JSFoundSet

A JSFoundset is a scriptable object, it contains record objects defined by its SQL. It does lazy load the PKs and records.
Each foundset has its own record set and selected index

Property Summary

String[]	alldataproviders Get all dataproviders of the foundset.
Boolean	multiSelect Get or set the multiSelect flag of the foundset.

Method Summary

Boolean	addFoundSetFilterParam(dataprovider, operator, value) Add a filter parameter that is permanent per user session to limit a specified foundset of records.
Boolean	addFoundSetFilterParam(dataprovider, operator, value, name) Add a filter parameter that is permanent per user session to limit a specified foundset of records.
void	clear() Clear the foundset.
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.
JSFoundSet	duplicateFoundSet() Get a duplicate of the foundset.
Number	duplicateRecord() Duplicate current record, change selection to new record, place on top.
Number	duplicateRecord(onTop) Duplicate selected record, change selection to new record.
Number	duplicateRecord(onTop, changeSelection) Duplicate selected record.
Number	duplicateRecord(index) Duplicate record at index in the foundset, change selection to new record, place on top.
Number	duplicateRecord(index, onTop) Duplicate record at index in the foundset, change selection to new record.
Number	duplicateRecord(index, onTop, changeSelection) Duplicate record at index in the foundset.
Number	duplicateRecord(index, location) Duplicate record at index in the foundset, change selection to new record.
Number	duplicateRecord(index, location, changeSelection) Duplicate record at index in the foundset.
Boolean	find() Set the foundset in find mode.
String	getCurrentSort() Get the current sort columns.
Object	getDataProviderValue(dataProviderID) Get a value based on a dataprovider name.
String	getDataSource() Get the datasource used.
Object[][]	getFoundSetFilterParams() Get the list of previously defined foundset filters.
Object[][]	getFoundSetFilterParams(filterName) Get a previously defined foundset filter, using its given name.
QBSelect	getQuery() Get the query that the foundset is currently using.
JSRecord	getRecord(index) Get the record object at the index.
Number	getRecordIndex(record) Get the record index.
String	getRelationName() Gets the relation name (null if not a related foundset).
Number	getSelectedIndex() Get the current record index of the foundset.

Number[]	getSelectedIndexes() Get the indexes of the selected records.
JSRecord	getSelectedRecord() Get the selected record.
JSRecord[]	getSelectedRecords() Get the selected records.
Number	getSize() Get the number of records in this foundset.
Boolean	hasConditions() Check wether the foundset has any conditions from a previous find action.
void	invertRecords() Invert the foundset against all rows of the current table.
Boolean	isInFind() Check if this foundset is in find mode.
Boolean	loadAllRecords() Loads all accessible records from the datasource into the foundset.
Boolean	loadOmittedRecords() Loads the records that are currently omitted as a foundset.
Boolean	loadRecords() Reloads all last (related) records again, if, for example, after search in tabpanel.
Boolean	loadRecords(foundset) Copies foundset data from another foundset.
Boolean	loadRecords(dataset) Loads a primary key dataset, will remove related sort.
Boolean	loadRecords(querybuilder) Loads records into form foundset based on a query builder object (also known as 'Form by query').
Boolean	loadRecords(uuidpk) Loads a single record by primary key, will remove related sort.
Boolean	loadRecords(numberpk) Loads a single record by primary key, will remove related sort.
Boolean	loadRecords(queryString) Loads records into form foundset based on a query (also known as 'Form by query').
Boolean	loadRecords(queryString, argumentsArray) Loads records into form foundset based on a query (also known as 'Form by query').
Number	newRecord() Create a new record on top of the foundset and change selection to it.
Number	newRecord(onTop) Create a new record in the foundset and change selection to it.
Number	newRecord(onTop, changeSelection) Create a new record in the foundset.
Number	newRecord(index) Create a new record in the foundset and change selection to it.
Number	newRecord(index, changeSelection) Create a new record in the foundset.
Boolean	omitRecord() Omit current record, to be shown with loadOmittedRecords.
Boolean	omitRecord(index) Omit record under the given index, to be shown with loadOmittedRecords.
void	relookup() Perform a relookup for the currently selected records Lookups are defined in the dataprovider (columns) auto-enter setting and are normally performed over a relation upon record creation.
void	relookup(index) Perform a relookup for the record under the given index Lookups are defined in the dataprovider (columns) auto-enter setting and are normally performed over a relation upon record creation.
Boolean	removeFoundSetFilterParam(name) Remove a named foundset filter.
Number	search() Start the database search and use the results, returns the number of records, make sure you did "find" function first.
Number	search(clearLastResults) Start the database search and use the results, returns the number of records, make sure you did "find" function first.
Number	search(clearLastResults, reduceSearch) Start the database search and use the results, returns the number of records, make sure you did "find" function first.
Boolean	selectRecord(pkid1, [pkid2], [pkidn]) Select the record based on pk data.
void	setDataProviderValue(dataProviderID, value) Set a value based on a dataprovider name.
void	setSelectedIndex(index) Set the current record index.
void	setSelectedIndexes(indexes) Set the selected records indexes.
void	sort(sortString) Sorts the foundset based on the given sort string.
void	sort(sortString, defer) Sorts the foundset based on the given sort string.
void	sort(recordComparisonFunction) Sorts the foundset based on the given record comparator function.

JSFoundSet unrelate()
Create a new unrelated foundset that is a copy of the current foundset.

Property Details

alldataproviders

Get all dataproviders of the foundset.

Returns

String[]

Sample

```
var dataprovidersNames = forms.customer.alldataproviders;
application.output("This foundset has " + dataprovidersNames.length + " data providers.")
for (var i=0; i<dataprovidersNames.length; i++)
        application.output(dataprovidersNames[i]);
```

multiSelect

Get or set the multiSelect flag of the foundset.

Returns

Boolean

Sample

```
// allow user to select multiple rows.
forms.customer.foundset.multiSelect = true;
```

Method Details

addFoundSetFilterParam

Boolean **addFoundSetFilterParam** (dataprovider, operator, value)

Add a filter parameter that is permanent per user session to limit a specified foundset of records.
Use clear() or loadAllRecords() to make the filter effective.
Multiple filters can be added to the same dataprovider, they will all be applied.

Parameters

{String} dataprovider - String column to filter on.
{String} operator - String operator: =, <, >, >=, <=, !=, (NOT) LIKE, (NOT) IN, (NOT) BETWEEN and IS (NOT) NULL optionally augmented with modifiers "#" (ignore case) or "^||" (or-is-null).
{Object} value - Object filter value (for in array and between an array with 2 elements)

Returns

Boolean - true if adding the filter succeeded, false otherwise.

Sample

```
// Filter a fondset on a dataprovider value.
// Note that multiple filters can be added to the same dataprovider, they will all be applied.

var success = forms.customer.foundset.addFoundSetFilterParam('customerid', '=', 'BLONP', 'custFilter');
//possible to add multiple
forms.customer.foundset.loadAllRecords();//to make param(s) effective
// Named filters can be removed using forms.customer.foundset.removeFoundSetFilterParam(filterName)
```

addFoundSetFilterParam

Boolean **addFoundSetFilterParam** (dataprovider, operator, value, name)

Add a filter parameter that is permanent per user session to limit a specified foundset of records.
Use clear() or loadAllRecords() to make the filter effective.
The filter is removed again using removeFoundSetFilterParam(name).

Parameters

{String} dataprovider - String column to filter on.
{String} operator - String operator: =, <, >, >=, <=, !=, (NOT) LIKE, (NOT) IN, (NOT) BETWEEN and IS (NOT) NULL optionally augmented with modifiers "#" (ignore case) or "^||" (or-is-null).
{Object} value - Object filter value (for in array and between an array with 2 elements)
{String} name - String name, used to remove the filter again.

Returns

Boolean - true if adding the filter succeeded, false otherwise.

Sample

```
var success = forms.customer.foundset.addFoundSetFilterParam('customerid', '=', 'BLONP', 'custFilter');
//possible to add multiple
// Named filters can be removed using forms.customer.foundset.removeFoundSetFilterParam(filterName)

// you can use modifiers in the operator as well, filter on companies where companyname is null or equals-
ignore-case 'servoy'
var ok = forms.customer.foundset.addFoundSetFilterParam('companyname', '#^||=', 'servoy')

forms.customer.foundset.loadAllRecords();//to make param(s) effective
```

clear

void **clear** ()

Clear the foundset.

Returns

void

Sample

```
//Clear the foundset, including searches that may be on it
forms.customer.foundset.clear();
```

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
```

duplicateFoundSet

JSFoundSet **duplicateFoundSet** ()

Get a duplicate of the foundset.

Returns

JSFoundSet - foundset duplicate.

Sample

```
var dupFoundset = forms.customer.foundset.duplicateFoundSet();
forms.customer.foundset.find();
//search some fields
var count = forms.customer.foundset.search();
if (count == 0)
{
        plugins.dialogs.showWarningDialog('Alert', 'No records found','OK');
        forms.customer.foundset.loadRecords(dupFoundset);
}
```

duplicateRecord

Number **duplicateRecord** ()

Duplicate current record, change selection to new record, place on top.

Returns

Number - 0 if record was not created or the record index if it was created.

Sample

```
forms.customer.foundset.duplicateRecord();
forms.customer.foundset.duplicateRecord(false); //duplicate the current record, adds at bottom
forms.customer.foundset.duplicateRecord(1,2); //duplicate the first record as second record
//duplicates the record (record index 3), adds on top and selects the record
forms.customer.foundset.duplicateRecord(3,true,true);
```

duplicateRecord

Number **duplicateRecord** (onTop)

Duplicate selected record, change selection to new record.

Parameters

{Boolean} onTop - when true the new record is added as the topmost record.

Returns

Number - 0 if record was not created or the record index if it was created.

Sample

```
forms.customer.foundset.duplicateRecord();
forms.customer.foundset.duplicateRecord(false); //duplicate the current record, adds at bottom
forms.customer.foundset.duplicateRecord(1,2); //duplicate the first record as second record
//duplicates the record (record index 3), adds on top and selects the record
forms.customer.foundset.duplicateRecord(3,true,true);
```

duplicateRecord

Number **duplicateRecord** (onTop, changeSelection)

Duplicate selected record.

Parameters

{Boolean} onTop - when true the new record is added as the topmost record.
{Boolean} changeSelection - when true the selection is changed to the duplicated record.

Returns

Number - 0 if record was not created or the record index if it was created.

Sample

```
forms.customer.foundset.duplicateRecord();
forms.customer.foundset.duplicateRecord(false); //duplicate the current record, adds at bottom
forms.customer.foundset.duplicateRecord(1,2); //duplicate the first record as second record
//duplicates the record (record index 3), adds on top and selects the record
forms.customer.foundset.duplicateRecord(3,true,true);
```

duplicateRecord

Number **duplicateRecord** (index)

Duplicate record at index in the foundset, change selection to new record, place on top.

Parameters

{Number} index - The index of the record to duplicate; defaults to currently selected index. Ignored if first given parameter is a boolean value.

Returns

Number - 0 if record was not created or the record index if it was created.

Sample

```
forms.customer.foundset.duplicateRecord();
forms.customer.foundset.duplicateRecord(false); //duplicate the current record, adds at bottom
forms.customer.foundset.duplicateRecord(1,2); //duplicate the first record as second record
//duplicates the record (record index 3), adds on top and selects the record
forms.customer.foundset.duplicateRecord(3,true,true);
```

duplicateRecord

Number **duplicateRecord** (index, onTop)

Duplicate record at index in the foundset, change selection to new record.

Parameters

{Number} index - The index of the record to duplicate; defaults to currently selected index. Ignored if first given parameter is a boolean value.
{Boolean} onTop - when true the new record is added as the topmost record.

Returns

Number - 0 if record was not created or the record index if it was created.

Sample

```
forms.customer.foundset.duplicateRecord();
forms.customer.foundset.duplicateRecord(false); //duplicate the current record, adds at bottom
forms.customer.foundset.duplicateRecord(1,2); //duplicate the first record as second record
//duplicates the record (record index 3), adds on top and selects the record
forms.customer.foundset.duplicateRecord(3,true,true);
```

duplicateRecord

Number **duplicateRecord** (index, onTop, changeSelection)

Duplicate record at index in the foundset.

Parameters

{Number} index - The index of the record to duplicate; defaults to currently selected index. Ignored if first given parameter is a boolean value.
{Boolean} onTop - when true the new record is added as the topmost record.
{Boolean} changeSelection - when true the selection is changed to the duplicated record.

Returns

Number - 0 if record was not created or the record index if it was created.

Sample

```
forms.customer.foundset.duplicateRecord();
forms.customer.foundset.duplicateRecord(false); //duplicate the current record, adds at bottom
forms.customer.foundset.duplicateRecord(1,2); //duplicate the first record as second record
//duplicates the record (record index 3), adds on top and selects the record
forms.customer.foundset.duplicateRecord(3,true,true);
```

duplicateRecord

Number **duplicateRecord** (index, location)

Duplicate record at index in the foundset, change selection to new record.

Parameters

{Number} index - The index of the record to duplicate; defaults to currently selected index. Ignored if first given parameter is a boolean value.
{Number} location - the new record is added at specified index

Returns

Number - 0 if record was not created or the record index if it was created.

Sample

```
forms.customer.foundset.duplicateRecord();
forms.customer.foundset.duplicateRecord(false); //duplicate the current record, adds at bottom
forms.customer.foundset.duplicateRecord(1,2); //duplicate the first record as second record
//duplicates the record (record index 3), adds on top and selects the record
forms.customer.foundset.duplicateRecord(3,true,true);
```

duplicateRecord

Number **duplicateRecord** (index, location, changeSelection)

Duplicate record at index in the foundset.

Parameters

{Number} index - The index of the record to duplicate; defaults to currently selected index. Ignored if first given parameter is a boolean value.
{Number} location - the new record is added at specified index
{Boolean} changeSelection - when true the selection is changed to the duplicated record.

Returns

Number - 0 if record was not created or the record index if it was created.

Sample

```
forms.customer.foundset.duplicateRecord();
forms.customer.foundset.duplicateRecord(false); //duplicate the current record, adds at bottom
forms.customer.foundset.duplicateRecord(1,2); //duplicate the first record as second record
//duplicates the record (record index 3), adds on top and selects the record
forms.customer.foundset.duplicateRecord(3,true,true);
```

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()
}
```

getCurrentSort

String **getCurrentSort** ()

Get the current sort columns.

Returns

String - String sort columns

Sample

```
//reverse the current sort

//the original sort "companyName asc, companyContact desc"
//the inversed sort "companyName desc, companyContact asc"
var foundsetSort = foundset.getCurrentSort()
var sortColumns = foundsetSort.split(',')
var newFoundsetSort = ''
for(var i=0; i<sortColumns.length; i++)
{
	var currentSort = sortColumns[i]
	var sortType = currentSort.substring(currentSort.length-3)
	if(sortType.equalsIgnoreCase('asc'))
	{
		newFoundsetSort += currentSort.replace(' asc', ' desc')
	}
	else
	{
		newFoundsetSort += currentSort.replace(' desc', ' asc')
	}
	if(i != sortColumns.length - 1)
	{
		newFoundsetSort += ','
	}
}
foundset.sort(newFoundsetSort)
```

getDataProviderValue

Object **getDataProviderValue** (dataProviderID)

Get a value based on a dataprovider name.

Parameters

{String} dataProviderID - data provider name

Returns

Object - Object value

Sample

```
var val = forms.customer.foundset.getDataProviderValue('contact_name');
```

getDataSource

String **getDataSource** ()

Get the datasource used.
The datasource is an url that describes the data source.

Returns

String - String data source.

Sample

```
var dataSource = forms.customer.foundset.getDataSource();
```

getFoundSetFilterParams

Object[][] **getFoundSetFilterParams** ()

Get the list of previously defined foundset filters.
The result is an array of:
[tableName, dataprovider, operator, value, name]

Returns

Object[][] - Array of filter definitions.

Sample

```
var params = foundset.getFoundSetFilterParams()
for (var i = 0; params != null && i < params.length; i++)
{
	application.output('FoundSet filter on table ' + params[i][0]+ ': '+ params[i][1]+ ' '+params[i][2]+
' '+params[i][3] +(params[i][4] == null ? ' [no name]' : ' ['+params[i][4]+']'))
}
```

getFoundSetFilterParams

Object[][] **getFoundSetFilterParams** (filterName)

Get a previously defined foundset filter, using its given name.
The result is an array of:
[tableName, dataprovider, operator, value, name]

Parameters

{String} filterName - name of the filter to retrieve.

Returns

Object[][] - Array of filter definitions.

Sample

```
var params = foundset.getFoundSetFilterParams()
for (var i = 0; params != null && i < params.length; i++)
{
	application.output('FoundSet filter on table ' + params[i][0]+ ': '+ params[i][1]+ ' '+params[i][2]+
' '+params[i][3] +(params[i][4] == null ? ' [no name]' : ' ['+params[i][4]+']'))
}
```

getQuery

QBSelect **getQuery** ()

Get the query that the foundset is currently using.
When the founset is in find mode, the find conditions are included in the resulting query.
So the query that would be used when just calling search() (or search(true,true)) is returned.
Note that foundset filters are included and table filters are not included in the query.

Returns

QBSelect - query.

Sample

```
var q = foundset.getQuery()
q.where.add(q.columns.x.eq(100))
foundset.loadRecords(q);
```

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);
```

getRelationName

String **getRelationName** ()

Gets the relation name (null if not a related foundset).

Returns

String - String relation name when related.

Sample

```
var relName = forms.customer.foundset.getRelationName();
```

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);
```

getSelectedIndexes

Number[] **getSelectedIndexes** ()

Get the indexes of the selected records.
When the founset is in multiSelect mode (see property multiSelect), a selection can consist of more than one index.

Returns

Number[] - Array current indexes (1-based)

Sample

```
// modify selection to the first selected item and the following row only
var current = forms.customer.foundset.getSelectedIndexes();
if (current.length > 1)
{
        var newSelection = new Array();
        newSelection[0] = current[0]; // first current selection
        newSelection[1] = current[0] + 1; // and the next row
        forms.customer.foundset.setSelectedIndexes(newSelection);
}
```

getSelectedRecord

JSRecord **getSelectedRecord** ()

Get the selected record.

Returns

JSRecord - Record record.

Sample

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

getSelectedRecords

JSRecord[] **getSelectedRecords** ()

Get the selected records.
When the founset is in multiSelect mode (see property multiSelect), selection can be a more than 1 record.

Returns

JSRecord[] - Array current records.

Sample

```
var selectedRecords = forms.customer.foundset.getSelectedRecords();
```

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);
}
```

hasConditions

Boolean **hasConditions** ()

Check wether the foundset has any conditions from a previous find action.

Returns

Boolean - wether the foundset has find-conditions

Sample

```
if (forms.customer.foundset.hasConditions())
{
                // foundset had find actions
}
```

invertRecords

void **invertRecords** ()

Invert the foundset against all rows of the current table.
All records that are not in the foundset will become the current foundset.

Returns

void

Sample

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

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();
```

loadOmittedRecords

Boolean **loadOmittedRecords** ()

Loads the records that are currently omitted as a foundset.

Before loading the omitted 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),
omitted records will not be loaded,

Returns

Boolean - true if records are loaded, false otherwise.

Sample

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

loadRecords

Boolean **loadRecords** ()

Reloads all last (related) records again, if, for example, after search in tabpanel.
When in find mode, this will reload the records from before the find() call.

Returns

Boolean - true if successful

Sample

```
//to reload all last (related) records again, if for example when searched in tabpanel
 forms.customer.foundset.loadRecords();
```

loadRecords

Boolean **loadRecords** (foundset)

Copies foundset data from another foundset.
This will alter the foundset state to the state of the foundset that is given.
If you really just want to use the given foundset on the form itself, then you need to use controller.loadRecords(foundset)
that will change the instance of the foundset that is used for this form. Not just update an existing form.

If you copy over a relation into this foundset, then this foundset will not be a related foundset, it will not automatically update its state
of records are updated or added that belong to that relation. It will only be a snapshot of that related foundsets state.

Parameters

{JSFoundSet} foundset - The foundset to load records from

Returns

Boolean - true if successful

Sample

```
//Copies foundset data from another foundset
forms.customer.foundset.loadRecords(fs);
```

loadRecords

Boolean **loadRecords** (dataset)

Loads a primary key dataset, will remove related sort.

Parameters

{JSDataSet} dataset - pkdataset

Returns

Boolean - true if successful

Sample

```
// loads a primary key dataset, will remove related sort!
//var dataset = databaseManager.getDataSetByQuery(...);
// dataset must match the table primary key columns (alphabetically ordered)
forms.customer.foundset.loadRecords(dataset);
```

loadRecords

Boolean **loadRecords** (querybuilder)

Loads records into form foundset based on a query builder object (also known as 'Form by query').
When the founset is in find mode, the find states are discarded, the foundset will go out of find mode and the foundset will be loaded using the query.
If the foundset is related, the relation-condition will be added to the query.

Parameters

{QBSelect} querybuilder - the query builder

Returns

Boolean - true if successful

Sample

```
forms.customer.foundset.loadRecords(qbselect);
```

loadRecords

Boolean **loadRecords** (uuidpk)

Loads a single record by primary key, will remove related sort.

NOTE: This function will return true if the foundset was altered/changed. It is up to the developer to check for the presence of actual data using getSize().

Parameters

{UUID} uuidpk - single-column pk value

Returns

Boolean - true if successful

Sample

```
//Loads a single record by primary key, will remove related sort!
forms.customer.foundset.loadRecords(application.getUUID('6b5e2f5d-047e-45b3-80ee-3a32267b1f20'));
```

loadRecords

Boolean **loadRecords** (numberpk)

Loads a single record by primary key, will remove related sort.

NOTE: This function will return true if the foundset was altered/changed. It is up to the developer to check for the presence of actual data using getSize().

Parameters

{Number} numberpk - single-column pk value

Returns

Boolean - true if successful

Sample

```
//Loads a single record by primary key, will remove related sort!
forms.customer.foundset.loadRecords(123);
```

loadRecords

Boolean **loadRecords** (queryString)

Loads records into form foundset based on a query (also known as 'Form by query'). The query must be a valid sql select.
If the foundset is related this function is not allowed.

see foundset.loadRecords(QBSelect).

Parameters

{String} queryString - select statement

Returns

Boolean - true if successful

Sample

```
//loads records in to the foundset based on a query (also known as 'Form by query')
forms.customer.foundset.loadRecords(sqlstring);
```

loadRecords

Boolean **loadRecords** (queryString, argumentsArray)

Loads records into form foundset based on a query (also known as 'Form by query'). The query must be a valid sql select.
If the foundset is related this function is not allowed.

see foundset.loadRecords(QBSelect).

Parameters

{String} queryString - select statement
{Object[]} argumentsArray - arguments to query

Returns

Boolean - true if successful

Sample

```
//loads records in to the foundset based on a query (also known as 'Form by query')
forms.customer.foundset.loadRecords(sqlstring,parameters);
```

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** (onTop)

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

Parameters

{Boolean} onTop - when true the new record is added as the topmost 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
}
```

newRecord

Number **newRecord** (onTop, changeSelection)

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

Parameters

{Boolean} onTop - when true the new record is added as the topmost record.
{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
}
```

newRecord

Number **newRecord** (index)

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

Parameters

{Number} index - the new record is added at specified index.

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
}
```

omitRecord

Boolean **omitRecord** ()

Omit current record, to be shown with loadOmittedRecords.
If the foundset is in multiselect mode, all selected records are omitted (when no index parameter is used).

Note: The omitted records list is discarded when these functions are executed: loadAllRecords, loadRecords(dataset), loadRecords(sqlstring), invertRecords()

Returns

Boolean - boolean true if all records could be omitted.

Sample

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

omitRecord

Boolean **omitRecord** (index)

Omit record under the given index, to be shown with loadOmittedRecords.
If the foundset is in multiselect mode, all selected records are omitted (when no index parameter is used).

Note: The omitted records list is discarded when these functions are executed: loadAllRecords, loadRecords(dataset), loadRecords(sqlstring), invertRecords()

Parameters

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

Returns

Boolean - boolean true if all records could be omitted.

Sample

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

relookup

void **relookup** ()

Perform a relookup for the currently selected records
Lookups are defined in the dataprovider (columns) auto-enter setting and are normally performed over a relation upon record creation.

Returns

void

Sample

```
forms.customer.foundset.relookup(1);
```

relookup

void **relookup** (index)

Perform a relookup for the record under the given index
Lookups are defined in the dataprovider (columns) auto-enter setting and are normally performed over a relation upon record creation.

Parameters

{Number} index - record index (1-based)

Returns

void

Sample

```
forms.customer.foundset.relookup(1);
```

removeFoundSetFilterParam

Boolean **removeFoundSetFilterParam** (name)

Remove a named foundset filter.
Use clear() or loadAllRecords() to make the filter effective.

Parameters

{String} name - String filter name.

Returns

Boolean - true if removing the filter succeeded, false otherwise.

Sample

```
var success = forms.customer.foundset.removeFoundSetFilterParam('custFilter');// removes all filters with
this name
forms.customer.foundset.loadAllRecords();//to make param(s) effective
```

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
```

search

Number **search** (clearLastResults)

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

Parameters

{Boolean} clearLastResults - boolean, clear previous search, default true

Returns

Number - the recordCount

Sample

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

search

Number **search** (clearLastResults, reduceSearch)

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

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

Parameters

{Boolean} clearLastResults - boolean, clear previous search, default true
{Boolean} reduceSearch - boolean, reduce (true) or extend (false) previous search results, default true

Returns

Number - the recordCount

Sample

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

selectRecord

Boolean **selectRecord** (pkid1, [pkid2], [pkidn])

Select the record based on pk data.
Note that if the foundset has not loaded the record with the pk, selectrecord will fail.

In case of a table with a composite key, the pk sequence must match the alphabetical ordering of the pk column names.

Parameters

pkid1 - primary key
[pkid2] - second primary key (in case of composite primary key)
[pkidn] - nth primary key

Returns

Boolean - true if succeeded.

Sample

```
forms.customer.foundset.selectRecord(pkid1,pkid2,pkidn);//pks must be alphabetically set! It is also
possible to use an array as parameter.
```

setDataProviderValue

void **setDataProviderValue** (dataProviderID, value)

Set a value based on a dataprovider name.

Parameters

{String} dataProviderID - data provider name
{Object} value - value to set

Returns

void

Sample

```
forms.customer.foundset.setDataProviderValue('contact_name','mycompany');
```

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);
```

setSelectedIndexes

void **setSelectedIndexes** (indexes)

Set the selected records indexes.

Parameters

{Number[]} indexes - An array with indexes to set.

Returns

void

Sample

```
// modify selection to the first selected item and the following row only
var current = forms.customer.foundset.getSelectedIndexes();
if (current.length > 1)
{
        var newSelection = new Array();
        newSelection[0] = current[0]; // first current selection
        newSelection[1] = current[0] + 1; // and the next row
        forms.customer.foundset.setSelectedIndexes(newSelection);
}
```

sort

void **sort** (sortString)

Sorts the foundset based on the given sort string.
TIP: You can use the Copy button in the developer Select Sorting Fields dialog to get the needed syntax string for the desired sort fields/order.

Parameters

{String} sortString - the specified columns (and sort order)

Returns

void

Sample

```
forms.customer.foundset.sort('columnA desc,columnB asc');
```

sort

void **sort** (sortString, defer)

Sorts the foundset based on the given sort string.
TIP: You can use the Copy button in the developer Select Sorting Fields dialog to get the needed syntax string for the desired sort fields/order.

Parameters

{String} sortString - the specified columns (and sort order)
{Boolean} defer - when true, the "sortString" will be just stored, without performing a query on the database (the actual sorting will be deferred until the next data loading action).

Returns

void

Sample

```
forms.customer.foundset.sort('columnA desc,columnB asc');
```

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;
}
```

unrelate

JSFoundSet **unrelate** ()

Create a new unrelated foundset that is a copy of the current foundset.
If the current foundset is not related, no copy will made.

Returns

JSFoundSet - FoundSet unrelated foundset.

Sample

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