JSONConverter
Converts between JSON encoded text data and XML. The converter supports both directions.

Handled payload-types:
json

On payload-type conversion between JSON and XML (and vice versa), it always is possible to choose one side (either the inbound or the outbound) to have the data in an arbitrary shape (means, in the natural format the data comes in or shall be produced). The other side (be it the outbound or inbound one) then must deal with the data in an artificial internal format as described by the payload-type converter.
The choice about which of these both sides (inbound or outbound) shall cope with the natural data format sometimes is governed by the concrete situation and therefore implies the obvious option:

- If (JSON-) data comes in from an external party, it needs to be processed "as is". Therefore, the natural format in terms of data representation would be required to cope with on the inbound side so that any kind of (JSON-) data as delivered by the external party can be processed. In this case, the format of the inbound data is determined by the external party.
- On the other hand side, if (JSON-) data shall be emitted to an external party in the format the party requires, the natural format in terms of data representation would be required to be produced on the outbound side so that any kind of (JSON-) data can be delivered to the external party. In this case, the format of the outbound data is determined by the external party.

The options mentioned above most likely will be the most common cases (when the JSON format "rules"), but there also exist cases where the XML format must be treated in its natural representation of the data of question on inbound or outbound: This just reversed situation most commonly is given when processing Javascript transformations on a message-level that in fact are embedded in an XML-environment.
Thus, as a summary, the converter either allows freestyle / natural format inbound data and in turn produces artificial output or vice versa:

- If the converter is configured to accept the inbound-data "as is" in any shape (be it JSON or XML), the benefit is that there is no need for a pre-processing in order to bring the data to the artificial shape otherwise expected by the converter if it operates in the mode to produce any arbitrary outbound data. That is why this kind of operation mode is called "ad hoc". The drawback of this operation mode is that there sometimes has to happen a pos-processing in order to decode / retrieve this artificial format. Pretty often, though, there is no need for an explicit pos-processing step in order to fulfill this purpose as that retrieval implicitly can be done in the semantical transformation of the data that would happen anyway.
- The other operation mode is that the converter is configured to expect the inbound-data in its internal pre-defined artificial shape (be it JSON or XML). As a drawback for this case, there most likely is a pre-processing necessary in order to bring the data to this artificial shape. The benefit in this case is that there would be no need for a pos-processing on outbound after the conversion of the data, as the data would be produced in the exact format the consuming (external) party would need to get it.

Thus, the operation mode (no matter which conversion direction) would be controlled / adjusted by a flag that either activates or de-activates the ad-hoc conversion mode (ad-hoc depicts the trait that any inbound data w/o previous preparation can be input).

Recently, there only did exist one kind of bi-directional conversion that always had an artificial internal XML format and in turn a free-style, arbitrary JSON format (no matter which direction was processed).
Hence, using this conversion from JSON to XML would represent the ad-hoc operation mode (as the inbound data can be any JSON).
On the other hand side, using this conversion from XML to JSON would not be an ad-hoc operation mode, as in this case, the inbound XML had to conform to the internal artificial structure.
The current converter now enhances this capabilities with another bi-directional conversion mode that in turn has an artificial internal JSON format and in turn a free-style, arbitrary XML format (no matter which direction gets processed).
Hence, using this conversion from XML to JSON would represent the ad-hoc operation mode (as the inbound data can be any XML).
On the other hand side, using this conversion from JSON to XML would not be an ad-hoc operation mode, as in this case, the inbound JSON had to conform to the internal artificial structure.
Having these new capabilities and still staying backward-compatible, the default is to use the ad-hoc mode on a conversion from JSON to XML. On XML to JSON, the ad-hoc mode by default is deactivated.

The artificial internal XML format for the converter is described in the BizStore XML schema-document
bizstore:/com.sap.b1i.system/xsd/json_pltype.xsd

The artificial internal JSON format for the converter is as follows:

- XML elements:
```
{
  "type" : "element",
  "name" : "ns-prefix:element-name",
  "value" :
  [
  ]
}
```

- XML attributes (and namespace declarations):
```
{
  "type" : "attribute",
  "name" : "the-name-of-the-attribute",
  "value" : "the value of the attribute"
}
```

- XML text:
```
{
  "type" : "text",
  "name" : "",
  "value" : "the text itself"
}
```

- XML processing-instructions (and XML declarations):
```
{
  "type" : "processing-instruction",
  "name" : "the-name-of-the-PI",
  "value" : "the value of the PI"
}
```

- XML comments:
```
{
  "type" : "comment",
  "name" : "",
  "value" : "the comment itself"
}
```

The overall represented XML-document always must be started by a JSON-array, as beside the single root-element, there also could occur processing-instructions (the XML declaration would not be specified as it will be handled internally automatically).

Note that XML namespaces, namespace-declarations and namespace-prefixes of XML elements and attributes don't have any specific treatment / interpretation on JSON side. Thus, having the XML markup on JSON side, the XML namespace semantics must be interpreted properly by the party that processes the produced JSON. As namespaces are an enhanced capability of XML compared to JSON, it would be recommended not to use them on such a conversion if possible / feasible in order to ease the processing on JSON side.
On the other hand side, if producing XML markup from artificial JSON that shall include namespace handling, it must be made sure to input proper JSON data structures in order to produce both syntactical and semantical correct XML markup. Again for this case, it is recommended not to cope with namespace information if possible / feasible in order to ease the processing on JSON side.

Also note that - due to the lack of reliable XML data-type handling w/o a strong XML-schema in backhand - there happens no coping of distinct data-types for the artificial JSON format. Thus, technically spoken, all data is represented as JSON-strings. It is the responsibility of the overall processing to pass content in a feasible format back and forth.

Find below a tabular summary that describes the traits of the offered conversion options:

Input Data Shape	Direction	Output Data Shape	Deflt.	Ad Hoc	Pre-Processing recommended	Pos-Processing recommended	Typical Usage
Natural	JSON->XML	Artific	X	X		X	in BizFlows input of arbitrary JSON
Natural	XML->JSON	Artific		X		X	in scripting input of arbitrary XML
Artific	JSON->XML	Natural			X		in scripting output of arbitrary XM
Artific	XML->JSON	Natural	X		X		in BizFlows output of arbitrary JSO

As the default conversion options in the table tell, the treatment of natural JSON was the original offering available for a long time. These conversion offerings in turn had been enhanced for natural XML treatment. Note that for the most concrete application use cases, there will come up an obvious choice about which of the conversion options to use!

Recognized BizFlow-Properties:
XXX -> XML:

bpm.adhoc -
If not existing or set to 'true', the ad-hoc conversion mode is activated (this is the default). If set to 'false', the ad-hoc mode is de-activated.

bpm.encoding -
The overall encoding of the input data. Defaults to 'UTF-8'

XML -> XXX:

bpm.adhoc -
If set to 'true', the ad-hoc conversion mode is activated. If not existing or set to 'false', the ad-hoc mode is de-activated (this is the default).

bpm.encoding -
The overall encoding of the output data. Defaults to 'UTF-8'

bpm.bom -
If existent and set to 'true', a suitable byte-order-mark will be prepended to the output document if applicable: This comes true for the encodings 'UTF-8', 'UTF-16BE' and 'UTF-16LE'.