Providing Converters and Validators from Plugins

Column converters as well as validators can be provided via client plugins. See Creating Client Plugins for more information on how to create client plugins.

Column Converters

In order to build a column converter, the plugin class implementing the IClientPlugin interface (see Implement the Plugin Interface) also needs to implement the interface IColumnConverterProvider to produce ITypedColumnConverter instances in the `getColumnConverters()` method. See API docs.

> ⚠ Note that IColumnConverter is considered to be deprecated, therefore in getColumnConverters() method do return an ITypedColumnConverter array.

A column converter class implementing the ITypedColumnConverter interface has to implement the following three methods that will be displayed in the **Con version** tab:

- convertFromObject()
  Converts from dataprovider value to db value
- convertToObject()
  Converts from db value to dataprovider value
- getToObjectType()
  The dataprovider data type (so the resulting type of the convertToObject()), being one of the IColumnTypes constants: TEXT, INTEGER, NUMBER, DATETIME or MEDIA

A column converter class can define properties in the `getDefaultProperties()` method, which will be displayed in the **Conversion** tab as well.

The column types that the column converter supports are to be specified in the `getSupportedColumnTypes()` method.

For an example of a column converter, see the NrToJodaConverter class in the example given on Providing UI Converters from Plugins page.

All the implemented custom column converters must be returned by the `getColumnConverters()` method of the plugin class implementing the IColumnConverterProvider interface, so that they become available within Servoy Developer.

Column Validators

In order to build a column validator, the plugin class implementing the IClientPlugin interface (see Implement the Plugin Interface) also needs to implement the interface IColumnValidatorProvider to produce IColumnValidator instances in the getColumnValidators() method. See API docs.

The validation rule of an IColumnValidator is defined in the `validate()` method.

An IColumnValidator can also define properties in the `getDefaultProperties()` method, whose values will be set by the developer in UI, in the **Validatio** tab.

The column types that the validator supports are specified in the `getSupportedColumnTypes()` method.

All the implemented custom column validators must be returned by the `getColumnValidators()` method of the plugin class implementing the IColumnValidatorProvider interface, so that they become available within Servoy Developer.

With 2020.06 there is a IColumnValidator2 which extends IColumnValidator which adds public void validate(Map<String, String> props, Object value, String dataprovider, IValidationObject validationObject, Object state); method that works with the new validation framework: Data/Record/Column validation

As an example how the new IColumnValidator2 interface works look here (see also below the example)

Example

This is an example of validating the size of a column of type TEXT or MEDIA. The validation rule is that the size must be lower or equal to the value given by the developer by setting the `length` property in the Validation tab.

```
import java.util.Map;

import com.servoy.j2db.dataprocessing.IColumnValidator2;
import com.servoy.j2db.dataprocessing.IValidationObject;
import com.servoy.j2db.persistence.IColumnTypes;
import com.servoy.j2db.util.ILogLevel;
import com.servoy.j2db.util.Utils;

@SuppressWarnings("nls")
public class NumberRangeValidator implements IColumnValidator2
{
	private static final String FROM_PROPERTY = "from";
	private static final String TO_PROPERTY = "to";

	public Map<String, String> getDefaultProperties()
	{
		Map<String, String> props = new java.util.LinkedHashMap<>();
		props.put(FROM_PROPERTY, "");
		props.put(TO_PROPERTY, "");
		return props;
	}

	public String getName()
	{
		return "servoy.NumberRangeValidator";
	}

	public int[] getSupportedColumnTypes()
	{
		return new int[] { IColumnTypes.INTEGER, IColumnTypes.NUMBER };
	}

	/**
     * IColumnValidator2 interface with the new IValidationObject, if given then the error is reported on that
instead of throwing an exception.
     */
	@Override
	public void validate(Map<String, String> props, Object value, String dataprovider, IValidationObject
validationObject, Object state)
	{
		if (value == null || value.toString().trim().length() == 0) return;

		double from = Utils.getAsDouble(props.get(FROM_PROPERTY));
		double to = Utils.getAsDouble(props.get(TO_PROPERTY));
		double val = Utils.getAsDouble(value);
		if (val < from || val > to)
		{
			if (validationObject != null)
				validationObject.report("i18n:servoy.validator.range", dataprovider, ILogLevel.
ERROR, state,
					new Object[] { value, dataprovider, Double.valueOf(from), Double.valueOf
(to) });
			else throw new IllegalArgumentException();
		}
	}

    // the old validate method of IColumnValidator interface, just call the IColumnValidator2 valdiate method
with null.
	public void validate(Map<String, String> props, Object arg) throws IllegalArgumentException
	{
		validate(props, arg, null, null, null);
	}
}
```