschreibung

Erzeugt oder aktualisiert eine Werbesperre in der Tabelle `COMMRESTRICTION` aus dem Wert einer Eingabespalte. Der Wert für die Eigenschaft `Medium` muss als direkter Wert des Keyvalues angegeben werden.

Parameter	
RelationID	Die Relations-ID, kann Platzhalter für die Eingabespalten oder die Ausgabespalten enthalten.
Medium	Der Wert für die `MEDIUM_ID` (aus den Schlüsselwörtern). Kann Platzhalter für die Eingabespalten oder die Ausgabespalten enthalten.
Reason (opt.)	Der Grund für die Werbesperre. Kann Platzhalter für die Eingabespalten oder die Ausgabespalten enthalten.

Beispiel

```
[iCommRestriction, {DoIf: "'{17}' != ''",
                    Medium: "{17}",
```

```
                    RelationID: "R_{0}",
                    Reason: "Example Reason"}
]
```

3.5.4. iDocument(pObject)

Beschreibung

Legt ein Dokument im Dokumentencontainer (ASYS_BINARIES) an.

Parameter	
Container	Der String mit dem Namen der Dokumentenzuordnung.
Rowid	Name der Ausgabespalte, die den entsprechenden Primärschlüssel des Datensatzes enthält, mit dem dieses Dokument verbunden werden soll.
Source	Indexnummer der Eingabespalte mit dem BLOB-Inhalt.
Filename	Indexnummer der Eingabespalte mit dem Dateinamen.
Tablename	String mit dem Namen der Datentabelle zu diesem Dokument.
Description	(Opt.) Eingabespaltenid mit einer Beschreibung des Dokuments.
Keywords	(Opt.) Eingabespaltenindex mit Schlüsselwörtern zum Dokument.
Action (opt.)	Die entsprechende Aktion, für die die Mapping-Funktion ausgeführt werden soll.

Beispiel

```
[iDocument, {Container: "HISTORYDOCS",
   Rowid: "HISTORY.HISTORYID",
   Source: 3,
   Filename: 4,
   Tablename: "HISTORY" }]

[iDocument, {Container: "HISTORYDOCS",
   Rowid: "HISTORY.HISTORYID",
   Source: 3,
   Filename: 4,
   Tablename: "HISTORY",
   Description: 7,
   Keywords: 8,
   Action: "I" }]
```

3.5.5. iHistoryLink(pObject)

Beschreibung

Erzeugt einen Historylink für die angegebenen Eingabedaten und verknüpft einen bestehenden Historieneintrag mit anderen Datensätzen. (Diese Funktion besitzt keine Update-Funktion! Es werden hier immer HistoryLinks neu angelegt!)

Parameter	
Source	Indexnummer der Eingabespalte mit dem Wert, mit dem der Historieneintrag verknüpft wird (die Spalte `HISTORYLINK.ROW_ID`).
Value	Eingabedaten mit dem Wert, mit dem der Historieneintrag verknüpft ist, kann Platzhalter enthalten (entspricht `HISTORYLINK.ROW_ID`).
Type	Der Typ für den Historylink, entspricht dem Objekttyp (der `FrameID`). Muss angegeben werden. Kann bei Angabe von `PersID` und `OrgID` der Typ nicht komplett gefüllt werden (z.B.: `Type: 3`, aber nur `OrgID` enthält einen Wert), wird der Datensatz nicht übernommen.
PersID (opt.)	Falls die `ROW_ID` für die Verknüpfung mit einer Person oder Organisation nicht bekannt ist, kann hier die Quellspalte angegeben werden, in der die `PersID` gespeichert ist. Akzeptiert Angaben im `Source`- und in `Value`-Format.
OrgID (opt.)	Falls die `ROW_ID` für die Verknüpfung mit einer Person oder Organisation nicht bekannt ist, kann hier die Quellspalte angegeben werden, in der die `OrgID` gespeichert ist. Akzeptiert Angaben im `Source`- und in `Value`-Format.
HistoryID (opt.)	Definition der Spalte aus dem Ausgabepuffer, in dem die `HistoryID` gespeichert ist (optional, default ist `HISTORY.HISTORYID`).
Action (opt.)	Die entsprechende Aktion, für die die Mapping-Funktion ausgeführt werden soll. (`Action „U“` sowie die Upate-Funktion bei `„I+U“` haben keinerlei Auswirkungen!)

Beispiele

```
[ iHistoryLink, { Source: 3,
                  Type: 11   }
  iHistoryLink, { PersID: 3,
                  OrgID: 5   } ]
```

3.5.6. iKeyword(pObject)

Beschreibung

Erzeugt einen Verweis auf ein Schlüsselwort aus einer Eingabespalte. Der Wert der Eingabespalte muss dem Inhalt der Spalte `KEYNAME1` in der Tabelle `KEYWORD` entsprechen.

Ist ein Keyword noch nicht in der Tabelle `KEYWORD` angelegt, wird dieses – wenn der Parameter `Autocreate` auf `true` gesetzt ist – angelegt.

Parameter	
Source	Der Index der Eingabespalte (alternativ zu `Value`).
Value	Der String mit den Eingabedaten, kann Platzhalter für die Eingabespalten oder die Ausgabespalten enthalten (alternativ zu `Source`).
Keytype	Entspricht dem internen Namen der Schlüsselwortkategorie, dies ist der Inhalt der Spalte `KEYNAME2` in der Tabelle `KEYWORD`.
Target	Die Definition der Zielspalte.
Autocreate (opt.)	Ist diese Funktion auf `true` gesetzt, werden Keywords, die nicht vorhanden sind, angelegt.
Action (opt.)	Die entsprechende Aktion, für die die Mapping-Funktion ausgeführt werden soll.
KeynameColumn	Angabe von `KEYNAME1` oder `KEYNAME2`. Wird der Parameter nicht angegeben, wird `KEYNAME1` verwendet. Wird `KEYNAME1` verwendet oder der Parameter nicht angegeben, so wird beim Import ein `KEYVALUE` und `KEYNAME1` angelegt. Wird `KEYNAME2` verwendet, so wird ein `KEYVALUE` verwendet und in `KEYNAME1` und `KEYNAME2` der gleiche Wert angegeben.

Beispiel

```
[iKeyword, {Source: 4,
            Keytype: "bundesland",
            Target: "RELATION.STATE",
            Action: "I+U" }]

[iKeyword, {Value: "100{4}",
            Keytype: "regalcode",
            Target: "STOCK.SHELFCODE",
            Autocreate: true,
            Action: "I" }]
```

4. Erweitern des Importers

4.1. Neue Importfunktionen erstellen

Für die meisten Anwendungsfälle sollten die vorhandenen Funktionen ausreichen. Bei besonderen Anforderungen ist das Modul über eigene Importerfunktionen, sogenannte `iFuncs`, jederzeit ausbaubar.

Ziel war es, innerhalb einer Zeitspanne von zehn Minuten eine erste eigene Funktion erstellen zu können und diese modular in jedes vorhandene Importermodul einhängen zu können. Bei der Erstellung der `iFuncs` gilt der Grundsatz „*convention over configuration*“, so dass sich Autoren von Funktionen darauf verlassen können sollen, dass Ihr Code auch in anderen System problemlos läuft.

Die Funktionsfähigkeit des Importers kann nicht mehr gewährleistet werden, wenn Sie bestehende Funktionen bearbeiten. Darüber hinaus kann es vorkommen, dass in zukünftigen Versionen des Importers die Funktionen verändert werden und somit Ihre Anpassungen nicht mehr gültig sind.

`iRelation` ist eine Sonderform von `iInsertUpdate`!

4.2. Definition einer iFunc

4.2.1. Funktionsname

Als Konvention beginnen alle Importfunktionen mit dem kleinen Buchstaben „i“, gefolgt vom restlichen Namen, bei dem jeweils der Anfangsbuchstabe des Wortes groß geschrieben wird. Unterstriche im Funktionsnamen sollten vermieden werden.

4.2.2. Parameter

Jede `iFunc` erhält einen Parameter `pObject`, in dem die für die Funktion definierten Parameter in Form eines Objekts übergeben werden. Dieses Objekt enthält das Objektliteral, das in der Mappingkonfiguration angegeben wurde. Somit können die einzelnen Parametereigenschaften leicht ausgelesen werden.

4.2.3. Funktionsergebnis

Jede Importfunktion muss bei erfolgreicher Abarbeitung den logischen Wert `true` als Ergebnis liefern. Falls Daten nicht erfolgreich verarbeitet werden können oder der aktuelle Datensatz der Importdaten übersprungen werden soll, kann die Importfunktion den logischen Wert `false` liefern. Dies signalisiert dem Importer, dass der aktuelle Datensatz nicht übernommen wird.

4.2.4. Fehlerbehandlung

Falls während der Abarbeitung der Funktion eine nicht abgefangene Exception auftritt, wird diese vom Importer abgefangen, eine entsprechende Logmeldung erzeugt und das Ergebnis der Funktion als `false` gewertet, der gerade bearbeitete Datensatz also übersprungen.

4.2.5. Zugriff auf Daten des Imports

Jede Importfunktion besitzt Zugriff auf alle Spalten des Eingaberesultsets und Zugriff auf alle Spalten des Ausgabepuffers. Damit sind Sie in der Lage, sehr mächtige Importfunktionen zu erstellen. Allerdings sind Sie damit auch in der Lage, in einer Funktion den kompletten Import zu ruinieren!

Über `this.InputRecord` erhalten Sie Zugriff auf die Spalten des Eingaberesultsets für den aktuellen Datensatz in Form eines Arrays. Der Index 0 bezeichnet dabei die erste Spalte des Eingaberesultsets. Dieses Array läuft bis zum Index `this.InputRecord[this.InputRecord.length - 1]`.

Über `this.setOutput` schreiben Sie die transformierten Daten in den Ausgabepuffer. Hierbei handelt es sich um eine Funktion, bei der Sie über die Eigenschaften das Objekt sowie den Wert als zweiten Parameter übermitteln. Hierbei muss in pObjekt der Wert `Target` im Format `Tabelle.Spalte` vorhanden sein. Um den Wert der Hausnummer zu schreiben, würden Sie also

```
this.setOutput(pObject, „42“);
```

verwenden. Falls Sie eine Tabelle befüllen wollen, können Sie einfach den Tabellennamen angeben, der Importer erzeugt später die entsprechenden Insert- bzw. Updateanweisungen. Das Tabellenobjekt selbst muss allerdings bereits in der Datenbank vorhanden sein.

Die Importer-Option für die Laufzeitsteuerung erhalten Sie direkt über den Zugriff auf das Objekt `this`. Damit können Sie beispielsweise über `this.Preview` (liefert `true` oder `false`) abfragen, ob der Simulationsmodus aktiviert wurde.

4.2.6. Beispiel

```
// diese Funktion wird später im mapping folgendermaßen
// eingesetzt:
// [ iSelbstGebaut, { Source: 4,
                      Target: "TABELLE.SPALTE" } ]

function iSelbstGebaut(pObject)
{
   var s = this.InputRecord[pObject.Source];
   // machwas ...
   this.setOutput(pObject, s);
   return true;
}
```

4.2.7. Auflösen von Platzhaltern

Falls Sie in Ihrer Funktion Platzhalter unterstützen wollen, können sie dies durch den Aufruf der folgenden Methode in Ihrer Funktion erledigen:

```
var value;

if(pObject.Value != undefined)
```

```
{
   value = this.resolveSymbol(pObject, pObject.Value, false);
}
else
{
   value = "";  // oder andere Problembehandlung
}
```

Der Aufruf von `this.resolveSymbol()` ersetzt alle in geschweiften Platzhalter der Form `{#}` (um den Index einer Spalte des Eingaberesultsets zu verwenden) oder `{tbl.col}` (um auf Werte im Ausgabepuffer zuzugreifen). Im obigen Codebeispiel kann damit der Parameter `Value` in der Mappingdefinition auch Platzhalter enthalten.

Der zweite Parameter ist per default `false`, wenn nicht angegeben und steuert die Auswertung als JavaScript-Ausdruck. Wird hier ein `true` angegeben, wird das Ergebnis der Platzhalterersetzung als JavaScript ausgeführt und das Ergebnis zurückgeliefert! Damit sind mächtige Konvertierungen möglich, da Sie Zugriff auf das JDito-Laufzeitsystem haben. Das folgende Beispiel konvertiert den Inhalt der Eingabespalte mit dem Index 2 in Kleinbuchstaben.

```
  [iMove, { Value: "String({2}).toLowerCase()",
            Eval: true,
            Target: "RELATION.ADDRESS" } ]
```

4.2.8. Loggen von Informationen

Mit der Methode `this.writeLog( <nachricht> )` können Sie Ausgaben in der Importprotokoll schreiben. Vor alle Ausgaben wir der String `[IMPORT]`, gefolgt von einem Leerzeichen gestellt.

```
if(this.Debug == true) this.writeLog("Adding COMM entry");
```

4.3. Parameterkonventionen

4.3.1. Allgemeines

Die Namen der Parameter sollten ebenso wie die Namen der Importfunktion selbst sorgfältig gewählt werden und den Zweck der Funktion möglichst allgemein, aber präzise ausdrücken. Parameter werden in PascalCasing geschrieben (jeweils erster Buchstabe eines Wortes groß). Verweise auf Schlüsselfelder beinhalten die Bezeichnung `ID` in Großbuchstaben, z.B. `RelationID`.

Der Datentyp von Optionen, die logische Schalter definieren, sollte Boolean sein, damit die Werte für die Parameter direkt als `true` oder `false` angegeben werden können.

4.3.2. Eingabedaten

Dieser Parameter sollte `Value` genannt werden und nach Möglichkeit Platzhalter akzeptieren (siehe auch „Auflösen von Platzhaltern“ weiter oben).

4.3.3. Ausgabedaten

Als Parameter für das Ziel einer Transformation sollte der Parameter `Target` verwendet werden. Erlaubte Ziele sind eine Spaltenspezifikation aus `Tabelle.Spalte`.

5. Beispiele

Entsprechende Beispiele finden Sie im xRM Basic in den Prozessen `sp_importExample` und `IMPORTERTEST`.

6. Performance

6.1. Tipps zur Verbesserung der Performance

- Auf `iSQL` sollte verzichtet werden. Nutzen Sie stattdessen `iMove` mit Datenquelle Map. Dieses lädt nur ganz am Anfang die Daten und verwendet später die zuvor geladenen.
- Setzen Sie `completeUpdate = true`.
- Verwenden Sie zum Testen nur wenige Daten.
- Verwenden Sie den Debugger wie folgt: `Debug = true`, `Debug.log = Console`.
- Führen Sie Joins immer in der Datenbank aus, nicht danach im Code.