Dynamische Stichwortlisten

[Stand: 19.03.2014 | Programmversion: 9.00.000]

Bei dynamischen Stichwortlisten handelt es sich um Stichwortlisten, deren Inhalt zur Laufzeit erzeugt wird. Der Inhalt von dynamischen Stichwortlisten kann auch auf bereits eingegeben Daten im Verschlagwortungsdialog reagieren und die Stichworte kontextabhängig filtern.

Inhalt

1 Beispiel

Man hat in der Verschlagwortungsmaske die Felder *Abteilung* und *Mitarbeiter*. Die Definition der Abteilungen und ihrer Mitarbeiter wird in einer Datenbank außerhalb von ELO gepflegt. Um die aktuellen Abteilungen abzufragen, wird ein Skript ausgeführt, welches sich mit der externen Datenbank verbindet, vorhandene Abteilungen abfragt und als Stichwortliste zurückliefert. Aus dieser Liste kann dann die gewünschte Abteilung ausgewählt werden.

Das Feld für den Namen des Mitarbeiters sollte nun mit einem Mitarbeiter aus der zuvor ausgewählten Abteilung gefüllt werden. Hierzu sendet die Anwendung die ausgewählte Abteilung an das Skript mit der Information, jetzt die Mitarbeiter abzufragen. Der weitere Verlauf ist wie gehabt.

2 Administration

Ein Indexfeld erhält die neue Option *ServerScript*. Hier ist der Name des Skriptes einzutragen, welches zum Laden der Daten für die dynamische Stichwortliste ausgeführt werden soll.

Das Skript müssen Sie im Ordner *Administration* unter *IndexServer Scripting Base* hinterlegen.

3 Client-Anwendung

Es wird ein Objekt verschlagwortet. Im Dialog gibt der Benutzer Daten in ein Indexfeld mit hinterlegter dynamischer Stichwortliste ein. Die Client-Anwendung erkennt ein solches Indexfeld daran, dass das Feld *DocMaskLine.serverScriptName* gesetzt ist.

Sobald die Client-Anwendung eine Eingabeverzögerung erkennt, ruft diese die IX-API-Funktion

```
checkoutKeywordsDynamic(ClientInfo,KeywordsDynamicInfo)
```

auf. Im IX wird sodann das zu den gegebenen Parametern entsprechende Skript aufgerufen und das Ergebnis als

```
KeywordsDynamicResult
```

zurückgegeben. Das Ergebnis wird in der Client-Anwendung als Liste oder als Tabelle unterhalb des Indexfeldes angezeigt. Der Benutzer kann daraus die für das Objekt richtige Zeile auswählen.

Mit den Informationen aus diesem Ergebnis kann die Client-Anwendung das in Bearbeitung befindliche Indexfeld vervollständigen. Sind im Ergebnis weitere Spalten mit der Information darüber gegeben, zu welchem Schlüssel die Daten aus dieser weiteren Spalte gehören, besteht für die Client-Anwendung sogar die Möglichkeit weitere Indexfelder zu vervollständigen.

4 Beispiel-Skript

Im Folgenden ist ein funktionierendes Beispiel-Skript gelistet, welches zum einen die benötigten Funktionen zeigen und zum anderen als Vorlage für Skripte dienen soll. Rudimentäre Beschreibungen der Funktionen sind im Listing dokumentiert.

4.1 SampleScript.js

```
/**
 * Für dieses Beispiel-Skript wird eine Datenbankverbindung benötigt.
 * Hierbei bedient es sich den Hilfsfunktionen aus dem Skript DataSource.js.
 *
 * @include DataSource.js
 */
importPackage(Packages.de.elo.ix.jscript);

/**
 * Diese Funktion muss im Skript implementiert werden. Sie wird ohne Argumente aufgerufen. Als
 * Rückgabe wird ein neues Objekt erwartet, welches die abstrakten Funktionen der Klasse
 * DynamicKeywordDataProvider implementiert.
 */
function getDataIterator() {
  return new DynamicKeywordDataProvider(new SimpleDatabaseQuery());
}
```

```
/**
 * Implementierung der Klasse DynamicKeywordDataProvider
 * var index: Laufvariable für das ResultSet
 * var resultset: Ergebnis der DB-Abfrage
 */
function SimpleDatabaseQuery() {
  var index = 0;
  var resultset = undefined;
  /**
   * Öffnet die Datenquelle. Diese Funktion wird als erstes aufgerufen.
   * @param ec: Ausführungskontext des Skripts.
   * Enthält ec.user: UserInfo, ec.ci: ClientInfo,
   * @param sord: Das derzeit in Bearbeitung befindliche Sord. Belegte Indexfelder können
   * als Filter für die Datenquelle dienen.
   * @param key: Schlüssel der aktuell im Client ausgewählten Zeile.
   */
  this.open = function(ec, sord, key) {

    /* this.getObjKeyValue erleichtert den Zugriff auf Werte in Indexfeldern */
    /* Rückgabe ist ein Array von Strings. */
    var filter = this.getObjKeyValue(sord, key);

    /* SQL-Statement mit Platzhaltern. Diese werden dann von params belegt */
    var statement = "SELECT userid, data_1, data_2, data_3"
       + " FROM testkeywordsdynamic"
       + " WHERE userid LIKE ? AND data_1 LIKE ? ORDER BY id;";

    /* params[0] ersetzt das erste ? in statement, params[1] das zweite ‚?‘ */
    var params = [];

    if (ec.user.id == 0) {
       /* Der Administrator soll alle Daten geliefert bekommen */
       params.push("%");
    } else {
       /* Ein normaler User soll nur seine Daten geliefert bekommen */
       params.push(ec.user.guid);
    }

    if (filter && filter[0]) {
       /* Falls das Sord Daten für das Indexfeld ‘key’ bereitstellt,
        * eine Präfixsuche in der DB durchführen. */
       params.push(filter[0] + "%");
    } else {
       /* Ansonsten nichts filtern. */
       params.push("%");
    }

    var database = new DBConnection("jdbc/TestDataSource", "elo90");
    resultset = database.query(statement, params);
  }

  /* Gibt jeweils die erste noch nicht zurückgegeben Zeile zurück */
  this.getNextRow = function() {
    return resultset[index++];
  }


  /* Wird aufgerufen, nachdem getNextRow() keine neuen Ergebnisse mehr zurückgibt oder
   * bereits 1000 Ergebniszeilen abgeholt wurden. */
  this.close = function() {
    /* DB Verbindung ist bereits geschlossen, sobald database.query(…) zurückkommt.
     * Daher in diesem Fall leer. */
  }
```

```
  /* Gibt Überschriften für Spalten als Array zurück. Die Anzahl an Spalten muss mit der
   * Anzahl an Spalten von getNextRow() übereinstimmen. Der IX prüft die Anzahl nicht. */
  this.getHeader = function() {
    return ["Nothing", "Something", "Special", "Important"];
  }

  /* Gibt die Schlüssel für Spalten als Array zurück. Die Anzahl an Spalten muss mit der
   * Anzahl an Spalten von getNextRow() übereinstimmen. Der IX prüft die Anzahl nicht. */
  this.getKeyNames = function() {
    return ["", "testKeywordScriptUser", "", ""];
  }

  /* Wird aufgerufen, nachdem der IX alle Zeilen abgeholt hat. Sollten aber noch Zeilen
   * übrig sein (also insgesamt >1000), gibt hasMoreRows() true zurück, sonst false. */
  this.hasMoreRows = function() {
    return (index < (resultset.length));
  }

  /* Gibt eine Nachricht zurück, die der Client anzeigen soll. Können Fehlermeldungen
   * oder Informationsmeldungen sein (z.B. „Bitte zunächst Feld XYZ ausfüllen“)*/
  this.getMessage = function() {
    return "";
  }

  /* Gibt einen aussagekräftigen Namen für die
   * */
  this.getTitle = function() {
    return "Auftragsdaten";
  }
}
```