controller

Property Summary

Boolean	#enabled Gets or sets the enabled state of a form; also known as "grayed-out".
Boolean	#readOnly Gets or sets the read-only state of a form; also known as "editable" Note: The field(s) in a form set as read-only can be selected and the field data can be copied to clipboard.
Number	#view Get/Set the current type of view of this form.

Method Summary

Boolean	#deleteAllRecords() Deletes all records in foundset, resulting in empty foundset.
Boolean	#deleteRecord() Delete current selected record, deletes mulitple selected records incase the foundset is using multiselect.
Boolean	#duplicateRecord([location]) Duplicate current record or record at index in the form foundset.
Boolean	#find() Set the foundset in find mode.
void	#focusField(fieldName, skipReadonly) Sets focus to a field specified by its name.
void	#focusFirstField() Sets focus to the first field of the form; based on tab order sequence.
Number	#getDataProviderMaxLength(name) Returns the maximum length allowed in the specified dataprovider.
Object	#getDataProviderValue(dataProvider) Gets a value based on the specified dataprovider name.
String	#getDataSource() Get the used datasource.
Boolean	#getDesignMode() Returns the state of this form designmode.
JSDataSet	#getFormContext() Gets the forms context where it resides, returns a dataset of its structure to the main controller.
Number	#getFormWidth() Gets the form width in pixels.
Number	#getMaxRecordIndex() Returns the current cached record count of the current foundset.
String	#getName() Get the name of this form.
Number	#getPartHeight(partType) Gets the part height in pixels.
Number	#getPartYOffset(partType) Returns the Y offset of a given part of the form.
Number	#getSelectedIndex() Gets the current record index of the current foundset.
String[]	#getTabSequence() Get an array with the names of the components that are part of the tab sequence.
JSWindow	#getWindow() Returns the JSWindow that the form is shown in, or null if the form is not currently showing in a window.
Boolean	#invertRecords() Inverts the current foundset against all rows of the current table; all records that are not in the foundset will become the current foundset.
Boolean	#loadAllRecords() Loads all accessible records from the datasource into the form foundset.
Boolean	#loadOmittedRecords() Loads the records that are currently omitted in the form foundset.
Boolean	#loadRecords([data], [queryArgumentsArray]) Load records via a (related) foundset, primary key (dataset/number/uuid) or query into the form.
Boolean	#newRecord([location]) Create a new record in the form foundset.
Boolean	#omitRecord() Omit current record in form foundset, to be shown with loadOmittedRecords.
void	#print([printCurrentRecordOnly], [showPrinterSelectDialog], [printerJob]) Print this form with current foundset, without preview.
String	#printXML([printCurrentRecordOnly]) Print this form with current foundset records to xml format.

Boolean	#recreateUI() Recreates the forms UI components, to reflect the latest solution model.
void	#relookup() Performs a relookup for the current foundset record dataproviders.
Number	#search([clearLastResults], [reduceSearch]) Start the database search and use the results, returns the number of records, make sure you did "find" function first.
void	#setDataProviderValue(dataprovider, value) Sets the value based on a specified dataprovider name.
void	#setDesignMode(designMode) Sets this form in designmode with param true, false will return to normal browse/edit mode.
void	#setDesignMode(ondrag, ondrop, onselect, onresize) Sets this form in designmode with one or more callback methods.
void	#setPageFormat(width, height, leftmargin, rightmargin, topmargin, bottommargin, [orientation], [units]) Set the page format to use when printing.
void	#setPreferredPrinter(printerName) Set the preferred printer name to use when printing.
void	#setSelectedIndex(index) Sets the current record index of the current foundset.
void	#setTabSequence(arrayOfElements) Set the tab order sequence programatically, by passing the elements references in a javascript array.
void	#show([window]) Shows the form (makes the form visible), optionally showing it in the specified JSWindow.
void	#showPrintPreview([printCurrentRecordOnly], [printerJob], [zoomFactor]) Show this form in print preview.
void	#showRecords(data, [window]) Load data into the form and shows the form, is a shortcut for the functions 'loadRecords' and 'show'.
void	#sort(sortString, [defer]) Sorts the form foundset based on the given sort string.
void	#sortDialog([sortString]) Show the sort dialog to the user a preselection sortString can be passed, to sort the form foundset.

Property Details

enabled

Gets or sets the enabled state of a form; also known as "grayed-out".

Notes:
-A disabled element(s) cannot be selected by clicking the form.
-The disabled "grayed" color is dependent on the LAF set in the Servoy Smart Client Application Preferences.

Returns
Boolean

Sample

```
//gets the enabled state of the form
var state = forms.customer.controller.enabled;
//enables the form for input
forms.customer.controller.enabled = true;
```

readOnly

Gets or sets the read-only state of a form; also known as "editable"

Note: The field(s) in a form set as read-only can be selected and the field data can be copied to clipboard.

Returns
Boolean

Sample

```
//gets the read-only state of the form
var state = forms.customer.controller.readOnly;
//sets the read-only state of the form
forms.customer.controller.readOnly = true
```

view

Get/Set the current type of view of this form.

Returns
Number

Sample

```
//gets the type of view for this form
var view = forms.customer.controller.view;
//sets the form to Record view
forms.customer.controller.view = 0;//RECORD_VIEW
//sets the form to List view
forms.customer.controller.view = 1;//LIST_VIEW
```

Method Details

deleteAllRecords

Boolean **deleteAllRecords**()

Deletes all records in foundset, resulting in empty foundset.

Returns
Boolean – false incase of related foundset having records and orphans records are not allowed by the relation

Sample

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

deleteRecord

Boolean **deleteRecord**()

Delete current selected record, deletes mulitple selected records incase the foundset is using multiselect.

Returns
Boolean – false incase of related foundset having records and orphans records are not allowed by the relation

Sample

```
var success = forms.customer.controller.deleteRecord();
```

duplicateRecord

Boolean **duplicateRecord**([location])

Duplicate current record or record at index in the form foundset.

Parameters

{Boolean} [location] – true adds the new record as the topmost record, or adds at specified index

Returns

Boolean – true if succesful

Sample

```
forms.customer.controller.duplicateRecord(); //duplicate the current record, adds on top
//forms.customer.controller.duplicateRecord(false); //duplicate the current record, adds at bottom
//forms.customer.controller.duplicateRecord(1,2); //duplicate the first record as second record
```

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()
}
```

focusField

void **focusField**(fieldName, skipReadonly)

Sets focus to a field specified by its name.
If the second parameter is set to true, then readonly fields will be skipped
(the focus will be set to the first non-readonly field located after the field with the specified name; the tab sequence is respected when searching for the non-readonly field).

Parameters

{String} fieldName – the name of the field to be focussed
{Boolean} skipReadonly – indication to skip read only fields, if the named field happens to be read only

Returns

void

Sample

```
var tabseq = forms.customer.controller.getTabSequence();
if (tabseq.length > 1) {
        // If there is more than one field in the tab sequence,
        // focus the second one and skip over readonly fields.
        forms.customer.controller.focusField(tabseq[1], true);
}
else {
        // If there is at most one field in the tab sequence, then focus
        // whatever field is first, and don't bother to skip over readonly fields.
        forms.customer.controller.focusField(null, false);
}
```

focusFirstField

void **focusFirstField**()

Sets focus to the first field of the form; based on tab order sequence.

Returns

void

Also see

controller.focusField

Sample

```
forms.customer.controller.focusFirstField();
```

getDataProviderMaxLength

Number **getDataProviderMaxLength**(name)

Returns the maximum length allowed in the specified dataprovider.

Parameters

{String} name – the dataprovider name

Returns

Number – the length

Sample

```
forms.customer.controller.getDataProviderMaxLength('name');
```

getDataProviderValue

Object **getDataProviderValue**(dataProvider)

Gets a value based on the specified dataprovider name.

Parameters

{String} dataProvider – the dataprovider name to retieve the value for

Returns

Object – the dataprovider value (null if unknown dataprovider)

Sample

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

getDataSource

String **getDataSource**()

Get the used datasource.

Returns

String – the datasource

Sample

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

getDesignMode

Boolean **getDesignMode**()

Returns the state of this form designmode.

Returns

Boolean – the design mode state (true/fase)

Sample

```
var success = forms.customer.controller.getDesignMode();
```

getFormContext

JSDataSet **getFormContext**()

Gets the forms context where it resides, returns a dataset of its structure to the main controller.
Note: can't be called in onload, because no context is yet available at this time.

Returns

JSDataSet – the dataset with form context

Also see

.

Sample

```
var dataset = forms.customer.controller.getFormContext();
if (dataset.getMaxRowIndex() > 1)
{
        // form is in a tabpanel
        //dataset columns: [containername(1),formname(2),tabpanel or beanname(3),tabname(4),tabindex(5)]
        //dataset rows: mainform(1) -> parent(2)  -> current form(3) (when 3 forms deep)
        var parentFormName = dataset.getValue(1,2)
}
```

getFormWidth

Number **getFormWidth**()

Gets the form width in pixels.

Returns

Number – the width in pixels

Sample

```
var width = forms.customer.controller.getFormWidth();
```

getMaxRecordIndex

Number **getMaxRecordIndex**()

Returns the current cached record count of the current foundset.
To return the full foundset count, use: databaseManager.getFoundSetCount(...)
Tip: get the the table count of all rows in a table, use: databaseManager.getTableCount(...)

Returns
Number – the max record index

Also see
databaseManager.getFoundSetCount

Sample

```
for ( var i = 1 ; i <= forms.customer.controller.getMaxRecordIndex() ; i++ )
{
        forms.customer.controller.setSelectedIndex(i);
        //do some action per record
}
```

getName

String **getName**()

Get the name of this form.

Returns
String – the name

Sample

```
var formName = forms.customer.controller.getName();
```

getPartHeight

Number **getPartHeight**(partType)

Gets the part height in pixels.

Parameters
{Number} partType – The type of the part whose height will be returned.

Returns
Number – the part height in pixels

Sample

```
var height = forms.customer.controller.getPartHeight(JSPart.BODY);
```

getPartYOffset

Number **getPartYOffset**(partType)

Returns the Y offset of a given part of the form.

Parameters
{Number} partType – The type of the part whose Y offset will be returned.

Returns
Number – A number holding the Y offset of the specified form part.

Sample

```
var offset = forms.customer.controller.getPartYOffset(JSPart.BODY);
```

getSelectedIndex

Number **getSelectedIndex**()

Gets the current record index of the current foundset.

Returns
Number – the index

Sample

```
//gets the current record index in the current foundset
var current = forms.customer.controller.getSelectedIndex();
//sets the next record in the foundset, will be reflected in UI
forms.customer.controller.setSelectedIndex(current+1);
```

getTabSequence

String[] **getTabSequence**()

Get an array with the names of the components that are part of the tab sequence.
The order of the names respects the order of the tab sequence.
Components that are not named will not appear in the returned array, although they may be in the tab sequence.

Returns
String[] – array of names

Sample

```
var tabseq = forms.customer.controller.getTabSequence();
if (tabseq.length > 1) {
        // If there is more than one field in the tab sequence,
        // focus the second one and skip over readonly fields.
        forms.customer.controller.focusField(tabseq[1], true);
}
else {
        // If there is at most one field in the tab sequence, then focus
        // whatever field is first, and don't bother to skip over readonly fields.
        forms.customer.controller.focusField(null, false);
}
```

getWindow

JSWindow **getWindow**()

Returns the JSWindow that the form is shown in, or null if the form is not currently showing in a window.

Returns
JSWindow – the JSWindow that the form is shown in, or null if the form is not currently showing in a window.

Sample

```
var currentWindow = controller.getWindow();
if (currentWindow != null) {
                currentWindow.title = 'We have a new title';
} else {
                currentWindow = application.createWindow("Window Name", JSWindow.WINDOW, null);
                currentWindow(650, 700, 450, 350);
                currentWindow = "Window Title";
                controller.show(currentWindow);
}
```

invertRecords

Boolean **invertRecords**()

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

Returns
Boolean – true if successful

Sample

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

loadAllRecords

Boolean **loadAllRecords**()

Loads all accessible records from the datasource into the form foundset.
When the form contains a related foundset it will be replaced by a default foundset on same datasource.

Notes:
-the default foundset is always limited by filters, if databaseManager.addFoundSetFilterParam function is used.
-typical use is loading the normal foundset again after form usage in a related tabpanel

Returns
Boolean – true if successful

Also see
databaseManager.addTableFilterParam

Sample

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

loadOmittedRecords

Boolean **loadOmittedRecords**()

Loads the records that are currently omitted in the form foundset.

Returns
Boolean – true if successful
Sample

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

loadRecords

Boolean **loadRecords**([data], [queryArgumentsArray])
Load records via a (related) foundset, primary key (dataset/number/uuid) or query into the form.

Load records can be used in 5 different ways
1) to load a (related)foundset into the form.
the form will no longer share the default foundset with forms of the same datasource, use loadAllRecords to restore the default foundset
controller.loadRecords(order_to_orderdetails);

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

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

4) to reload all last related records again, if for example after a search in related tabpanel
controller.loadRecords();

5) to load records in to the form based on a query (also known as 'Form by query')
controller.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
[data] – the foundset/pkdataset/singlenNmber_pk/UUIDpk/queryString to load
[queryArgumentsArray] – the arguments to replace the questions marks in the queryString
Returns
Boolean – true if successful
Also see
.loadRecords

Sample

```
//Load records can be used in 5 different ways
//1) to load a (related)foundset into the form.
//the form will no longer share the default foundset with forms of the same datasource, use loadAllRecords to
restore the default foundset
//forms.customer.controller.loadRecords(order_to_orderdetails);

//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.controller.loadRecords(dataset);

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

//4) to reload all last related records again, if for example after a search in related tabpanel
//forms.customer.controller.loadRecords();

//5) to load records in to the form based on a query (also known as 'Form by query')
//forms.customer.controller.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

Boolean **newRecord**([location])

Create a new record in the form foundset.

Parameters

{Boolean} [location] – true adds the new record as the topmost record, or adds at specified index

Returns

Boolean – true if succesful

Sample

```
// foreign key data is only filled in for equals (=) relation items
forms.customer.controller.newRecord();//default adds on top
//forms.customer.controller.newRecord(false); //adds at bottom
//forms.customer.controller.newRecord(2); //adds as second record
```

omitRecord

Boolean **omitRecord**()

Omit current record in form foundset, to be shown with loadOmittedRecords.

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

Returns

Boolean – true if successful

Also see

controller.loadOmittedRecords

Sample

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

print

void **print**([printCurrentRecordOnly], [showPrinterSelectDialog], [printerJob])

Print this form with current foundset, without preview.

Parameters
[printCurrentRecordOnly] – to print the current record only
{Boolean} [showPrinterSelectDialog] – to show the printer select dialog (default printer is normally used)
[printerJob] – print to plugin printer job, see pdf printer plugin for example

Returns
void

Sample

```
//print this form (with foundset records)
forms.customer.controller.print();
//print only current record (no printerSelectDialog) to pdf plugin printer
//forms.customer.controller.print(true,false,plugins.pdf_output.getPDFPrinter('c:/temp/out.pdf'));
```

printXML

String **printXML**([printCurrentRecordOnly])

Print this form with current foundset records to xml format.

Parameters
[printCurrentRecordOnly] – to print the current record only

Returns
String – the XML

Sample

```
//TIP: see also plugins.file.writeXMLFile(...)
var xml = forms.customer.controller.printXML();
//print only current record
//var xml = forms.customer.controller.printXML(true);
```

recreateUI

Boolean **recreateUI**()

Recreates the forms UI components, to reflect the latest solution model.
Use this after altering the elements via solutionModel at the JSForm of this form.

Returns
Boolean – true if successful

Also see
.

Sample

```
// get the solution model JSForm
var form = solutionModel.getForm("myForm");
// get the JSField of the form
var field = form.getField("myField");
// alter the field
field.x = field.x + 10;
// recreate the runtime forms ui to reflect the changes.
forms.customer.controller.recreateUI();
```

relookup

void **relookup**()

Performs a relookup for the current foundset record dataproviders.
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.controller.relookup();
```

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
```

setDataProviderValue

void **setDataProviderValue**(dataprovider, value)

Sets the value based on a specified dataprovider name.

Parameters
{String} dataprovider – the dataprovider name to set the value for
{Object} value – the value to set in the dataprovider
Returns
void
Sample

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

setDesignMode

void **setDesignMode**(designMode)

Sets this form in designmode with param true, false will return to normal browse/edit mode.

Parameters
{Boolean} designMode – sets form in design mode if true, false ends design mode.
Returns
void
Sample

```
//Set the current form in designmode with no callbacks
forms.customer.controller.setDesignMode(true);

//Set the current form out of designmode (to normal browse)
forms.customer.controller.setDesignMode(false);
```

setDesignMode

void **setDesignMode**(ondrag, ondrop, onselect, onresize)

Sets this form in designmode with one or more callback methods.

Parameters
{Function} ondrag – onDrag method reference
{Function} ondrop – onDrop method reference
{Function} onselect – onSelect method reference
{Function} onresize – onResize method reference
Returns
void
Sample

```
//Set the current form in designmode with callbacks
forms.customer.controller.setDesignMode(onDragMethod,onDropMethod,onSelectMethod,onResizeMethod);

//Set the current form out of designmode (to normal browse mode)
forms.customer.controller.setDesignMode(false);
```

setPageFormat

void **setPageFormat**(width, height, leftmargin, rightmargin, topmargin, bottommargin, [orientation], [units])

Set the page format to use when printing.

Orientation values:
0 - Landscape mode
1 - Portrait mode

Units values:
0 - millimeters
1 - inches
2 - pixels

Note: The unit specified for width, height and all margins MUST be the same.

Parameters
width – the specified width of the page to be printed.
height – the specified height of the page to be printed.
leftmargin – the specified left margin of the page to be printed.
rightmargin – the specified right margin of the page to be printed.
topmargin – the specified top margin of the page to be printed.
bottommargin – the specified bottom margin of the page to be printed.
[orientation] – the specified orientation of the page to be printed; the default is Portrait mode
[units] – the specified units for the width and height of the page to be printed; the default is pixels

Returns
void

Sample

```
//Set page format to a custom size of 100x200 pixels with 10 pixel margins on all sides in portrait mode
forms.customer.controller.setPageFormat(100, 200, 10, 10, 10, 10);

//Set page format to a custom size of 100x200 mm in landscape mode
forms.customer.controller.setPageFormat(100, 200, 0, 0, 0, 0, 0, 0);

//Set page format to a custom size of 100x200 inch in portrait mode
forms.customer.controller.setPageFormat(100, 200, 0, 0, 0, 0, 1, 1);
```

setPreferredPrinter

void **setPreferredPrinter**(printerName)

Set the preferred printer name to use when printing.

Parameters
{String} printerName – The name of the printer to be used when printing.

Returns
void

Sample

```
forms.customer.controller.setPreferredPrinter('HP Laser 2200');
```

setSelectedIndex

void **setSelectedIndex**(index)

Sets the current record index of the current foundset.

Parameters
{Number} index – the index to select

Returns
void

Sample

```
//gets the current record index in the current foundset
var current = forms.customer.controller.getSelectedIndex();
//sets the next record in the foundset, will be reflected in UI
forms.customer.controller.setSelectedIndex(current+1);
```

setTabSequence

void **setTabSequence**(arrayOfElements)

Set the tab order sequence programatically, by passing the elements references in a javascript array.

Parameters
{Object[]} arrayOfElements – array containing the element references

Returns
void

Sample

```
forms.customer.controller.setTabSequence([forms.customer.elements.fld_order_id, forms.customer.elements.
fld_order_amount]);
```

show

void **show**([window])

Shows the form (makes the form visible), optionally showing it in the specified JSWindow.
This function does not affect the form foundset in any way.

Parameters

[window] – the window in which this form should be shown. If it is unspecified the current window will be used.

Returns

void

Also see

application.createWindow
application.getWindow

Sample

```
// show the form in the current window/dialog
forms.customer.controller.show();
// show the form in newly created named modal dialog
var w = application.createWindow("mydialog", JSWindow.MODAL_DIALOG);
forms.customer.controller.show(w);
// show the form in an existing window/dialog
var w = application.getWindow("mydialog"); // use null name for main app. window
forms.customer.controller.show(w);
//show the form in the main window
//forms.customer.controller.show(null);
```

showPrintPreview

void **showPrintPreview**([printCurrentRecordOnly], [printerJob], [zoomFactor])

Show this form in print preview.

Parameters

[printCurrentRecordOnly] – to print the current record only
[printerJob] – print to plugin printer job, see pdf printer plugin for example (incase print is used from printpreview)
[zoomFactor] – a specified number value from 10-400

Returns

void

Sample

```
//shows this form (with foundset records) in print preview
forms.customer.controller.showPrintPreview();
//to print preview current record only
//forms.customer.controller.showPrintPreview(true);
//to print preview current record only with 125% zoom factor;
//forms.customer.controller.showPrintPreview(true, null, 125);
```

showRecords

void **showRecords**(data, [window])

Load data into the form and shows the form, is a shortcut for the functions 'loadRecords' and 'show'.

Parameters

data – the foundset/pkdataset/singleNumber_pk/UUIDpk to load before showing the form.
[window] – the window in which this form should be shown.

Returns

void

Also see

application.createWindow
application.getWindow
controller.loadRecords
controller.show

Sample

```
forms.customer.controller.showRecords(foundset);
// load foundset & show the form in newly created named modal dialog
var w = application.createWindow("mydialog", JSWindow.MODAL_DIALOG);
forms.customer.controller.showRecords(foundset, w);
// load foundset & show the form in an existing window/dialog
var w = application.getWindow("mydialog"); // use null name for main app. window
forms.customer.controller.showRecords(foundset, w);
```

sort

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

Sorts the form 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
sortString – the specified columns (and sort order)
[defer] – 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.controller.sort('columnA desc,columnB asc');
```

sortDialog

void **sortDialog**([sortString])

Show the sort dialog to the user a preselection sortString can be passed, to sort the form foundset.
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
[sortString] – the specified columns (and sort order)

Returns
void

Sample

```
forms.customer.controller.sortDialog('columnA desc,columnB asc');
```