JSFoundSet

Method Summary

Boolean	deleteAllRecords() Delete all records in foundset, resulting in empty foundset.
Boolean	deleteRecord() Delete currently selected record(s).
Boolean	deleteRecord(record) Delete record from foundset.
Boolean	deleteRecord(index) Delete record with the given index.
Boolean	find() Set the foundset in find mode.
JSRecord	getRecord(index) Get the record object at the index.
Number	getRecordIndex(record) Get the record index.
Number	getSelectedIndex() Get the current record index of the foundset.
JSRecord	getSelectedRecord() Get the selected record.
Number	getSize() Get the number of records in this foundset.
Boolean	isInFind() Check if this foundset is in find mode.
Boolean	loadAllRecords() Loads all accessible records from the datasource into the foundset.
Number	newRecord() Create a new record on top of the foundset and change selection to it.
Number	newRecord(index, changeSelection) Create a new record in the foundset.
Number	search() Start the database search and use the results, returns the number of records, make sure you did "find" function first.
void	setSelectedIndex(index) Set the current record index.
void	sort(recordComparisonFunction) Sorts the foundset based on the given record comparator function.

Method Details

deleteAllRecords

Boolean **deleteAllRecords** ()

Delete all records in foundset, resulting in empty foundset.

Returns

Boolean - boolean true if all records could be deleted.

Sample

```
var success = forms.customer.foundset.deleteAllRecords();
```

deleteRecord

Boolean **deleteRecord** ()

Delete currently selected record(s).
If the foundset is in multiselect mode, all selected records are deleted.

Returns

Boolean - boolean true if all records could be deleted.

Sample

```
var success = forms.customer.foundset.deleteRecord();
//can return false incase of related foundset having records and orphans records are not allowed by the
relation
```

deleteRecord

Boolean **deleteRecord** (record)

Delete record from foundset.

Parameters

{JSRecord} record - The record to delete from the foundset.

Returns

Boolean - boolean true if record could be deleted.

Sample

```
var success = forms.customer.foundset.deleteRecord(rec);
//can return false incase of related foundset having records and orphans records are not allowed by the
relation
```

deleteRecord

Boolean **deleteRecord** (index)

Delete record with the given index.

Parameters

{Number} index - The index of the record to delete.

Returns

Boolean - boolean true if record could be deleted.

Sample

```
var success = forms.customer.foundset.deleteRecord(4);
//can return false incase of related foundset having records and orphans records are not allowed by the
relation
```

find

Boolean **find** ()

Set the foundset in find mode. (Start a find request), use the "search" function to perform/exit the find.

Before going into find mode, all unsaved records will be saved in the database.
If this fails (due to validation failures or sql errors) or is not allowed (autosave off), the foundset will not go into find mode.
Make sure the operator and the data (value) are part of the string passed to dataprovider (included inside a pair of quotation marks).
Note: always make sure to check the result of the find() method.

When in find mode, columns can be assigned string expressions (including operators) that are evaluated as:
General:
c1||c2 (condition1 or condition2)
c|format (apply format on condition like 'x|dd-MM-yyyy')
!c (not condition)
#c (modify condition, depends on column type)
^ (is null)
^= (is null or empty)
<x (less than value x)
>x (greater than value x)
<=x (less than or equals value x)
>=x (greater than or equals value x)
x...y (between values x and y, including values)
x (equals value x)

Number fields:
=x (equals value x)
^= (is null or zero)

Date fields:
#c (equals value x, entire day)
now (equals now, date and or time)
// (equals today)
today (equals today)

Text fields:
#c (case insensitive condition)
= x (equals a space and 'x')
^= (is null or empty)
%x% (contains 'x')
%x_y% (contains 'x' followed by any char and 'y')
\% (contains char '%')
_ (contains char '_')

Related columns can be assigned, they will result in related searches.
For example, "employees_to_department.location_id = headoffice" finds all employees in the specified location).

Searching on related aggregates is supported.
For example, "orders_to_details.total_amount = '>1000'" finds all orders with total order details amount more than 1000.

Arrays can be used for searching a number of values, this will result in an 'IN' condition that will be used in the search.
The values are not restricted to strings but can be any type that matches the column type.
For example, "record.department_id = [1, 33, 99]"

Returns

Boolean - true if the foundset is now in find mode, false otherwise.

Sample

```
if (forms.customer.foundset.find()) //find will fail if autosave is disabled and there are unsaved records
{
	columnTextDataProvider = 'a search value'
	// for numbers you have to make sure to format it correctly so that the decimal point is in your
locales notation (. or ,)
	columnNumberDataProvider = '>' + utils.numberFormat(anumber, '####.00');
	columnDateDataProvider = '31-12-2010|dd-MM-yyyy'
	forms.customer.foundset.search()
}
```

getRecord

JSRecord **getRecord** (index)
Get the record object at the index.

Parameters

{Number} index - record index

Returns

JSRecord - Record record.

Sample

```
var record = forms.customer.foundset.getRecord(index);
```

getRecordIndex

Number **getRecordIndex** (record)

Get the record index. Will return -1 if the record can't be found.

Parameters

{JSRecord} record - Record

Returns

Number - int index.

Sample

```
var index = forms.customer.foundset.getRecordIndex(record);
```

getSelectedIndex

Number **getSelectedIndex** ()

Get the current record index of the foundset.

Returns

Number - int current index (1-based)

Sample

```
//gets the current record index in the current foundset
var current = forms.customer.foundset.getSelectedIndex();
//sets the next record in the foundset
forms.customer.foundset.setSelectedIndex(current+1);
```

getSelectedRecord

JSRecord **getSelectedRecord** ()

Get the selected record.

Returns

JSRecord - Record record.

Sample

```
var selectedRecord = forms.customer.foundset.getSelectedRecord();
```

getSize

Number **getSize** ()

Get the number of records in this foundset.
This is the number of records loaded, note that when looping over a foundset, size() may
increase as more records are loaded.

Returns

Number - int current size.

Sample

```
var nrRecords = forms.customer.foundset.getSize()

// to loop over foundset, recalculate size for each record
for (var i = 1; i <= forms.customer.foundset.getSize(); i++)
{
        var rec = forms.customer.foundset.getRecord(i);
}
```

isInFind

Boolean **isInFind** ()

Check if this foundset is in find mode.

Returns

Boolean - boolean is in find mode.

Sample

```
//Returns true when find was called on this foundset and search has not been called yet
forms.customer.foundset.isInFind();
```

loadAllRecords

Boolean **loadAllRecords** ()

Loads all accessible records from the datasource into the foundset.
Filters on the foundset are applied.

Before loading the records, all unsaved records will be saved in the database.
If this fails (due to validation failures or sql errors) or is not allowed (autosave off),
records will not be loaded,

Returns

Boolean - true if records are loaded, false otherwise.

Sample

```
forms.customer.foundset.loadAllRecords();
```

newRecord

Number **newRecord** ()

Create a new record on top of the foundset and change selection to it. Returns -1 if the record can't be made.

Returns

Number - int index of new record.

Sample

```
// foreign key data is only filled in for equals (=) relation items
var idx = forms.customer.foundset.newRecord(false); // add as last record
// forms.customer.foundset.newRecord(); // adds as first record
// forms.customer.foundset.newRecord(2); //adds as second record
if (idx >= 0) // returned index is -1 in case of failure
{
        forms.customer.foundset.some_column = "some text";
        application.output("added on position " + idx);
        // when adding at the end of the foundset, the returned index
        // corresponds with the size of the foundset
}
```

newRecord

Number **newRecord** (index, changeSelection)

Create a new record in the foundset. Returns -1 if the record can't be made.

Parameters

{Number} index - the new record is added at specified index.
{Boolean} changeSelection - when true the selection is changed to the new record.

Returns

Number - int index of new record.

Sample

```
// foreign key data is only filled in for equals (=) relation items
var idx = forms.customer.foundset.newRecord(false); // add as last record
// forms.customer.foundset.newRecord(); // adds as first record
// forms.customer.foundset.newRecord(2); //adds as second record
if (idx >= 0) // returned index is -1 in case of failure
{
        forms.customer.foundset.some_column = "some text";
        application.output("added on position " + idx);
        // when adding at the end of the foundset, the returned index
        // corresponds with the size of the foundset
}
```

search

Number **search** ()

Start the database search and use the results, returns the number of records, make sure you did "find" function first.
Clear results from previous searches.

Note: Omitted records are automatically excluded when performing a search - meaning that the foundset result by default will not include omitted records.

Returns

Number - the recordCount

Sample

```
var recordCount = forms.customer.foundset.search();
//var recordCount = forms.customer.foundset.search(false,false); //to extend foundset
```

setSelectedIndex

void **setSelectedIndex** (index)

Set the current record index.

Parameters

{Number} index - index to set (1-based)

Returns

void

Sample

```
//gets the current record index in the current foundset
var current = forms.customer.foundset.getSelectedIndex();
//sets the next record in the foundset
forms.customer.foundset.setSelectedIndex(current+1);
```

sort

void **sort** (recordComparisonFunction)

Sorts the foundset based on the given record comparator function.
The comparator function is called to compare
two records, that are passed as arguments, and
it will return -1/0/1 if the first record is less/equal/greater
then the second record.

The function based sorting does not work with printing.
It is just a temporary in-memory sort.

NOTE: starting with 7.2 release this function doesn't save the data anymore

Parameters

{Function} recordComparisonFunction - record comparator function

Returns

void

Sample

```
forms.customer.foundset.sort(mySortFunction);

function mySortFunction(r1, r2)
{
        var o = 0;
        if(r1.id < r2.id)
        {
                o = -1;
        }
        else if(r1.id > r2.id)
        {
                o = 1;
        }
        return o;
}
```