Baumtabelle

AID 087 DE

© 2017 ADITO Software GmbH

Diese Unterlagen wurden mit größtmöglicher Sorgfalt hergestellt. Dennoch kann für Fehler in den Beschreibungen und Erklärungen keine Haftung übernommen werden. Wir sind für Feedback zu den Themen, Inhalten, aber auch noch vorhandenen Fehlern dankbar und würden uns freuen, Ihre Meinung zu hören. Die in diesen Unterlagen enthaltenen Daten und Angaben, einschließlich URLs und anderer Verweise können ohne vorherige Ankündigung geändert werden. Alle in diesen Unterlagen aufgeführten Produkt- und Firmennamen sind unter Umständen Marken oder geschützte Zeichen der einzelnen Firmen. Ohne ausdrückliche schriftliche Einverständniserklärung der ADITO Software GmbH darf kein Teil dieses Dokumentes vervielfältigt oder in einer Datenverarbeitungsanlage gespeichert oder in diese eingelesen werden. Diese Einschränkung gilt unabhängig von Art und Weise der Datenerfassung.

Autor: FA, MW, KN. Version 10.1. Zuletzt geändert 19.09.2017

Version	Änderungen
10.1	Anpassung der Formatierungen
10.0	Anpassung an ADITO4.6
2.3	Beispiele für die Verwendung der Formatierung hinzugefügt
2.2	Letzte Version vor Übernahme in die Versionshistorie

Inhaltsverzeichnis

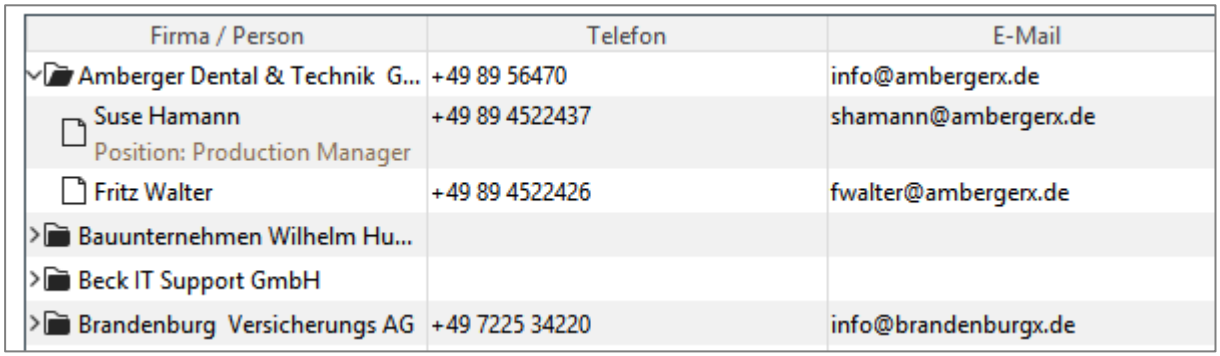
1. Die Baumkomponente

In den folgenden Kapiteln lernen Sie die ADITO Baumtabelle kennen, wie Benutzer sie bedienen können und wie sie über den ADITO4-Designer eingefügt wird.

Die Baumkomponente steht in ADITO4 ab Version 4.2 zur Verfügung.

Dieses Dokument ist nur für die Swing-Anwendung gültig.

1.1. Für die Benutzer



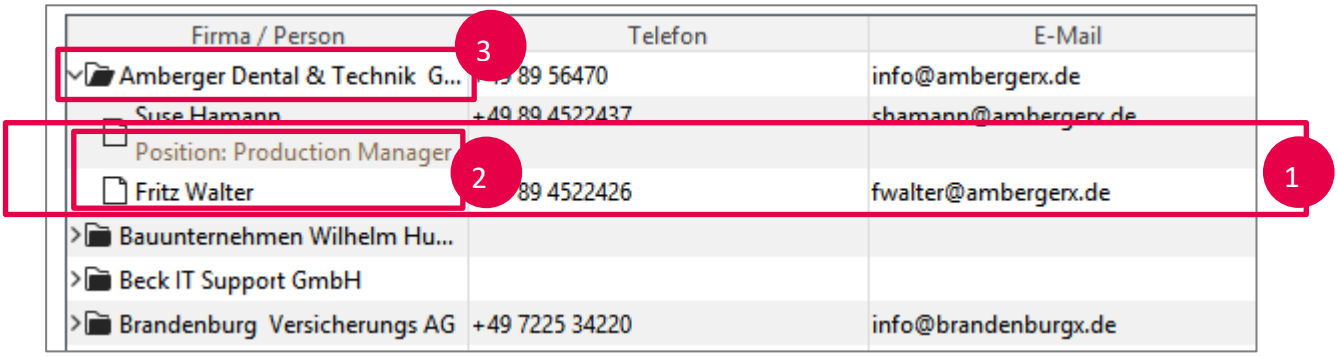
Für die ADITO-Anwender ist die Baumtabelle das, was der Name verheißt – eine Mischung aus Baum und Tabelle. Das bedeutet, dass die tabellarisch aufgebauten Informationen strukturiert sowie auf- und zugeklappt werden können.

Im Gegensatz zum Baum werden die Knoten hier durch Klick auf das Icon auf- und zugeklappt.

Des Weiteren können natürlich auch Doppelklickaktionen und ein Kontextmenü definiert werden.

2. Baumtabellen anlegen und füllen

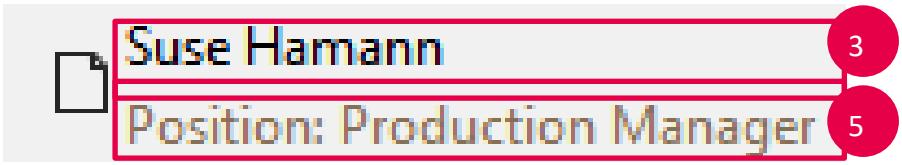
2.1. Technischer Aufbau



Die Komponente besteht aus den folgenden Elementen:

(1) **ARow**: Eine Zeile der Baumtabelle.
(2) **ACell**: Eine Zelle einer Baumtabelle.
(3) **AHeaderCell**: Eine Überschriftenzeile in der Baumtabelle.

Eine **ACell** kann wiederum aus mehreren **AData**-Elementen bestehen.



AData-Elemente können innerhalb einer ACell nur untereinander angezeigt werden.

2.2. Was braucht man?

Neben der Komponente selbst im Designer sind die Hilfsprozesse

- `aditoTreeTable`
- `aditoTreeTableAdditional`

unerlässlich zum Aufbau der notwendigen Objekte. Diese Prozesse können Sie, falls in Ihrem Basissystem nicht vorhanden, als Kunde mit Wartungsvertrag kostenlos von ADITO beziehen.

Diese Funktionsbibliotheken sind ausführlich innerhalb des Codes dokumentiert.

2.2.1. aditoTreeTableAdditional

Der `TreeTableAdditional`-Prozess bietet vereinfachte Funktionen, um Baumtabellen zu erstellen.

- **buildTreeTable** ermöglicht die Erstellung komplexer Baumtabellen mit ein- oder mehrzeiligen Zellen.
- **buildSimpleTable** ermöglicht die Erstellung einfacher Baumtabellen auf Basis eines zweidimensionalen Arrays.

2.3. Wichtige Eigenschaften

Eigenschaft	Beschreibung
contentProcess	Füllt die Baumtabelle.
showHeader	Zeigt die Headerzeile an. Der Inhalt der Headerzeile kann mit der Funktion `aHeaderCell` definiert werden.
useTreeColumn	Zeigt die Icon-Spalte an, aufgrund derer der Baum auf- und zugeklappt werden kann.
showTreeIcon	Zeigt die klickbaren Icons an.

3. Aufbau einer Tabelle

3.1. Grundlegendes Beispiel mit buildTreeTable

Das folgende Beispiel verwendet die Funktion `buildTreeTable` aus der Funktionsbibliothek `aditoTreeTableAdditional`.

Das Beispiel basiert auf einem einfachen Zahlenspiel. Kommt ein Wert über 1,0 heraus, wird in der Tabelle ein Knoten angezeigt, ansonsten ein Blatt.

3.1.1. Codebeispiel

```
import("system.result");
import("aditoTreeTable");
import("aditoTreeTableAdditional");

var root = Math.random();


var getCellFn = function(id, colIndex)
{
    return new ACell([new AData(id + "/" + (colIndex+1))]);

}

var getChildFn = function(id, index)
{
    return id * Math.random();
}

var getChildCountFn = function(id)
{
    return ~~(id * 10);
}

var getIdFn = function(id)
{
    return id;
}
var getIconFn = function(id)
{
```

```
    var icon = null;
    return icon;
}

var getTreeOpen = function(id)
{
    return false;
}

var tt = buildTreeTable(root, 1, getCellFn, getChildFn,
getChildCountFn, getIdFn, getIconFn, getTreeOpen)

tt.header = [ new AHeaderCell([new AData("Knoten")], 160)]

result.object(tt);
```

3.2. Beispiel mit Firmen- / Personendaten

Das folgende Beispiel baut eine Tabelle nach diesem Muster auf:

Firma / Person	Telefon	E-Mail
Amberger Dental & Technik G...	+49 89 56470	info@ambergerx.de
Suse Hamann Position: Production Manager	+49 89 4522437	shamann@ambergerx.de
Fritz Walter	+49 89 4522426	fwalter@ambergerx.de
Bauunternehmen Wilhelm Hu...		
Beck IT Support GmbH		
Brandenburg Versicherungs AG	+49 7225 34220	info@brandenburgx.de
Martin Heilingbrunner	+49 7225 978422	m.heiligbrunner@brandenburgx.de
Heidi Schulz	+49 7225 978482	h.schulz@brandenburgx.de
Buchberger & Partner		
CCK GmbH		
Claude Altendorf SA	+33 57156730	info@altendorfx.fr
Financial Group Chesterfield	+44 201234560	fg_chesterfield@domain.local
Förderverband Doritz		
Frankenmann Co.KG		
Friedrich & Frank GmbH		
GfK AG	+49 911 23400	gfk@domain.local
GHG Consulting		
Gorch Systems Engineering G...		
Grün Versicherung AG	+49 7824 5460	info@gruenx.de
Gußberg Software KG		

3.2.1. Aufbau des Objektes

Der Select, welcher dieser Tabelle zugrunde liegt (Beispiel weiter unten), liefert diese Tabellenstruktur:

RELATION ID	ORGNAME	FIRSTNAME + '' + LASTNAME	PERSID	ORGID	ADDR	ADDR	RELPOSITION
1010	Amberger Dental & Technik GmbH & Co.KG			O1010	+49 89 56470	info@ambergerx.de	
1073	Amberger Dental & Technik GmbH & Co.KG	Suse Hamann	P1023	O1010	+49 89 4522437	shamann@ambergerx.de	Production Manager
1075	Amberger Dental & Technik GmbH & Co.KG	Fritz Walter	P1024	O1010	+49 89 4522426	fwalter@ambergerx.de	

Das Objekt, welches die Baumtabelle befüllt, wird nun nach folgendem Muster angegeben:

- Wurzelknoten
  - Firma1
    - Daten der Firma (Name, Telefonnummer, E-Mail-Adresse)
    - PERSIDs aller zugrundeliegenden Personen
      - Daten der Person
  - Firma2
    - …

Ausgegeben, sieht dieses Objekt (Beispiel im Code weiter unten) so aus:

```
###ROOT: abe5730e-bb2d-457b-97da-87a81a41ce51; subprop: ids: O1010
,[…]
 ORG prop: O1010                                    ; subprop: ids: P1023
,P1024
 ORG prop: O1010                                    ; subprop: data:
Amberger Dental & Technik
GmbH & Co.KG
,+49 89 56470,info@ambergerx.de
 **** PERS prop: P1023                                        ; subprop: ids:
 **** PERS prop: P1023                                        ; subprop:
data: Suse Hamann,+49 89 4522437,shamann@ambergerx.de,Position:
Production Manager
 **** PERS prop: P1024                                        ; subprop: ids:
```

```
 **** PERS prop: P1024                                    ; subprop:
data: Fritz Walter,+49 89 4522426,fwalter@ambergerx.de
```

Dieses Objekt wird über die Standard-Funktionen aus `aditoTreeTable` aufgeteilt und schlussendlich zum Aufbau der Baumtabelle verwendet.

3.2.2. Beispielcode für den contentProcess

Nun der Gesamtcode, welcher die Baumtabelle füllt.

```
import("system.db");
import("system.util");
import("system.result");
import("aditoTreeTable");
import("aditoTreeTableAdditional");

var dsql = "select relationid, "
+ "orgname, firstname + ' ' + lastname, persid, orgid "
+ "    ,(select addr from comm where relation_id = relationid and
medium_id = 1 and standard = 1)"
+ "    ,(select addr from comm where relation_id = relationid and
medium_id = 3 and standard = 1)"
+ " ,RELPOSITION "
+ "from org "
+ "join relation on org_id = orgid "
+ "left join pers on pers_id = persid "
+ "where orgname <> 'privat'"
+ "order by orgname, persid asc";

var data = db.table(dsql);

var root = util.getNewUUID();

var treeStruc = new Object();
treeStruc[root] = {ids: []};

for(var i = 0; i < data.length; i++)
{
    //Übergeordnete Firmenknoten initialisieren
    if( treeStruc[data[i][4]] == undefined)
    {
        treeStruc[data[i][4]] =
```

```
        {
            ids: []//children-ids
            ,data: [ [data[i][1], data[i][5], data[i][6]] ]//orgname
        };
        /* Aktuellen Knoten bei übergeordnetem Knoten registrieren.
           Da es sich beim aktuellen Knoten um den obersten,
sichtbaren Knoten handelt ist der logische parent ein unsichtbarer
root-Knoten
         */
        treeStruc[root].ids.push(data[i][4]);
    }



    if(data[i][3] != "")
    {
        //Untergeordnete Personen initialisieren
        if( treeStruc[data[i][3]] == undefined)
        {
            treeStruc[data[i][3]] =
            {
                ids: []//children-ids
                ,data: [ [data[i][2], data[i][5], data[i][6]]
]//persname
            };

            if(data[i][7] != "")
            {
                treeStruc[data[i][3]].data.push(["Position: " +
data[i][7]]);
            }

            /* Aktuellen Knoten bei übergeordnetem Knoten
registrieren.
               Das ist bei den Personen die Firma.
             */
            treeStruc[data[i][4]].ids.push(data[i][3]);
        }
    }
}
```

```
var getCellFn = function(id, colIndex)
{
    var cellData = new Array();

    var fgColor = -16777216;//Schwarz
    for ( var i = 0; i < treeStruc[id].data.length; i++)
    {
        if(i > 0)
            fgColor = -8032171;//Grau

        cellData.push(new AData(treeStruc[id].data[i][colIndex],
undefined, undefined, fgColor));
    }
    return new ACell(cellData, undefined, undefined, undefined, 0);
}

var getChildFn = function(id, index)
{
    return treeStruc[id].ids[index];
}

var getChildCountFn = function(id)
{
    return treeStruc[id].ids.length;
}

var getIdFn = function(id)
{
    return id;
}
var getIconFn = function(id)
{
    var icon = null;
    return icon;
}
```

```
var getTreeOpen = function(id)
{
    return false;
}
var tt = buildTreeTable(root, 3, getCellFn, getChildFn,
getChildCountFn, getIdFn, getIconFn, getTreeOpen)


tt.header = [ new AHeaderCell([new AData("Name")], 160), new
AHeaderCell([new AData("Telefon")], 160), new AHeaderCell([new
AData("E-Mail")], 160) ]


result.object(tt);
```

3.3. Tipps und Tricks

3.3.1. Arbeiten mit Kopfdaten

Es bestehen zwei Möglichkeiten, die Kopfzeilen einer Baumtabelle zu bearbeiten. Entweder über das Kontextmenü im Navigator ("`Spalte hinzufügen`") und anschließender Benennung der Spalte.

Der zweite Weg führt über die `AHeaderCell`. Dazu ist der über `buildTreeTable` aufgebauten Tabelle eine Headerzelle hinzuzufügen (siehe im Beispiel oben). Die Funktion `AHeaderCell` ist wie folgt definiert:

```
AHeaderCell(data, width, fgColor, bgColor, font, alignment)
```

Die wichtigsten Parameter:

- **data**: Text, der in der Kopfzeile erscheint.
- **width**: Breite der Spalte. Wird vom Benutzer vorgegeben.

3.3.2. Arbeiten mit Zellendaten

Über `ACell` kann eine Zelle der Baumtabelle erzeugt werden. Diese Funktion ist wie folgt definiert:

```
ACell(data, fgColor, bgColor, font, alignment)
```

3.3.3. Ausrichten von Daten

Um Daten in einer Zelle auszurichten kann bei Aufruf der `ACell`-Funktion im letzten Parameter ein Alignment mitgegeben werden. Dieses Alignment wird in Kommazahlen eingegeben:

- **linksbündig**: 0.0
- **rechtsbündig**: 1.0
- **zentriert**: 0.5

Alle Werte dazwischen entsprechen der Positionierung nach links und rechts.

3.3.4. Verwendung der Formatierung

Um die zusätzlichen Konfigurationsmöglichkeiten, wie zum Beispiel Hintergrundfarbe und Schriftart, nutzen zu können, muss in der Funktion `buildTreeTable` des Prozesses `aditoTreeTableAdditional` folgender Code am Anfang der Funktion eingefügt werden, da der `AFontStyle` sonst nicht gefunden wird:

```
AFontStyle = {
      /**
       * Plain style.
       */
      PLAIN: 0,
      /**
       * Bold style.
       */
      BOLD: 1,
      /**
       * Italic style.
       */
      ITALIC: 2,
      /**
       * Both bold and italic style.
       */
      BOLD_ITALIC: 3
    };

    /**
     * Type of data.
     */
    ADataType = {
      /**
       * Plain text.
       */
      TEXT: "TEXT",
      /**
       * Text which is html formatted.
       */
      TEXT_HTML: "TEXT_HTML",
      /**
```

```
       * Text which is richtext formatted.
       */
      TEXT_RTF: "TEXT_RTF",
      /**
       * A number.
       */
      NUMBER: "NUMBER",
      /**
       * A timestamp. A timestamp is the time in milliseconds
starting from
       * 1. january 1970.
       */
      TIME: "TIME",
      /**
       * An image. An image is an Id from ASYS_BINARIES or a Base64
encoded string.
       */
      IMAGE: "IMAGE",
      /**
       * True or false.
       */
      BOOLEAN: "BOOLEAN",
      /**
       * A color. A color is best expressed by a hex number like
'0xff0000' for red.
       */
      COLOR: "COLOR"
    };
```

Beispiel

```
var getCellFn = function(id)
    {
        return new ACell(new AData(tree[id].name), '-11508066', '-
3355444', new
        AFont(AFontStyle.ITALIC, 12, "SansSerif"), 0.0);
    }
```