Find Mode

In This Chapter

Find Mode is a special mode that can be assumed by a foundset object to perform data searches using a powerful, high-level abstraction. When in Find Mode, the foundset's *Data Providers*, normally used to read/write data, are instead used to enter search criteria. Any data provider can be assigned a search condition which should evaluate to a String, Number or Date. Because forms typically bind to a foundset, criteria may be entered from the GUI by the user or programmatically.

A foundset enters Find Mode when its *find* method is invoked. This method returns a Boolean, because under certain circumstances, the foundset may fail to enter find mode. Therefore, it is good practice to enclose a find in an an *if* statement, so as not to accidentally modify the selected record. A foundset exits Find Mode when its *search* method is executed, upon which the foundset's SQL query is modified to reflect the expressed criteria and the matching records are loaded. The *search* method returns an integer, which is the number of records loaded by the find. However this doesn't necessarily represent the total number of matching records as foundset records are loaded in blocks.

Example:

```
// Find all customers in the city of Berlin
if(foundset.find()){   // Enter find mode
    city = 'Berlin';   // Assign a search criteria
    foundset.search(); // Execute the query and load the records
}
```

Results in the foundset's SQL query:

```
 SELECT customerid FROM customers WHERE city = ? ORDER BY customerid ASC //Query params: ['Berlin']
```

Search Criteria with Logical AND

When multiple search criteria are entered for multiple data providers, the criteria will be concatenated with a SQL *AND* operator.

Example:

```
// Find all customers in the city of Berlin AND in the postal code 12209
if(foundset.find()){     // Enter find mode    city = 'Berlin';     // Assign city search criterion
    city = 'Berlin';     // Assign a search criteria
    postalcode = '12209' // Assign postal code criterion
    foundset.search();   // Execute the query and load the records
}
```

Results in the foundset's SQL query:

```
 SELECT customerid FROM customers WHERE city = ?  AND postalcode = ? ORDER BY customerid ASC //Query params:
['Berlin','12209']
```

Multiple Find Records for Logical OR

It's important to note that when in Find Mode, a foundset will initially contain one record object. However, multiple record objects may be used to articulate search criteria. This has the effect that the criteria described in each record are concatenated by a SQL *OR*.

Example:

```
// Find customers in the city of Berlin AND in the postal code 12209...
// OR customers in the city of San Francisco AND in the postal code 94117
if(foundset.find()){        // Enter find mode    city = 'Berlin';
    city = 'Berlin';   // Assign a search criteria
    postalcode = '12209';
    foundset.newRecord();   // Create a new search record
    city = 'San Francisco'
    postalcode = '94117';
    foundset.search();      // Execute the query and load the records
}
```

Results in the foundset's SQL query:

```
 SELECT customerid FROM customers WHERE (city = ?  AND postalcode = ?) OR (city = ?  AND postalcode = ?) ORDER
BY customerid ASC //Query params: ['Berlin','12209','San Fransisco','94117']
```

Finding Records Through a Relation

Find Mode is very flexible as searches can traverse the entire data model. When a foundset enters find mode, any foundset related to a search record can be used to enter criteria. Moreover, related foundsets can use multiple search records so any permutation of Logical AND / OR is possible.

Example:

```
// Find customers that have 1 or more orders which were shipped to Argentina
if(foundset.find()){                            // Enter find mode
    customers_to_orders.shipcountry = 'Argentina'; // enter criteria in a related foundset
    foundset.search();                          // Execute the query and load the records
}
```

Results in the foundset's SQL query:

```
SELECT DISTINCT customers.customerid FROM customers
LEFT OUTER JOIN orders ON customers.customerid=orders.customerid
WHERE orders.shipcountry = ? ORDER BY customers.customerid ASC
```

And there are no limitations to the number of traversals across related foundsets.

Example:

```
// Find customers with one or more orders containing one or more products supplied by a vendor in USA
if(foundset.find()){
    customers_to_orders.orders_to_order_details.order_details_to_products.products_to_suppliers.country = 'USA';
    foundset.search();
}
```

Finding Records within a Related Foundset

It is worth pointing out that related foundsets may be put into Find Mode as well. The foundset will maintain the constraints imposed by the relation in addition to the criteria specified in the data providers.

Example: This operation is nearly identical to the previous search on ship country, however it matters which foundset is in Find Mode. The difference is that this operation searches for order records of a particular customer.

```
// Find orders of THE SELECTED CUSTOMER that were shipped to Argentina
if(customers_to_orders.find()){
    customers_to_orders.shipcountry = 'Argentina';
    customers_to_orders.search();
}
```

Results in the foundset's SQL query (notice the relation constraint is preserved):

```
SELECT orderid FROM orders  WHERE customerid = ? AND shipcountry = ? ORDER BY orderid ASC
```

Special Operators

Servoy's Find Mode provides several special operators that when used in combination can articulate the most sophisticated search requirements. Operators and operands should be concatenated as strings.

Operator	Description	Applicable Data Types	Example
||	OR: Used to implement a logical OR for two or more search conditions in the same data provider	Any	`// Cities of London or Berlin` `city = 'Berlin||London';`
|	Format: Used to separate a value and an implied format.	Date	`// exactly 01/01/2001 (00:00:00 implied)` `orderdate = '01/01/2001|MM/dd/yyyy';`
!	Not: Used to implement a logical NOT for a search condition.	Any	`// Anything but Berlin` `city = '!Berlin';`

#	Sensitivity Modifier: Implies a case-insensitive search for text columns. Implies a match on entire day for date columns.	Text, Date	`// i.e. Los Angeles, lOS aNGeLES`
			`city = '#los angeles';`
			
			`// any time on 01/01/2001`
			`orderdate = '#01/01/2001|MM/dd/yyyy';`

^	Is Null: Matches records where a column is null.	Any	`// All null contact names, not including empty strings` `contactname = '^';`
^=	Is Null/Empty/Zero: Matches records where a column is null, empty string value or zero numeric value	Text, Numeric	`// All freights which are null or 0` `freight = '^=';`
<	Less than: Matches records where the column is less than the operand	Any	`// i.e. 50, 99.99, but not 100, 101` `freight = '<100';`
<=	Less than or equal to: Matches records where the column is less than or equals the operand	Any	`// i.e. Atlanta, Baghdad, Berlin, but not Buenos Aires, Cairo` `city = '<=Berlin';`
>=	Greater than or equal to: Matches records where the column is greater than or equals the operand	Any	`// Any time on/after 12am new year's day 2001` `orderdate = '>=01/01/2001|MM/dd /yyyy';`
>	Greater than: Matches records where the column is greater than the operand	Any	`// i.e. 100.01, 200, but not 99,100` `freight = '>100';`

...	Between: Matches records where the column is between (inclusive) the left and right operands.	Any	`// Any time during the year 2001`
			`orderdate = '01/01/2001...01/01/2002|MM/dd/yyyy';`
			
			`// i.e.`
			`freight = '100...200';`
			
			`// i.e. London, Lyon, Madrid, Omaha, Portland`
			`city = 'London...Portland';`

%	Wild Card String: Matches records based on matching characters and wild cards	Text	`city = 'New%';  // Starts with: i.e. New York, New Orleans`
			`city = '%Villa%; // Contains: i.e. Villa Nova, La Villa Linda`
			`city = '%s';  // Ends with: i.e. Athens, Los Angeles`

_	Wild Card Character: Matches records based on	Text	`// i.e. Toledo, Torino` `city = '%To___o%';`
\	Escape Character: Used to escape other string operators	Text	`// Escape the wild card, i.e. ...50% of Capacity...` `notes = '%\%%';`
now	Now: Matches records where the condition is right now, including time	Date	`// exact match on this second` `creationdate = 'now';`
today	Today: Matches records where the condition is any time today	Date	`// match on anytime today` `orderdate = 'today';`

Using find mode from scripting without using special operators or spaces

You can use find mode with non-strings as well. For example, dates, numbers are not interpreted and will be used literally.

Arrays can be used when searching for multiple values, these are also not interpreted.

```
if(foundset.find()) {
    city = ['Berlin', 'Amsterdam'] // city in (?, ?) {'Berlin', 'Amsterdam'}
    companyid = 42; // literal numerical value
    startdate = new Date(99,5,24,11,33,30,0); // literal date value
    foundset.search();   // Execute the query and load the records
}
```

Note that when you use a string for searching, it will be trimmed (except in case of a CHAR column, which is padded with spaces by the database).

If you want to make sure the argument is not interpreted, us a single-element array:

```
if (foundset.find()) {
    // tag = ' Hello Servoy ';     // would search for trimmed
    tag = [' Hello Servoy '];     // will search for literal (untrimmed)
    foundset.search();   // select ... from ... where tag = ? {' Hello Servoy '}
}
```

Find Mode and the User Interface

The above examples deal with find mode in which find mode is entered, criteria are expressed and the search is run, all in a single action. The effect of the search is entirely up to the developer. However, find mode can also be entered in one action and searched in another action. In between, the user may manually enter values into fields to express the search criteria. They can then run the search action and a form's foundset will show the results of the search. Any of the above search criteria may be used.

Example In this example there is a method which can both enter find mode as well as run a search when in find mode. In between the two different invocations of this method, the user interface is ready to receive input from the user. When complete, the user may run the method again, this time the foundset will search for results.

```
/**
 * @AllowToRunInFind
 *
 * @properties={typeid:24,uuid:"088B830C-2A4F-483C-A135-5FA32A010AE9"}
 */
function doFind(){
	if(foundset.isInFind()){	// if the foundset is already in find mode, run the search
		foundset.search();
	} else {
		foundset.find();	// otherwise, enter find mode
	}
}
```


Find mode blocks the execution of any methods which are normally invoked from the user interface. This is a good thing as these methods may have unintended consequences when a form's foundset is in find mode. Notice the JSDocs tag **@AllowToRunInFind** in the comment block which precedes the method. This tag provides the metadata to let Servoy know that this method should be allowed to run while the form's foundset is in find mode. Without this exception, this method would be blocked from execution, and there would be no recourse to programmatically exit find mode.

Read-Only Fields

By default, even read-only fields will become editable for the duration of the find mode. This is often useful, because while a data provider may not be available to edit, in find mode, it becomes a vehicle to enter a search criterion and should be editable to the user. However, in some cases it may be desired that read-only fields remain so for the duration of find mode as well. Servoy provides a UI property which may be set through the Application API using the method setUIProperty.

Example This example is identical to the above example with the exception, that for the duration of this find, the read-only property of fields is maintained. After a find, it is set back to the default so as not to interfere with other functionality throughout the rest of the application.

```
/**
 * @AllowToRunInFind
 *
 * @properties={typeid:24,uuid:"088B830C-2A4F-483C-A135-5FA32A010AE9"}
 */
function doFind(){
	if(foundset.isInFind()){
		foundset.search();
		application.setUIProperty(APP_UI_PROPERTY.LEAVE_FIELDS_READONLY_IN_FIND_MODE, false)	// reset
to the default
	} else {
		application.setUIProperty(APP_UI_PROPERTY.LEAVE_FIELDS_READONLY_IN_FIND_MODE, true);	//
before entering find mode, enforce read-only fields
		foundset.find();
	}
}
```

Canceling Find Mode

Find mode can be programmatically cancelled by invoking the loadAllRecords method of the foundset. The foundset will revert to the query prior to entering find mode. Within the Smart Client the user can cancel Find mode by pressing Escape. This will trigger the loadAllRecords command of the Form to which the foundset is bound.

Complex Searches

Servoy's find mode can be used to easily satisfy even complex search requirements. Remember that any related foundset may be used to enter criteria and that any number of search records may be used in any foundset and any operators may be used in combination for every data provider.