Converter Implementation

View level converters must extend the abstract class **com.hof.mi.interfaces.Converter**. Configuration UI and persistence is available through the normal UserInputParameters API. You can see the documentation on this API here.

The following methods need to be implemented in order to create a view level converter:

- public String getName()
- public boolean acceptsNativeType(int type)
- public abstract int getReturnType()
- public Object convertObject(Object data) throws Exception
- public Object convertObjectReverse(Object data) throws Exception

public String getName()

This defines the name that will be shown to the user when applying this converter.

top

---

public boolean acceptsNativeType(int type)

This method returns whether or not this converter will accept a field of the type specified by the “type” argument. This argument will be one of UserInputParameters.TYPE_* (ie. TYPE_NUMERIC, TYPE_TEXT, etc.).

Below is an example of a TextToNumericConverter implementing this method:

```
@Override
public boolean acceptsNativeType(int type) {
    if (type == UserInputParameters.TYPE_TEXT) {
        return true;
    } else {
        return false;
    }
}
```

top

---

public abstract int getReturnType()

This defines the converted data type that this converter will generate values of. This can be one of UserInputParameters.TYPE_* (ie. TYPE_NUMERIC, TYPE_TEXT, etc.).

Here's a simple TextToNumericConverter example using this method:

```
@Override
public int getReturnType() {
    return UserInputParameters.TYPE_NUMERIC;
}
```

top

---

public Object convertObject(Object data) throws Exception

This is the method that will receive and convert values. The “data” argument will be an input object of a data type that represents the backend implementation for the UserInputParameters type of the input field. It is generally good practice to check the object type before blindly casting; usually data will be in the obvious data type, such as TYPE_TEXT being a String. However, there are some cases to look out for:

- TYPE_NUMERIC
    - BigDecimal
    - BigInteger
    - Other basic Java numeric data types which extend Number
- TYPE_DATE
    - java.util.Date
    - java.sql.Date
- TYPE_TIMESTAMP
    - java.sql.Timestamp

This method may throw an exception if the data type conversion fails in a way which would indicate that the converter was set up incorrectly, but if the conversion fails for generic reasons such as the data type could not be rectified, or the value was null, then it is accepted to return null.

The TextToNumericConverter example using this method:

```
@Override
public Object convertObject(Object data) {
   if (data == null) return null;
   try {
      return new BigDecimal((String)data);
   }  catch (NumberFormatException e) {
      return null;
   }
}
```

top

---

public Object convertObjectReverse(Object data) throws Exception

This method is used in order for Yellowfin to retrieve the initial values of a converted data set after-the-fact. It should perform the opposite conversion which was applied in convertObject and will receive an object of the type specified as the return type of this Converter.

```
@Override
public Object convertObjectReverse(Object data) {
   if (data == null) return null;
   return data.toString();
}
```

top

Previous topic: Converter overview
Next topic: Data transformation extras