Column Conversion

In This Chapter

Some scenarios require that a value is stored in a database column in one form and written to and read from the database column in another form. Servoy supports this requirement with a feature called Column Conversion and it has three applications: String Serialization, Blob Serialization and Global Method Conversion.

Servoy also allows the contribution of a column converter by a java plugin.

String Serialization

Servoy supports object persistence using String Serialization, which involves the conversion of a runtime object into a string format, which can then be persisted in a database column. When the column is read from the database, the persistent string will be deserialized back into a runtime object. Because Servoy uses JavaScript as its scripting language, runtime objects will be serialized into standard JSON format.

String Serialization can only be used for column type TEXT.

```
//  Construct an object to capture some custom settings and write it directly to a Text column called
'custom_settings'
var obj = new Object();
obj.name = 'foobar';
obj.message = 'Hello World';

// at this point it is serialized into the string: "{name:'foobar',message:'Hello World'}"
custom_settings = obj;
databaseManager.saveData();

// ...read object properties at a later time...
application.output(custom_settings.message + 'My name is: ' + custom_settings.name);
```


Remember that only by assigning an object to a data provider will the serialized string be actually stored. It is not possible to set individual instance properties of an object to directly modify the serialized string.

```
// For Example
my_data_provider.property = 'Foobar'; // This will have no effect on the data provider

// Instead
var obj = my_data_provider; // read the data provider into a runtime object
obj.property = 'Foo Bar';   // Modify the Object's instance properties
my_data_provider = obj;     // And reassign it to the data providerdatabaseManager.saveData();
```

Blob Serialization

Servoy provides Blob Serialization for persisting an object as a Blob. This involves converting the runtime object into a Blob, which is then persisted in the database column. When retrieving the column data from the database, the Blob is deserialized back into a runtime object.

See code example at String Serialization.

⚠ Blob Serialization can only be used for column type MEDIA.

Global Method Conversion

Servoy allows a database column to be bound to custom business logic, giving developers control over how a value is converted when it is written to, and read from the data provider.

⚠ The nomenclature refers to the **Object Value**, seen in the GUI, as well as used programmatically, and the **Database Value**, the value stored in the data provider and persisting in the database.

The column is bound to two methods which facilitate the conversion between the Object Value and the Database Value. A developer may also specify an optional **Object Data Type**, prompting Servoy to provide the data in an alternate data type in lieu of the default column type. This is useful when values are stored in a non-standard storage type to accommodate legacy systems, but should be treated like standard data type in the runtime.

Object to Database Method

This method is called anytime a value is written to the data provider. It will be called regardless of the origin of the action, i.e. GUI event or programmatically. It will be called before data is committed to the database.

Parameters

Object - The value that is being written to the data provider

String - The column's data type: TEXT, INTEGER, NUMBER, DATETIME, MEDIA

Returns

Object - The converted value that will actually be written to the data provider.

Example

Perhaps the most classic use case is the conversion between SI Units, where a database is standardized on a certain unit, but an application requires that values be written and read in multiple units, often to support different locales / preferences. Imagine a database column for temperature, which is standardized on Celsius, but an application which allows data entry in Celsius, Fahrenheit and Kelvin.

```
/**
 * This method auto-converts from client units to Celsius as the value is being written to the data provider
 * @parameter {Object} value The value of the runtime object
 * @parameter {String} columnType The data type of the column
 * @returns {Object} The value converted into celsius
 * @properties={typeid:24,uuid:"303ACB93-3B0E-4B9C-9550-D78FF17343C2"}
 */
function objectToDB(value, columnType) {

	// evaluate client unit settings
	switch(tempUnits){

		// Already in C, just return it as is
		case C :
			return value;

		// Fahrenheit,use conversion formula
		case F :
			return (5/9)*(value-32);

		// Kelvin,use conversion formula
		case K :
			return value - 273;
	}
}
```

Database to Object Method

This method is called anytime a value is read from the data provider. It will be called when it is displayed in the GUI or read programmatically.

Parameters

Object - The value that is being read from the data provider

String - The column's data type: TEXT, INTEGER, NUMBER, DATETIME, MEDIA

Returns

Object - The converted value that will actually be displayed in the GUI and read programmatically.

Example

Perhaps the most classic use case is the conversion between SI Units, where a database is standardized on a certain unit, but an application requires that values be written and read in multiple units, often to support different locales / preferences. Imagine a database column for temperature, which is standardized on Celsius, but an application which allows data entry in Celsius, Fahrenheit and Kelvin.

```
/**
 * This method converts database values (Celsius) into the current client units for degrees
 * @parameter {Object} value The value stored in the column
 * @parameter {String} columnType The data type of the column
 * @returns {Object} The value that was converted into current client units
 * @properties={typeid:24,uuid:"63C4D552-531C-48DB-A6C6-ED02F4603C20"}
 */
function dbToObject(value, columnType) {

	//	evaluate client unit settings
	switch(tempUnits){

		// Already using C, just return it as is
		case C :
			return value;

		// Fahrenheit, use conversion formula
		case F :
			return (9/5) * value + 32;

		// Kelvin, use conversion formula
		case K :
			return value + 273;
	}
}
}
```

Converted Object Type

One can optionally specify the data type of the Object Value. This is useful in situations where the stored value is a different data type than the object value.

Example

The application talks to a database that is storing dates as 8-character text columns to support legacy applications. By setting the **Converted Object Type** setting to DATETIME, Servoy will treat the column as a date object. Moreover, the two conversion methods written by the developer should assume the Object Value is a Date object.

```
/**
 * This method converts Text data stored in the database column, presenting it as Date object
 * @parameter {Object} value The value stored in the column
 * @parameter {String} columnType The data type of the column
 * @returns {Object} The value that was converted
 * @properties={typeid:24,uuid:"16BDC049-E63B-47C4-B49C-595D916FD51B"}
 */
function dbToObj(value, columnType) {
	return utils.dateFormat(value,'MMddyyyy');
}
```

Column Converter from Java Code (plugin)

A Column converter can be contributed by a java plugin. See Providing Converters and Validators from Plugins for more information.