DOCUMENTS 4

GENTABLE (INVOICE PLUG-IN)
Configuration and Administration

VERSION 2.0

Table of Contents

1. Introduction

1.1 Overview

The Web-based software component GenTable is integrated into DOCUMENTS 4; in addition to running with *Internet Explorer* version 6.x or later, it runs with *Mozilla/Firefox* version 1.0 or later.

GenTable is generally suited for representing any number of table or item-related data in a user-friendly manner. This data can then be edited and saved by the user.

Moreover, even your own implemented scripts can be integrated. External data from relational databases can also be embedded.

Important features are additionally saving the table data and the configuration data in *XML* file format as well as client-side execution of the bulk of the application to minimize data traffic and easing the load on the server.

1.2 The most important properties & features

The GenTable component's most important properties and features:

- User-defined representation of any type of table or item-related data.
- Saving the table data and configuration data in *XML* file format.
- Client-side processing of data to minimize the server load.
- Allowed element types of table output are text fields, multiline text fields, drop-down lists, checkboxes, static text, and (graphic) buttons.
- More application-specific helper functions, e.g. split transaction as well as automatic calculation of individual item amounts and the total amount for invoice files.
- Extensibility by integrating your own scripts using the script API.
- User-defined integration of any external data from relational databases to the different text fields, multiline text fields, drop-down lists and checkboxes.
- External data may additionally be dependent on the logged-in user, on any values of the DOCUMENTS 4 file, or on specific values of the same row.
- Table rows can be dependent on the visible, invisible or *readonly* definition (not changeable.
- Linking automatically executed events with specific table columns/fields.

1.3 Using the software component as an Invoice plug-in

The best known option available of GenTable so far is its use as a plug-in for invoice verification in invoice files (Invoice plug-in). The table columns in this case correspond to the individual categories of an invoice (e.g. purchase order number, quantity, unit price, total price, cost center, etc.) and the table rows correspond to the individual invoice items.

2. Configuration & Customization

2.1 Integration into DOCUMENTS 4

If the GenTable data files have not yet been installed, the `gentable_www.zip` file will be unpacked to the `www` subdirectory of the installation directory on the Web server.

To integrate GenTable as a plug-in into DOCUMENTS 4, you need to start the DOCUMENTS Manager. The individual, existing file types are listed under *Documents / File Types* in the tree structure in the left section of the main window.

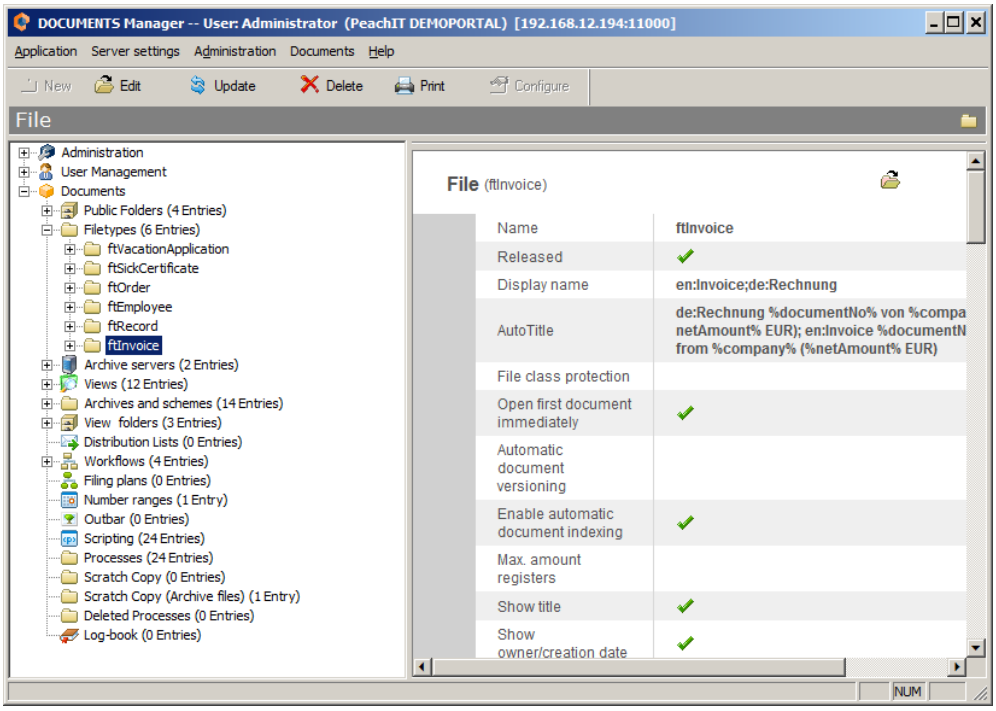

Fig. 1: Tree structure in the DOCUMENTS Manager

Now you either open one of the existing file types by double-clicking the respective file type entry in the tree structure, or you create a new file type via the *New* button on the top left of the main menu. To do this, the *File Types* entry in the tree structure must be enabled.

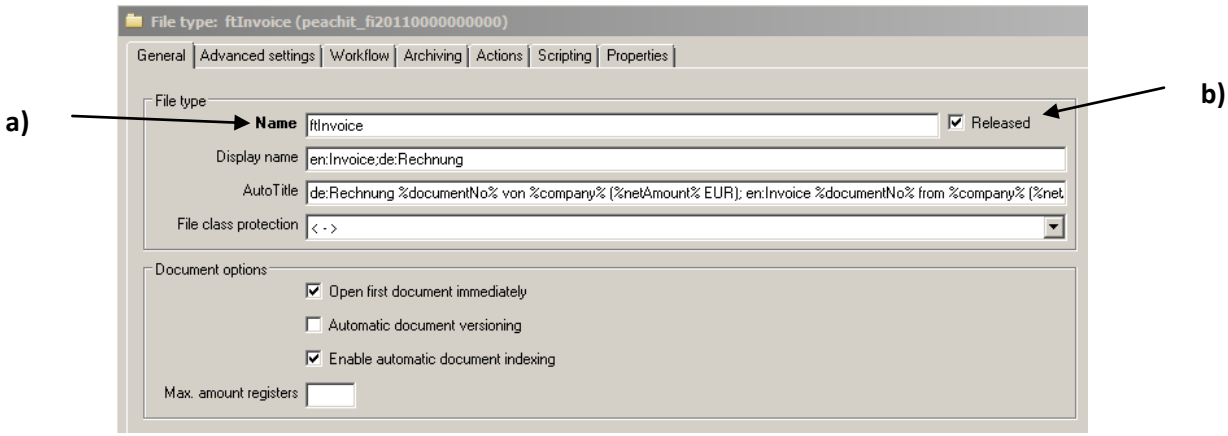

Fig. 2: Editing or creating a file type

The following items in the *General* category are important settings for a file type:

a) In the text field *Name* you need to specify the unique name of the file type. This name must also be used analogously when defining the XML configuration file of GenTable.

b) The checkmark in the *Released* checkbox must be set to release the file type in DOCUMENTS 4.

To integrate GenTable as a *plug-in* into the file type, you need to make the following settings in the *Properties* category:

Right-clicking in this window and then selecting the *Add new property* function, you need to create the property with the label `hasInvoicePlugin` with the value `true`.

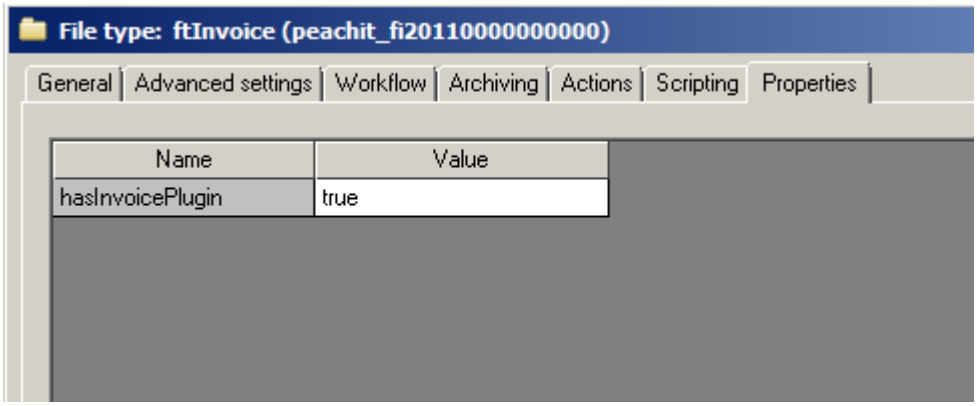

Fig. 3: Properties of a file type

The individual fields of the file type are listed in the *Fields* category in the bottom section of the window for editing the file type. The existing fields can be edited by double-clicking the respective entry. Other fields can be created by right-clicking in this section of the window and selecting the *Create new data record and insert it in fields*.

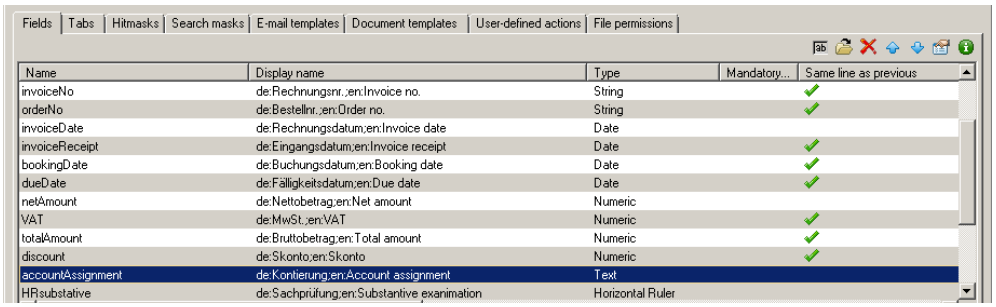

Fig. 4: Editing or creating fields

GenTable requires that an additional field for saving the XML table data exist or be created. In this example, this is the *account Assignment* field.

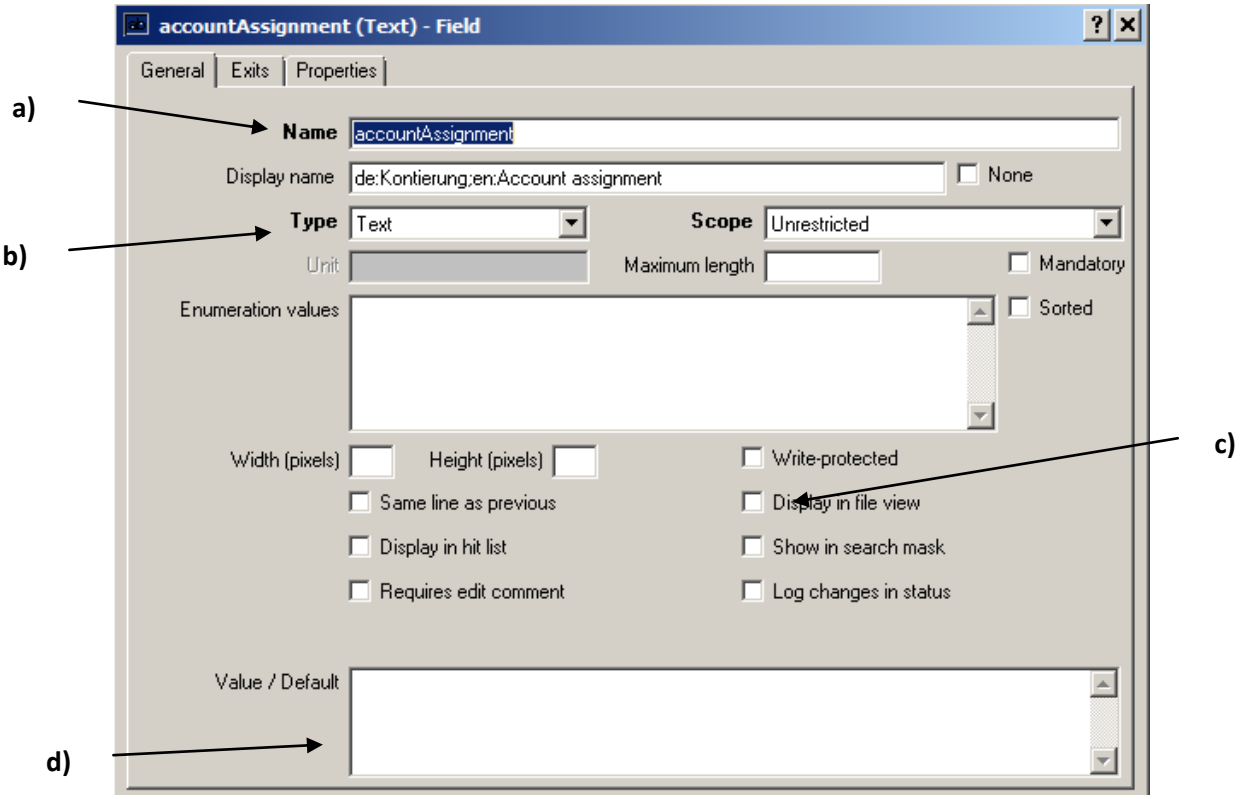

Fig. 5: File field dialog

The following settings are the most important items for configuring the field for the XML table data:

a) In the Name text field, you need to spe*cify a unique name for this field. This name will analogously be used by* GenTable in the XML configuration file.

b) The type of field for the XML table data should be *Text* or *String*.

c) Because the field may not be displayed with the XML table data, the checkmark will not be set for the *Display in file view* checkbox.

d) You can optionally enter predefined XML table data in the *Value/Default* text window. The data entered here is entered in the GenTable table by default every time you create a new file of this type. By simply setting the XML tag for rows `<tr></tr>` you can define beforehand how many rows are automatically displayed on creating a new file. The entries of columns containing static text or of columns defined as uneditable should be set because these entries can no longer be made by the user. The table data is always saved in the following XML structure (in this case, a table with three columns and three rows, for example):

```
<table>
	<tr>
		<td title="Column 1">SampleEntry1</td>
		<td title="Column 2">SampleEntry2</td>
		<td title="Column 3">SampleEntry3</td>
	</tr>
	<tr>
		<td title="Column 1">SampleEntry4</td>
		<td title="Column 2">SampleEntry5</td>
		<td title="Column 3">ColumnEntry6</td>
	</tr>
	<tr>
		<td title="Column 1">SampleEntry7</td>
		<td title="Column 2">SampleEntry8</td>
		<td title="Column 3">SampleEntry9</td>
	</tr>
</table>
```

Thus, the basic settings for integrating GenTable into the DOCUMENTS 4 file are complete.

2.2 Defining the configuration file

Each configuration of GenTable always refers to a specific file type in DOCUMENTS 4. For this, an XML configuration file named `FileTypeName_Def.xml` is created in the www\WEB-INF directory of the DOCUMENTS 4 installation *(e.g.* `ftInvoice_Def.xml`*)*. The DTD file `table_def.dtd` can be used with an XML editor to make creation easier.

Below we will explain the individual elements of the configuration file.

2.2.1 Basic settings

`<table_def name=""></table_def>`: Root element of the configuration file. The `name` attribute contains the name of the file type. All other elements are noted as child element `of table_def`.

`<xmlfield></xmlfield>`***:*** Name of the invisible field of the file type where the XML data is to be saved.

`<js_include></js_include>`***:*** Contains the file name of the JS/JSP file that includes the additional file-specific script functions. This entry is always required because this file also contains general standard functions (see `UserdefinedScripts.jsp`).

The `<database></database>` element allows defining the required parameters for connecting to a database for this table. Database, driver, user name and password URL can be specified via the subelements `<url></url>`, `<driver></driver>`, `<user></user> and <password></password>`.

`<init></init>`***:*** Contains a JavaScript statement or a JavaScript function call which is executed on initializing GenTable (initially starting or reloading the page).

`<manipulatorClass></ manipulatorClass>`***:*** Name of a self-implemented Java class (manipulator class) which enables server-side manipulation of specific table data. In this class it is defined which table rows are visible, invisible or `read-only`, and which table data should be preliminarily changed in which manner (e.g. appending the row number to each entry, etc.). If no entry is made, the data will not be manipulated.

The class must implement the functions `setXMLTableVisibility(HttpServletRequest request, String rowNumber)` and `manipulateXMLTableData(String rowNumber)`, and be inferred from the abstract parent class `XMLTableDataManipulatorImpl` that comes with the software and that provides important variables and help functions.

`<buttonPositions></buttonPositions>`***:*** Can accept the values `North` and `South` and determines whether the buttons should be arranged above or below the table rows.

`<indexNumbers></indexNumbers>`***:*** Defines via `true` or `false` whether a sequential row number should be displayed prior to each row.

`<indexCheckboxes></indexCheckboxes>`***:*** Defines via `true` or `false` whether a checkbox for selecting the rows should be displayed prior to each row.

`<saveAll></saveAll>`*:* Defines via `true` or `false` whether all (including invisible) or only visible table columns should be saved.

2.2.2 Defining individual table columns/fields

The `<field></field>` element is used to define a column of the table. You need to define **at least one table column**.

The detailed settings of the respective column are defined via the following subelements.

`<title></title>` ***(mandatory field:)*** Contains the column's technical name.

`<label></label>`***:*** Contains an optional column heading that is displayed in the user interface. (This can be internationalized, see section 2.4.1)

`<type></type>`***:*** Field's data type. The following data types are available:

- **Text,** Default setting
- **TEXTAREA,** Text field for longer text
- **SELECT,** Selection list
- **CHECKBOX,** Checkbox
- **STATICTEXT,** Static, unmodifiable display text
- **BUTTON,** Column contains a button

`<default></default>`**:** Default value of the field on creating a row.

`<editable></editable>`**:** Determines the table column's visibility.

- `true`**:** The column is visible and editable.
- `readonly`**:** The column is visible but not editable.
- `disabled`**:** The column is invisible.

By default, **ANY FIELD** is editable when **creating a new** row. This is also true of the fields which have the `readonly` value for the `<editable>` element. To alter this behavior, version 2.1.0 or higher includes the global setting (the element must be set to the highest XML document level) named `<lastroweditable>`. If the value is set to `false` here, the `readonly` fields will not be editable, either, when creating a new row.

`<maxLength></maxLength>`***:*** Specifies the maximum number of characters for fields of the TEXT type. This setting has no function for fields of other data types.

`<width></width>`**:** Width of the respective table column in pixels. By default, the width of the column depends on the column heading.

<height></height>: Height of table column cells in pixels. By default, the height depends on the cell content.

<class></class>: Name of an additional CSS class for the column's cells.

<event type=""></event>: Registers a JavaScript event using the column's input element. The name of the event is used here as the type (e.g. onBlur). The event element must include a JavaScript function call.

<isLogin></isLogin>: When setting isLogin to true (see above), the login name of the currently logged-in user will be used as the presetting when creating a new row for the respective column.

<option value=""></option>: (May occur several times). Defines the options if the column is of the **SELECT** type. A technical value that will be saved when the option is selected can be specified in the value attribute. This value can be internationalized (see section 2.4.1).

<clearExisting></clearExisting>: If external dynamic data is integrated into drop-down lists (SELECT) in a table column but the currently saved value can no longer be found in the database query, this entry will be removed from the drop-down list (true**,** default). Otherwise, this entry will be additionally included in the selection (false).

<buttonLabel></buttonLabel>: Contains the label of the button if the column is of the **BUTTON** type.

<img></img>: To convert conventional buttons as the column element type to graphic image buttons, you can specify an image file (file name with the .gif or .png extension) here. The image file(s) should reside in the www/images/dlc/gentable folder of the DOCUMENTS 4 installation on the Web server. Otherwise, the image path must be specified in relative terms from the www directory (e.g. images/testimage.gif). To be able to additionally highlight a disabled image button, you need to store the corresponding image with the _dis extension in the same folder (e.g. testimage_dis.gif). The width and height attributes can also be used to determine the size of the image.

2.2.3 Restricting the content

Various restrictions can be set for a field:

For a text field with numeric input, you need to add a constraint name `<constraint>NUMBER</constraint>`. The field will then be checked for correct input on saving it.

If the field should be created as a mandatory field, you can specify `<constraint>MANDATORY</constraint>` for this field.

2.2.4 Restricting visibility

The option to control visibility of rows and columns in the Gentable plug-in depending on various conditions is available. Both on rows and column level you can define rules that:

- Compare a field of a row with a fixed value
- Compare a field of a row with a fixed value and check whether the current user is a member of a specific access profile
- Check whether the current user is a member of a specific access profile
- Compare a file field with a fixed value
- Compare a file field with a fixed value and check whether the current user is a member of a specific access profile
- Compare a file field with the value of a field of a row
- Compare a file field with the value of a field of a row and check whether the current user is a member of a specific access profile
- Compare a file field and/or a field of a row with the value of the system-wide auto text or a property of the current user

These different combinations will be used in a sample definition file at the end of this section.

Additional settings options:

- The "columnsAlwaysVisible" parameter is optional; it ensures that the column heading is always displayed, even if none of the rows contains a visible value in a column.
- There are two rule types: "READONLY" and "INVISIBLE"
- If the "invert" attribute is set to "true" in a rule, the logic of that rule will be reversed (see example of Field3). Here the READONLY column is set for all users who are NOT members of the "Administrator" access profile.

Important note:

Restricting visibility only impacts on the Web interface. The underlying data model is generally fully available to the browser, so visibility restriction must not be considered a restriction of user permissions.

Sample definition file

```
<?xml version="1.0" encoding="windows-1252"?>
<table_def name="Invoice">
	<js_include>UserdefinedScripts.jsp</js_include>
	<xmlfield>InvoiceField</xmlfield>
	<saveAll>true</saveAll>
	<columnsAlwaysVisible>false</columnsAlwaysVisible>

	<rowCondition>
		<rule type="READONLY" field="Field1" value="2222" />
		<rule type="READONLY" filefield="Creditor" value="ALFKI" />
		<rule type="READONLY" accessprofile="Warehouse" />
		<rule type="READONLY" field="Field5"
		 value="Field5" accessprofile="Directors" />
		<rule type="READONLY" filefield="Creditor"
		 value="OTRIS" accessprofile="Administration" />
		<rule type="READONLY" autotext="%currentUser.$AllowedAmount%"
		 field="Field4" />
		<rule type="READONLY" autotext="%currentUser.$AllowedAmount%"
		 filefield="Amount" />
	</rowCondition>

	<field number="1">
		<label>Field1</label>
		<title>Field1</title>
		<type>TEXT</type>
		<editable>true</editable>
		<condition>
			<rule type="READONLY" field="Field2" value="Field2" />
		</condition>
	</field>

	<field number="2">
		<label>Field2</label>
		<title>Field2</title>
		<type>TEXT</type>
		<editable>true</editable>
		<condition>
		 <rule type="READONLY" filefield="DocumentNr" value="1000" />
```

```
        </condition>
    </field>

    <field number="3">
        <label>Field3</label>
        <title>Field3</title>
        <type>TEXT</type>
        <editable>true</editable>
        <condition>
          <rule type="READONLY" invert=”true”
           accessprofile="Administration" />
        </condition>
    </field>

    <field number="4">
        <label>Field4</label>
        <title>Field4</title>
        <type>TEXT</type>
        <editable>true</editable>
        <condition>
         <rule type="READONLY" field="Field3" value="2000"
          accessprofile="Administration" />
        </condition>
    </field>

    <field number="5">
        <label>Field5</label>
        <title>Field5</title>
        <type>TEXT</type>
        <editable>true</editable>
        <condition>
           <rule type="READONLY" filefield="DocumentNr"
            value="1111" accessprofile="Warehouse" />
        </condition>
    </field>

    <field number="6">
        <label>Field6</label>
        <title>Field6</title>
        <type>TEXT</type>
        <editable>true</editable>
        <condition>
          <rule type="READONLY" autotext="file:%Creditor%"
           field="Field6" />
        </condition>
```

```
        </field>

        <field number="7">
                <label>Field7</label>
                <title>Field7</title>
                <type>TEXT</type>
                <editable>true</editable>
                <condition>
                  <rule type="READONLY"
                   autotext="%currentUser.$AllowedAmount%"
                   filefield="CostCenter" />
                </condition>
        </field>
</table_def>
```

2.2.5 Querying external databases

`<sql></sql>`: Can be used to get the field contents from an external data source. In doing so, the database connection specified at the beginning of the document is used to get the data.

Querying the database can be set for the field in the `<sql></sql>` element using the following subelements:

`<query></query>`: Contains the SQL command to be used for the query. Various placeholders can be used when specifying this command.

- `%userLogin%` **:** Will be replaced with the login name of the currently logged-in user.
- `%FILE_FIELD:FileFieldName%` **:** Will be replaced with the content of the respective file field of the current DOCUMENTS file.
- `%FIELD:ColumnName%` **:** Will be replaced with the value of the respective column of the current table row.

It is necessary to ensure that all placeholders are always enclosed in **single quotes** (e.g. `'%FIELD:Item%'`). External data can be integrated into drop-down lists, text fields and multiline text fields and, depending on that, checkboxes can be checked (DB entry `true`) or not (DB entry `false`).

`<result></result>`: Contains the name of the database column to be used to query the data.

`<key></key>`: Contains the name of the database column that contains the technical value of the result.

2.2.6 Defining buttons

The `<button></button>` element **optionally** allows creating one or more user-defined buttons.

The buttons can be set respectively via the following subelements:

`<label></label>`: Button label. (This can be internationalized, see section 2.4.1).

`<function></function>`: Function call that will be executed when clicking the button. **CAUTION**: Enter functions always without parameter, even if their signatures contain parameters!

`<accessKey></accessKey>`: Contains a letter. Pressing the ALT key in combination with this letter may trigger the button function via the keyboard.

`<img></img>`: To convert conventional buttons as column element type to graphic image buttons, you can specify an image file here (file name with the .gif or .jpg extension). The image file(s) should reside in the `www/images/dlc/gentable` folder of the DOCUMENTS 4 installation on the Web server. Otherwise, the image path must be specified in relative terms from the `www` directory (e.g. `images/testimage.gif`). To be able to additionally highlight a disabled image button, you need to store the corresponding image with the `_dis` extension in the same folder (e.g. `testimage_dis.gif`). The `width` and `height` attributes can also be used to determine the size of the image.

2.2.7 Multiple configuration files for a file type

Facilitating the use of multiple definition files for a file type is possible. The user can decide here which definition file is to be used for the gentable plug-in at a specific time or with a specific workflow state. This can be set via the "genTableDefField" property. The value of this property must match the technical name of a selection field in the file type. This selection field must include the names of different available definition files as selection options.

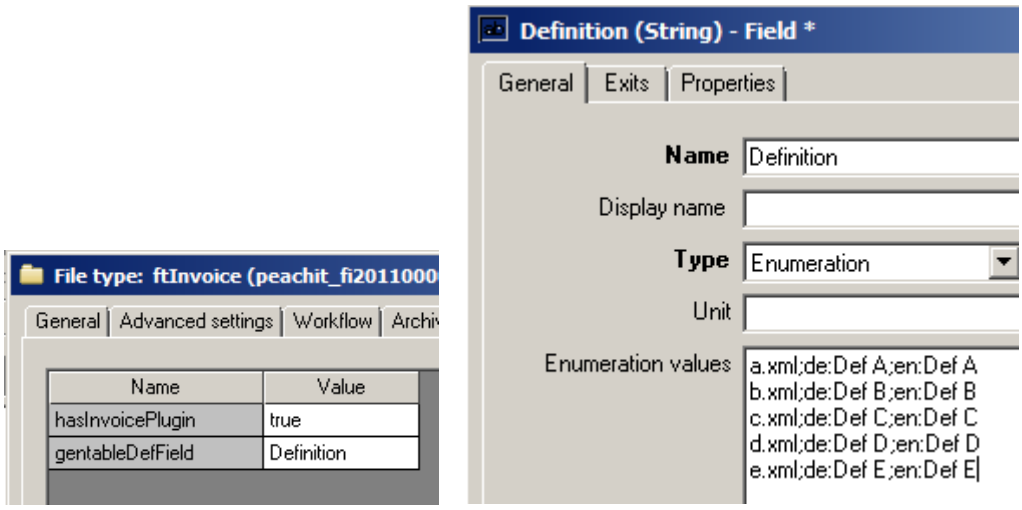

Fig. 6: The gentableDefField property

> *It is imperative that you enable the "saveAll" property in the definition files to be able to use multiple definition files for a file type.*

2.3 Customizing the user-defined script file

An option for integrating any script files or JSP files containing user-defined JavaScript functions into GenTable is available. By default, this is the `UserdefinedScripts.jsp` file that comes with the software. This file contains important functions that have been implemented, such as copying table rows, deleting table rows, appending a new table row, and moving table rows. Furthermore, more specific functions for the use of invoice verification have been implemented as an Invoice plug-in. For instance, these are the functions for calculating individual item prices, calculating and verifying the invoice total, formatting price fields, and split transaction.

These functions, however, partially require the exact column names, so that they can access individual cells, and work correctly. That is why the user-defined script file `UserdeinfedScripts.jsp` must be customized to the respective file type and the changing configuration settings. For this purpose, a separate section has been set up directly at the beginning of this file. The corresponding variables can be filled with the correct names of specific file fields and specific table columns there. Moreover, there you can define whether the additional script file `crmUserdefinedScripts.jsp` that contains functions for the *Scorecard* file (to supply evaluations/votes/ratings???) should be integrated.

This section looks as follows; the terms to be changed are in bold:

```
/*************************************************************************
*************************************************************************
*/

/** BEGIN script configuration !!! **/ **/

/** CAUTION: Names/IDs for various file fields and invoice item columns,
etc. (in this case, define centrally according to file type) *******/

/*************************************************************************
*************************************************************************
*/


      /* ID of Total amount field of the corresponding file */

      String fileFieldAmount = "Amount";   /* e.g. "TotalAmount_Gross" or
"Total" or "Amount" */


      /* ID of order quantity column of individual invoice items (see
<title> tag of the corresponding table column in the Def file) */

      String posCount = "amount";   /* e.g. "Order quantity" or "number" or
"quantity" */


      /* ID of unit price column of individual invoice items (see <title>
tag of the corresponding table column in the Def file) */

      String posPrice = "unitprice";       /* e.g. "UnitPrice_Net" or "unit
price" or "price" */


      /* ID of the total price column of individual invoice items (see
<title> tag of the corresponding table column in the Def file) */

      String posAmount = "value";   /* e.g. "TotalAmount_Net" or "Total
price_Net" or "total price" or "Amount" */




      /* ID of audit field of the corresponding file */

      String fileFieldChecking = "substantiveExamination";


      /* ID of the verifed column of individual invoice items (see <title>
tag of the corresponding table column in the Def file) */

      String posChecking = "checked";




      /************************ for CRM script
*********************************************/

      /* Information for integrating the additional CRM script */

      String crmScripts = "no";     /* "yes" or "no" */

%>
```

The section on customizing variables use is followed by the definition of callback functions that can be customized to individual requirements.

These are functions that are called prior to and/or after important operations such as saving the table or table output, and that allow you to trigger execution of

more desired functions. The callback functions include the `callbackDrawtablePost()` (execution directly after table output), `callbackSavePre()` (execution directly prior to save operation), and `callbackSavePost()` (execution directly after the save operation) functions. The required operations or function calls should be defined in these functions, and unwanted operations should be commented out.

The `callbackDrawtablePost()` function, for example, could look as follows:

```
function callbackDrawTablePost()
{
	//Automatically format table price fields accordingly
	formatPriceFields();

//Calculate and enter rating scores
	//calculateScores();
}
```

Here the function for formatting price fields is enabled, whereas the function for calculating the rating scores on the *Scorecard* file is commented out.

2.4 Internationalization

Gentable can be employed and used for multiple languages. Thus, for instance, all software component messages are output in the corresponding language. Language conversion takes place via the DOCUMENTS Manager. Preconfigured languages are English and German. Individual language versions can be easily changed, and you can also add other languages. The language files reside in the `www/WEB INF/classes/` subdirectory of the DOCUMENTS 4 installation on the Web server; they are all named

```
GenTablePinstrings_xx.properties.
```

Instead of the *xx*, the locale code of the corresponding language, according to ISO standard (e.g. *en* for English, and *de* for German) is displayed. The individual language terms are defined as value pairs in these `properties` files (e.g. `saveButton`=*Save*). English is the default language in the `GenTablePinstrings.properties` file.

To create a new language file, you simply need to copy one of the existing language files, change the locale code in the file name accordingly, and customize the individual language versions in the file to the corresponding language.

2.4.1 Internationalization within the configuration file

All `<label></label>` and `<option></option>` elements within the configuration file may contain internationalized contents. For this, the following syntax is used:

```
<label>DefaultValue;de:German;en:English;fr:French</label>, etc. ...
```

In the case of `<label>` the default value will be displayed if the description does not include the language set for the current user. In the case of `<option>` the default value is additionally used as a technical value for saving the option.

3. Implementing Your Own Scripts & Script API

3.1 Introduction

In addition to customizing GenTable to the respective requirements, you can integrate other script functions you have implemented yourself. However, this requires knowledge of JavaScript and of the *Document Object Model (DOM)*. The JSP file `UserdefinedScripts.jsp` that comes with the software and is integrated by default includes the bulk of important standard functions, and functions tailored to the use of the Invoice plug-in, which are implemented as script functions there. An editor enables you to additionally insert own functions in this file, or you can create a new JSP file containing the desired script functions, and integrate it into the software via the XML configuration file. Like the *UserdefinedScripts.jsp* file, this user-defined file must reside in the `www/jsp/dlc/gentable/scripts` subdirectory of the DOCUMENTS 4 installation on the Web server. The `UserdefinedScripts.jsp` file can be used as a template. Below we will briefly introduce the individual functions of the Script API which roughly break down into two areas.

3.2 The internal data model's functions

The internal data model represents an important area of the Script API. In this data model, the entire data of a table in an XML element structure are temporarily saved at internal level in the browser's runtime environment. The implemented *API* functions allow accessing all elements or data. You can then work with that data. The functions can be subdivided into simple access functions, and in some important editing functions. You should generally use this data model to make more fundamental changes to the table.

3.2.1 The access functions

The `getModelRows()` function returns all table rows temporarily saved in the data model as elements in an array. If no table rows are present, the value *null* will be returned.

The `getModelCells(rowNumber)` function returns all cells/fields of a specific table row that are temporarily saved in the data model as elements in an array. The `rowNumber` parameter specifies the desired row number; the first row corresponds to the number 0. If the table row does not exist, the value *null* will be returned.

The `getModelCell(rowNumber, columnTitle)` function returns a single table cell temporarily saved in the data model as an element. This table cell is uniquely determined via the `rowNumber` (row number; the first row corresponds to the number 0), and `columnTitle` (internal column name) parameters. If the table cell does not exist, the value *null* will be returned.

The `getModelValue(rowNumber, columnTitle)` function returns the *value* element of a single table cell temporarily saved in the data model. The *value* element contains the saved table entry. The table cell is uniquely determined via the `rowNumber` (row number; the first row corresponds to the number 0), and `columnTitle` (internal column name) parameters. If the table cell does not exist, the value *null* will be returned.

The `getModelValueEntry(rowNumber, columnTitle)` function returns the actual, saved table entry of an individual table cell temporarily saved in the data model. This table cell is uniquely determined via the `rowNumber` (row number; the first row corresponds to the number 0), and `columnTitle` (internal column name) parameters. If the table cell does not exist, the value *null* will be returned.

The `getModelOptions(rowNumber, columnTitle)` function returns all existing (row dependent) selection options of a single table cell temporarily saved in the data model as elements in an array. The table cell is uniquely determined via the `rowNumber` (row number; the first row corresponds to the number 0), and `columnTitle` (internal column name) parameters. If no selection options are present or the table cell does not exist, the value *null* will be returned.

The `getGeneralData(columnTitle, nodeType)` function returns the data elements temporarily saved in the data model and generally valid for a specific table column in an array. The data that is saved in these elements is valid for the entire table column, i.e. it is row independent. This is the case if the table column is filled with external data via a row independent database query. The `columnTitle` parameter specifies the internal name of the table column, and the `nodeType` parameter specifies the type of the element to be returned ("`value`" or "`options`"). That is, either the (row independent) *value* element is returned as a single element in an array (for single entry such as for text fields, multiline text fields, etc.), or the existing (row independent) selection options are returned as elements in an array (for multiline entries such as drop-down lists). If no row independent data exists for this table column or it does not contain any *value* element or selection option, the value *null* will be returned.

The `getVisibleRowNumbers()` function returns an array containing the row numbers of the visible/displayed table rows temporarily saved in the data model. These include *read-only* table rows. The row numbers of invisible table rows are omitted. Row number count starts with 0. If no visible table rows exist, the value *null* will be returned.

The `isLastModelRow(rowNumber)` function specifies whether the specified table row is the row that has been most recently saved temporarily in the data

model. The `rowNumber` parameter specifies the row number; the first row corresponds to the number 0. Depending on the result, either `true` or `false` is returned.

3.2.2 The editing functions

The `deleteModelRow(rowNumber)` function deletes a specific row of the table rows temporarily saved in the data model, including all entries and subelements. However, this only refers to rows declared visible (or *read-only*) in the data model; rows declared invisible remain unaffected. The `rowNumber` specifies the row number; the number 0 corresponds to the first (visible) table row.

The `cloneModelRow(rowNumber)` function copies a specific row of the table rows temporarily saved in the data model, including all entries and subelements. However, this only refers to rows declared visible (or *read-only*) in the data model; rows declared invisible remain unaffected. The `rowNumber` specifies the row number; the number 0 corresponds to the first (visible) table row.

The `appendNewModelRow()` function creates a new table row temporarily saved in the data model, including all subelements, which is declared as visible. The row contains empty entries or the corresponding predefined default values; it is appended to the end of the table in the data model.

The `moveModelRow(rowNumber, direction)` function moves a specific row of the table rows temporarily saved in the data model, including all entries and subelements, up or down one position. However, this only refers to rows declared visible (or *read-only*) in the data model; rows declared invisible remain unaffected, and are skipped. The `rowNumber` specifies the row number; the number 0 corresponds to the first (visible) table row. The `direction` parameter determines whether the row is moved up (*'UP'*) or down (*'DOWN'*).

3.3 Functions of the displayed table

The other important area of the Script API includes the functions of the displayed (*HMTL*) table. These functions allow direct access to the *HTML* element structure of the table output, and to all table data. This enables modifying individual entries in a more direct way.

3.3.1 The access functions

The `getHTMLRows()` returns all displayed table rows as elements in an array. If no table rows are present, the value *null* will be returned.

The `getHTMLCells(rowNumber)` returns all cells/fields of a specific, displayed table row as elements in an array. The `rowNumber` parameter specifies the desired row number; the first row corresponds to the number 0. If the table row does not exist, the value *null* will be returned.

The `getHTMLCell(rowNumber, columnTitle)` returns an individual, displayed table row as an element. This table cell is uniquely determined via the `rowNumber` (row number; the first row corresponds to the number 0), and `columnTitle` (internal column name) parameters. If the table cell does not exist, the value *null* will be returned.

The `getHTMLEntry(htmlCell)` returns the actual entry of an individual, displayed table row, regardless of its element type. The `htmlCell` parameter is used to pass the table row itself as an element. If the table cell does not exist, the value *null* will be returned.

The `setHTMLEntry(htmlCell, value)` function occupies a displayed table row with a specific value/entry if the cell's element type is a text field, a multiline text field or static text. The `htmlCell` passes the table row itself as an element; the `value` parameter is used to pass the entry/value to be set.

The `getActiveHTMLCell(evt)` function returns the displayed table row as an element in which the current event was triggered. The `evt` parameter is used to pass the current (looped through) JavaScript event object.

The `getActiveHTMLRow(evt)` function returns the displayed table row in which the current event was triggered as an element. The `evt` parameter is used to pass the current (looped through) JavaScript event object.

The `getActiveHTMLRowNumber(evt)` function returns the row number of the displayed table row in which the current event was triggered. The `evt` parameter is used to pass the current (looped through) JavaScript event object.

If this table row does not exist, the value *null* will be returned.

The `getSelectedHTMLRowNumbers()` function returns an array containing the row numbers of the displayed table rows selected by the user (respective index checkboxes checked).

The `getHTMLFileField(fileFieldName)` function returns a specific, displayed file filed as an element. It therefore does not directly refer to the table, but to the file. The `fileFieldName parameter` specifies the internal name of the file field.

The `setHTMLFileField(fileFieldName, value)` function occupies a specific, displayed file field that contains a specific value/entry. It therefore does not directly refer to the table, but to the file. The `fileFieldName` parameter specifies the internal name of the file field; the `value` parameter passes the value/entry to be set.

3.3.2 The most important main functions

The `drawTable()` function redraws the complete table to be displayed. For instance, you should open it with the internal data model after these changes to make the effects of the editing processes visible.

The `prestoreTable(checkData)` function triggers temporary saving of currently displayed table and its data in the internal data model. You should start this function after the changes to the table data, so that the changes are imported. The `checkData` parameter determines whether the data is checked for adhering to the predefined conditions/constraints (`true`) or not (`false`). Normally, `false` can be passed here because verification is by default performed while saving permanently.

The `reloadRow(evt)` function triggers reloading a specific, displayed table row through a new server-side data query, and the redrawing of this individual table row. This function is used as an event function if the entries of a table row depend on the entries of another table row, for example. The table row is determined by the JavaScript event object passed in the `evt` parameter which is automatically passed through linking the fields of a table column with a specific type of event.

When implementing your own script functions, you should remember that the `prestoreTable(false)` function is executed prior to calling the functions of the internal data model to incorporate the changes that the user may have made in the displayed table into the data model. After editing via the internal data model function calls you need to call the `drawtable()` function to make the changes to the table visible.

4. Usage Examples & Standard Functions

On the basis of sections of various files, we will now represent some basic standard functions as Invoice plug-in. After loading a file of the "*ftInvoice*" type, the display looks something like this:

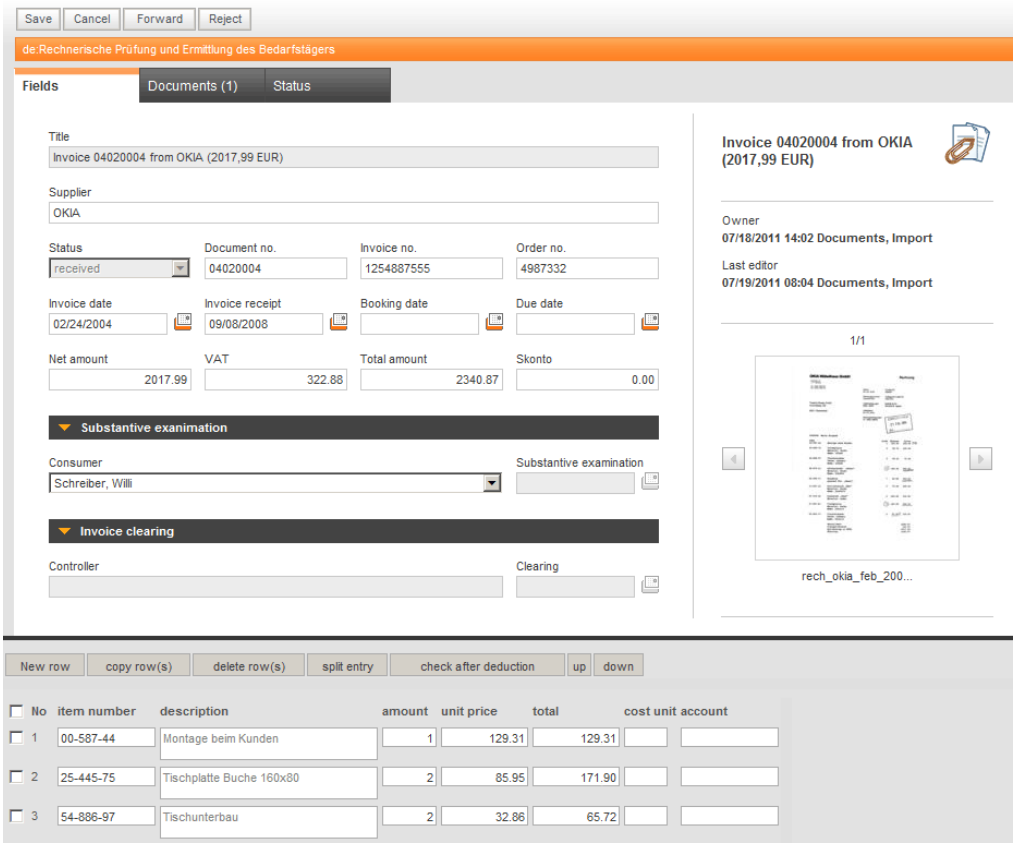

Fig. 7: File with Invoice plug-in

Some sample entries were made in the file fields and in the invoice items (table rows). In the figure below, a new row has been appended via the corresponding button.

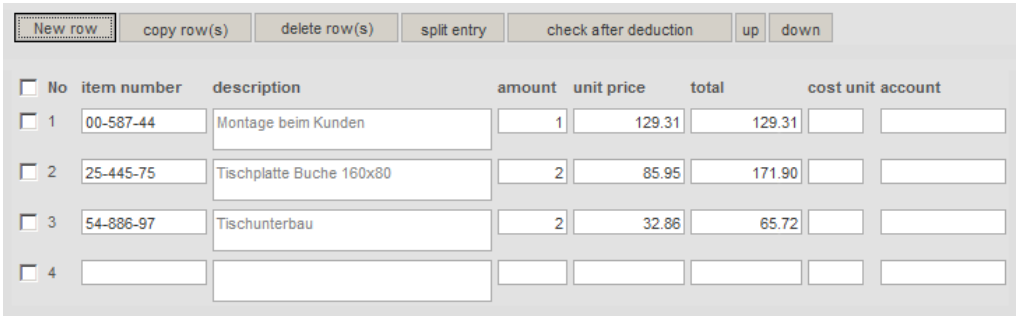

Fig. 8: Appending a new row

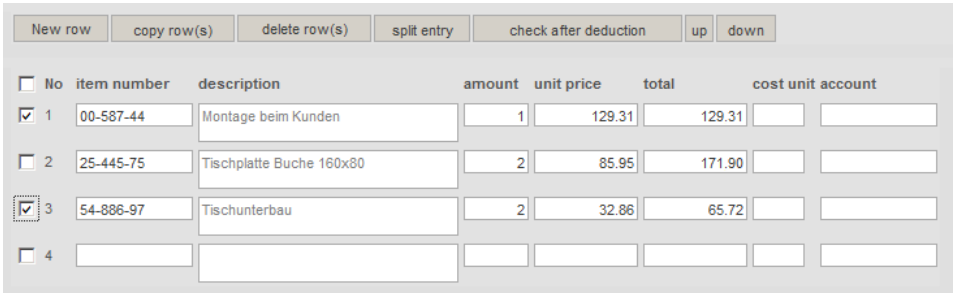

Fig. 9: Selecting row(s)

The two rows selected via the index checkboxes are to be copied.

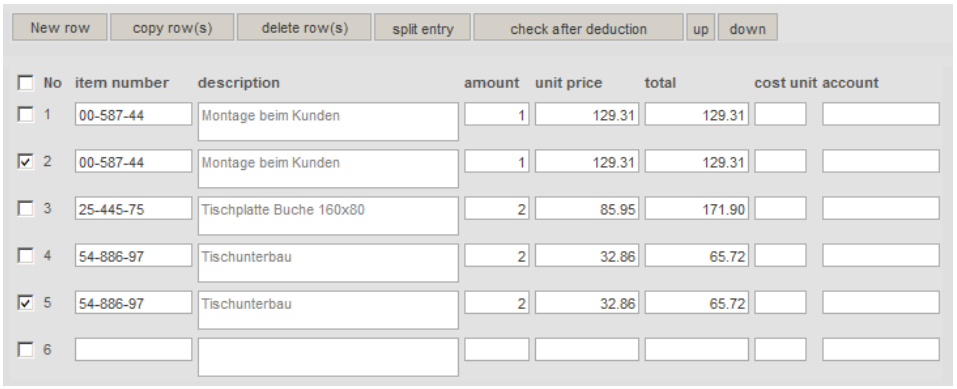

Fig. 10: Copying row(s)

After copying the rows, rows are selected, and then deleted.

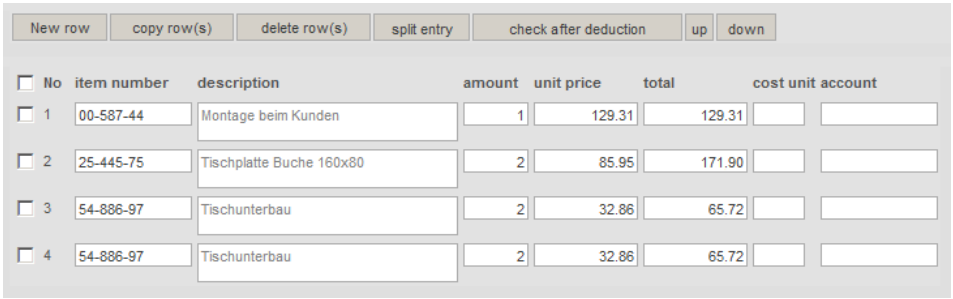

Fig. 11: Deleting row(s)

The "*Split transaction*" function (in this case, row 2) is used to copy selected rows at the same time, and the entries for quantity and amount are halved.

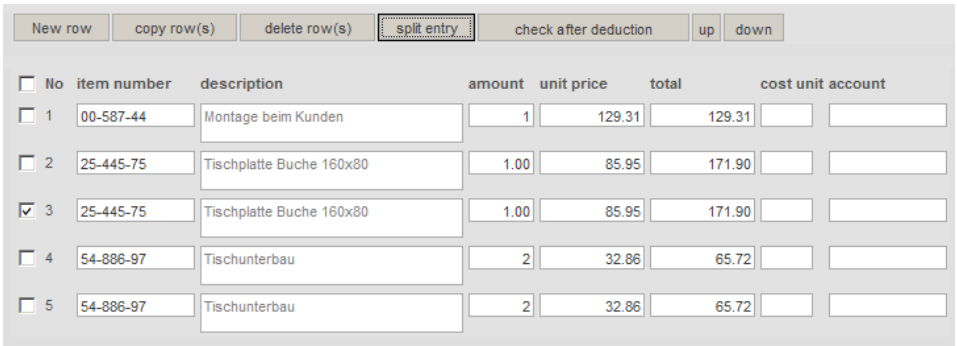

Fig. 12: Performing split transaction

Specific fields can also be automatically calculated from the entries of other fields, e.g. the amount from quantity and unit price.

When verifying the net amount, you will receive a specific message depending on the comparison to the file amount.

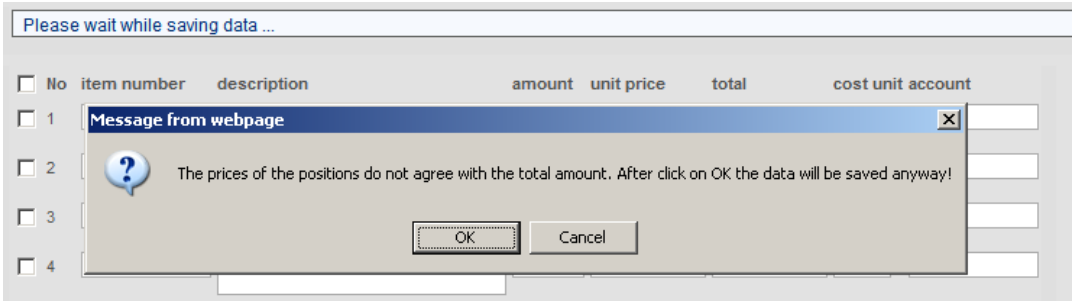

Fig. 13: Verifying the net amount

If, depending on the configuration, specific table columns/fields may only contain numeric values or may not remain empty, the corresponding message will appear. Only when all conditions are met can the table be saved.

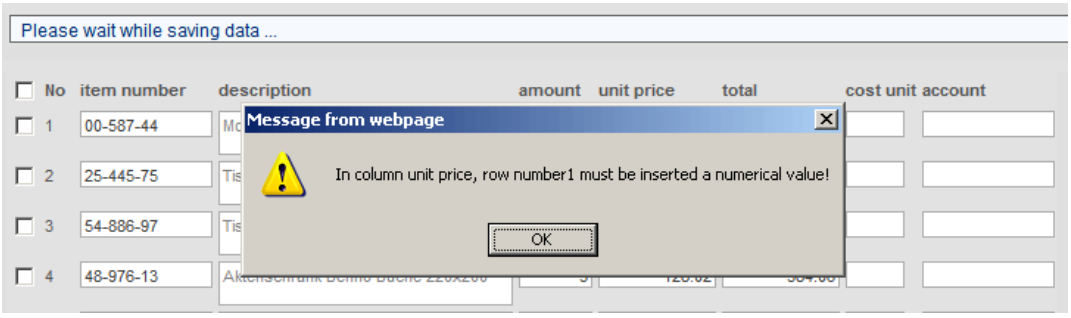

Fig. 14: Verification on saving

5. Errors & Causes

For errors occurring directly after installing GenTable, the cause of which is not obvious, deleting the `work/` subdirectory from the *Tomcat* directory of the DOCUMENTS 4 installation may be helpful.

You should also remember that *JavaScript* must be enabled in the client/user browsers.

If no XML configuration file exists for the respective file type, or if the name is wrongly typed or the configuration file is empty, GenTable cannot be loaded and will terminate with the following error message:

Fig. 15: Missing XML configuration file

If the XML configuration file cannot be read correctly, or the file does not include a valid XML or specific illegal special characters have been used as XML entries (e.g. <, >, &, etc.), GenTable will terminate with the following error message:

Fig. 16: Corrupt XML configuration file

The following error occurs if the XML tag `<xmlfield>` is missing from the XML configuration file, wrongly typed, or empty.

Fig. 17: No configuration entry for file field, or empty

The following error message will appear if the file field for the table data that is defined in the XML configuration file is not defined/present in the file type itself.

Fig. 18: File field not present

At least a table column in the XML configuration file should be defined:

Fig. 19: No table columns

When integrating external data from a database, the database driver should be present:

Fig. 20: Database driver

Moreover, the SQL command syntax must be correct to avoid query errors. The error message contains more information.

Fig. 21: Database query

6. Table of Figures