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

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([index/record]) Delete current/parameter record or the record under the given index.
JSFoundset	#duplicateFoundSet() Get a duplicate of the foundset.
Number	#duplicateRecord([index], [location], [changeSelection]) Duplicate current record or 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.
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 selected records indexes.
JSRecord	#getSelectedRecord() Get the selected record.
JSRecord[]	#getSelectedRecords() Get the selected records.
Number	#getSize() Get the number of records in this foundset.
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([input], [queryArgumentsArray]) Load records with primary key (dataset/number/uuid) or query.
Number	#newRecord([location], [changeSelection]) Create a new record in the foundset.
Boolean	#omitRecord([index]) Omit current record or the record under the given index, to be shown with loadOmittedRecords.
void	#relookup([index]) Perform a relookup for the current record or 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([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/recordComparator, [defer]) Sorts the foundset based on the given sort string or record comparator function.
JSFoundset	#unrelate() Create a new unrelated foundset that is a copy of the current foundset.

Property Details

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
{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**([index/record])

Delete current/parameter record or the record under the given index.
If the foundset is in multiselect mode, all selected records are deleted (when no parameter is used).

Parameters

[index/record] – index of record to delete or record itself.

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
```

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');
        controller.loadRecords(dupFoundset);
}
```

duplicateRecord

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

Duplicate current record or record at index in the foundset.

Parameters

[index] – index of record to duplicate; defaults to currently selected index. Ignored if first given parameter is a boolean value.
[location] – a boolean or number when true the new record is added as the topmost record, when a number, the new record is added at specified index ; defaults to 1.
{Boolean} [changeSelection] – when true the selection is changed to the duplicated record; defaults to true.

Returns

Number – true if succesful

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.

Also see
.search
databaseManager.setAutoSave
controller.find
controller.search

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

Also see

.sort

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.

Also see
databaseManager.getDataSourceServerName
databaseManager.getDataSourceTableName

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]+']'))
}
```

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.

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

Returns

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

Sample

```
var current = forms.customer.foundset.getSelectedIndexes();
var newSelection = new Array();
newSelection[0] = current[0];
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);
        }
```

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.

Also see
.addFoundSetFilterParam

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**([input], [queryArgumentsArray])

Load records with primary key (dataset/number/uuid) or query.

Load records can be used in 5 different ways
1) to copy foundset data from another foundset
foundset.loadRecords(fs);

2) to load a primary key dataset, will remove related sort!
var dataset = databaseManager.getDataSetByQuery(...);
foundset.loadRecords(dataset);

3) to load a single record by primary key, will remove related sort! (pk should be a number or UUID)
foundset.loadRecords(123);
foundset.loadRecords(application.getUUID('6b5e2f5d-047e-45b3-80ee-3a32267b1f20'));

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

5) to load records in to the form based on a query (also known as 'Form by query')
foundset.loadRecords(sqlstring,parameters);
limitations/requirements for sqlstring are:
-must start with 'select'
-the selected columns must be the (Servoy Form) table primary key columns (alphabetically ordered like 'select a_id, b_id,c_id ...')
-can contain '?' which are replaced with values from the array supplied to parameters argument
if the sqlstring contains an 'order by' clause, the records will be sorted accordingly and additional constraints apply:
-must contain 'from' keyword
-the 'from' must be a comma separated list of table names
-must at least select from the table used in Servoy Form
-cannot contain 'group by', 'having' or 'union'
-all columns must be fully qualified like 'orders.order_id'

Parameters

[input] – foundset/pkdataset/single_pk/query
[queryArgumentsArray] – used when input is a query

Returns

Boolean – true if successful

Sample

```
//Load records can be used in 5 different ways
//1) to copy foundset data from another foundset
//forms.customer.foundset.loadRecords(fs);

//2) to load 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);

//3) to load a single record by primary key, will remove related sort! (pk should be a number or UUID)
//forms.customer.foundset.loadRecords(123);
//forms.customer.foundset.loadRecords(application.getUUID('6b5e2f5d-047e-45b3-80ee-3a32267b1f20'));

//4) to reload all last related records again, if for example when searched in tabpanel
//forms.customer.foundset.loadRecords();

//5) to load records in to the form based on a query (also known as 'Form by query')
//forms.customer.foundset.loadRecords(sqlstring,parameters);
//limitations/requirements for sqlstring are:
//-must start with 'select'
//-the selected columns must be the (Servoy Form) table primary key columns (alphabetically ordered like
'select a_id, b_id,c_id ...')
//-can contain '?' which are replaced with values from the array supplied to parameters argument
//if the sqlstring contains an 'order by' clause, the records will be sorted accordingly and additional
constraints apply:
//-must contain 'from' keyword
//-the 'from' must be a comma separated list of table names
//-must at least select from the table used in Servoy Form
//-cannot contain 'group by', 'having' or 'union'
//-all columns must be fully qualified like 'orders.order_id'
```

newRecord

Number **newRecord**([location], [changeSelection])

Create a new record in the foundset.

Parameters

[location] – a boolean or number when true the new record is added as the topmost record, when a number, the new record is added at specified index ; defaults to 1.

{Boolean} [changeSelection] – when true the selection is changed to the new record; defaults to true.

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**([index])

Omit current record or the 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

[index] – index of record to omit.

Returns

Boolean – boolean true if all records could be omitted.

Also see
.loadOmittedRecords
Sample

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

relookup

void **relookup**([index])

Perform a relookup for the current record or 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
[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**([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
[clearLastResults] – boolean, clear previous search, default true
[reduceSearch] – boolean, reduce (true) or extend (false) previous search results, default true
Returns
Number – the recordCount
Also see
.find
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

{Object[]} indexes – An array with indexes to set.

Returns

void

Sample

```
var current = forms.customer.foundset.getSelectedIndexes();
var newSelection = new Array();
newSelection[0] = current[0];
forms.customer.foundset.setSelectedIndexes(newSelection);
```

sort

void **sort**(sortString/recordComparator, [defer])

Sorts the foundset based on the given sort string or record comparator function.
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.
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.

Parameters

sortString/recordComparator – the specified columns (and sort order) or record comparator function

{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');

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();
```