



# Custom Types in Java Snippets

Jonathan Hale  
Intern, KNIME

# Custom DataTypes

---

- KNIME AP has an **Extension Point** to register a custom **Data Type**
- Requires implementing `DataCellFactory`

# Custom Types in Java Snippets

---



Currently cannot be used in Java Snippets

# Custom Type Support in Java Snippets – Extension Point

---

One interface to implement for the Extension Point itself:

```
public interface ConverterProvider {  
  
    /**  
     * @return a Collection of factories to create “KNIME DataValue -> Java type”  
     *         converters  
     */  
    Collection<ConverterFactory<?, ?>> getConverterFactory();  
  
    /**  
     * @return a Collection of factories to create “Java type -> KNIME DataType”  
     *         converters  
     */  
    Collection<ToDataValueConverterFactory<?>> getToDataValueConverterFactory();  
  
}
```

# Custom Type Support in Java Snippets – Extension Point

---

Sample implementation code snippet:

```
@Override
public Collection<ToDataValueConverterFactory> getToDataValueConverterFactories() {
    return Arrays.asList(
        new SimpleToDataValueConverterFactory(
            Integer.class, // input type
            IntCell.TYPE, // output type
            (source) -> new IntCell(source) // conversion function
        )
    );
}
```

# Custom Type Support in Java Snippets – Annotations

---

In an existing DataCellFactory implementation:

```
public MyDataCell createFromFoo(Foo source) {  
    return new MyDataCell(source);  
}
```

# Custom Type Support in Java Snippets – Annotations

---

In an existing DataCellFactory implementation:

```
@FactoryMethod // <-  
public MyDataCell createFromFoo(Foo source) {  
    return new MyDataCell(source);  
}
```

# Support for Existing DataTypes

The screenshot displays the KNIME software interface. On the left, there are two panels: 'Column List' containing 'ROWID', 'ROWINDEX', 'ROWCOUNT', and 'ArraySizes'; and 'Flow Variable List' containing 'knime.workspace'. The central area is a code editor with the following Java code:

```
1 // system imports
12 // Your custom imports:
13
14 import java.io.InputStream;
15 import java.io.ByteArrayInputStream;
16
17 // system variables
18 // Your custom variables:
19
20 // expression start
21 // Enter your code here:
22 assert(c_ArraySizes <= Byte.MAX_VALUE);
23
24 byte[] byteArray = new byte[c_ArraySizes];
25 for (int i = 0; i < c_ArraySizes; ++c_ArraySizes) {
26     byteArray[i] = (byte)i;
27 }
28
29 out_Blob = new ByteArrayInputStream(byteArray);
30
31 // expression end
32
```

Below the code editor are two configuration panels:

**Input**

Column / Flow variable	Java Type	Java Field
ArraySizes	Integer	c_ArraySizes

**Output**

Field Type	Replace	Column / Flow Variable	Output Type	Array	Java Type	Java Field
Column	<input type="checkbox"/>	Blob	Binary object	<input type="checkbox"/>	InputStream	out_Blob

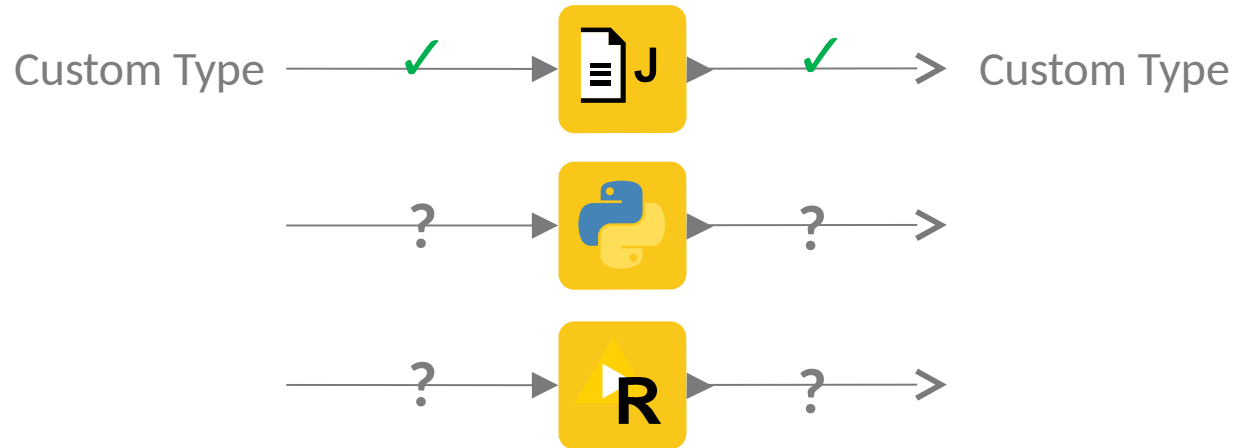
A yellow arrow points to the 'Binary object' output type in the Output table.



# Extendibility

---

- A same/similar API for Python and R could possibly happen
- Current focus lies on **Java** Snippets



The KNIME® trademark and logo and OPEN FOR INNOVATION® trademark are used by KNIME.com AG under license from KNIME GmbH, and are registered in the United States.