

JSON

Go Up to [Using the RTL in Multi-Device Applications](#)

JSON (<http://www.json.org/>) (JavaScript Object Notation) is a language independent lightweight data-interchange format. JSON can be used as an alternative to other data-interchange formats such as XML or YAML.

RAD Studio provides [JSON frameworks](#) that contain classes and methods to store, parse, read, write, and generate data in JSON format.

JSON Frameworks

RAD Studio provides different frameworks to handle JSON data.

- [JSON Objects Framework](#): This framework creates temporary objects to read and write JSON data.
- [Readers and Writers JSON Framework](#): This framework allows you to read and write JSON data directly.

JSON Objects Framework

The JSON objects framework requires you to create a temporary object to parse or generate JSON data. To read or write JSON data, you have to create an intermediate memory object such as [TJSONObject](#), [TJSONArray](#), or [TJSONString](#) before reading and writing the JSON.

For further details about this framework, see [JSON Objects Framework](#).

Readers and Writers JSON Framework

The readers and writers JSON framework allows you to read and write JSON data directly to a stream, without creating a temporary object. Since you do not have to create temporary object to read and write a JSON, this framework has a better performance, and improves memory consumption.

Unlike the JSON objects framework; the readers and writers JSON framework provides [BSON](#) support.

For further details about this framework, see [Readers and Writers JSON Framework](#).

JSON Framework Differences

You can use any of the two frameworks when working with JSON data. You may choose one or the other according to the needs of the project. The below table highlights some key differences among both JSON frameworks.

JSON Objects Framework	<ul style="list-style-type: none">▪ Easier way to read a JSON structure, allows iterations and modifications in the model tree.
Readers and Writers JSON Framework	<ul style="list-style-type: none">▪ Reads and writes JSON in a stream in a sequential manner, which reduces memory consumption.▪ BSON support.▪ Extensible.

The below table has three code snippets that illustrate the differences between the frameworks when writing a JSON.

Writing a JSON With Different Frameworks		
JSON Objects Framework	Readers and Writers JSON Framework: TJsonWriter	Readers and Writers JSON Framework: TJSONObjectBuilder
<pre>JSONColor := TJSONObject.Create; JSONColor.AddPair('name', 'red'); JSONColor.AddPair('hex', '#f00'); JSONArray := TJSONArray.Create; JSONArray.Add(JSONColor); JSONObject := TJSONObject.Create; JSONObject.AddPair('colors', JSONArray);</pre>	<pre>Writer.WriteStartObject; Writer.WritePropertyName('colors'); Writer.WriteStartArray; Writer.WriteStartObject; Writer.WritePropertyName('name'); Writer.WriteValue('red'); Writer.WritePropertyName('hex'); Writer.WriteValue('#f00'); Writer.WriteEndObject; Writer.WriteEndArray; Writer.WriteEndObject;</pre>	<pre>Builder := TJSONObjectBuilder.Create(Writer); Builder .BeginObject .BeginArray('colors') .BeginObject .Add('name', 'red') .Add('hex', '#f00') .EndObject .EndArray .EndObject;</pre>

The three codes snippets of the above table write the following JSON:

```
{
  "colors": [
```

Contents

- JSON Frameworks**
 - JSON Objects Framework
 - Readers and Writers JSON Framework
- JSON Framework Differences**
- JSON Processing and Parsing Improvements in 10.3**
- JSON Topics**
- See Also**
 - Code Samples

```
    {
      "name": "red",
      "hex": "#f00"
    }
  ]
}
```

The below table has three code snippets that illustrate the differences between the frameworks when reading a JSON.

Reading a JSON With Different Frameworks		
JSON Objects Framework	Readers and Writers JSON Framework: <u>TJsonReader</u>	Readers and Writers JSON Framework: <u>TJSONIterator</u>
<pre>JSONValue := TJSONObject.ParseJSONValue('{ "colors": [{"name": "red", "hex": "#f00"}]}'); Memo1.Lines.Add('READER:'); if JSONValue is TJSONArray then //... else if JSONValue is TJSONObject then Memo1.Lines.Add('colors'); Memo1.Lines.Add('name: ' + JSONValue.GetValue<string>('colors[0].name')); Memo1.Lines.Add('hex: ' + JSONValue.GetValue<string>('colors[0].hex'));</pre>	<pre>LStringReader := TStringReader.Create('{ "colors": [{"name": "red", "hex": "#f00"}]}'); LJsonTextReader := TJsonTextReader.Create(LStringReader); while LJsonTextReader.read do case LJsonTextReader.TokenType of TJsonToken.PropertyName: Memo1.Lines.Add(LJsonTextReader.Value.AsString); TJsonToken.String: Memo1.Lines[Memo1.Lines.Count-1] := Memo1.Lines[Memo1.Lines.Count-1] + ': ' + LJsonTextReader.Value.AsString; end;</pre>	<pre>LStringReader := TStringReader.Create('{ "colors": [{"name": "red", "hex": "#f00"}]}') LJsonTextReader := TJsonTextReader.Create(LStringReader) LIterator := TJSONIterator.Create(LJsonTextReader) LIterator.Recurse; LIterator.Next; Memo1.Lines.Add(LIterator.Key); LIterator.Recurse; LIterator.Recurse; LIterator.Next; LIterator.Recurse; LIterator.Next; Memo1.Lines.Add(LIterator.Key + LIterator.AsString); LIterator.Next; Memo1.Lines.Add(LIterator.Key + LIterator.AsString);</pre>

The three codes snippets of the above table add the following to a TMemo.

```
colors
name: red
hex: #f00
```

JSON Processing and Parsing Improvements in 10.3

- Improved the correctness of JSON code, in terms of the JSON code generated by the TJSONValue class and derived ones, but also in terms of parsing. We have also worked on performance improvements.
- New TAsciiStreamWriter class: this class can be combined with a TJsonTextWriter to give the best JSON string generation performance (fewer conversions).
- Additional "pretty print" JSON output with the introduction of the new TJSONAncestor.Format(Indentation: Integer = 4). As a consequence, TJSON.Format has been deprecated.
- TJSONAncestor.ToJSON now always produces a formally valid JSON string, while TJSONAncestor.ToString produce a similar JSON string, but without converting non-ASCII symbols to \uNNNN.
- JSON parsing support has a new behavior in case of errors in the JSON source text. There is a new option, TJSONObject.TJSONParseOption.RaiseExc, which determines whether the ParseJSONValue method raises an exception of type EJSONParseException in case of invalid JSON, or it returns nil (as in the past). If the exception is enabled, TJSONObject.ParseJSONValue now returns an error position, raising the new System.JSON.EJSONParseException (which has the properties Path, Offset, Line, and Position). Additionally, the method TJSONObject.ParseJSONValue has a third new parameter: RaiseExc, which overrides the global setting causing the exception to be raised, in case of JSON parsing errors.

JSON Topics

- JSON Frameworks:
 - JSON Objects Framework
 - Readers and Writers JSON Framework
- BSON

See Also

- System.JSON
- System.JSON.Builders
- System.JSON.Writers
- System.JSON.Readers
- System.JSON.BSON

Code Samples

- [RTL.JSON Iterator Code Snippet](#)
- [RTL.JSON Reader Code Snippet](#)
- [RTL.JSON Builder Code Snippet](#)
- [RTL.JSON Writer Code Snippet](#)
- [RTL.JSON Workbench Sample](#)

Retrieved from "<http://docwiki.embarcadero.com/RADStudio/Rio/e/index.php?title=JSON&oldid=267999>"

[This page was last edited on 26 October 2018, at 09:39.](#)