Revised on: 2022-01-19

JSON Essentials™ for COM/ActiveX v1.2.1

> PINERY_®

Table of Contents

1 General

1.1 Scope

1. This document describes API interfaces, basic use cases and related information of the JSON Essentials v1.2.1 COM/ActiveX library (below referred to as The library).
2. The library uses the proprietary implementations of JSON parser, JSON schema and encoding processors that have been created with focus on performance, robustness, stability and standard compliance.
3. The documentation does not describe, change or imply any JSON and JSON Schema normative reference, however the library implementation is aimed to follow and implement requirements from those normative references.
4. In addition to following JSON and JSON Schema normative references the library may provide additional non-standardized functionality that:
    - By no means is meant to substitute any of the standardized normative references and does not contradict them
    - Is chosen to be provided only by the author's sole discretion as a potentially beneficial extended functionality

1.2 Getting started

1. A package contains:
    - core library file (*.OCX, *.LIC)
    - convenience scripts
    - test and use cases source files, scripts and related data files
    - additional information and documentation files
2. Only core library files are required for distribution if it is allowed by the chosen license.
3. These steps are necessary in order to be able to use the library:

- Component registration, see Deployment
- Referencing in a development environment, see Hosting and Referencing
- Library unlocking, see Licensing

4. The convenience scripts automate deployment in a development environment and require execution from a command line with **Administrator** privileges:

   - **install.cmd** – Registers components to the system, saves license token to the system registry for automated tests execution. Run **install.cmd --help** for more available execution options.
   - **test.cmd** – Executes Windows Script Host (WSH) scripts included into the package. Test execution requires a valid license token to be provided during the preceding execution of **install.cmd**.
   - **uninstall.cmd** – Unregisters the previously registered components and deletes the saved license token from the system registry.

5. Refer to the included samples and tests implementation for examples of how the library can be used in various use cases, start with Use Cases.
6. For a complete reference of the exposed library API refer to API Reference and IDL Listings.

1.3 Configurations

1. The library comes in a number of configurations in separate modules aimed to cover the most common use cases and hosting environments.
2. Platform specific configurations:

   - 32-bit – The module to be used in either or both 32- and 64-bit platforms, but can be hosted only in a 32-bit application or a 32-bit component.
   - 64-bit – The module to be used in 64-bit platforms and can be hosted only in a 64-bit application or a 64-bit component.

3. Functional specific configurations:

- DOM for Workstation (DWS) – The module implements and provides only JSON DOM related functionality and provides only the basic, if any, security and behavior customization means
- Schema for Workstation (SWS) - The module implements and provides both JSON DOM and JSON Schema related functionality and provides only the basic, if any, security and behavior customization means
- DOM for Server (DSR) – **[Unavailable in v1.2.1]** The module implements and provides only JSON DOM related functionality and provides advanced security and behavior customization means
- Schema for Server (SSR) - **[Unavailable in v1.2.1]** The module implements and provides both JSON DOM and JSON Schema related functionality and provides advanced security and behavior customization means

4. The combination of the functional and platform configuration names forms full module configuration name:
    - DWS32, DWS64
    - SWS32, SWS64
    - DSR32, DSR64
    - SSR32, SSR64

1.4 Licensing

1. Each module of the library is licensed to use by the licensed user(s) only, regardless of the kind of the obtained license. Refer to the terms of the end user license agreements depending on a licensing type at https://pinery.systems/legal.html.
2. The license must be kept in a .LIC files with the same file name as the subject module and in the same directory where the subject module is stored.
3. The license kind and the license subject can be retrieved from .LIC file as well as from the subject module version properties.
4. The .LIC file can be recovered at any time using command line provided that execution write permissions allow creating and writing files in the current directory:

```
rundll32.exe <SUBJECT_MODULE>.ocx,ExtractLicense
```

5. The license can be shown in a system message box using command line:

```
rundll32.exe <SUBJECT_MODULE>.ocx,ShowLicense
```

6. The license can be printed into STDOUT using command line:

```
rundll32.exe <SUBJECT_MODULE>.ocx,PrintLicense
```

7. Each license is associated with the unique order ID is used in the following below entities in order to eliminate any possible version conflicts when used in environments where other library modules are registered:

   - Module files names
   - ProgID registry records
   - TypeLib GUID (TLBID)
   - CoClass GUIDs (CLSID)
   - Interface and enumeration GUIDs

8. Mixing different module versions, configurations and licenses within one process address space is allowed, however exposed types, classes and interfaces will be completely incompatible for any kinds of cross referencing and thus must not be attempted.

9. In addition to the .LIC file the library must be unlocked using the unique personal license token provided in the order delivery e-mail. The license token is a long random string that is generated for the very first order and remains immutable for all subsequent linked orders. In order to unlock the library the very first call to it should be a call to IJsonEssentials.**unlock** method with the license token value provided. See the JScript helper implementation as an example. A locked library will not be able to create instances of any of its classes. Refer to the IJsonEssentials interface description for more details.

1.5 Versioning

1. The library and its components are versioned using 3 level version number: <MAJOR>.<MINOR>.<PATCH>. For example: 1.2.1

2. Each registered user will receive library updates as stipulated by the requested license type: either every new PATCH update (frequent) or MINOR update (less frequent). All registered users are subject to receiving all MAJOR updates.
3. PATCH update means:
   - no new features added
   - urgent bug fixes, tweaks or some internal improvements were added
   - the module is completely compatible with any previous version with identical MAJOR version component
4. MINOR update means:
   - new features and/or behavior have been added
   - non urgent bug fixes, tweaks or some internal improvements were added
   - the module is completely compatible with any previous version with identical MAJOR version component
5. MAJOR update means:
   - new features and/or behavior have been added
   - bug fixes, tweaks or some internal improvements were added
   - the module may be partly or fully incompatible with any previous version

1.6 Deployment

1. All deployed files must be kept in the same directory with file names preserved.
2. Workstation configurations require deployment only 2 files: .OCX and .LIC.
3. Server configurations require deployment of .OCX and .LIC files but also the corresponding .JSON file with module settings can be optionally deployed.
4. Default server module configuration file can be extracted using the command line:

   ```
   rundll32.exe <SUBJECT_MODULE>.ocx,ExtractConfig
   ```
5. File names of the related .OCX, .LIC and .JSON files must be identical and follow the following scheme:

jess-<CONFIGURATION>-<VERSION>_<ORDER_ID>.<EXTENSION>

The following example shows JSON Schema 1.2.1 module file names ready for deployment onto a 64-bit server machine for order #1010101010:

jess-ssr64-1.2.1_1010101010.ocx

jess-ssr64-1.2.1_1010101010.lic

jess-ssr64-1.2.1_1010101010.json

6. Once deployed onto a target machine the .OCX file has to be registered either by calling the exported DllRegisterServer function (refer to the Windows SDK official documentation on it), or using the command line executed with administrative privileges:

   regsvr32.exe jess-<CONFIGURATION>-<VERSION>_<ORDER_ID>.ocx

1.7 Hosting and Referencing

1. The library requires Windows XP or any newer Windows OS version.
2. Refer to the official documentation on the development environment or programming language of your choice for information on support and using of COM/ActiveX components. The library can be used in all environments and programming languages that support COM/ActiveX components, the most known include, but not limited:
   - Visual Basic (all versions)
   - Visual Basic for Applications (VBA)
   - Windows Scripting Host (WSH)
   - C/C++ (if compiled for Windows platform)
   - .NET (all languages)
   - JScript
   - JavaScript
   - TypeScript
   - Delphi

- Java
- Rust
- Python
- Go
- NodeJS
- Ruby
- Erlang
- TCL
- PowerShell
- Visual Fox Pro (VFP)
- PowerBuilder
- PHP
- ASP
- Clarion
- Visual COBOL
- Matlab
- Internet Explorer (up to version 11)
- etc.

3. 32-bit components should only be referenced from 32-bit environments and 64-bit components should only be referenced from 64-bit environments. However some environments may require 32-bit component to be registered in order to be used in visual designers even when they are assumed to produce 64-bit application or module on output, like Visual Studio WinForms designer.

4. Different environments may or may not have built-in tools for automatic handling component upgrades. Refer to the official documentation for your development environment to find out how to upgrade an existing component reference with a newer one.
5. Different environments may or may not extract type information from the library or automatically generated wrappers for browsing API interfaces, auto-completion and coding hints. If such information is accessible it should be used as the first source of interface description. If such information is not available it can be extracted in IDL language from any of the deployed .OCX files using OleView utility from Microsoft SDK by picking File -> View TypeLib menu item and selecting the .OCX file.
6. Each exposed library interface is dual, which means that interface members can be accessible both via IDispatch or via virtual table.
7. All exposed library interfaces are OLE Automation compatible.

1.8 Conventions

1. As the library can be referenced and used in a number of mainstream programming languages and domain specific languages, the document describes API using a generalized simple pseudo language that should be easy to map to any real programming language intuitively.
2. Interface representation using a pseudo language:

```
interface interface_name {
  # Optional traits
  ...
  # Optional properties
  ...
  # Optional methods
  ...
}
```

where:

- **interface_name** – actual interface name

3. Type traits currently can contain only one supported optional trait:
   - **[enumerator]** – indicates that a type is a collection type that allows enumerating its values using any language specific means or wrappers, if any, available for representing IEnumVARIANT COM interface. Most programming languages allow traversing enumerable objects using language specific reserved words and constructions like **for**, **foreach**, etc. Some languages iterate through collection key names, like JavaScript, other languages iterate through actual objects in a collection. Both iteration methods are supported by an interface marked as enumerable.
4. Method representation using a pseudo language:

   name(arg1 : type1, arg2 : type2, arg3 : type3 = default_value) : result_type
   where:

   - **name** – case sensitive method name
   - **arg1, arg2, arg3** – argument names
   - **type1, type2, type3** – generalized argument type names
   - **= default_value** – default provided value for an optional argument
   - **result_type** – generalized type name of the returned result. If empty no result is returned.
5. Property representation using a pseudo language:

   name : type = default_value [optional_traits]
   Where possible type names include:

   - string – a 16-bit wide zero-ending UTF-16-LE encoded string
   - boolean – **true** or **false**
   - object – any type
   - integer – a signed integer that is at least 32-bit wide
   - interface – any interface exposed by the library
   - OtherType – a specific type exposed by the library

And optional supported traits are:

- **read-only** – A property cannot be assigned to a different value
- **case-insensitive** – Letter case of a value does not affect behavior and is not guaranteed to be preserved as specified.

1.9 Encodings

1. JSON Essentials v1.2.1 is capable of loading and saving JSON data in different encodings, however string values used internally and exposed via library interfaces are always zero-ending sequences of 16-bit wide characters encoded using UTF-16-LE encoding. Environments that support COM/ActiveX components normally either are able to support such string types natively or should provide special data types for encapsulating string conversion to the native formats.
2. Though JSON standard does not mention using Byte Order Marks (BOM) in JSON documents in order to specify used encoding, the library does support BOMs when loading and saving JSON data as follows:
   - On loading a JSON document if BOM is detected it is compared to the BOM of the expected encoding and if they are not matched the document loading is interrupted and an error is reported.
   - On saving a JSON document BOM is not added unless it is explicitly specified that BOM has to be added for the current call.
3. Supported encodings use RFC-6365 encoding names and currently include:

Supported encoding	Byte Order Mark (BOM)
utf-8	0xEF, 0xBB, 0xBF
utf-16	0xFF, 0xFE
utf-32	0xFF, 0xFE, 0x00, 0x00

Table 1 - Supported encodings

2 API Reference

2.1 Enumerations

2.1.1 JsonNodeKind

```
enum JsonNodeKind {
   jsonNodeObject = 1
   jsonNodeArray = 2
   jsonNodeNull = 3
   jsonNodeNumeric = 4
   jsonNodeString = 5
   jsonNodeBoolean = 6
}
```

1. The **JsonNodeKind** enumeration defines all possible kinds of a JSON node.

jsonNodeObject = 1

2. Meaning: JSON node is an object: **{}.**

jsonNodeArray = 2

3. Meaning: JSON node is an array: **[].**

jsonNodeNull = 3

4. Meaning: JSON node is a **null** value.

jsonNodeNumeric = 4

5. Meaning: JSON node is a number.

jsonNodeString = 5

6. Meaning: JSON node is a string.

jsonNodeBoolean = 6

7. Meaning: JSON node is either **true** or **false.**

2.1.2 JsonError

```
enum JsonError {
   jsonErrorSuccess = 0
   jsonErrorCommonFirst = 256
   jsonErrorCommonInvalidArgument = 256
   jsonErrorCommonInvalidObjectState = 257
   jsonErrorCommonUnsupportedByObjectKind = 258
   jsonErrorCommonIndexOutOfBoundary = 259
   jsonErrorCommonInvalidEncodingSequence = 260
   jsonErrorCommonBadDereferencing = 261
   jsonErrorCommonNoMemoryAllocated = 262
   jsonErrorCommonLast = 262
   jsonErrorParserFirst = 4096
   jsonErrorParserUnexpectedEndOfData = 4096
   jsonErrorParserUnexpectedToken = 4097
   jsonErrorParserInvalidData = 4098
   jsonErrorParserInvalidEscapeSequence = 4099
   jsonErrorParserLast = 4099
   jsonErrorDomFirst = 12288
   jsonErrorDomCannotBeAddedToObject = 12288
   jsonErrorDomCannotBeAddedToArray = 12289
   jsonErrorDomNoReadAccess = 12290
   jsonErrorDomNoWriteAccess = 12291
   jsonErrorDomNodeNotFound = 12292
   jsonErrorDomAlreadyAtRoot = 12293
   jsonErrorDomNodeHasNoName = 12294
   jsonErrorDomNoAssociatedNode = 12295
   jsonErrorDomForeignNode = 12296
```

```
    jsonErrorDomLast = 12296
    jsonErrorSchemaFirst = 16384
    jsonErrorSchemaInvalidUri = 16384
    jsonErrorSchemaCannotBeResolved = 16385
    jsonErrorSchemaUnexpectedValue = 16386
    jsonErrorSchemaUnexpectedValueType = 16387
    jsonErrorSchemaDuplicatedAssertion = 16388
    jsonErrorSchemaInvalidAssertionSyntax = 16389
    jsonErrorSchemaReferenceCycle = 16390
    jsonErrorSchemaDuplicatedId = 16391
    jsonErrorSchemaUnexpectedSchema = 16392
    jsonErrorSchemaUnsupportedSchema = 16393
    jsonErrorSchemaRegex = 16394
    jsonErrorSchemaSchemaRequired = 16395
    jsonErrorSchemaLast = 16395
    jsonErrorValidationFirst = 20480
    jsonErrorValidationFalseAssertion = 20480
    jsonErrorValidationValueType = 20481
    jsonErrorValidationValue = 20482
    jsonErrorValidationRequiredProperty = 20483
    jsonErrorValidationNodeCount = 20484
    jsonErrorValidationNotUnique = 20485
    jsonErrorValidationRequiredNode = 20486
    jsonErrorValidationLast = 20486
}
```

1. **JsonError** defines all possible library specific error codes. Error code values are split in error code ranges, some of which are available in certain configurations only:

Error code range name	Configurations
Common	All
Parser	All
Dom	All
Schema	SWS, SSR
Validation	SWS, SSR

Table 2 - Error code ranges

jsonErrorSuccess = 0

2. Message: “No error”

jsonErrorCommonFirst = 256

3. Meaning: Indicates the first error code in the Common error code range

jsonErrorCommonInvalidArgument = 256

4. Message: “Invalid argument”

jsonErrorCommonInvalidObjectState = 257

5. Message: "The object is in invalid state"

jsonErrorCommonUnsupportedByObjectKind = 258

6. Message: "The action is not supported by object kind"

jsonErrorCommonIndexOutOfBoundary = 259

7. Message: "Array index is out of array boundary"

jsonErrorCommonInvalidEncodingSequence = 260

8. Message: "Byte sequence unsupported by selected encoding is encountered"

jsonErrorCommonBadDereferencing = 261

9. Message: "End iterator or invalid pointer dereferencing attempt"

jsonErrorCommonNoMemoryAllocated = 262

10. Message: "Required memory was not allocated"

jsonErrorCommonLast = 262

11. Meaning: Indicates the last error code in the Common error code range

jsonErrorParserFirst = 4096

12. Meaning: Indicates the first error code in the Parser error code range

jsonErrorParserUnexpectedEndOfData = 4096

13. Message: "Unexpected end of data"

jsonErrorParserUnexpectedToken = 4097

14. Message: "Unexpected token"

jsonErrorParserInvalidData = 4098

15. Message: ”Invalid data”

jsonErrorParserInvalidEscapeSequence = 4099

16. Message: "Unrecognized escape sequence"

jsonErrorParserLast = 4099

17. Meaning: Indicates the last error code in the Parser error code range

jsonErrorDomFirst = 12288

18. Meaning: Indicates the first error code in the DOM error code range

jsonErrorDomCannotBeAddedToObject = 12288

19. Message: "A value cannot be added to DOM object"

jsonErrorDomCannotBeAddedToArray = 12289

20. Message: "A value cannot be added to DOM array"

jsonErrorDomNoReadAccess = 12290

21. Message: "Node data reading is prohibited"

jsonErrorDomNoWriteAccess = 12291

22. Message: "Node data writing is prohibited"

jsonErrorDomNodeNotFound = 12292

23. Message: "Requested node not found"

jsonErrorDomAlreadyAtRoot = 12293

24. Message: "The node is already a root node"

jsonErrorDomNodeHasNoName = 12294

25. Message: "The node has no name"

jsonErrorDomNoAssociatedNode = 12295

26. Message: "No node is associated with the request"

jsonErrorDomForeignNode = 12296

27. Message: "The node provided belongs to another DOM document"

jsonErrorDomLast = 12296

28. Meaning: Indicates the last error code in the DOM error code range.

JsonErrorSchemaFirst = 16384

29. Meaning: Indicates the first error code in the Schema error code range.

jsonErrorSchemaInvalidUri = 16384

30. Message: "URI provided is not valid"

jsonErrorSchemaCannotBeResolved = 16385

31. Message: "The requested schema cannot be resolved"

jsonErrorSchemaUnexpectedValue = 16386

32. Message: "Unexpected value"

jsonErrorSchemaUnexpectedValueType = 16387

33. Message: "Unexpected value type"

jsonErrorSchemaDuplicatedAssertion = 16388

34. Message: "Duplicated assertion"

jsonErrorSchemaInvalidAssertionSyntax = 16389

35. Message: "Invalid assertion syntax"

jsonErrorSchemaReferenceCycle = 16390

36. Message: "Reference cycle detected"

jsonErrorSchemaDuplicatedId = 16391

37. Message: "Duplicated schema ID"

jsonErrorSchemaUnexpectedSchema = 16392

38. Message: "Unexpected '$schema' node location"

jsonErrorSchemaUnsupportedSchema = 16393

39. Message: "Unsupported '$schema' URI"

jsonErrorSchemaRegex = 16394

40. Message: "Regular expression instance cannot be created"

jsonErrorSchemaSchemaRequired = 16395

41. Message: "At least one schema is required"

jsonErrorSchemaLast = 16395

42. Meaning: Indicates the last error code in the Schema error code range.

jsonErrorValidationFirst = 20480

43. Meaning: Indicates the last error code in the Validation error code range.

jsonErrorValidationFalseAssertion = 20480

44. Message: "False assertion"

jsonErrorValidationValueType = 20481

45. Message: "Value type doesn't meet requirements"

jsonErrorValidationValue = 20482

46. Message: "Value doesn't meet requirements"

jsonErrorValidationRequiredProperty = 20483

47. Message: "Required property is missing"

jsonErrorValidationNodeCount = 20484

48. Message: "Node count doesn't meet limit requirement"

jsonErrorValidationNotUnique = 20485

49. Message: "Node doesn't meet uniqueness requirement"

jsonErrorValidationRequiredNode = 20486

50. Message: "Required node is missing"

jsonErrorValidationLast = 20486

51. Meaning: Indicates the last error code in the Validation error code range.

2.1.3 JsonStatusKind

```
enum JsonStatusKind {
  jsonStatusUnknown = 0
  jsonStatusJess = 1
  jsonStatusSystem = 2
  jsonStatusProvider = 3
}
```

1. JsonStatusKind defines a kind of a status information returned from any Input-Output operation attempted by the library.

jsonStatusUnknown = 0

2. Meaning: No status information is provided.

jsonStatusJess = 1

3. Meaning: The returned status information describes a library specific error.

jsonStatusSystem = 2

4. Meaning: The returned status information describes an OS error.

jsonStatusProvider = 3

5. Meaning: The returned status information is specific to a data provider used for invoking the requested operation. See Data Providers.

2.1.4 JsonSchemaStandard

```
enum JsonSchemaStandard {
  jsonSchemaStandardUndefined = 0
  jsonSchemaStandardDraft07 = 7
}
```

1. JsonSchemaStandard defines JSON Schema standard used.

jsonSchemaStandardUndefined = 0

2. Meaning: JSON Schema standard is undefined or not specified.

jsonSchemaStandardDraft07 = 7

3. Meaning: JSON Schema Draft-07.

2.2 Interfaces

2.2.1 IJsonEssentials

```
interface IJsonEssentials {
  # Properties
  id : integer [read-only]
  versionMajor : integer [read-only]
  versionMinor : integer [read-only]
  versionPatch : integer [read-only]
  licenseText : string [read-only]
  tools : IJsonTools [read-only]

  # Methods
  unlock(licenseToken : string) : boolean
```

```
    createDocument() : IJsonDocument
    createSchemas() : IJsonSchemas
  }
```

1. The IJsonEssentials defines the interface for the JsonEssentials class and is responsible for providing methods for accessing basic library properties as well as factory methods. See JScript helper object implementation for an example of using it.

id : integer [read-only]

2. Returns: Unique order ID associated with this library instance.

versionMajor : integer [read-only]

3. Returns: The major library version component value.

versionMinor : integer [read-only]

4. Returns: The minor library version component value.

versionPatch : integer [read-only]

5. Returns: The patch library version component value.

licenseText : string [read-only]

6. Returns: The text of the license file.

tools : IJsonTools [read-only]

7. Returns: A global instance of the JsonTools object. Using this property is more efficient than creating a new instance of the JsonTools class every time it's needed, besides that there is no difference between both ways of accessing the IJsonTools interface.

unlock(licenseToken : string) : boolean

8. Effect: Unlocks the library using the specified **licenseToken** string. It must be the very first call to the library as until the library is unlocked it is impossible to create any other library object. The following properties of the IJsonEssentials interface do not require library unlocking: **id**, **versionMajor**, **versionMinor**, **versionPatch**. Any subsequent calls of this method after the first successful unlocking attempt will be ignored.

IMPORTANT: A license token must be accessible only by the licensed developers and should never be stored unencrypted in the end user's environment or publicly exposed to it in any other way. The best practice is to keep it hardcoded and/or encrypted.

Returns: **true** if the library has been unlocked successfully, **false** otherwise.

createDocument() : IJsonDocument

9. Effect: Creates a new instance of an empty JSON document. Using the method is a more efficient way than creating a new instance of the JsonDocument class every time it's needed, besides that there is no differences between both ways to create a JSON document.

   Returns: The IJsonDocument interface of the newly created JSON document object.

createSchemas() : IJsonSchemas

10. Effect: Creates a new instances of a schema collection class. Using the method is a more efficient way than creating a new instance of the JsonSchemas class every time it's needed, besides that there is no differences between both ways to create a schema collection. The method is available on in configurations that include JSON Schema implementation, see Configurations.

    Returns: The IJsonSchemas interface of the newly created JSON schema collection.

2.2.2 IJsonDocument

```
interface IJsonDocument {
  # Properties
  json : string [read-only]
  root : IJsonNode [read-only]
  encoding : string [case-insensitive]
  format : IJsonFormat [read-only]
  location : string [read-only]

  # Methods
  makeEmptyObject() : IJsonNode
```

```
    makeEmptyArray() : IJsonNode
    makeValue(value : object) : IJsonNode
    save(
      target : object,
      encoding : string = undefined,
      request : object = undefined,
      writeBom : boolean = false) : IJsonStatus
    load(
      source : object,
      encoding : string = undefined,
      request : string = undefined) : IJsonStatus
    parse(json : string) : IJsonStatus
  }
```

1. The IJsonDocument defines the interface for the JsonDocument class and is responsible for providing methods for JSON document manipulation and accessing its properties.

json : string [read-only]

2. Returns: A non-empty string that contains JSON data of the entire document. JSON data is formatted using IJsonFormat options exposed by the **format** property. The order of JSON object elements is not guaranteed to match the order in which those elements were added or loaded. See Use case: Create DOM.

root : IJsonNode [read-only]

3. Returns: Root object node interface. See Use case: Traverse DOM.

encoding : string [read-write, case-insensitive]

4. Returns: Encoding name used for saving and loading JSON document. See Encodings, Use case: FileSystem provider.

location : string [read-only]

5. Returns: A data provider specific string that indicates the location or the source of the JSON document object. See Data Providers, Use case: FileSystem provider.

makeEmptyObject() : IJsonNode

6. Effect: Makes root node of the current document an empty JSON object. All existing references to the nested DOM objects get invalidated and must not be used. See Use case: FileSystem provider.

   Returns: Root object node interface.

makeEmptyArray() : IJsonNode

7. Effect: Makes root node of the current document an empty JSON array. All existing references to the nested DOM objects get invalidated and must not be used.

   Returns: Root object node interface.

makeValue(value : object) : IJsonNode

8. Requires: **value** is one of:

   - string
   - number
   - boolean
   - null

   Effect: Makes root node of the current document an JSON value. All existing references to the nested DOM objects get invalidated and must not be used.

   Returns: Root object node interface.

save(

  target : object,

  encoding : string = undefined,

  request : object = undefined,

  writeBom : boolean = false) : IJsonStatus

9. Requires:

- **target** is an object supported by any of the Data Providers that are not read-only.
- **encoding** is either not specified (optional), or a string that indicates one of the supported Encodings. If not specified, the value of the **encoding** property is used.
- **request** is an optional data provider specific parameter. See Data Providers.
- **writeBom** is an optional boolean value that specifies whether Byte Order Mark (BOM) is to be added at the beginning of the saved JSON document in order to specify the used encoding. See Encodings.

Effect: Attempts writing JSON data of the current document into the specified target using provided settings. See Use case: FileSystem provider, Use case: HTTP PUT.

Returns: A status object.

load(
  source : object,
  encoding : string = undefined,
  request : string = undefined) : IJsonStatus

10. Requires:

- **source** is an object supported by any of the Data Providers.
- **encoding** is either not specified (optional), or a string that indicates one of the supported Encodings.
- **request** is an optional data provider specific parameter. See Data Providers.

Effect: Attempts loading and parsing JSON data from the specified target using provided settings. All existing references to the nested DOM objects get invalidated and must not be used. In case of loading failure either the previous DOM is preserved or the document becomes an empty JSON object, depending on a step at which an issue is encountered. See Use case: Parse DOM, Use case: FileSystem provider, Use case: JSON provider, Use case: LSO provider, Use case: HTTP GET, Use case: HTTP PUT, Use case: Schema validation.

Returns: A status object.

parse(json : string) : IJsonStatus

11. Requires: **json** is a valid string that contains JSON data to parse.

Effect: Attempts parsing JSON data provided. All existing references to the nested DOM objects get invalidated and must not be used. In case of parsing failure either the previous DOM is preserved or the document becomes an empty JSON object, depending on a step at which an issue is encountered. See Use case: Parse DOM.

Returns: A status object.

2.2.3 IJsonNode

```
interface IJsonNode {
  # Properties
  json : string [read-only]
  document : IJsonDocument [read-only]
  parent : IJsonNode [read-only]
  name : string [read-only]
  isRoot : boolean [read-only]
  isObject : boolean [read-only]
  isArray : boolean [read-only]
  isEmpty : boolean [read-only]
  isNull : boolean [read-only]
  isNumeric : boolean [read-only]
  isString : boolean [read-only]
  isBoolean : boolean [read-only]
  isContainer : boolean [read-only]
  hasName : boolean [read-only]
  kind : JsonNodeKind [read-only]
  nodes : IJsonNodeCollection [read-only]
  dom : object,
  index : integer [read-only],
  annotation : IJsonNodeAnnotation [read-only]
```

```
    # Methods
    equals(
        other : object,
        noName : boolean = false) : boolean
    save(
        target : object,
        encoding : string = undefined,
        request : object = undefined,
        writeBom : boolean = false) : IJsonStatus
    select(selector : object) : object
    createJsonPointer() : IJsonSelector
}
```

1. IJsonNode defines the interface for a JSON node object and is responsible for providing methods for node manipulation and accessing its properties.

json : string [read-only]

2. Returns: A non-empty string that contains JSON data of the current node and its descendants. JSON data is formatted using IJsonFormat options exposed by the IJsonDocument.**format** property. The order of JSON object elements is not guaranteed to match the order in which those elements were added or loaded. See Use case: Parse DOM.

document : IJsonDocument [read-only]

3. Returns: Owner document object interface.

parent : IJsonNode [read-only]

4. Returns: The parent's node interface or **null** if the current node is a root node.

name : string [read-only]

5. Returns: Name of the current node if the current node's parent node is a JSON object, otherwise **null**.

isRoot : boolean [read-only]

6. Returns: **true** if the current node is the root document node, otherwise **false**.

isObject : boolean [read-only]

7. Returns: **true** if the current node is a JSON object, otherwise **false**.

isArray : boolean [read-only]

8. Returns: **true** if the current node is a JSON array, otherwise **false**.

isEmpty : boolean [read-only]

9. Returns: **true** if the current node is an empty JSON object or an empty JSON array, otherwise **false**. For value nodes the result is always **true**.

isNull : boolean [read-only]

10. Returns: **true** if the current node is **null**, otherwise **false**.

isNumeric : boolean [read-only]

11. Returns: **true** if the current node is a numeric value node, otherwise **false**.

isString : boolean [read-only]

12. Returns: **true** if the current node is the string value node, otherwise **false**.

isBoolean : boolean [read-only]

13. Returns: **true** if the current node is the boolean values node, otherwise **false**.

isContainer : boolean [read-only]

14. Returns: **true** if the current node is either a JSON object or JSON array node, otherwise **false**. See Use case: Traverse DOM.

hasName : boolean [read-only]

15. Returns: **true** if the current node has a name associated with it, i.e. its parent's node is a JSON object node, otherwise **false**.

kind : JsonNodeKind [read-only]

16. Returns: The kind of the current node.

nodes : IJsonNodeCollection [read-only]

17. Returns: The interface of the collection of children nodes if the current node is either a JSON object or JSON array node, otherwise **null**. See Use case: Traverse DOM.

dom : object

18. Accepts: A value or an object the current node will be set to, it can be one of:

    - string
    - number
    - boolean
    - null
    - any object as described in `Language Specific Objects (LSO)

    Returns: Direct DOM Access (DDA) interface of a collection node or a language specific value of a value node. See Use case: LSO provider.

index : integer [read-only]

19. Returns: The zero-based index of the current node if its parent node is an array, otherwise **0**.

annotation : IJsonNodeAnnotation [read-only]

20. Returns: The interface of the node annotation object for the current node if library configuration (see Configurations) is either SWS or SSR, otherwise it's not supported and the return value is always **null**. By default the annotation object is not filled with annotation data for a node. In order to fill annotation data it is necessary to call the IJsonSchema.**annotate** method for the current node, its parent node or the root document node. See Use case: Schema annotation.

equals(

  other : object,

  noName : boolean = false) : boolean

21. Effect: Compares the current node to the **other** object that is either an IJsonNode interface of a node object or an IJsonDocument interface of a document object for which the root node will be compared to. If the nodes contain children nodes, those get compared recursively. By default the current node name is is compared to the **other** node name if such name exist, unless **noName** is set to **true**. The order of object elements is not considered. See Use case: JSON provider.

22. Returns: **true** if both JSON DOM trees starting from the current and the other nodes are identical, otherwise **false**.

save(
  target : object,
  encoding : string = undefined,
  request : object = undefined,
  writeBom : boolean = false) : IJsonStatus

23. Requires:

- **target** is an object supported by any of the Data Providers that are not read-only.
- **encoding** is either not specified (optional), or a string that indicates one of the supported Encodings. If not specified, the value of the IJsonDocument**.encoding** property is used.
- **request** is an optional data provider specific parameter. See Data Providers.
- **writeBom** is an optional boolean value that specifies whether Byte Order Mark (BOM) is to be added at the beginning of the saved JSON document in order to specify the used encoding. See Encodings.

Effect: Attempts writing JSON data of the current node and all its descendants into the specified target using provided settings. See Use case: FileSystem provider, Use case: HTTP PUT.

Returns: A status object.

select(selector : object) : object

24. Requires: **selector** is one of:

- a string query expression, if the string starts with **"rel:"** prefix the remaining part has to be a Relative JSON Pointer expression, otherwise the entire string has to be a JSON Pointer expression
- IJsonSelector interface of a selector object created using either IJsonTools**.createJsonPointer** or IJsonTools**.createRelativeJsonPointer**. Unlike passing a string query, which results in parsing the query string with each call, a selector object already contains a parsed query expression which results in a faster query processing.

Returns: Depending on the query the returned value can be one of:

- IJsonNode interface of the requested node
- string value of a node name (specific to Relative JSON Pointer)
- integer zero-based value of a node index (specific to Relative JSON Pointer)
- **null** if the query was unsuccessful.

See Use case: Select node (slow), Use case: Select node (fast).

createJsonPointer() : IJsonSelector

25. Returns: A new selector object that uses an absolute JSON Pointer from the root node of the document to the current element.

2.2.4 IJsonNodeCollection

```
interface IJsonNodeCollection {
  [enumerable]

  # Properties
  length : integer [read-only]

  # Methods
  item(key : object) : IJsonNode
  add(value : object, name : object = undefined) : IJsonNode
  addArray(name : object = undefined) : IJsonNode
  addObject(name : object = undefined) : IJsonNode
  contains(key : object) : boolean
  remove(key : object)
  clear()
}
```

1. IJsonNodeCollection defines an interface of an enumerable collection of JSON nodes that is exposed by either JSON array or JSON object nodes.

length : integer [read-only]

2. Returns: The number of nodes in the collection. See Use case: Traverse DOM.

item(key : object) : IJsonNode

3. Requires:

   - key – Depending on node kind the value of the argument is interpreted as:
     - an integer value of zero-based child node index of a JSON array or object node, if index value is not in the range of available elements the return value is **null.** Indices of an object node are not guaranteed to reflect the order in which child nodes have been added.
     - a string value of child node name for a JSON object node, if the node does not contain a child node with such a name the return value is **null**
     - a IJsonNode interface of a child node, if the node does not contain the provided child node the return value is **null**

   Returns: IJsonNode interface of the requested node or **null** if the key value is not acceptable or there is no child node associated with the key. See Use case: Traverse DOM.

add(value : object, name : object = undefined) : IJsonNode

4. Requires:

   - value – a value of a new node to be added, it can be one of:
     - string
     - number
     - boolean
     - null
     - any object as described in Language Specific Objects (LSO)

- **name** – the name of a new child node to be added to JSON object node, it must not be specified if a node is to be added to a JSON array node.

Effect: adds a new node to a JSON array or object node. The order of the named items is not guaranteed to be preserved during JSON serialization. Node name duplicates are not allowed. See Use case: Create DOM.

Returns: IJsonNode interface of the added child node or **null** if a node cannot be added.

addArray(name : object = undefined) : IJsonNode

5. Requires: **name** – the name of a new child node to be added to JSON object node, it must not be specified if a node is to be added to a JSON array node.

   Effect: Adds a new JSON array node to the current node. Node name duplicates are not allowed. See Use case: Create DOM.

   Returns: IJsonNode interface of the added child node or **null** if a node cannot be added.

addObject(name : object = undefined) : IJsonNode

6. Requires: **name** – the name of a new child node to be added to JSON object node, it must not be specified if a node is to be added to a JSON array node.

   Effect: Adds a new JSON object node to the current node. Node name duplicates are not allowed. See Use case: Create DOM.

   Returns: IJsonNode interface of the added child node or **null** if a node cannot be added.

contains(key : object) : boolean

7. Requires:

   - **key** – Depending on node kind the value of the argument is interpreted as:
     - an integer value of zero-based child node index for a JSON array node, if index value is not in the range of available elements the return value is **false**
     - a string value of child node name for a JSON object node, if the node does not contain a child node with such a name the return value is **false**

  - a IJsonNode interface of a child node, if the node does not contain the provided child node the return value is **false**

Returns: **true** if the requested node is the child of the current node, otherwise or if the key value is not acceptable the return value is **false**.

remove(key : object)

8. Requires:

- **key** – Depending on node kind the value of the argument is interpreted as:
  - an integer value of zero-based child node index for a JSON array or object node, if index value is not in the range of available elements the return value is **false**
  - a string value of child node name for a JSON object node, if the node does not contain a child node with such a name the return value is **false**
  - a IJsonNode interface of a child node, if the node does not contain the provided child node the return value is **false**

Effect: Removes a child node identified by the key from the collection. All remaining references to the removed node get invalidated and must not be used. After removing a node from an object node the order of the sibling nodes is not guaranteed to be preserved.

clear()

9. Effect: Removes all child nodes from the collection. All remaining references to the removed nodes get invalidated and must not be used.

2.2.5 IJsonNodeAnnotation

interface IJsonNodeAnnotation {

# Properties

title : IJsonNode [read-only]

description : IJsonNode [read-only]

contentEncoding : IJsonNode [read-only]

contentMediaType : IJsonNode [read-only]

```
    readOnly : IJsonNode [read-only]
    writeOnly : IJsonNode [read-only]
    defaultNode : IJsonNode [read-only]
    examples : IJsonNode [read-only]
}
```

1. The IJsonNodeAnnotation interface provides access to node annotation nodes assigned in the result of invoking IJsonSchema.**annotate** method on the owning node or any of its parent nodes. See Use case: Schema annotation.

title : IJsonNode [read-only]

2. Returns: The IJsonNode interface of the **title** node associated with the owning node, or **null** if no title is associated with the owning node by the annotation schema.

description : IJsonNode [read-only]

3. Returns: The IJsonNode interface of the **description** node associated with the owning node, or **null** if no description is associated with the owning node by the annotation schema.

contentEncoding : IJsonNode [read-only]

4. Returns: The IJsonNode interface of the **contentEncoding** node associated with the owning node, or **null** if no contain encoding is associated with the owning node by the annotation schema.

contentMediaType : IJsonNode [read-only]

5. Returns: The IJsonNode interface of the **contentMediaType** node associated with the owning node, or **null** if no content media type is associated with the owning node by the annotation schema.

readOnly : IJsonNode [read-only]

6. Returns: The IJsonNode interface of the **readOnly** node associated with the owning node, or **null** if no write protection is associated with the owning node by the annotation schema.

writeOnly : IJsonNode [read-only]

7. Returns: The IJsonNode interface of the **writeOnly** node associated with the owning node, or **null** if no read protection is associated with the owning node by the annotation schema.

defaultNode : IJsonNode [read-only]

8. Returns: The IJsonNode interface of the **default** node associated with the owning node, or **null** if no default value is associated with the owning node by the annotation schema.

examples : IJsonNode [read-only]

9. Returns: The IJsonNode interface of the **examples** node associated with the owning node, or **null** if no examples are associated with the owning node by the annotation schema.

2.2.6 IJsonStatus

```
interface IJsonStatus {
  # Properties
  success : boolean [read-only]
  kind : JsonStatusKind [read-only]
  code : integer [read-only]
  response : object [read-only]
  line : integer [read-only]
  column : integer [read-only]
  schema : IJsonNode [read-only]
  instance : IJsonNode [read-only]
}
```

1. The IJsonStatus interface describes the status of an operation performed by the library. The values of its properties depend on status kind. See Use case: HTTP GET, Use case: HTTP PUT, Use case: Schema validation.

success : boolean [read-only]

2. Returns: **true** if the performed operation is deemed to be completed successfully, otherwise **false**.

kind : JsonStatusKind [read-only]

3. Returns: The kind of the described status as described by JsonStatusKind.

code : integer [read-only]

4. Returns: The value of the status code determined by the status kind as follows:

- **jsonStatusJess** – The value is a library specific error code and it has one of JsonError values. Depending on the error code the **line**, **column**, **schema** and **instance** properties can have valid values that provide more context for an error occurred.
- **jsonStatusSystem** – The value is an OS specific HRESULT error code.
- **jsonStatusProvider** – The status code value is determined by the data provider used for the executed operation as described in Data Providers.

response : object [read-only]

5. Returns: The response value associated with the returned status code and determined by the status kind as follows:

- **jsonStatusJess** – The value is a string description of the library specific error code as defined by JsonError enumeration.
- **jsonStatusSystem** – The string value of the OS provided error message associated with the returned HRESULT error code.
- **jsonStatusProvider** – The object value and its type is determined by the data provider used for the executed operation as described in Data Providers. See Use case: HTTP PUT.

line : integer [read-only]

6. Returns: The zero-base line index of the parsed document where a parser error is detected or a negative value if not applicable. The value is applicable only to **jsonStatusJess** status kind if error code is in the Parser code range, see JsonError.

column : integer [read-only]

7. Returns: The zero-base column index of the parsed document where a parser error is detected or a negative value if not applicable. The value is applicable only to **jsonStatusJess** status kind if error code is in the Parser code range, see JsonError.

schema : IJsonNode [read-only]

8. Returns: The property is available only in SWS and SSR Configurations and the value is applicable only to **jsonStatusJess** status kind as follows:

   - if the error is in the Schema code range (see JsonError) then it indicates a schema node where an error is detected, or **null** if not applicable.
   - if the error is in the Validation code range (see JsonError) then it indicates a schema assertion node that did not pass validation of an **instance** node, or **null** if not applicable.

   See Use case: Schema validation.

instance : IJsonNode [read-only]

9. Returns: The property is available only in SWS and SSR Configurations and the value is applicable only to **jsonStatusJess** status kind if the error is in the Validation code range (see JsonError). It indicates an instance node that failed validation of the schema assertion node specified by the **schema** property, or **null** if not applicable. See Use case: Schema validation.

2.2.7 IJsonFormat

```
interface IJsonFormat {
  # Properties
  indentationSize : integer
  useDosNewLines : boolean
  useTabs : boolean
  placeBracketsOnNewLine : boolean
  placeValueOnNewLine : boolean
  addNewLineInEmptyBrackets : boolean
  addSpaceBeforeColon : boolean
  addSpaceAfterColon : boolean
  addSpaceInBrackets : boolean
  addSpaceAfterComma : boolean
  forceEscapeNonAscii : boolean
```

    forceEscapeNonEascii : boolean
  }

1. The IJsonFormat interfaces describes JSON document formatting options used for converting JSON documents and nodes into a string representation. See Use case: Format JSON.

indentationSize : integer

2. Effect: Gets or sets the number of space characters (**U+0020**) to be used as an indentation if tabs are not used.

useDosNewLines : boolean

3. Effect: Enables or disables using DOS newlines in the formatted documents. If this value is **true** then all newlines are specified by Carriage-Return+Line-Feed (CRLF) character sequence (**U+000D U+000A = "\r\n"**), otherwise all newlines are specified by the only Line-Feed (LF) character (**LF = U+000A = "\n"**).

useTabs : boolean

4. Effect: If **true** then all indentations are indicated by exactly one TAB character (**U+0009 = "\t"**) per an indentation level, otherwise each indentation level is filled with a number of space characters (**U+0020**) the number of times defined by the **indentationSize** property.

placeBracketsOnNewLine : boolean

5. Effect: If true then all opening brackets (**"{", "["**) is placed on a new line.

placeValueOnNewLine : boolean

6. Effect: If **true** then every value or node in JSON arrays and JSON objects is placed on a new line.

addNewLineInEmptyBrackets : boolean

7. Effect: If **true** then new lines are added into empty JSON arrays and JSON objects.

addSpaceBeforeColon : boolean

8. Effect: If **true** then a space character (**U+0020**) is added before the colon character that delimits node name from node value is JSON objects.

addSpaceAfterColon : boolean

9. Effect: If **true** then a space character (**U+0020**) is added after the colon character that delimits node name from node value is JSON objects.

addSpaceInBrackets : boolean

10. Effect: If **true** then a space character (**U+0020**) is added:
    - after opening and before closing brackets (**"{", "}", "[", "]"**) if **placeValueOnNewLine** is **false**
    - in empty brackets if **addNewLineInEmptyBrackets** is **false**

addSpaceAfterComma : boolean

11. Effect: If **true** then a space character (**U+0020**) is added after comas if **placeValueOnNewLine** is **false.**

forceEscapeNonAscii : boolean

12. Effect: If true then the generated JSON data is guaranteed to consist of only ASCII characters (**U+0000 – U+007F**), all non-ASCII characters in JSON strings will be escaped using JSON escape sequences.

forceEscapeNonEascii : boolean

13. Effect: If true then the generated JSON data is guaranteed to consist of only EASCII characters (**U+0000 – U+00FF**), all non-EASCII characters in JSON strings will be escaped using JSON escape sequences.

2.2.8 IJsonTools

```
interface IJsonTools {
  # Methods
  createJsonPointer(pointer : string) : IJsonSelector
  createRelativeJsonPointer(pointer : string) : IJsonSelector
}
```

1. The IJsonTools interface provides access to the algorithms and utilities exposed by the JsonTools object. See Use case: Select node (fast).

createJsonPointer(pointer : string) : IJsonSelector

2. Requires: **pointer** is a string JSON Pointer expression to be used for querying JSON DOM.

Effect: The query expression gets parsed during selector object creation thus increasing performance of IJsonNode.**select** method execution if compared to passing a string query directly to IJsonNode.**select**.

createRelativeJsonPointer(pointer : string) : IJsonSelector

3. Requires: **pointer** is a string Relative JSON Pointer expression to be used for querying JSON DOM.

   Effect: The query expression gets parsed during selector object creation thus increasing performance of IJsonNode.**select** method execution if compared to passing a string query prefixed with **"rel:"** directly to IJsonNode.**select**.

2.2.9 IJsonPolicy

```
interface IJsonPolicy {
  # Properties
  nullValue : object
  ddaEscape : string
}
```

1. The IJsonPolicy interface provides access to the global library configuration parameters.

nullValue : object

2. Effect: As various languages represent **null** value differently it may be necessary to specify a language specific **null** value to be used consistently in all operation that may involve JSON **null** value. The default value corresponds to **VT_EMPTY** variant type which used as **null** in C# and **Nothing** in Visual Basic.

   The overridden **null** value is not used for non JSON related results, for example if the result of IJsonNode.**parent** for a root node will always **null** in the meaning of 'no-object' regardless of the value of **nullValue** property. However if a JSON node has a **null** value or a new **null** node is being added using IJsonNodeCollection.**add** method, the value of **nullValue** property will be used.

   The overridden **null** value can be an object of any type, however setting it to a string, numeric or boolean value will make it impossible to distinct from any other JSON values of the same types. Also, it is not recommended to set its value to an object or interface reference value as in multi threaded environments it can lead to unexpected behavior. See JScript helpers.

Returns: The current value of a JSON **null** value.

ddaEscape : string

3. Effect: The specified Direct DOM Access (DDA) interface key can be used as a property name of a DDA collection object in order to escape from DDA interface back to IJsonNode interface. The default value is an underscore symbol "**_**". See VisualBasic sample from Use case: Traverse DOM.

2.2.10 IJsonSelector

```
interface IJsonSelector {
  # Properties
  expression : string [read-only]

  # Methods
  select(from : object) : object
}
```

1. The IJsonSelector interface is an interface for the selector object used to speed up JSON DOM query execution.

expression : string [read-only]

2. Returns: The query expression string.

select(from : object) : object

3. Requires: **from** is either IJsonNode node interface or IJsonDocument document interface to execute the query on.

   Effect: Performs a query defined by the **expression** string on a specified object using the abstracted query algorithm.

   Returns: Depending on the query implementation the returned value can be one of:

- IJsonNode interface of the requested node
- string value of a node name (specific to Relative JSON Pointer)
- integer zero-based value of a node index (specific to Relative JSON Pointer)
- **null** if the query was unsuccessful.

2.2.11 IJsonSchemas

```
interface IJsonSchemas {
  [enumerable]

  # Properties
  length : integer [read-only]

  # Methods
  item(key : object) : string
  add(
    schemaNode : object,
    defaultId : string) : IJsonSchema
  find(id : string) : IJsonSchema
  remove(id : string)
  clear()
  annotate(
    instance : object,
    id : string = undefined) : IJsonStatus
  validate(
    instance : object,
    id : string = undefined) : IJsonStatus
  onRequestSchema(handler : object)
}
```

1. The IJsonSchemas interface is the default interface for the JsonSchemas class and provides methods for working with a set of JSON schemas. See Use case: Schema validation, Use case: Schema request, Use case: Schema annotation.

2. A collection of schemas may contain both manually and automatically added or resolved schemas.

3. All schemas within a collection can reference each other as it is defined by the referencing schema's standard.

length : integer [read-only]

4. Returns: The number of schemas stored in the collection.

item(key : integer) : string

5. Effect: Obtains an absolute schema identifier by either schema index in the collection.

   Returns: The absolute schema identified value or **null** if the index value provided is outside the available range.

add(
  schemaNode : object,
  defaultId : string) : IJsonSchema

6. Requires:

   - **schemaNode** – is either a IJsonNode or IJsonDocument interface of a node or a document that starts a JSON Schema to be added to the collection.
   - **defaultId** – a string that can be used as a default schema **"$id"** if it's not provided by the schema itself. The collection is guaranteed to retrieve the schema by this ID value in addition to the schema provided ID.

   Effect: Adds a new schema identified by a unique default ID and an optional schema provided unique ID to the schema collection. Schema ID duplicates are not allowed, however default ID is allowed to be identical to the schema provided ID. See Use case: Schema validation, Use case: Schema request.

   Returns: A newly added schema object.

find(id : string) : IJsonSchema

7. Requires: **id** is either a default or schema provided ID.

   Returns: A registered JSON schema object if found by the provided ID or **null**.

remove(id : string)

8. Requires: **id** is either a default or schema provided ID.

   Effect: Removes a schema identified by the provided ID from the collection. All schema references get invalidated and thus must not be used.

clear()

9. Effect: Removes all schemas from the collection. All schema references get invalidated and thus must not be used.

annotate(
  instance : object,
  id : string = undefined) : IJsonStatus

10. Requires:

    - **instance** – is either a IJsonNode interface of a node or IJsonDocument interface of a document to be annotated.
    - **id** – if specified it must be one of the schema IDs registered in the collection, otherwise the ID value will be attempted to extract from the **"$schema"** attribute of a top-level instance JSON object if such is provided.

    Effect: Performs validation and annotation of the provided JSON instance using a registered schema determined by the provided or extracted schema ID. If the provided or determined schema ID is not registered in the collection then the handler previously set using IJsonSchemas**.onRequestSchema** is called in order to attempt loading and registering a new schema which then is used for validation and annotation. If no ID is provided, determined or no schema is loaded for the requested ID the operation cannot be completed. The result of the successful annotation is filling of the applicable properties of the IJsonNodeAnnotation interface exposed by the IJsonNode**.annotation** property for the instance node and all of its child nodes. If a node was already annotated prior to a new annotation call all previous annotation data gets reset. See Use case: Schema annotation.

Returns: The validation status object of the attempted operation.

validate(
  instance : object,
  id : string = undefined) : IJsonStatus

11. Requires:

- **instance** – is either a IJsonNode interface of a node or IJsonDocument interface of a document to be annotated.
- **id** – if specified it must be one of the schema IDs registered in the collection, otherwise the ID value will be attempted to extract from the **"$schema"** attribute of a top-level instance JSON object if such is provided.

Effect: Performs only validation of the provided JSON instance using a registered schema determined by the provided or extracted schema ID. If the provided or determined schema ID is not registered in the collection then the handler previously set using IJsonSchemas.**onRequestSchema** is called in order to attempt loading and registering a new schema which then is used for validation. If no ID is provided, determined or no schema is loaded for the requested ID the operation cannot be completed. See Use case: Schema validation.

Returns: The validation status object of the attempted operation.

onRequestSchema(handler : object)

12. Requires: handler – is one of:

- an IDispatch compatible object that either exposes a named **onRequestSchema** method
- an IDispatch compatible object that allows making a call to it as if it's a method (aka DISPID_VALUE call)
- a .NET delegate

In all cases the handler is treated as a function of type:

handler(schemas : IJsonSchemas, id : string) : IJsonSchema
where:

- **schemas** – The current collection of JSON schemas.
- **id** – An ID of the requested schema (not found in the collection).

- return value – The resulted loaded and added JSON schema object returned by **schemas.add** or **null** if the requested schema cannot be loaded or added due to any reason.

Effect: The provided handler is called automatically every time a requested schema ID cannot be found in the collection during annotation or validation. The handler is supposed to load and add the requested schema to the current collection thus making it available in the collection for the subsequent request if there is any. However the handler is never called when a schema is being looked up using IJsonSchemas**.contains**, IJsonSchemas**.find** or IJsonSchemas**.remove** calls. See Use case: Schema request.

2.2.12 IJsonSchema

```
interface IJsonSchema {
  # Properties
  id : string [read-only]
  standard : JsonSchemaStandard [read-only]
  schemas : IJsonSchemas [read-only]
  root : IJsonNode [read-only]

  # Methods
  annotate(instance : object) : IJsonStatus
  validate(instance : object) : IJsonStatus
}
```

1. The IJsonSchema interface provides access to JSON schema object properties and methods.

id : string [read-only]

2. Returns: The JSON schema ID value which is a unique ID in the owning schema collection.

standard : JsonSchemaStandard [read-only]

3. Returns: JSON schema standard identifier for the current schema.

schemas : IJsonSchemas [read-only]

4. Returns: The owning schema collection object.

root : IJsonNode [read-only]

5. Returns: The root schema document node. As JSON schemas can be nested it's not always a root document node.

annotate(instance : object) : IJsonStatus

6. Requires: **instance** – is either a IJsonNode interface of a node or IJsonDocument interface of a document to be validated and annotated.

   Effect: Performs validation and annotation of the provided JSON instance using the current schema regardless of the schema ID specified by the **"$schema"** value of the root instance node. The result of the successful annotation is filling of the applicable properties of the IJsonNodeAnnotation interface exposed by the IJsonNode**.annotation** property for the instance node and all of its child nodes. If a node was already annotated prior to a new annotation call all previous annotation data gets reset. See Use case: Schema annotation.

   Returns: The validation status object of the attempted operation.

validate(instance : object) : IJsonStatus

7. Requires: **instance** – is either a IJsonNode interface of a node or IJsonDocument interface of a document to be validated.

   Effect: Performs validation of the provided JSON instance using the current schema using the current schema regardless of the schema ID specified by the **"$schema"** value of the root instance node. See Use case: Schema validation.

   Returns: The validation status object of the attempted operation.

2.2.13 Direct DOM Access (DDA) interface

1. The Direct DOM Access (DDA) interface is a dynamic interface that allows using JSON element names as object properties and JSON arrays as array objects thus allowing reading and writing DOM values directly. See Use case: LSO provider.

2. Any identifier or an array element specified by its index used as a read/write property of an object returned by the IJsonNode.**dom** property or any of its child elements that represent JSON collections provides access to the respective JSON node data as if it was a class member or an array element. For example (JavaScript):

```
document.load({
   a: {
      b: true,
      c: [1, 2],
      “_”: null,
      “圓周率”: 3.14159
   }
});

var root_dda = document.root.dom;
root_dda.a.b = false;
root_dda.a.c[0] = root_dda.a.c[1] * 2;
```

The code will modify JSON document as follows:

```
{
   “a”: {
      “b”: false,
      “c”: [4, 2],
      “_”: null,
      “圓周率”: 3.14159
   }
}
```

3. If a DDA property or an array element represents a JSON value node (string, numeric, boolean, null) its value type will be the closest matching language specific type.

4. If a DDA property represents a collection JSON node (object or array) a special DDA escape key can be used as a property name in order to obtain IJsonNode interface for the current DDA object. The default value of the DDA escape key is the underscore symbol "_". For languages that don't support the underscore symbol as a property name the DDA escape key can be globally changed using IJsonPolicy.**ddaEscape** property (see VisualBasic sample from Use case: Traverse DOM). It can be any string value that is not likely to appear in a JSON document as node name. If such a name is expected in a JSON document, using it as a property name on a DDA interface will always return IJsonNode interface of that DDA interface, and never the interface of the named node. If such name collision is expected, IJsonNodeCollection.**item** must be used instead:

```
// DDA escape on collection nodes
Test.assertEqual(root_dda._, document.root);
Test.assertEqual(root_dda.a.c._.document, document);

// DDA escape key collision
var node_a = document.root.nodes.item("a");
Test.assertEqual(root_dda.a._, node_a);
Test.assertEqual(root_dda.a._, root_dda.a["_"]);
Test.assertNotEqual(root_dda.a._., node_a.nodes.item("_"));
```

5. If a language does not support property names in Unicode and such JSON node name is expected, it can be accessed from a DDA interface by its name specified as a string key provided that the language is capable to such a Unicode string into UTF-16-LE and file encoding used preserves code point bits:

```
Test.assertEqual(root_dda.a["圓周率" ], 3.14159);
```

2.3 Classes

2.3.1 JsonEssentials

1. ProgID: JsonEssentials_###.JsonEssentials.1

   where:

   - ### - is the unique order ID value.
   - 1 - the MAJOR component of the version number, see Versioning.

2. CLSID is unique for each order ID and can be obtained from the Windows registry from the default value of the HKEY_CLASSES_ROOT\JsonEssentials_###.JsonEssentials.1\CLSID key.
3. IJsonEssentials is the default interface of the created object.

2.3.2 JsonDocument

1. ProgID: JsonEssentials_###.JsonDocument.1

   where:

   - ### - is the unique order ID value.
   - 1 - the MAJOR component of the version number, see Versioning.

2. CLSID is unique for each order ID and can be obtained from the Windows registry from the default value of the HKEY_CLASSES_ROOT\JsonEssentials_###.JsonDocument.1\CLSID key.
3. IJsonDocument is the default interface of the created object.

2.3.3 JsonSchemas

1. ProgID: JsonSchemas_###.JsonSchemas.1

   where:

- **###** - is the unique order ID value.
- **1** - the MAJOR component of the version number, see Versioning.

2. CLSID is unique for each order ID and can be obtained from the Windows registry from the default value of the **HKEY_CLASSES_ROOT\JsonEssentials_###.JsonSchemas.1\CLSID** key.
3. IJsonSchemas is the default interface of the created object.

2.3.4 JsonTools

1. ProgID: **JsonTools_###.JsonTools.1**

   where:

   - **###** - is the unique order ID value.
   - **1** - the MAJOR component of the version number, see Versioning.

2. CLSID is unique for each order ID and can be obtained from the Windows registry from the default value of the **HKEY_CLASSES_ROOT\JsonEssentials_###.JsonTools.1\CLSID** key.
3. IJsonTools is the default interface of the created object.

2.4 Data Providers

1. A data provider is an implementation of a JSON saving/loading mechanism to/from a specified target/source.
2. Depending on the underlying concept the saving or loading operations can be unsupported by a data provider. But at least one operation is always supported.

2.4.1 FileSystem

1. The FileSystem data provider allows both loading and saving JSON data from/to a file system object specified by a path as described in Naming Files, Paths and Namespaces. See Use case: FileSystem provider.
2. If a relative path is provided the actual file system object location is determined by combining the current process directory with the provided relative path.
3. The specified file path must start with **"file://"** prefix.

4. The **source** and **target** arguments of the **load** and **save** methods of IJsonDocument and IJsonNode interfaces should be the file system object path values.

5. The **request** argument of the **load** and **save** methods of IJsonDocument and IJsonNode interfaces is not used and must not be specified.

6. If a system error occurs while attempting resolving, reading or writing the specified file system object path the requested operation fails and the status is reported as **jsonStatusSystem** (see IJsonStatus).

7. The value of IJsonDocument**.location** property of a successfully loaded JSON document using the FileSystem data provider is a full canonical file system path prefixed with **"file://"**.

2.4.2 Http

1. The Http data provider allows both loading and saving JSON data from/to a remote object specified by URL via HTTP protocol. See Use case: HTTP GET, Use case: HTTP PUT.

2. The specified URL must start either with **"http://"** or **"https://"** prefix.

3. The **source** and **target** arguments of the **load** and **save** methods of IJsonDocument and IJsonNode interfaces should be the URL values.

4. The optional **request** argument of the **load** and **save** methods of IJsonDocument and IJsonNode defines HTTP and content properties as described in Request format. The argument can be IJsonDocument or IJsonNode interface, or it will be used as a **source** argument for the internal IJsonDocument**.load** call to load configuration JSON DOM from. If argument specifies a remote JSON data source via HTTP protocol that data will be loaded with the default configuration values. For such use cases it is better to pre-load the configuration data pass IJsonNode or IJsonDocument interface of the pre-loaded configuration DOM as a value of the **request** argument.

5. If a system error occurs while attempting resolving, reading or writing the specified remote object the requested operation fails and the status is reported as **jsonStatusSystem** (see IJsonStatus).

6. If an HTTP error occurs while attempting reading or writing the specified remote object the requested operation fails and the status is reported as **jsonStatusProvider** (see IJsonStatus) and the HTTP status code

is used as the operation status code. An HTTP status is considered failed if its value it greater or equal to 400, i.e. Bad Request.

7. Upon successful completion of an operation it is possible to load a JSON response if such is available or the values of HTTP headers from the response. See the **"response"** request configuration (Request format) for further details.

8. The value of IJsonDocument**.location** property of a successfully loaded JSON document using the Http data provider is a supplied URL from which the document is loaded.

2.4.2.1 Request format

1. Generic format:

```
{
  "method": <string>,
  "query": {
    <name>: <value>,
    ...
  },
  "form": {
    <name>: <value>
    ...
  },
  "headers": {
    <name>: <value>,
    ...
  },
  "timeouts": {
    "resolve": <integer>
    "connect": <integer>
    "send": <integer>
    “receive”: <integer>
  },
  “response”: {
    “status”: <boolean>
    “headers”: <boolean>,
    “body”: <”skip”|”text”|”base64”|”json”>
  }
}
```

2. Though some of the values are used in HTTP request composition, the implementation does not verify compliance of the provided values with HTTP request standard specifications, so it’s up to the caller to ensure the standard compliant values to be provided, otherwise a server side error may be triggered.
3. If the request is a writing request, i.e. JSON data is being uploaded to the server, then the request gets send with **Content-Type** set to **application/json** unless additional **form** data is provided.
4. **method** is a string value that contains an HTTP method name (see RFC-7231 Request Methods) to be used for an HTTP request. By default it’s GET for a reading attempt and PUT for a writing attempt, but it can be set to any server supported HTTP method name.
5. **query** is an optional collection of non-encoded name/value string pairs that specify custom GET query parameters to be added to the URL provided as defined by RFC-3986 Query.
6. `form` is an optional collection of non-encoded key/value string pairs that specify HTTP form data as defined by RFC-1867 Form-based File Upload in HTML. If **form** is not empty and:
   - if the request is a reading only request, i.e. no JSON data is being uploaded to the server, then the request gets send with **Content-Type** set to **application/x-www-form-urlencoded**
   - if the request is a writing request, i.e. JSON data is being uploaded to the server, then the request gets send with **Content-Type** set to **multipart/form-data,** and the JSON payload part can be identified by the **name=“json”** or **filename=”data.json”**.

7. **headers** is an optional collection of non-encoded name/value string pairs that specify custom HTTP headers to be added to an HTTP request as defined by RFC-7231 Request Header Fields.

8. **timeouts** is an optional element that defines TCP/IP timeouts. If any timeout value is no provided or its value is 0 then its default value is used. All timeout values are in milliseconds. Supported timeouts include:

   - **resolve** – timeout for resolving a host name to an IP address. The default value 0 means no timeout, i.e. the operation can run infinite.
   - **connect** – the timeout for establishing a TCP/IP connection with a target host. The default value is 60000 (60 seconds).
   - **send** – the timeout for sending and individual TCP/IP packet to a target host. The default value is 30000 (30 seconds).
   - **receive** – the timeout for receiving and individual TCP/IP packet from a target host. The default value is 30000 (30 seconds).

9. **response** is an optional element that controls how an HTTP response will be handled upon successful request completion. If any its elements implies response data collection, the response data will be available via IJsonStatus.**response** property as an IJsonDocument object that follows Response format. The following nested elements are supported:

   - **status** – true indicates that HTTP status code and text will be collected into the status element, see Response format.
   - **headers** – **true** indicates that all response HTTP header name/value pairs will be collected into the **headers** element, see Response format. If response headers are not needed the value can be set to false or the element omitted.
   - **body** – controls if and how the entire HTTP response body will be collected. If the operation completes successfully the result can be extracted from a JSON document that follows Response format and is accessible via IJsonStatus.**response** property as IJsonDocument. The supported string values include:
     - **skip** – no body data is collected. It's the default value. Implied if **body** is omitted.
     - **text** – body data is collected and stored as a string value.

- **base64** – body data is collected as a binary data which is then encoded and stored as Base-64.
- **json** – body data is collected, parsed and stored as JSON document. If parsing fails the IJsonStatus.**kind** is set to **jsonStatusJess** and IJsonStatus.**code** takes one of JsonError values.

2.4.2.2 Response format

1. If an HTTP response data collection is implied by the response element of the request (see Request format) and the request operation is successful, the IJsonStatus.**response** property returns IJsonDocument interface of a JSON document object that has the following generic format:

```
{
  "status":  {
    "code": <integer>,
    "text": <string>
  },
  "headers": {
    <name>: <value>,
    ...
  },
  "body": <string|json>
}
```

2. **status** element is added if **response.status** is set to **true** in the request, if specified then:
   - **code** – an integer HTTP status code as per RFC-7231 Response Status Codes
   - **text** – the message string associated with the returned status code in the returned response.
3. **headers** element is added if **response.headers** is set to **true** in the request, if specified then the element is a JSON object that contains name/value string pairs for all HTTP headers returned in the response.

4. **body** element is added if **response.body** is specified in the request and its value is not set to skip. Depending on **response.body** value it can be either a raw response string, Base-64 encoded response or JSON of the parsed response body, refer to Request format for the detailed description.

2.4.3 Stream

1. Any object that exposes an IStream interface can be specified as a source of JSON data for the loading operations or a target for writing JSON data to. Such objects are usually language and library specific, refer to the official documentation of the objects intended to be used as input-output streams for JSON for IStream compatibility confirmation.
2. If stream I/O error occurs the status of operation is returned as **jsonStatusSystem** (see JsonStatusKind).
3. The value of IJsonDocument**.location** property of a successfully loaded JSON document using the Stream data provider is always **"[stream]"**.
4. The **request** argument of the **load** and **save** methods of IJsonDocument and IJsonNode interfaces is not used and must not be specified.

2.4.4 Json

1. Plain JSON data can be used as a source of JSON data for loading operations only if **"json://"** prefix is added. See Use case: JSON provider.
2. The data provider is read-only, it cannot be used as a target for writing operation.
3. The value of IJsonDocument**.location** property of a successfully loaded JSON document using the Json data provider is always **"[json]"**.
4. The **request** argument of the **load** and **save** methods of IJsonDocument and IJsonNode interfaces is not used and must not be specified.

2.4.5 Language Specific Objects (LSO)

1. Loading and assigning operations can accept objects of any language specific data types as JSON source provided that they are either COM compatible (derived from IUnknown or of VARIANT type) or the environment is able to provide a COM compatible wrapper implicitly, for example JavaScript and Visual Basic

objects and arrays, .NET arrays, etc. Such objects get examined by the library and treated differently depending on how they get classified:

- all IEnumVARIANT derived and other array-like detected interfaces get converted to JSON arrays
- all IDispatch derived and other object-like detected interfaces get converted to JSON objects
- all other value types get converted to the best suitable JSON value type.

See Use case: LSO provider.

2. The data provider is read-only, it cannot be used as a target for writing operations.
3. The value of IJsonDocument**.location** property of a successfully loaded JSON document for this data provider is always **"[object]"**.
4. In case a cycle is detected in LSO tree the respective JSON node gets assigned a string value **"[cycle]"** and recursion breaks.
5. The **request** argument of the **load** and **save** methods of IJsonDocument and IJsonNode interfaces is not used and must not be specified.

2.4.6 Registry

1. The Registry data provider allows both loading and saving JSON data from/to a registry value specified by a full registry value path that starts with **"reg://"** prefix. See Use case: Registry provider.
2. The first component of a registry path must specify a root registry key name. The following root registry key names are supported and described in Windows SDK as Predefined Keys:
   - HKEY_CLASSES_ROOT
   - HKEY_CURRENT_USER
   - HKEY_LOCAL_MACHINE
   - HKEY_USERS
   - HKEY_PERFORMANCE_DATA
   - HKEY_PERFORMANCE_TEXT

  - HKEY_PERFORMANCE_NLSTEXT
  - HKEY_CURRENT_CONFIG
  - HKEY_DYN_DATA
  - HKEY_CURRENT_USER_LOCAL_SETTINGS
  - HKCR (short alias for HKEY_CLASSES_ROOT)
  - HKCU (short alias for HKEY_CURRENT_USER)
  - HKLM (short alias for HKEY_LOCAL_MACHINE)

3. The last component of a registry path must specify a REG_BINARY registry value name that belongs to a registry key specified by the previous path component.
4. The **source** and **target** arguments of the **load** and **save** methods of IJsonDocument and IJsonNode interfaces should be the registry path values.
5. The optional **request** argument of the **load** and **save** methods of IJsonDocument and IJsonNode interfaces defines configuration JSON DOM and can be IJsonDocument or IJsonNode interface, or is used as the **source** argument for the internal IJsonDocument.**load** call to load JSON configuration data from. The configuration JSON uses the following format:

   ```
   {
     "forceCreateKey": <boolean>
   }
   ```

   Where **forceCreateKey** defines whether non-existent keys should be automatically created when attempted to access (**true**), or not (**false**). The default value is **false**.
6. If a system error occurs while attempting resolving, reading or writing the specified registry value the requested operation fails and the status is reported as **jsonStatusSystem** (see IJsonStatus).
7. The value of IJsonDocument.**location** property of a successfully loaded JSON document using the Registry data provider is a full registry value path prefixed with **"reg://"**.

3 Use Cases

3.1 Helpers

3.1.1 JScript

The following library factory object is used in all use cases for creating instances of library object. The ORDER_ID constant must be replaced with a valid order ID value prior to executing use cases.

```
Jess = (function () {
   return {
      orderId: ORDER_ID,
      version: {
         major: 1,
         minor: 2,
         patch: 1
      },

      features: {
         schema: 1
      },

      library: null,

      // Initializes JSON Essential library.
      init: function () {
         // Create a new JsonEssentials class instance.
         this.library = new ActiveXObject('JsonEssentials_' + this.orderId
            + '.JsonEssentials.' + this.version.major);
```

```
    // The very first call to the library has to be unlocking using
    // the license token supplied from command line.
    if (!this.library.unlock(Test.getLicenseToken())) {
        var msg = 'ERROR: Unable to unlock JSON Essentials';
        Test.message(msg);
        throw Error(msg);
    }

    // As the default JSON 'null' value is not compatible with
    // JScript null value it has to be overridden.
    this.library.tools.policy.nullValue = null;
},

// The method creates a new instance of the JSON document object.
createDocument: function () {
    return this.library.createDocument();

    // JSON Document object can also be created using its
    // class identifier, but it's a slower method:
    //   return new ActiveXObject('JsonEssentials_' + this.orderId
    //     + '.JsonDocument.' + this.version.major);
},

// The method creates a new JsonTools class instance.
createTools: function () {
    return this.library.tools;

    // JSON Tools object can also be created using its class
```

```
            // identifier, but it's a slower method:
            //   return new ActiveXObject('JsonEssentials_' + this.orderId
            //     + '.JsonTools.' + this.version.major);
        },

        // The method creates a new JsonSchemas class instance.
        createSchemas: function () {
            return this.library.createSchemas();

            // JSON Schemas object can also be created using its class
            // identifier, but it's a slower method:
            //   return new ActiveXObject('JsonEssentials_' + this.orderId
            //     + '.JsonSchemas.' + this.version.major);
        }
    };
})();

Jess.init();
```

3.1.2 C#

```
[ComVisible(true)]
[ClassInterface(ClassInterfaceType.AutoDual)]
public class SomeObject {
    public object c1 { get; set; }
    public object c2 { get; set; }
}
```

```
[ComVisible(true)]
[ClassInterface(ClassInterfaceType.AutoDual)]
public class RootObject {
   public bool a { get; set; }
   public double b { get; set; }
   public object c { get; set; }
   public object[] d { get; set; }
}
```

3.1.3 Visual Basic

3.1.3.1 EmptyDocumentModel.cls

```
' An empty class
```

3.1.3.2 SomeDocumentModel.cls

```
Option Explicit
Option Base 0

Public a As Variant
Public b As Variant
Public c As Variant
Public d As Variant
Public e As Variant
Public f As Variant
Public g As Variant
Public h As Variant
```

3.2 Use case: Create DOM

This example demonstrates how to build JSON DOM.

3.2.1 JScript

```
// Create JSON document object.
var document = Jess.createDocument();

// By default a JSON document is an empty object
// but for illustrative purposes it is made explicit.
var root = document.makeEmptyObject();

// Build JSON DOM:
// {
//   "a" : true,
//   "b" : 123,
//   "c" : [ { "x" : "y" } ]
// }
root.nodes.add(true, 'a');
root.nodes.add(123, 'b');
root.nodes.addObject('c');
root.nodes.addArray('d').nodes.add('abc');

// Ensure the DOM is built correctly.
Test.assertEqual(document.json, '{\"a\":true,\"b\":123,\"c\":[{\"x\":\"y\"}]}');
```

3.2.2 C#

```
// Create JSON document object.
```

```
var document = Jess.createDocument();

// By default a JSON document is an empty object
// but for illustrative purposes it is made explicit.
var root = document.makeEmptyObject();

// Build JSON DOM:
// {
//   "a" : true,
//   "b" : 123,
//   "c" : [ { "x" : "y" } ]
// }
root.nodes.add(true, "a");
root.nodes.add(123, "b");
root.nodes.addArray("c").nodes.addObject().nodes.add("y", "x");

// Ensure the DOM is built correctly.
Test.assertEqual(document.json, "{\"a\":true,\"b\":123,\"c\":[{\"x\":\"y\"}]}");
```

3.2.3 Visual Basic

```
' Create JSON document object.
Dim document As JsonDocument
Set document = New JsonDocument

' By default a JSON document is an empty object
' but for illustrative purposes it is made explicit.
Dim root As IJsonNode
```

```
Set root = document.makeEmptyObject

' Build JSON DOM:
' {
'   "a" : true,
'   "b" : 123,
'   "c" : [ { "x" : "y" } ]
' }
root.nodes.Add True, "a"
root.nodes.Add 123, "b"
root.nodes.addArray("c").nodes.addObject().nodes.Add "y", "x"

' Ensure the DOM is built correctly.
Test.assertEqual document.json, "{""a"":true,""b"":123,""c"":[{""x"":""y""}]}"
```

3.3 Use case: Parse DOM

This example demonstrates how to a JSON document specified by a string.

3.3.1 JScript

```
// Create JSON document object.
var document = Jess.createDocument();

// Parse JSON document specified by a string.
var json = '{\"a\":true,\"b\":123,\"c\":[{\"x\":\"y\"}]}';
document.parse(json);

// Ensure the DOM is parsed correctly.
```

```
Test.assertEqual(document.json, json);

// Load the same DOM from a string using JSON provider.
document.load('json://' + json);

// Ensure the DOM is loaded correctly.
Test.assertEqual(document.json, json);
```

3.3.2 C#

```
// Create JSON document object.
var document = Jess.createDocument();

// Parse JSON document specified by a string.
var json = "{\"a\":true,\"b\":123,\"c\":[{\"x\":\"y\"}]}";
document.parse(json);

// Ensure the DOM is parsed correctly.
Test.assertEqual(document.json, json);

// Load the same DOM from a string using JSON provider.
document.load("json://" + json);

// Ensure the DOM is loaded correctly.
Test.assertEqual(document.json, json);
```

3.3.3 Visual Basic

```
' Create JSON document object.
Dim document As JsonDocument
Set document = New JsonDocument

' Parse JSON document specified by a string.
Dim json As String
json = "{""a"":true,""b"":123,""c"":[{""x"":""y""}]}"
document.parse json

' Ensure the DOM is parsed correctly.
Test.assertEqual document.json, json

' Load the same DOM from a string using JSON provider.
document.Load "json://" + json

' Ensure the DOM is loaded correctly.
Test.assertEqual document.json, json
```

3.4 Use case: Traverse DOM

This example demonstrates ways to traverse JSON DOM.

3.4.1 JScript

```
// Create JSON document object.
var document = Jess.createDocument();

// Parse JSON.
```

```
document.parse('{\"a\":true,\"b\":123,\"c\":[{\"x\":\"y\"}]}');

var count = 0;
var traverse;

// This function counts the number of nodes in
// DOM by traversing the node tree sequentially.
traverse = function(node) {
  // Increment counter for the current node.
  ++count;

  // Proceed with child node traversal only if the
  // current node is a container.
  if(node.isContainer) {
    // Loop through all child node indices of the current node.
    for(var i = 0; i < node.nodes.length; ++i) {
      // Repeat traversal recursively for each child node
      // accessed by its index.
       traverse(node.nodes.item(i));
    }
  }
};

// Start node tree traversal from the root node.
traverse(document.root);

// Verify the number of nodes computed.
Test.assertEqual(count, 6);
```

```
// This function counts the number of nodes in DOM
// by DOM keys and accessing DOM elements directly
// using Direct DOM Access interface.
traverse = function(dom) {
  for(var key in dom) {
    // Increment counter for the current node.
    ++count;

    // Obtain DOM for a child node determined by the current key.
    var childDom = dom[key];

    // If the DOM exposes DDA Escape key then the DOM has child nodes.
    if(childDom._) {
      // Repeat traversal recursively for the current child DOM.
      traverse(childDom);
      continue;
    }
  }
};

// Start traversal from the root node.
count = 1;
traverse(document.root.dom);

// Verify the number of nodes computed.
Test.assertEqual(count, 6);
```

3.4.2 C#

```
// This function counts the number of nodes in
// DOM by traversing the node tree sequentially.
private static int TraverseNode(IJsonNode node) {
  int count = 1;

  // Proceed with child node traversal only if the
  // current node is a container.
  if (node.isContainer) {
    // Loop through all child node indices of the current node.
    for (var i = 0; i < node.nodes.length; ++i) {
      // Repeat traversal recursively for each child node
      // accessed by its index.
       count += TraverseNode(node.nodes[i]);
    }
  }

  return count;
}

// This function counts the number of nodes in DOM
// by DOM keys and accessing DOM elements directly
// using Direct DOM Access interface.
private static int TraverseNodeDom(dynamic dom) {
  int count = 1;

  foreach (dynamic childNode in dom) {
    // Proceed with child node traversal only if the
```

```
        // current node is a container.
        if (childNode.isContainer) {
          // Repeat traversal recursively for each child DOM.
          count += TraverseNodeDom(childNode.dom);
          continue;
        }

        ++count;
    }

    return count;
}

// Create JSON document object.
var document = Jess.createDocument();

// Parse JSON.
document.parse("{\"a\":true,\"b\":123,\"c\":[{\"x\":\"y\"}]}");

// Start traversal from the root node.
var count = TraverseNode(document.root);

// Verify the number of nodes computed.
Test.assertEqual(count, 6);

// Start traversal from the root node.
count = TraverseNodeDom(document.root.dom);
```

```
// Verify the number of nodes computed.
Test.assertEqual(count, 6);
```

3.4.3 Visual Basic

```
' This function counts the number of nodes in
' DOM by traversing the node tree sequentially.
Private Function CountJsonNodes(node As IJsonNode) As Integer
   Dim count As Integer
   count = 1

   ' Proceed with child node traversal only if the
   ' current node is a container.
   If node.isContainer Then
      ' Loop through all child node indices of the current node.
      Dim i As Integer
      For i = 0 To node.nodes.length - 1
         ' Repeat traversal recursively for each child node
         ' accessed by its index.
         count = count + CountJsonNodes(node.nodes(i))
      Next
   End If

   CountJsonNodes = count
End Function

' This function counts the number of nodes in DOM
' by DOM keys and accessing DOM elements directly
' using Direct DOM Access interface.
Private Function CountDomNodes(dom) As Integer
  Dim count As Integer
  count = 1

  ' Loop through all immediate nodes in DOM.
  Dim node
  For Each node In dom
    ' Proceed with child node traversal only if the
    ' current node is a container.
    If node.isContainer Then
      ' Repeat traversal recursively for each child DOM.
      count = count + CountDomNodes(node.dom)
    Else
      count = count + 1
    End If
  Next

  CountDomNodes = count
End Function

' Set DDA Escape key.
Dim tools As JsonTools
Set tools = New JsonTools
tools.policy.ddaEscape = "NODE_"

' Create JSON document object.
Dim document As JsonDocument
```

```
Set document = New JsonDocument

' Parse JSON.
document.parse "{""a"":true,""b"":123,""c"":[{""x"":""y""}]}"

' Start node tree traversal from the root node.
Dim count As Integer
count = CountJsonNodes(document.root)

' Verify the number of nodes computed.
Test.assertEqual count, 6

count = CountDomNodes(document.root.dom)
Test.assertEqual count, 6
```

3.5 Use case: Format JSON

This example demonstrates how to configure JSON output formatting rules for a JSON document.

3.5.1 JScript

```
// Create JSON document object.
var document = Jess.createDocument();

// Parse JSON.
document.parse('{\"a\":true,\"b\":123,\"c\":[{\"x\":\"y\"}]}');

// Change document format rules.
var format = document.format;
```

```
format.indentationSize = 2;
format.useTabs = false;
format.placeBracketsOnNewLine = true;
format.placeValueOnNewLine = true;
format.addSpaceBeforeColon = true;
format.addSpaceAfterColon = true;
format.addSpaceInBrackets = true;

// Ensure the document is formatted according
// to the updated formatting rules.
Test.assertEqual(document.json,
    '{\n' +
    '  "a" : true,\n' +
    '  "b" : 123,\n' +
    '  "c" :\n' +
    '  [\n' +
    '    {\n' +
    '      "x" : "y"\n' +
    '    }\n' +
    '  ]\n' +
    '}'
);
```

3.5.2 C#

```
// Create JSON document object.
var document = Jess.createDocument();
```

```
// Parse JSON.
document.parse("{\"a\":true,\"b\":123,\"c\":[{\"x\":\"y\"}]}");

// Change document format rules.
var format = document.format;
format.indentationSize = 2;
format.useTabs = false;
format.placeBracketsOnNewLine = true;
format.placeValueOnNewLine = true;
format.addSpaceBeforeColon = true;
format.addSpaceAfterColon = true;
format.addSpaceInBrackets = true;

// Ensure the document is formatted according
// to the update formatting rules.
Test.assertEqual(document.json,
  "{\n" +
  "  \"a\" : true,\n" +
  "  \"b\" : 123,\n" +
  "  \"c\" :\n" +
  "  [\n" +
  "    {\n" +
  "      \"x\" : \"y\"\n" +
  "    }\n" +
  "  ]\n" +
  "}"
);
```

3.5.3 Visual Basic

```
' Create JSON document object.
Dim document As JsonDocument
Set document = New JsonDocument

' Parse JSON.
document.parse "{""a"":true,""b"":123,""c"":[{""x"":""y""}]}"

' Change document format rules.
Dim format As IJsonFormat
Set format = document.format
format.indentationSize = 2
format.useTabs = False
format.placeBracketsOnNewLine = True
format.placeValueOnNewLine = True
format.addSpaceBeforeColon = True
format.addSpaceAfterColon = True
format.addSpaceInBrackets = True

' Ensure the document is formatted according
' to the updated formatting rules.
Test.assertEqual document.json, _
  "{" + Chr(10) + _
  "  ""a"" : true," + Chr(10) + _
  "  ""b"" : 123," + Chr(10) + _
  "  ""c"" :" + Chr(10) + _
  "  { }," + Chr(10) + _
  "  [" + Chr(10) + _
  "    {" + Chr(10) + _
  "      ""x"" : ""y""" + Chr(10) + _
  "    }" + Chr(10) + _
  "  ]" + Chr(10) + _
  "}"
```

3.6 Use case: Select node (slow)

This example demonstrates how to select a node from a JSON document using JSON and Relative JSON Pointers specified by a string, i.e. slow select method.

3.6.1 JScript

```
// Create JSON document object.
var document = Jess.createDocument();

// Parse JSON
document.parse('{\"a\":true,\"b\":123,\"c\":{},\"d\":[\"abc\"]}');

// Select the first node of the 'd' node using JSON Pointer
// starting from the root document node.
var node_abc = document.root.select('/d/0');

// Select node 'a' starting from the previously selected
// first child node of node 'd' and traversing first up to
// the root node and then down to node 'a' using Relative
// JSON Pointer.
var node_a = node_abc.select('rel:2/a');
```

```
// Ensure both nodes have been selected properly.
Test.assertEqual(node_abc.json, '\"abc\"');
Test.assertEqual(node_a.json, 'true');
```

3.6.2 C#

```
// Create JSON document object.
var document = Jess.createDocument();

// Parse JSON
document.parse("{\"a\":true,\"b\":123,\"c\":{},\"d\":[\"abc\"]}");

// Select the first node of the 'd' node using JSON Pointer
// starting from the root document node.
var node_abc = document.root.select("/d/0");

// Select node 'a' starting from the previously selected
// first child node of node 'd' and traversing first up to
// the root node and then down to node 'a' using Relative
// JSON Pointer.
var node_a = node_abc.select("rel:2/a");

// Ensure both nodes have been selected properly.
Test.assertEqual(node_abc.json, "\"abc\"");
Test.assertEqual(node_a.json, "true");
```

3.6.3 Visual Basic

```
' Create JSON document object.
Dim document As JsonDocument
Set document = New JsonDocument

' Parse JSON.
document.parse "{""a"":true,""b"":123,""c"":{},""d"":[""abc""]}"

' Select the first node of the 'd' node using JSON Pointer
' starting from the root document node.
Dim node_abc As IJsonNode
Set node_abc = document.root.Select("/d/0")

' Select node 'a' starting from the previously selected
' first child node of node 'd' and traversing first up to
' the root node and then down to node 'a' using Relative
' JSON Pointer.
Dim node_a As IJsonNode
Set node_a = node_abc.Select("rel:2/a")

' Ensure both nodes have been selected properly.
Test.assertEqual node_abc.json, """abc"""
Test.assertEqual node_a.json, "true"
```

3.7 Use case: Select node (fast)

This example demonstrates how to select a node from a JSON document using pre-compiled JSON and Relative JSON Pointers, i.e. fast select method.

3.7.1 JScript

```
// Create JSON document object.
var document = Jess.createDocument();

// Create JSON Tools object.
var tools = Jess.createTools();

// Create a pre-compiled JSON Pointer for selecting
// the first child node of root's node 'd'.
var pointer = tools.createJsonPointer('/d/0');

// Create a pre-compiled Relative JSON Pointer for selecting
// node 'a' after ascending 2 levels up from a node the
// selector is executed for.
var relativePointer = tools.createRelativeJsonPointer('2/a');

// Parse JSON.
document.parse('{\"a\":true,\"b\":123,\"c\":{},\"d\":[\"abc\"]}');

// Execute JSON pointer.
var node_abc = document.root.select(pointer);

// Execute Relative JSON pointer on the previously selected node.
var node_a = node_abc.select(relativePointer);

// Ensure both nodes have been selected properly.
Test.assertEqual(node_abc.json, '\"abc\"');
Test.assertEqual(node_a.json, 'true');
```

3.7.2 C#

```
// Create JSON document object.
var document = Jess.createDocument();

// Create JSON Tools object.
var tools = Jess.createTools();

// Create a pre-compiled JSON Pointer for selecting
// the first child node of root's node 'd'.
var pointer = tools.createJsonPointer("/d/0");

// Create a pre-compiled Relative JSON Pointer for selecting
// node 'a' after ascending 2 levels up from a node the
// selector is executed for.
var relativePointer = tools.createRelativeJsonPointer("2/a");

// Parse JSON.
document.parse("{\"a\":true,\"b\":123,\"c\":{},\"d\":[\"abc\"]}");

// Execute JSON pointer.
var node_abc = document.root.select(pointer);

// Execute Relative JSON pointer on the previously selected node.
var node_a = node_abc.select(relativePointer);

// Ensure both nodes have been selected properly.
```

```
Test.assertEqual(node_abc.json, "\"abc\"");
Test.assertEqual(node_a.json, "true");
```

3.7.3 Visual Basic

```
' Create JSON document object.
Dim document As JsonDocument
Set document = New JsonDocument

' Create JSON Tools object.
Dim tools As JsonTools
Set tools = New JsonTools

' Create a pre-compiled JSON Pointer for selecting
' the first child node of root's node 'd'.
Dim pointer As IJsonSelector
Set pointer = tools.CreateJsonPointer("/d/0")

' Create a pre-compiled Relative JSON Pointer for selecting
' node 'a' after ascending 2 levels up from a node the
' selector is executed for.
Dim relativePointer As IJsonSelector
Set relativePointer = tools.createRelativeJsonPointer("2/a")

' Parse JSON.
document.parse "{""a"":true,""b"":123,""c"":{},""d"":[""abc""]}"

' Execute JSON pointer.
```

```
Dim node_abc As IJsonNode
Set node_abc = document.root.Select(pointer)

' Execute Relative JSON pointer on the previously selected node.
Dim node_a As IJsonNode
Set node_a = node_abc.Select(relativePointer)

' Ensure both nodes have been selected properly.
Test.assertEqual node_abc.json, """abc"""
Test.assertEqual node_a.json, "true"
```

3.8 Use case: FileSystem provider

This example demonstrates how to use FileSystem provider for saving and loading JSON data to and from a file specified by its path.

3.8.1 JScript

```
// Create JSON document object.
var document = Jess.createDocument();

// Parse JSON.
var json = '{\"a\":true,\"b\":123,\"c\":[{\"x\":\"y\"}]}';
document.parse(json);

// Save document to test.json in the current
// directory using UTF-16 encoding.
document.save('file://test.json', 'utf-16');
```

```
// Make document an empty JSON object to ensure
// loading the saved document reconstructs the DOM.
document.makeEmptyObject();

// Load the previously saved document.
document.load('file://test.json', 'utf-16');

// Ensure the loaded document matches the initial JSON.
Test.assertEqual(document.root.json, json);
Test.assertEqual(document.encoding, 'utf-16');
Test.assertTrue(0 < document.location.indexOf('test.json'));
```

3.8.2 C#

```
// Create JSON document object.
var document = Jess.createDocument();

// Parse JSON.
var json = "{\"a\":true,\"b\":123,\"c\":[{\"x\":\"y\"}]}";
document.parse(json);

// Save document to test.json in the current
// directory using UTF-16 encoding.
document.save("file://test.json", "utf-16");

// Make document an empty JSON object to ensure
// loading the saved document reconstructs the DOM.
document.makeEmptyObject();
```

```
// Load the previously saved document.
document.load("file://test.json", "utf-16");

// Ensure the loaded document matches the initial JSON.
Test.assertEqual(document.root.json, json);
Test.assertEqual(document.encoding, "utf-16");
Test.assertTrue(0 < document.location.IndexOf("test.json"));
```

3.8.3 Visual Basic

```
' Create JSON document object.
Dim document As JsonDocument
Set document = New JsonDocument

' Parse JSON.
Dim json As String
json = "{""a"":true,""b"":123,""c"":[{""x"":""y""}]}"
document.parse json

' Save document to test.json in the current directory using UTF-16 encoding.
document.save "file://test.json", "utf-16"

' Make document an empty JSON object to ensure
' loading the saved document reconstructs the DOM.
document.makeEmptyObject

' Load the previously saved document.
```

```
document.Load "file://test.json", "utf-16"

' Ensure the loaded document matches the initial JSON.
Test.assertEqual document.root.json, json
Test.assertEqual document.encoding, "utf-16"
Test.assertTrue 0 < InStr(document.location, "test.json")
```

3.9 Use case: Registry provider

This example demonstrates how to use Registry provider for saving and loading JSON data to and from a registry value specified by its path.

3.9.1 JScript

```
// Create JSON document object.
var document = Jess.createDocument();

// Parse JSON.
var json = '{\"a\":true,\"b\":123,\"c\":[{\"x\":\"y\"}]}';
document.parse(json);

// Save document to the specified registry location using UTF-16 encoding.
var regPath = 'reg://HKCU\\Software\\Pinery\\JsonEssentials_' + Jess.orderId;
document.save(regPath, 'utf-16');

// Make document an empty JSON object to ensure
// loading the saved document reconstructs the DOM.
document.makeEmptyObject();
```

```
// Load the previously saved document.
document.load(regPath, 'utf-16');

// Ensure the loaded document matches the initial JSON.
Test.assertEqual(document.root.json, json);
Test.assertEqual(document.encoding, 'utf-16');
Test.assertEqual(document.location, regPath);
```

3.9.2 C#

```
// Create JSON document object.
var document = Jess.createDocument();

// Parse JSON.
var json = "{\"a\":true,\"b\":123,\"c\":[{\"x\":\"y\"}]}";
document.parse(json);

// Save document to the specified registry location using UTF-16 encoding.
var regPath = "reg://HKCU\\Software\\Pinery\\JsonEssentials_" + Jess.Library.id;
document.save(regPath, "utf-16");

// Make document an empty JSON object to ensure
// loading the saved document reconstructs the DOM.
document.makeEmptyObject();

// Load the previously saved document.
document.load(regPath, "utf-16");
```

```
// Ensure the loaded document matches the initial JSON.
Test.assertEqual(document.root.json, json);
Test.assertEqual(document.encoding, "utf-16");
Test.assertEqual(document.location, regPath);
```

3.9.3 Visual Basic

```
' Create JsonEssentials library facade
Dim library As JsonEssentials
Set library = New JsonEssentials

' Create JSON document object.
Dim document As JsonDocument
Set document = New JsonDocument

' Parse JSON.
Dim json As String
json = "{""a"":true,""b"":123,""c"":[{""x"":""y""}]}"
document.parse json

' Save document to the specified registry location using UTF-16 encoding.
Dim regPath As String
regPath = "reg://HKCU\Software\Pinery\JsonEssentials_" + Str(library.id)
document.save regPath, "utf-16"

' Make document an empty JSON object to ensure
' loading the saved document reconstructs the DOM.
document.makeEmptyObject
```

```
' Load the previously saved document.
document.Load regPath, "utf-16"

' Ensure the loaded document matches the initial JSON.
Test.assertEqual document.root.json, json
Test.assertEqual document.encoding, "utf-16"
Test.assertEqual document.location, regPath
```

3.10 Use case: JSON provider

This example demonstrates how to use JSON provider for loading JSON data from the specified source string directly.

3.10.1 JScript

```
// Create JSON document objects.
var document1 = Jess.createDocument();
var document2 = Jess.createDocument();

var json = '{\"a\":true,\"b\":123,\"c\":[{\"x\":\"y\"}]}';

// Load JSON directly from the specified source into the first document.
document1.load('json://' + json);

// Parse the same JSON as a string into the second document.
document2.parse(json);

// Ensure both documents are equal.
Test.assertTrue(document1.root.equals(document2.root));
```

3.10.2 C#

```
// Create JSON document objects.
var document1 = Jess.createDocument();
var document2 = Jess.createDocument();

var json = "{\"a\":true,\"b\":123,\"c\":[{\"x\":\"y\"}]}";

// Load JSON directly from the specified source into the first document.
document1.load("json://" + json);

// Parse the same JSON as a string into the second document.
document2.parse(json);

// Ensure both documents are equal.
Test.assertTrue(document1.root.equals(document2.root));
```

3.10.3 Visual Basic

```
' Create JSON document objects.
Dim document1 As JsonDocument
Set document1 = New JsonDocument

Dim document2 As JsonDocument
Set document2 = New JsonDocument

' Load JSON directly from the specified source into the first document.
Dim json As String
json = "{""a"":true,""b"":123,""c"":[{""x"":""y""}]}"
```

```
document1.Load "json://" + json

' Parse the same JSON as a string into the second document.
document2.parse json

' Ensure both documents are equal.
Test.assertTrue document1.root.equals(document2.root)
```

3.11 Use case: LSO provider

This example demonstrates capabilities of use the LSO provider for loading DOM from language objects and primitives and modifying DOM as if it was an object of its model class.

3.11.1 JScript

```
// Create JSON document object.
var document = Jess.createDocument();

// Load JSON DOM right from a JavaScript object.
document.load({
  a: true,
  b: 123,
  c: {},
  d: [ 'abc' ]
});

// Get root DOM direct access interface.
var dom = document.root.dom;
```

```
// Ensure element 'a' is loaded as 'true'.
Test.assertTrue(dom.a);

// Modify the value of 'a' right in place.
dom.a = false;

// Ensure element 'b' is loaded as number 123.
Test.assertEqual(dom.b, 123);

// Increase 'b' by 2 right in place and verify result.
dom.b += 2;
Test.assertEqual(dom.b, 125.0);

// Make element 'c' a new object right from a JavaScript object.
dom.c = {
   c1: 1,
   c2: 2
};

// Modify the first element of 'd' right in place.
dom.d[0] = 'ABC';

// Add new elements to array 'd' right from JavaScript values and objects.
var d = dom.d._;
d.nodes.add(321);
d.nodes.add([ 'XYZ', null ]);

Test.message(document);
```

```
// Output:
//
// {"d":["ABC",321,["XYZ",null]],"a":false,"b":125,"c":{"c1":1,"c2":2}}
```

3.11.2 C#

```
// Create JSON document object.
var document = Jess.createDocument();

// Load JSON DOM right from a .NET object.
document.load(new RootObject {
   a = true,
   b = 123,
   c = null,
   d = new object[] { "abc" }
});

// Get root DOM direct access interface.
var dom = document.root.dom;

// Ensure element 'a' is loaded as 'true'.
Test.assertTrue(dom.a);

// Modify the value of 'a' right in place.
dom.a = false;

// Ensure element 'b' is loaded as number 123.
```

```
Test.assertEqual(dom.b, 123.0);

// Increase 'b' by 2 right in place and verify result.
dom.b += 2;
Test.assertEqual(dom.b, 125.0);

// Make element 'c' a new object right from a .NET object.
dom.c = new SomeObject {
   c1 = 1,
   c2 = 2
};

// Modify the first element of 'd' right in place.
dom.d[0] = "ABC";

// Add new elements to array 'd' right from .NET values and objects.
var d = dom.d._;
d.nodes.add(321);
d.nodes.add(new object[] { "XYZ", null });

Test.message(document.json);

// Output:
//
// {"d":["ABC",321,["XYZ",null]],"a":false,"b":125,"c":{"c1":1,"c2":2}}
```

3.11.3 Visual Basic

```
' Create JSON document object.
Dim document As JsonDocument
Set document = New JsonDocument

Dim nodes As IJsonNodeCollection

' Create a document model object.
Dim model As SomeDocumentModel
Set model = New SomeDocumentModel
model.a = True
model.b = 123
Set model.c = New EmptyDocumentModel
model.d = Array("abc")

' Load document from a document model object.
document.Load model

' Remove unused nodes.
Set nodes = document.root.nodes
nodes.Remove "e"
nodes.Remove "f"
nodes.Remove "g"
nodes.Remove "h"

' Get root DOM direct access interface.
Dim dom
Set dom = document.root.dom
```

```
' Ensure element 'a' is loaded as 'true'.
Test.assertTrue dom.a

' Modify the value of 'a' right in place.
dom.a = False

' Ensure element 'b' is loaded as number 123.
Test.assertEqual dom.b, 123

' Increase 'b' by 2 right in place and verify result.
dom.b = dom.b + 2
Test.assertEqual dom.b, 125

' Make element 'c' a new object right from a document model object.
Set dom.c = New SomeDocumentModel
Set nodes = document.root.nodes("c").nodes
nodes("a").dom = 1
nodes("b").dom = 2

' Remove unused nodes.
nodes.Remove "c"
nodes.Remove "d"
nodes.Remove "e"
nodes.Remove "f"
nodes.Remove "g"
nodes.Remove "h"
```

```
' Modify the first element of 'd' right in place.
dom.d(0) = "ABC"

' Add new elements to array 'd' right from JavaScript values and objects.
Set nodes = document.root.nodes("d").nodes
nodes.Add 321
nodes.Add Array("XYZ", Null)

Test.message document.json

' Output:
'
' {"d":["ABC",321,["XYZ",null]],"a":false,"b":125,"c":{"a":1,"b":2}}
```

3.12 Use case: HTTP GET

This example demonstrates how to load JSON data from HTTP response.

3.12.1 JScript

```
// Create JSON document object.
var document = Jess.createDocument();

// Load document from HTTP response.
var status = document.load('http://postman-echo.com/get');
Test.assertTrue(status.success);

Test.message(document);
```

```
// Output:
//
// {"args":{},"headers":{"host":"postman-echo.com","x-forwarded-proto":"http","x-forwarded-port":"80","x-amzn-trace-id":"Root=1-603a958a-4452fd9f6e244e043dd26fc3","accept":"*/*","user-agent":"Mozilla/4.0 (compatible; Win32; WinHttp.WinHttpRequest.5)"},"url":"http://postman-echo.com/get"}
```

3.12.2 C#

```
// Create JSON document object.
var document = Jess.createDocument();

// Load document from HTTP response.
var status = document.load("http://postman-echo.com/get");
Test.assertTrue(status.success);

Test.message(document);

// Output:
//
// {"args":{},"headers":{"host":"postman-echo.com","x-forwarded-proto":"http","x-forwarded-port":"80","x-amzn-trace-id":"Root=1-603a958a-4452fd9f6e244e043dd26fc3","accept":"*/*","user-agent":"Mozilla/4.0 (compatible; Win32; WinHttp.WinHttpRequest.5)"},"url":"http://postman-echo.com/get"}
```

3.12.3 Visual Basic

```
' Create JSON document object.
Dim document As JsonDocument
Set document = New JsonDocument
```

```
' Load document from HTTP response.
Dim status As IJsonStatus
Set status = document.Load("http://postman-echo.com/get")
Test.assertTrue status.success

Test.message document.json

' Output:
'
' {"args":{},"headers":{"host":"postman-echo.com","x-forwarded-proto":"http","x-forwarded-
port":"80","x-amzn-trace-id":"Root=1-603a958a-4452fd9f6e244e043dd26fc3","accept":"*/*","user-
agent":"Mozilla/4.0  compatible Win32
WinHttp.WinHttpRequest.5"},"url":"http://postman-echo.com/get"}
```

3.13 Use case: HTTP PUT

This example demonstrates how to push JSON to HTTP PUT endpoint with specifying HTTP method, query data, form data, custom HTTP headers, TCP timeouts and then retrieving HTTP response status code, headers and parse response body as a new JSON document.

3.13.1 JScript

```
// Create JSON document object.
var document = Jess.createDocument();

// Load JSON data from a JavaScript object.
document.load({
  a: true,
  b: 123,
    c: {},
    d: [ 'abc' ]
});

// Save JSON document to the specified endpoint as HTTP PUT request
// that is encoded in UTF-8.
var status = document.save('http://postman-echo.com/put', 'utf-8', {
    // Specify HTTP method explicitly.
    method: 'PUT',

    // Add custom HTTP query parameters.
    query: {
        a: "#a",
        b: "#b",
        c: "#c"
    },

    // Add custom HTTP form data parameters.
    form: {
        d: "#d",
        e: "#e",
        f: "#f"
    },

    // Add custom HTTP headers.
    headers: {
        jess1: '#1',
        jess2: '#2',
        jess3: '#3'
    },

    // Override default TCP timeouts.
    timeouts: {
        resolve: 5000,
        connect: 5000,
        send: 5000,
        receive: 5000
    },

    // Require response JSON document to contains HTTP response status code,
    // HTTP response headers and HTTP response body nested as JSON.
    response: {
        status: true,
        headers: true,
        body: 'json'
    }
});

// Ensure the status of the operation indicates success.
Test.assertTrue(status.success);

Test.message(status.response.json);

// Output:
//
// {"status":{"code":200,"text":"OK"},"headers":{"Connection":"keep-alive","Vary":"Accept-
Encoding","Date":"Sun, 3 Apr 2021 11:20:07 GMT","Content-Length":"678","Set-Cookie":"sails.sid=s
```

%3Af3CRn9038HjgWzoRkxU0RxCHjjk_jUTs.MKfuT5LpKQQ7%2FQDoEz2XC3Lb2iehBACZ9SNI1%2BG21f0; Path=/; HttpOnly","Content-Type":"application/json; charset=utf-8","ETag":"W/\"2a6-R7s/F4SEKFJV+ODN/f54jLqZmwg\""},"body":{"args":{"a":"#a","b":"#b","c":"#c"},"data":{},"json":null,"files":{"data.json":"data:application/octet-stream;base64,eyJhIjp0cnVlLCJiIjoxMjMsImMiOnt9LCJkIjpbImFiYyJdfQ=="},"form":{"d":"#d","e":"#e","f":"#f"},"headers":{"host":"postman-echo.com","x-forwarded-proto":"http","x-forwarded-port":"80","jess2":"#2","content-length":"764","x-amzn-trace-id":"Root=1-600dac47-7ac67cc51fad912a38ae20a5","content-type":"multipart/form-data; boundary=8X5N71r9cOPolXTJuZ0YwKnW2EAnrnbP8IJIBOGnzIrgTLumu47bs9hBltergt0o","accept":"/*","user-agent":"Mozilla/4.0 (compatible; Win32; WinHttp.WinHttpRequest.5)","jess1":"#1","jess3":"#3"},"url":"http://postman-echo.com/put?a=%23a&b=%23b&c=%23c"}}*

3.13.2 C#

```
[ComVisible(true)]
[ClassInterface(ClassInterfaceType.AutoDual)]
public class HttpQueryParams {
   public object a { get; set; }
   public object b { get; set; }
   public object c { get; set; }
}

[ComVisible(true)]
[ClassInterface(ClassInterfaceType.AutoDual)]
public class HttpFormFields {
   public object d { get; set; }
   public object e { get; set; }
   public object f { get; set; }
}
```

```
[ComVisible(true)]
[ClassInterface(ClassInterfaceType.AutoDual)]
public class HttpHeaders {
   public object jess1 { get; set; }
   public object jess2 { get; set; }
   public object jess3 { get; set; }
}

[ComVisible(true)]
[ClassInterface(ClassInterfaceType.AutoDual)]
public class HttpTimeouts {
   public int resolve { get; set; }
   public int connect { get; set; }
   public int send { get; set; }
   public int receive { get; set; }
}

[ComVisible(true)]
[ClassInterface(ClassInterfaceType.AutoDual)]
public class HttpResponse {
   public bool status { get; set; }
   public bool headers { get; set; }
   public string body { get; set; }
}

[ComVisible(true)]
[ClassInterface(ClassInterfaceType.AutoDual)]
public class HttpRequest {
        public string method { get; set; }
        public HttpQueryParams query { get; set; }
        public HttpFormFields form { get; set; }
        public HttpHeaders headers { get; set; }
        public HttpTimeouts timeouts { get; set; }
        public HttpResponse response { get; set; }
    }
...

    // Create JSON document object.
    var document = Jess.createDocument();

    // Load JSON data from a .NET object.
    document.load(new RootObject {
        a = true,
        b = 123,
        c = { },
        d = new object[] { "abc" }
    });

    // Save JSON document to the specified endpoint as HTTP PUT request
    // that is encoded in UTF-8.
    var status = document.save("http://postman-echo.com/put", "utf-8", new HttpRequest {
        // Specify HTTP method explicitly.
        method = "PUT",

        // Add custom HTTP query parameters.
        query = new HttpQueryParams {
            a = "#a",
    b = "#b",
    c = "#c"
},

// Add custom HTTP form data parameters.
form = new HttpFormFields {
    d = "#d",
    e = "#e",
    f = "#f"
},

// Add custom HTTP headers.
headers = new HttpHeaders {
    jess1 = "#1",
    jess2 = "#2",
    jess3 = "#3"
},

// Override default TCP timeouts.
timeouts = new HttpTimeouts {
    resolve = 5000,
    connect = 5000,
    send = 5000,
    receive = 5000
},

// Require response JSON document to contains HTTP response status code,
// HTTP response headers and HTTP response body nested as JSON.
```

```
    response = new HttpResponse {
        status = true,
        headers = true,
        body = "json"
    }
});

// Ensure the status of the operation indicates success.
Test.assertTrue(status.success);

Test.message(status.response.json);

// Output:
//
// {"status":{"code":200,"text":"OK"},"headers":{"Connection":"keep-alive","Vary":"Accept-
Encoding","Date":"Sat, 03 Apr 2021 15:30:10 GMT","Content-Length":"678","Set-Cookie":"sails.sid=s
%3Asf-66WiqcyiuZ_2SJ17gx-3b5YuCwTL2.ypMc3CENuc0iKnH6h0wgzPFAtnwsSRZZpcey6b53cKk; Path=/;
HttpOnly","Content-Type":"application/json; charset=utf-8","ETag":"W/\"2a6-
zRlOfd1DNMYwlzZ5JE5I7etigBc\""},"body":{"args":{"a":"#a","b":"#b","c":"#c"},"data":
{},"json":null,"files":{"data.json":"data:application/octet-
stream;base64,eyJhIjp0cnVlLCJiIjoxMjMsImMiOm51bGwsImQiOlsiYWJjIl19"},"form":
{"d":"#d","e":"#e","f":"#f"},"headers":{"host":"postman-echo.com","x-forwarded-proto":"http","x-
forwarded-port":"80","jess2":"#2","content-length":"766","x-amzn-trace-id":"Root=1-60688a02-
5e3d1c3d7f9d9d3c0682ba22","jess1":"#1","content-type":"multipart/form-data;
boundary=kjKqsN68qM1S3pZ4JXMyqBuRgaJemafIyM23mkjRF7bHEewYemXQhyHK5z8UKE2n","accept":"*/
*","user-agent":"Mozilla/4.0 (compatible; Win32;
WinHttp.WinHttpRequest.5)","jess3":"#3"},"url":"http://postman-echo.com/put?a=%23a&b=%23b&c=
%23c"}}
```

3.13.3 Visual Basic

```
' Create JSON document object.
```

```
Dim document As JsonDocument
Set document = New JsonDocument

' Create and fill a new document model object.
Dim model As SomeDocumentModel
Set model = New SomeDocumentModel
model.a = True
model.b = 123
Set model.c = New EmptyDocumentModel
model.d = Array("abc")

' Load JSON data from a document model object.
document.Load model

' Create and fill a new HTTP request object.
Dim request As HttpRequest
Set request = New HttpRequest

' Specify HTTP method explicitly.
request.method = "PUT"

' Add custom HTTP query parameters.
Set request.query = New SomeDocumentModel
request.query.a = "#a"
request.query.b = "#b"
request.query.c = "#c"

' Add custom HTTP form data parameters.
```

```
Set request.Form = New SomeDocumentModel
request.Form.d = "#d"
request.Form.e = "#e"
request.Form.f = "#f"

' Add custom HTTP headers.
Set request.headers = New SomeDocumentModel
request.headers.a = "#1"
request.headers.b = "#2"
request.headers.c = "#3"

' Override default TCP timeouts.
Set request.timeouts = New HttpTimeouts
request.timeouts.resolve = 5000
request.timeouts.connect = 5000
request.timeouts.send = 5000
request.timeouts.receive = 5000

' Require response JSON document to contains HTTP response status code,
' HTTP response headers and HTTP response body nested as JSON.
Set request.response = New HttpResponseInfo
request.response.status = True
request.response.headers = True
request.response.body = "json"

' Save JSON document to the specified endpoint as HTTP PUT request
' that is encoded in UTF-8.
Dim status As IJsonStatus
```

```
Set status = document.save("http://postman-echo.com/put", "utf-8", request)

' Ensure the status of the operation indicates success.
Test.assertTrue status.success

Test.message status.response.json

' Output:
'
' {"status":{"code":200,"text":"OK"},"headers":{"Connection":"keep-alive","Vary":"Accept-Encoding","Date":"Sat, 03 Apr 2021 15:34:49 GMT","Content-Length":"617","Set-Cookie":"sails.sid=s%3Ahg3npKxy0FExUULz4-_s0IRGYzu_gmhK.NycNtiubkhAUGIIRaiMyXbhPvZoJlLMPi8krQy8%2Fbbo; Path=/; HttpOnly","Content-Type":"application/json; charset=utf-8",
' "ETag":"W/\"269-Vmwoxk1eelLslNkN/q7JBQGZx3E\""},"body":{"args":{"a":"#a","b":"#b","c":"#c"},"data":{"a":false,"e":null,"b":123,"f":null,"c":{},"g":null,"d":["abc"],"h":null},"json":{"a":false,"e":null,"b":123,"f":null,"c":{},"g":null,"d":["abc"],"h":null},"files":{},"form":{},"headers":{"host":"postman-echo.com",
' "x-forwarded-proto":"http","x-forwarded-port":"80","content-length":"73","x-amzn-trace-id":"Root=1-60688b19-4ac615b46defe08c78e72f33","content-type":"application/json; charset=\"utf-8\"",
' "accept":"*/*","user-agent":"Mozilla/4.0 (compatible; Win32; WinHttp.WinHttpRequest.5)","a":"#1","b":"#2","c":"#3"},"url":"http://postman-echo.com/put?a=%23a&b=%23b&c=%23c"}}
```

3.14 Use case: Schema validation

This example demonstrates how add a JSON schema to schema collection and perform schema validation.

3.14.1 JScript

```
// Create schema JSON document object.
var schemaDoc = Jess.createDocument();
```

```
// Load JSON schema that requires a node to be an array of numeric values.
schemaDoc.load({
   '$id': 'json:numeric_array',
   type: 'array',
   items: {
      type: 'number'
   }
});

// Create schema collection and add the schema document to it.
var schemas = Jess.createSchemas();
var schema = schemas.add(schemaDoc, 'json:numeric_array');

// Create JSON document object.
var instanceDoc = Jess.createDocument();

// Load JSON, an array of numeric values that is expected to
// satisfy schema requirements.
instanceDoc.load([ 0, 1, 2 ]);

// Validate JSON instance document against the added schema.
var status = schema.validate(instanceDoc);

// Ensure the validation passed successfully.
Test.assertTrue(status.success);

// Change the second element in the array to a string value.
```

```
instanceDoc.root.dom[1] = 'abc';

// Validate JSON instance document against the added schema.
status = schema.validate(instanceDoc);

// Ensure the validation failed.
Test.assertFalse(status.success);

// Ensure the reason of the failure is that a changed
// element type is prohibited by the schema.
Test.assertEqual(status.response, 'Value type doesn\'t meet requirements');

// Ensure the schema node for which validation has
// failed is the one that requires a numeric type.
Test.assertEqual(status.schema.json, '"number"');

// Ensure the instance node that has caused the validation
// failure is not a numeric value.
Test.assertEqual(status.instance.json, '"abc"');
```

3.14.2 C#

```
// Create schema JSON document object.
var schemaDoc = Jess.createDocument();

// Load JSON schema that requires a node to be an array of numeric values.
schemaDoc.parse("{\"$id\": \"json:numeric_array\", \"type\": \"array\", \"items\":
{ \"type\": \"number\" }}");
```

```
// Create schema collection and add the schema document to it.
var schemas = Jess.createSchemas();
var schema = schemas.add(schemaDoc, "json:numeric_array");

// Create JSON document object.
var instanceDoc = Jess.createDocument();

// Load JSON, an array of numeric values that is expected to
// satisfy schema requirements.
instanceDoc.load(new double[] { 0, 1, 2 });

// Validate JSON instance document against the added schema.
var status = schema.validate(instanceDoc);

// Ensure the validation passed successfully.
Test.assertTrue(status.success);

// Change the second element in the array to a string value.
instanceDoc.root.dom[1] = "abc";

// Validate JSON instance document against the added schema.
status = schema.validate(instanceDoc);

// Ensure the validation failed.
Test.assertFalse(status.success);

// Ensure the reason of the failure is that a changed
// element type is prohibited by the schema.
Test.assertEqual(status.response, "Value type doesn't meet requirements");

// Ensure the schema node for which validation has
// failed is the one that requires a numeric type.
Test.assertEqual(status.schema.json, "\"number\"");

// Ensure the instance node that has caused the validation
// failure is not a numeric value.
Test.assertEqual(status.instance.json, "\"abc\"");
```

3.14.3 Visual Basic

```
' Create schema JSON document object.
Dim schemaDoc As JsonDocument
Set schemaDoc = New JsonDocument

' Load JSON schema that requires a node to be an array of numeric values.
schemaDoc.parse _
"{" + _
  """$id"": ""json:numeric_array""," + _
  """type"": ""array""," + _
  """items"": {" + _
    """type"": ""number""" + _
  "}" + _
"}"

' Create schema collection and add the schema document to it.
```

```
Dim schemas As JsonSchemas
Set schemas = New JsonSchemas
Dim schema As IJsonSchema
Set schema = schemas.Add(schemaDoc, "json:numeric_array")

' Create JSON document object.
Dim instanceDoc As JsonDocument
Set instanceDoc = New JsonDocument

' Load JSON, an array of numeric values that is expected to
' satisfy schema requirements.
instanceDoc.Load Array(0, 1, 2)

' Validate JSON instance document against the added schema.
Dim status As IJsonStatus
Set status = schema.Validate(instanceDoc)

' Ensure the validation passed successfully.
Test.assertTrue status.success

' Change the second element in the array to a string value.
instanceDoc.root.dom(1) = "abc"

' Validate JSON instance document against the added schema.
Set status = schema.Validate(instanceDoc)

' Ensure the validation failed.
Test.assertFalse status.success
```

```
' Ensure the reason of the failure is that a changed
' element type is prohibited by the schema.
Test.assertEqual status.response, "Value type doesn't meet requirements"

' Ensure the schema node for which validation has
' failed is the one that requires a numeric type.
Test.assertEqual status.schema.json, """number"""

' Ensure the instance node that has caused the validation
' failure is not a numeric value.
Test.assertEqual status.instance.json, """abc"""
```

3.15 Use case: Schema request

This example demonstrates how to provide a schema on demand when validation process attempts to reference a schema that has not been previously added to the schema collection.

3.15.1 JScript

```
// Create schema JSON document object.
var schemaDoc = Jess.createDocument();

// Load JSON schema that does not contain any constrains and
// merely references some schema that is not added to the collection.
schemaDoc.load({ '$ref': 'json:numeric_array' });

// Create schema collection object.
var schemas = Jess.createSchemas();
```

```
// Set schema request callback function.
schemas.onRequestSchema(schemaRequestCallback);

// Add the test schema to the collection.
var schema = schemas.add(schemaDoc, 'uri:test');

// Create JSON document object.
var instanceDoc = Jess.createDocument();

// Load JSON, an array of numeric values.
instanceDoc.load([ 0, 1, 2 ]);

// Validate the test JSON against the added schema.
// The implementation will attempt to reference 'json:numeric_array'
// schema, which is not in the collection yet, so the
// schema request callback function will be executed
// and the schema it'll return, if any, will be added
// to the collection and used for further validation.
var status = schema.validate(instanceDoc);

// Ensure validation was successful.
Test.assertTrue(status.success);
```

3.15.2 C# (using callback)

```
// The function handles schema requests from a schema collection.
var handler = new SchemaHandler((all, uri) => {
    // If the requested schema is a numeric array schema then
    // proceed with loading and adding a new schema.
    if (uri == "json:numeric_array") {
        // Create new schema document object.
        var doc = Jess.createDocument();

        // Load schema JSON that requires a node
        // to be an array of numeric values.
        doc.parse("{\"$id\": \"json:numeric_array\",
            \"type\": \"array\", \"items\": { \"type\": \"number\" }}");

        // Add the new schema document to the schema collection
        // and return the result schema object.
        return all.add(doc, uri);
    }

    // For all unexpected schema request there is no schema object to return.
    return null;
});

// Create schema JSON document object.
var schemaDoc = Jess.createDocument();

// Load JSON schema that does not contain any constrains and
// merely references some schema that is not added to the collection.
schemaDoc.parse("{ \"$ref\": \"json:numeric_array\" }");

// Create schema collection object.
```

```
var schemas = Jess.createSchemas();

// Set schema request callback object.
schemas.onRequestSchema(handler);

// Add the test schema to the collection.
var schema = schemas.add(schemaDoc, "uri:test");

// Create JSON document object.
var instanceDoc = Jess.createDocument();

// Load JSON, an array of numeric values.
instanceDoc.parse("[0, 1, 2]");

// Validate the test JSON against the added schema.
// The implementation will attempt to reference 'json:numeric_array'
// schema, which is not in the collection yet, so the
// 'onRequestSchema' method of the schema request callback
// object will be executed and the schema it'll return, if any, will be added
// to the collection and used for further validation.
var status = schema.validate(instanceDoc);

// Ensure validation was successful.
Test.assertTrue(status.success);
```

SchemaHandler.cs:

```
public delegate IJsonSchema SchemaHandlerCallback(IJsonSchemas schemas, string uri);
```

```
[ComVisible(true)]
[ClassInterface(ClassInterfaceType.AutoDual)]
public class SchemaHandler {
  private Func<IJsonSchemas, string, IJsonSchema> callback_;

  public SchemaHandler(Func<IJsonSchemas, string, IJsonSchema> callback) {
    this.callback_ = callback;
  }

  [DispId(0)]
  public IJsonSchema onRequestSchema(IJsonSchemas schemas, string uri) {
    return this.callback_(schemas, uri);
  }
}
```

3.15.3 C# (using delegate)

```
// Create schema JSON document object.
var schemaDoc = Jess.createDocument();

// Load JSON schema that does not contain any constrains and
// merely references some schema that is not added to the collection.
schemaDoc.parse("{ \"$ref\": \"json:numeric_array\" }");

// Create schema collection object.
var schemas = Jess.createSchemas();

// Set schema request callback delegate.
```

```
schemas.onRequestSchema(new Func<IJsonSchemas, string, IJsonSchema>((all, uri) => {
   if (uri == "json:numeric_array") {
      var doc = Jess.createDocument();
      doc.parse("{\"$id\": \"json:numeric_array\", \"type\": \"array\", \"items\":
{ \"type\": \"number\" }}");
      return all.add(doc, uri);
   }

   return null;
}));

// Add the test schema to the collection.
var schema = schemas.add(schemaDoc, "uri:test");

// Create JSON document object.
var instanceDoc = Jess.createDocument();

// Load JSON, an array of numeric values.
instanceDoc.parse("[0, 1, 2]");

// Validate the test JSON against the added schema.
// The implementation will attempt to reference 'json:numeric_array'
// schema, which is not in the collection yet, so the
// schema request callback delegate will be executed
// and the schema it'll return, if any, will be added
// to the collection and used for further validation.
var status = schema.validate(instanceDoc);

// Ensure validation was successful.
```

```
Test.assertTrue(status.success);
```

3.15.4 Visual Basic

```
' Create schema JSON document object.
Dim schemaDoc As JsonDocument
Set schemaDoc = New JsonDocument

' Load JSON schema that does not contain any constrains and
' merely references some schema that is not added to the collection.
schemaDoc.parse "{ ""$ref"": ""json:numeric_array"" }"

' Create schema collection object.
Dim schemas As JsonSchemas
Set schemas = New JsonSchemas

' Set schema request callback object.
schemas.onRequestSchema New SchemaRequestHandler

' Add the test schema to the collection.
Dim schema As IJsonSchema
Set schema = schemas.Add(schemaDoc, "uri:test")

' Create JSON document object.
Dim instanceDoc As JsonDocument
Set instanceDoc = New JsonDocument

' Load JSON, an array of numeric values.
```

```
instanceDoc.Load Array(0, 1, 2)

' Validate the test JSON against the added schema.
' The implementation will attempt to reference 'json:numeric_array'
' schema, which is not in the collection yet, so the
' 'onRequestSchema' method of the schema request callback
' object will be executed and the schema it'll return, if any,
' will be added to the collection and used for further validation.
Dim status As IJsonStatus
Set status = schema.Validate(instanceDoc)

' Ensure validation was successful.
Test.assertTrue status.success
```

SchemaRequestHandler.cls:

```
Option Explicit
Option Base 0

' The function handles schema requests from a schema collection.
Public Function onRequestSchema(schemas As JsonSchemas, uri As String) As IJsonSchema

  ' If the requested schema is a numeric array schema then
  ' proceed with loading and adding a new schema.
  If uri = "json:numeric_array" Then

    ' Create new schema document object.
    Dim doc As JsonDocument
    Set doc = New JsonDocument
```

```
        ' Parse schema JSON that requires a node to be an array of numeric values.
        doc.parse _
        "{" + _
          """$id"": ""json:numeric_array""," + _
          """type"": ""array""," + _
          """items"": {" + _
            """type"": ""number""" + _
          "}" + _
        "}"

        ' Add the new schema document to the schema collection
        ' and return the result schema object.
        Set onRequestSchema = schemas.Add(doc, uri)
      Else
        ' For all unexpected schema request there is no schema object to return.
        onRequestSchema = Null
      End If
    End Function
```

3.16 Use case: Schema annotation

This example demonstrates how to annotate JSON instance nodes using a JSON schema

3.16.1 JScript

```
// Create JSON schema document object.
var schemaDoc = Jess.createDocument();
```

```
// Load JSON schema data that not only requires a node
// being validated to be a JSON numeric value but also
// provides additional metadata that describe
// characteristics of a node being verified.
schemaDoc.load({
   '$id': 'json:integer',
   type: 'integer',
   title: 'Test schema',
   description: 'This simple schema demonstrates how to annotate JSON document',
   examples: [0, 1, 2],
   readOnly: false,
   writeOnly: true,
   contentEncoding: 'utf-8',
   contentMediaType: 'plaint/text',
   'default': 100
});

// Create schema collection object and add the loaded schema to it.
var schemas = Jess.createSchemas();
var schema = schemas.add(schemaDoc, 'json:integer');

// Create JSON instance document and make it a single numeric value.
var instanceDoc = Jess.createDocument();
instanceDoc.load(123);

// Annotate the JSON instance document using the test schema.
var status = schema.annotate(instanceDoc);
```

```
// Ensure annotation was successful.
Test.assertTrue(status.success);

// As the JSON instance document is a numeric value expect its root
// node to have all schema defined annotation data attached to it
// as a result of successful annotation.
var root = instanceDoc.root;
Test.assertEqual(root.annotation.title.dom, 'Test schema');
Test.assertEqual(root.annotation.defaultNode.dom, 100);
Test.assertEqual(root.annotation.description.dom, 'This simple schema demonstrates how to
annotate JSON document');
Test.assertEqual(root.annotation.examples.dom[0], 0);
Test.assertEqual(root.annotation.examples.dom[1], 1);
Test.assertEqual(root.annotation.examples.dom[2], 2);
Test.assertFalse(root.annotation.readOnly.dom);
Test.assertTrue(root.annotation.writeOnly.dom);
Test.assertEqual(root.annotation.contentEncoding.dom, 'utf-8');
Test.assertEqual(root.annotation.contentMediaType.dom, 'plaint/text');
```

3.16.2 C#

```
// Create JSON schema document object.
var schemaDoc = Jess.createDocument();

// Load JSON schema data that not only requires a node
// being validated to be a JSON numeric value but also
// provides additional metadata that describe
// characteristics of a node being verified.
```

```
schemaDoc.parse(
  "{" +
  "   \"$id\": \"json:integer\"," +
  "   \"type\": \"integer\"," +
  "   \"title\": \"Test schema\"," +
  "   \"description\": \"This simple schema demonstrates how to annotate JSON document\"," +
  "   \"examples\": [0, 1, 2]," +
  "   \"readOnly\": false," +
  "   \"writeOnly\": true," +
  "   \"contentEncoding\": \"utf-8\"," +
  "   \"contentMediaType\": \"plaint/text\"," +
  "   \"default\": 100" +
  "}");

// Create schema collection object and add the loaded schema to it.
var schemas = Jess.createSchemas();
var schema = schemas.add(schemaDoc, "json:integer");

// Create JSON instance document and make it a single numeric value.
var instanceDoc = Jess.createDocument();
instanceDoc.load(123);

// Annotate the JSON instance document using the test schema.
var status = schema.annotate(instanceDoc);

// Ensure annotation was successful.
Test.assertTrue(status.success);
```

```
// As the JSON instance document is a numeric value expect its root
// node to have all schema defined annotation data attached to it
// as a result of successful annotation.
var root = instanceDoc.root;
Test.assertEqual(root.annotation.title.dom, "Test schema");
Test.assertEqual(root.annotation.defaultNode.dom, 100.0);
Test.assertEqual(root.annotation.description.dom, "This simple schema demonstrates how to
annotate JSON document");
Test.assertEqual(root.annotation.examples.dom[0], 0.0);
Test.assertEqual(root.annotation.examples.dom[1], 1.0);
Test.assertEqual(root.annotation.examples.dom[2], 2.0);
Test.assertFalse(root.annotation.readOnly.dom);
Test.assertTrue(root.annotation.writeOnly.dom);
Test.assertEqual(root.annotation.contentEncoding.dom, "utf-8");
Test.assertEqual(root.annotation.contentMediaType.dom, "plaint/text");
```

3.16.3 Visual Basic

```
' Create JSON schema document object.
Dim schemaDoc As JsonDocument
Set schemaDoc = New JsonDocument

' Parse JSON schema data that not only requires a node
' being validated to be a JSON numeric value but also
' provides additional metadata that describe
' characteristics of a node being verified.
schemaDoc.parse _
"{" + _
```

```
    """$id"": ""json:integer""," + _
    """type"": ""integer""," + _
    """title"": ""Test schema""," + _
    """description"": ""This simple schema demonstrates how to annotate JSON document""," + _
    """examples"": [0, 1, 2]," + _
    """readOnly"": false," + _
    """writeOnly"": true," + _
    """contentEncoding"": ""utf-8""," + _
    """contentMediaType"": ""plaint/text""," + _
    """default"": 100" + _
"}"

' Create schema collection object and add the loaded schema to it.
Dim schemas As JsonSchemas
Set schemas = New JsonSchemas
Dim schema As IJsonSchema
Set schema = schemas.Add(schemaDoc, "json:integer")

' Create JSON instance document and make it a single numeric value.
Dim instanceDoc As JsonDocument
Set instanceDoc = New JsonDocument
instanceDoc.Load 123

' Annotate the JSON instance document using the test schema.
Dim status As IJsonStatus
Set status = schema.annotate(instanceDoc)

' Ensure annotation was successful.
```

```
Test.assertTrue status.success

' As the JSON instance document is a numeric value expect its root
' node to have all schema defined annotation data attached to it
' as a result of successful annotation.
Dim root As IJsonNode
Set root = instanceDoc.root
Test.assertEqual root.annotation.Title.dom, "Test schema"
Test.assertEqual root.annotation.defaultNode.dom, 100
Test.assertEqual root.annotation.Description.dom, "This simple schema demonstrates how to annotate JSON document"
Test.assertEqual root.annotation.examples.dom(0), 0
Test.assertEqual root.annotation.examples.dom(1), 1
Test.assertEqual root.annotation.examples.dom(2), 2
Test.assertFalse root.annotation.ReadOnly.dom
Test.assertTrue root.annotation.writeOnly.dom
Test.assertEqual root.annotation.contentEncoding.dom, "utf-8"
Test.assertEqual root.annotation.contentMediaType.dom, "plaint/text"
```

4 IDL Listings

1. The Interface Definition Language (IDL) listing provided below can be used as a brief cumulative language independent API reference for each configuration (see Configurations).
2. The listings omit values of all uuid attributes as those contain a license and version specific information of each distributed copy of the library.
3. The library name in the listings does not included order ID as described in Classes.
4. Complete IDL listing for a specific copy of the library can be extracted using Microsoft OleViewer utility using View TypeLib menu and selecting the library module OCX file (see Deployment) or by looking up for a registered **JsonEssential** type library in the Type Libraries tree node.

4.1 DWS configuration

```
library JsonEssentials
{
   importlib("stdole2.tlb");

   // Forward declare all types defined in this typelib
   interface IJsonDocument;
   interface IJsonNode;
   interface IJsonNodeCollection;
   interface IJsonNodeAnnotation;
   interface IJsonStatus;
   interface IJsonSelector;
   interface IJsonFormat;
   interface IJsonTools;
   interface IJsonPolicy;
   interface IJsonEssentials;

   [
     odl,
     uuid(...),
     dual,
     nonextensible,
     oleautomation
   ]
   interface IJsonDocument : IDispatch {
      [id(00000000), propget]
      HRESULT json([out, retval] BSTR* json);
      [id(0x00000001), propget]
```

```
HRESULT root([out, retval] IJsonNode** root);
[id(0x00000002), propget]
HRESULT encoding([out, retval] BSTR* encoding);
[id(0x00000002), propput]
HRESULT encoding([in] BSTR encoding);
[id(0x00000003), propget]
HRESULT format([out, retval] IJsonFormat** format);
[id(0x00000004), propget]
HRESULT location([out, retval] BSTR* location);
[id(0x00000064)]
HRESULT makeEmptyObject([out, retval] IJsonNode** root);
[id(0x00000065)]
HRESULT makeEmptyArray([out, retval] IJsonNode** root);
[id(0x00000066)]
HRESULT makeValue(
        [in] VARIANT value,
        [out, retval] IJsonNode** root);
[id(0x00000067)]
HRESULT save(
        [in] VARIANT target,
        [in, optional] VARIANT encoding,
        [in, optional] VARIANT request,
        [in, optional] VARIANT writeBom,
        [out, retval] IJsonStatus** status);
[id(0x00000068)]
HRESULT load(
        [in] VARIANT source,
        [in, optional] VARIANT encoding,
                [in, optional] VARIANT request,
                [out, retval] IJsonStatus** status);
    [id(0x00000069)]
    HRESULT parse(
                [in] BSTR json,
                [out, retval] IJsonStatus** status);
};

[
  odl,
  uuid(...),
  dual,
  nonextensible,
  oleautomation
]
interface IJsonNode : IDispatch {
    [id(00000000), propget]
    HRESULT json([out, retval] BSTR* json);
    [id(0x00000001), propget]
    HRESULT document([out, retval] IJsonDocument** document);
    [id(0x00000002), propget]
    HRESULT parent([out, retval] IJsonNode** parent);
    [id(0x00000003), propget]
    HRESULT name([out, retval] BSTR* name);
    [id(0x00000004), propget]
    HRESULT isRoot([out, retval] VARIANT_BOOL* result);
    [id(0x00000005), propget]
    HRESULT isObject([out, retval] VARIANT_BOOL* result);
```

```
[id(0x00000006), propget]
HRESULT isArray([out, retval] VARIANT_BOOL* result);
[id(0x00000007), propget]
HRESULT isEmpty([out, retval] VARIANT_BOOL* result);
[id(0x00000008), propget]
HRESULT isNull([out, retval] VARIANT_BOOL* result);
[id(0x00000009), propget]
HRESULT isNumeric([out, retval] VARIANT_BOOL* result);
[id(0x0000000a), propget]
HRESULT isString([out, retval] VARIANT_BOOL* result);
[id(0x0000000b), propget]
HRESULT isBoolean([out, retval] VARIANT_BOOL* result);
[id(0x0000000c), propget]
HRESULT isContainer([out, retval] VARIANT_BOOL* result);
[id(0x0000000d), propget]
HRESULT hasName([out, retval] VARIANT_BOOL* result);
[id(0x0000000e), propget]
HRESULT kind([out, retval] JsonNodeKind* kind);
[id(0x0000000f), propget]
HRESULT nodes([out, retval] IJsonNodeCollection** nodes);
[id(0x00000010), propget]
HRESULT dom([out, retval] VARIANT* dom);
[id(0x00000010), propput]
HRESULT dom([in] VARIANT dom);
[id(0x00000011), propget]
HRESULT index([out, retval] long* index);
[id(0x00000012), propget]
HRESULT annotation([out, retval] IJsonNodeAnnotation** annotation);
```

```
    [id(0x00000064)]
    HRESULT equals(
                [in] VARIANT other,
                [in, optional] VARIANT noName,
                [out, retval] VARIANT_BOOL* result);
    [id(0x00000065)]
    HRESULT save(
                [in] VARIANT target,
                [in, optional] VARIANT encoding,
                [in, optional] VARIANT request,
                [in, optional] VARIANT writeBom,
                [out, retval] IJsonStatus** status);
    [id(0x00000068)]
    HRESULT select(
                [in] VARIANT selector,
                [out, retval] VARIANT* result);
    [id(0x00000069)]
    HRESULT createJsonPointer([out, retval] IJsonSelector** selector);
};

typedef [uuid(...)]
enum {
    jsonNodeObject = 1,
    jsonNodeArray = 2,
    jsonNodeNull = 3,
    jsonNodeNumeric = 4,
    jsonNodeString = 5,
    jsonNodeBoolean = 6
```

```
} JsonNodeKind;

[
  odl,
  uuid(...),
  dual,
  nonextensible,
  oleautomation
]
interface IJsonNodeCollection : IDispatch {
    [id(0xfffffffc), propget, restricted, hidden]
    HRESULT _newEnum([out, retval] IUnknown** punk);
    [id(00000000), propget]
    HRESULT item(
                [in] VARIANT key,
                [out, retval] IJsonNode** node);
    [id(0x00000002), propget]
    HRESULT length([out, retval] long* result);
    [id(0x00000064)]
    HRESULT add(
                [in] VARIANT value,
                [in, optional] VARIANT name,
                [out, retval] IJsonNode** newNode);
    [id(0x00000065)]
    HRESULT addObject(
                [in, optional] VARIANT name,
                [out, retval] IJsonNode** newNode);
    [id(0x00000066)]
    HRESULT addArray(
                [in, optional] VARIANT name,
                [out, retval] IJsonNode** newNode);
    [id(0x00000067)]
    HRESULT contains(
                [in] VARIANT key,
                [out, retval] VARIANT_BOOL* result);
    [id(0x00000068)]
    HRESULT remove([in] VARIANT key);
    [id(0x00000069)]
    HRESULT clear();
};

[
  odl,
  uuid(...),
  dual,
  nonextensible,
  oleautomation
]
interface IJsonNodeAnnotation : IDispatch {
    [id(0x00000001), propget]
    HRESULT title([out, retval] IJsonNode** node);
    [id(0x00000002), propget]
    HRESULT description([out, retval] IJsonNode** node);
    [id(0x00000003), propget]
    HRESULT contentEncoding([out, retval] IJsonNode** node);
    [id(0x00000004), propget]
    HRESULT contentMediaType([out, retval] IJsonNode** node);
    [id(0x00000005), propget]
    HRESULT readOnly([out, retval] IJsonNode** node);
    [id(0x00000006), propget]
    HRESULT writeOnly([out, retval] IJsonNode** node);
    [id(0x00000007), propget]
    HRESULT defaultNode([out, retval] IJsonNode** node);
    [id(0x00000008), propget]
    HRESULT examples([out, retval] IJsonNode** node);
};

[
  odl,
  uuid(...),
  dual,
  nonextensible,
  oleautomation
]
interface IJsonStatus : IDispatch {
    [id(00000000), propget]
    HRESULT success([out, retval] VARIANT_BOOL* success);
    [id(0x00000001), propget]
    HRESULT kind([out, retval] JsonStatusKind* kind);
    [id(0x00000002), propget]
    HRESULT code([out, retval] long* code);
    [id(0x00000003), propget]
    HRESULT response([out, retval] VARIANT* response);
    [id(0x00000004), propget]
    HRESULT line([out, retval] long* line);
    [id(0x00000005), propget]
    HRESULT column([out, retval] long* column);
};

typedef [uuid(...)]
enum {
    jsonStatusUnknown = 0,
    jsonStatusJess = 1,
    jsonStatusSystem = 2,
    jsonStatusProvider = 3
} JsonStatusKind;

[
  odl,
  uuid(...),
  dual,
  nonextensible,
  oleautomation
]
interface IJsonSelector : IDispatch {
    [id(00000000), propget]
    HRESULT expression([out, retval] BSTR* expression);
    [id(0x00000064)]
    HRESULT select(
                    [in] IDispatch* from,
                    [out, retval] VARIANT* result);
};
```

```
[
 odl,
 uuid(...),
 dual,
 nonextensible,
 oleautomation
]
interface IJsonFormat : IDispatch {
   [id(0x00000001), propget]
   HRESULT indentationSize([out, retval] long* value);
   [id(0x00000001), propput]
   HRESULT indentationSize([in] long value);
   [id(0x00000002), propget]
   HRESULT useDosNewLines([out, retval] VARIANT_BOOL* value);
   [id(0x00000002), propput]
   HRESULT useDosNewLines([in] VARIANT_BOOL value);
   [id(0x00000003), propget]
   HRESULT useTabs([out, retval] VARIANT_BOOL* value);
   [id(0x00000003), propput]
   HRESULT useTabs([in] VARIANT_BOOL value);
   [id(0x00000004), propget]
   HRESULT placeBracketsOnNewLine([out, retval] VARIANT_BOOL* value);
   [id(0x00000004), propput]
   HRESULT placeBracketsOnNewLine([in] VARIANT_BOOL value);
   [id(0x00000005), propget]
   HRESULT placeValueOnNewLine([out, retval] VARIANT_BOOL* value);
   [id(0x00000005), propput]
```

```
HRESULT placeValueOnNewLine([in] VARIANT_BOOL value);
[id(0x00000006), propget]
HRESULT addNewLineInEmptyBrackets([out, retval] VARIANT_BOOL* value);
[id(0x00000006), propput]
HRESULT addNewLineInEmptyBrackets([in] VARIANT_BOOL value);
[id(0x00000007), propget]
HRESULT addSpaceBeforeColon([out, retval] VARIANT_BOOL* value);
[id(0x00000007), propput]
HRESULT addSpaceBeforeColon([in] VARIANT_BOOL value);
[id(0x00000008), propget]
HRESULT addSpaceAfterColon([out, retval] VARIANT_BOOL* value);
[id(0x00000008), propput]
HRESULT addSpaceAfterColon([in] VARIANT_BOOL value);
[id(0x00000009), propget]
HRESULT addSpaceInBrackets([out, retval] VARIANT_BOOL* value);
[id(0x00000009), propput]
HRESULT addSpaceInBrackets([in] VARIANT_BOOL value);
[id(0x0000000a), propget]
HRESULT addSpaceAfterComma([out, retval] VARIANT_BOOL* value);
[id(0x0000000a), propput]
HRESULT addSpaceAfterComma([in] VARIANT_BOOL value);
[id(0x0000000b), propget]
HRESULT forceEscapeNonAscii([out, retval] VARIANT_BOOL* value);
[id(0x0000000b), propput]
HRESULT forceEscapeNonAscii([in] VARIANT_BOOL value);
[id(0x0000000c), propget]
HRESULT forceEscapeNonEascii([out, retval] VARIANT_BOOL* value);
[id(0x0000000c), propput]
```

```
    HRESULT forceEscapeNonEascii([in] VARIANT_BOOL value);
};

typedef [uuid(...)]
enum {
    jsonErrorSuccess = 0,
    jsonErrorCommonFirst = 256,
    jsonErrorCommonInvalidArgument = 256,
    jsonErrorCommonInvalidObjectState = 257,
    jsonErrorCommonUnsupportedByObjectKind = 258,
    jsonErrorCommonIndexOutOfBoundary = 259,
    jsonErrorCommonInvalidEncodingSequence = 260,
    jsonErrorCommonBadDereferencing = 261,
    jsonErrorCommonNoMemoryAllocated = 262,
    jsonErrorCommonLast = 262,
    jsonErrorParserFirst = 4096,
    jsonErrorParserUnexpectedEndOfData = 4096,
    jsonErrorParserUnexpectedToken = 4097,
    jsonErrorParserInvalidData = 4098,
    jsonErrorParserInvalidEscapeSequence = 4099,
    jsonErrorParserLast = 4099,
    jsonErrorDomFirst = 12288,
    jsonErrorDomCannotBeAddedToObject = 12288,
    jsonErrorDomCannotBeAddedToArray = 12289,
    jsonErrorDomNoReadAccess = 12290,
    jsonErrorDomNoWriteAccess = 12291,
    jsonErrorDomNodeNotFound = 12292,
    jsonErrorDomAlreadyAtRoot = 12293,
    jsonErrorDomNodeHasNoName = 12294,
    jsonErrorDomNoAssociatedNode = 12295,
    jsonErrorDomForeignNode = 12296,
    jsonErrorDomLast = 12296
} JsonError;

[
  uuid(...),
  helpstring("JSON DOM Document"),
  licensed
]
coclass JsonDocument {
    [default] interface IJsonDocument;
};

[
  odl,
  uuid(...),
  dual,
  nonextensible,
  oleautomation
]
interface IJsonTools : IDispatch {
    [id(0x00000001), propget]
    HRESULT policy([out, retval] IJsonPolicy** policy);
    [id(0x0000006a)]
    HRESULT createJsonPointer(
                [in] BSTR pointer,
                [out, retval] IJsonSelector** selector);
    [id(0x0000006b)]
    HRESULT createRelativeJsonPointer(
                [in] BSTR pointer,
                [out, retval] IJsonSelector** selector);
};

[
  odl,
  uuid(...),
  dual,
  nonextensible,
  oleautomation
]
interface IJsonPolicy : IDispatch {
    [id(0x00000001), propget]
    HRESULT nullValue([out, retval] VARIANT* nullValue);
    [id(0x00000001), propput]
    HRESULT nullValue([in] VARIANT nullValue);
    [id(0x00000002), propget]
    HRESULT ddaEscape([out, retval] BSTR* key);
    [id(0x00000002), propput]
    HRESULT ddaEscape([in] BSTR key);
};

[
  uuid(...),
  helpstring("JSON Tools"),
  licensed
]
coclass JsonTools {
    [default] interface IJsonTools;
};

[
  odl,
  uuid(...),
  dual,
  nonextensible,
  oleautomation
]
interface IJsonEssentials : IDispatch {
    [id(0x00000001), propget]
    HRESULT id([out, retval] long* id);
    [id(0x00000002), propget]
    HRESULT versionMajor([out, retval] long* major);
    [id(0x00000003), propget]
    HRESULT versionMinor([out, retval] long* minor);
    [id(0x00000004), propget]
    HRESULT versionPatch([out, retval] long* patch);
    [id(0x00000005), propget]
    HRESULT licenseText([out, retval] BSTR* license);
    [id(0x00000064)]
    HRESULT unlock(
                    [in] BSTR licenseToken,
                    [out, retval] VARIANT_BOOL* unlocked);
```

```
        [id(0x00000065), propget]
        HRESULT tools([out, retval] IJsonTools** tools);
        [id(0x00000066)]
        HRESULT createDocument([out, retval] IJsonDocument** document);
    };

    [
      uuid(...),
      helpstring("JSON Essentials library facade"),
      licensed
    ]
    coclass JsonEssentials {
        [default] interface IJsonEssentials;
    };
}
```

4.2 SWS configuration

```
library JsonEssentials
    // TLib :     // TLib : OLE Automation : {...}
    importlib("stdole2.tlb");

    // Forward declare all types defined in this typelib
    interface IJsonDocument;
    interface IJsonNode;
    interface IJsonNodeCollection;
    interface IJsonNodeAnnotation;
    interface IJsonStatus;
interface IJsonSelector;
interface IJsonFormat;
interface IJsonTools;
interface IJsonPolicy;
interface IJsonSchemas;
interface IJsonSchema;
interface IJsonEssentials;

[
 odl,
 uuid(...),
 dual,
 nonextensible,
 oleautomation
]
interface IJsonDocument : IDispatch {
   [id(00000000), propget]
   HRESULT json([out, retval] BSTR* json);
   [id(0x00000001), propget]
   HRESULT root([out, retval] IJsonNode** root);
   [id(0x00000002), propget]
   HRESULT encoding([out, retval] BSTR* encoding);
   [id(0x00000002), propput]
   HRESULT encoding([in] BSTR encoding);
   [id(0x00000003), propget]
   HRESULT format([out, retval] IJsonFormat** format);
   [id(0x00000004), propget]
   HRESULT location([out, retval] BSTR* location);
```

```
    [id(0x00000064)]
    HRESULT makeEmptyObject([out, retval] IJsonNode** root);
    [id(0x00000065)]
    HRESULT makeEmptyArray([out, retval] IJsonNode** root);
    [id(0x00000066)]
    HRESULT makeValue(
            [in] VARIANT value,
            [out, retval] IJsonNode** root);
    [id(0x00000067)]
    HRESULT save(
            [in] VARIANT target,
            [in, optional] VARIANT encoding,
            [in, optional] VARIANT request,
            [in, optional] VARIANT writeBom,
            [out, retval] IJsonStatus** status);
    [id(0x00000068)]
    HRESULT load(
            [in] VARIANT source,
            [in, optional] VARIANT encoding,
            [in, optional] VARIANT request,
            [out, retval] IJsonStatus** status);
    [id(0x00000069)]
    HRESULT parse(
            [in] BSTR json,
            [out, retval] IJsonStatus** status);
};

[
   odl,
   uuid(...),
   dual,
   nonextensible,
   oleautomation
]
interface IJsonNode : IDispatch {
    [id(00000000), propget]
    HRESULT json([out, retval] BSTR* json);
    [id(0x00000001), propget]
    HRESULT document([out, retval] IJsonDocument** document);
    [id(0x00000002), propget]
    HRESULT parent([out, retval] IJsonNode** parent);
    [id(0x00000003), propget]
    HRESULT name([out, retval] BSTR* name);
    [id(0x00000004), propget]
    HRESULT isRoot([out, retval] VARIANT_BOOL* result);
    [id(0x00000005), propget]
    HRESULT isObject([out, retval] VARIANT_BOOL* result);
    [id(0x00000006), propget]
    HRESULT isArray([out, retval] VARIANT_BOOL* result);
    [id(0x00000007), propget]
    HRESULT isEmpty([out, retval] VARIANT_BOOL* result);
    [id(0x00000008), propget]
    HRESULT isNull([out, retval] VARIANT_BOOL* result);
    [id(0x00000009), propget]
    HRESULT isNumeric([out, retval] VARIANT_BOOL* result);
    [id(0x0000000a), propget]
```

```
HRESULT isString([out, retval] VARIANT_BOOL* result);
[id(0x0000000b), propget]
HRESULT isBoolean([out, retval] VARIANT_BOOL* result);
[id(0x0000000c), propget]
HRESULT isContainer([out, retval] VARIANT_BOOL* result);
[id(0x0000000d), propget]
HRESULT hasName([out, retval] VARIANT_BOOL* result);
[id(0x0000000e), propget]
HRESULT kind([out, retval] JsonNodeKind* kind);
[id(0x0000000f), propget]
HRESULT nodes([out, retval] IJsonNodeCollection** nodes);
[id(0x00000010), propget]
HRESULT dom([out, retval] VARIANT* dom);
[id(0x00000010), propput]
HRESULT dom([in] VARIANT dom);
[id(0x00000011), propget]
HRESULT index([out, retval] long* index);
[id(0x00000012), propget]
HRESULT annotation([out, retval] IJsonNodeAnnotation** annotation);
[id(0x00000064)]
HRESULT equals(
        [in] VARIANT other,
        [in, optional] VARIANT noName,
        [out, retval] VARIANT_BOOL* result);
[id(0x00000065)]
HRESULT save(
        [in] VARIANT target,
        [in, optional] VARIANT encoding,
                [in, optional] VARIANT request,
                [in, optional] VARIANT writeBom,
                [out, retval] IJsonStatus** status);
    [id(0x00000068)]
    HRESULT select(
                [in] VARIANT selector,
                [out, retval] VARIANT* result);
    [id(0x00000069)]
    HRESULT createJsonPointer([out, retval] IJsonSelector** selector);
};

typedef [uuid(...)]
enum {
    jsonNodeObject = 1,
    jsonNodeArray = 2,
    jsonNodeNull = 3,
    jsonNodeNumeric = 4,
    jsonNodeString = 5,
    jsonNodeBoolean = 6
} JsonNodeKind;

[
  odl,
  uuid(...),
  dual,
  nonextensible,
  oleautomation
]
```

```
interface IJsonNodeCollection : IDispatch {
   [id(0xfffffffc), propget, restricted, hidden]
   HRESULT _newEnum([out, retval] IUnknown** punk);
   [id(00000000), propget]
   HRESULT item(
               [in] VARIANT key,
               [out, retval] IJsonNode** node);
   [id(0x00000002), propget]
   HRESULT length([out, retval] long* result);
   [id(0x00000064)]
   HRESULT add(
               [in] VARIANT value,
               [in, optional] VARIANT name,
               [out, retval] IJsonNode** newNode);
   [id(0x00000065)]
   HRESULT addObject(
               [in, optional] VARIANT name,
               [out, retval] IJsonNode** newNode);
   [id(0x00000066)]
   HRESULT addArray(
               [in, optional] VARIANT name,
               [out, retval] IJsonNode** newNode);
   [id(0x00000067)]
   HRESULT contains(
               [in] VARIANT key,
               [out, retval] VARIANT_BOOL* result);
   [id(0x00000068)]
   HRESULT remove([in] VARIANT key);
```

```
    [id(0x00000069)]
    HRESULT clear();
};

[
  odl,
  uuid(...),
  dual,
  nonextensible,
  oleautomation
]
interface IJsonNodeAnnotation : IDispatch {
    [id(0x00000001), propget]
    HRESULT title([out, retval] IJsonNode** node);
    [id(0x00000002), propget]
    HRESULT description([out, retval] IJsonNode** node);
    [id(0x00000003), propget]
    HRESULT contentEncoding([out, retval] IJsonNode** node);
    [id(0x00000004), propget]
    HRESULT contentMediaType([out, retval] IJsonNode** node);
    [id(0x00000005), propget]
    HRESULT readOnly([out, retval] IJsonNode** node);
    [id(0x00000006), propget]
    HRESULT writeOnly([out, retval] IJsonNode** node);
    [id(0x00000007), propget]
    HRESULT defaultNode([out, retval] IJsonNode** node);
    [id(0x00000008), propget]
    HRESULT examples([out, retval] IJsonNode** node);
```

```
};

[
  odl,
  uuid(...),
  dual,
  nonextensible,
  oleautomation
]
interface IJsonStatus : IDispatch {
    [id(00000000), propget]
    HRESULT success([out, retval] VARIANT_BOOL* success);
    [id(0x00000001), propget]
    HRESULT kind([out, retval] JsonStatusKind* kind);
    [id(0x00000002), propget]
    HRESULT code([out, retval] long* code);
    [id(0x00000003), propget]
    HRESULT response([out, retval] VARIANT* response);
    [id(0x00000004), propget]
    HRESULT line([out, retval] long* line);
    [id(0x00000005), propget]
    HRESULT column([out, retval] long* column);
    [id(0x00000006), propget]
    HRESULT schema([out, retval] IJsonNode** schema);
    [id(0x00000007), propget]
    HRESULT instance([out, retval] IJsonNode** instance);
};
```

```
typedef [uuid(...)]
enum {
   jsonStatusUnknown = 0,
   jsonStatusJess = 1,
   jsonStatusSystem = 2,
   jsonStatusProvider = 3
} JsonStatusKind;

[
  odl,
  uuid(...),
  dual,
  nonextensible,
  oleautomation
]
interface IJsonSelector : IDispatch {
   [id(00000000), propget]
   HRESULT expression([out, retval] BSTR* expression);
   [id(0x00000064)]
   HRESULT select(
                  [in] IDispatch* from,
                  [out, retval] VARIANT* result);
};

[
  odl,
  uuid(...),
  dual,
  nonextensible,
  oleautomation
]
interface IJsonFormat : IDispatch {
   [id(0x00000001), propget]
   HRESULT indentationSize([out, retval] long* value);
   [id(0x00000001), propput]
   HRESULT indentationSize([in] long value);
   [id(0x00000002), propget]
   HRESULT useDosNewLines([out, retval] VARIANT_BOOL* value);
   [id(0x00000002), propput]
   HRESULT useDosNewLines([in] VARIANT_BOOL value);
   [id(0x00000003), propget]
   HRESULT useTabs([out, retval] VARIANT_BOOL* value);
   [id(0x00000003), propput]
   HRESULT useTabs([in] VARIANT_BOOL value);
   [id(0x00000004), propget]
   HRESULT placeBracketsOnNewLine([out, retval] VARIANT_BOOL* value);
   [id(0x00000004), propput]
   HRESULT placeBracketsOnNewLine([in] VARIANT_BOOL value);
   [id(0x00000005), propget]
   HRESULT placeValueOnNewLine([out, retval] VARIANT_BOOL* value);
   [id(0x00000005), propput]
   HRESULT placeValueOnNewLine([in] VARIANT_BOOL value);
   [id(0x00000006), propget]
   HRESULT addNewLineInEmptyBrackets([out, retval] VARIANT_BOOL* value);
   [id(0x00000006), propput]
   HRESULT addNewLineInEmptyBrackets([in] VARIANT_BOOL value);
```

```
    [id(0x00000007), propget]
    HRESULT addSpaceBeforeColon([out, retval] VARIANT_BOOL* value);
    [id(0x00000007), propput]
    HRESULT addSpaceBeforeColon([in] VARIANT_BOOL value);
    [id(0x00000008), propget]
    HRESULT addSpaceAfterColon([out, retval] VARIANT_BOOL* value);
    [id(0x00000008), propput]
    HRESULT addSpaceAfterColon([in] VARIANT_BOOL value);
    [id(0x00000009), propget]
    HRESULT addSpaceInBrackets([out, retval] VARIANT_BOOL* value);
    [id(0x00000009), propput]
    HRESULT addSpaceInBrackets([in] VARIANT_BOOL value);
    [id(0x0000000a), propget]
    HRESULT addSpaceAfterComma([out, retval] VARIANT_BOOL* value);
    [id(0x0000000a), propput]
    HRESULT addSpaceAfterComma([in] VARIANT_BOOL value);
    [id(0x0000000b), propget]
    HRESULT forceEscapeNonAscii([out, retval] VARIANT_BOOL* value);
    [id(0x0000000b), propput]
    HRESULT forceEscapeNonAscii([in] VARIANT_BOOL value);
    [id(0x0000000c), propget]
    HRESULT forceEscapeNonEascii([out, retval] VARIANT_BOOL* value);
    [id(0x0000000c), propput]
    HRESULT forceEscapeNonEascii([in] VARIANT_BOOL value);
};

typedef [uuid(...)]
enum {
```

```
jsonErrorSuccess = 0,
jsonErrorCommonFirst = 256,
jsonErrorCommonInvalidArgument = 256,
jsonErrorCommonInvalidObjectState = 257,
jsonErrorCommonUnsupportedByObjectKind = 258,
jsonErrorCommonIndexOutOfBoundary = 259,
jsonErrorCommonInvalidEncodingSequence = 260,
jsonErrorCommonBadDereferencing = 261,
jsonErrorCommonNoMemoryAllocated = 262,
jsonErrorCommonLast = 262,
jsonErrorParserFirst = 4096,
jsonErrorParserUnexpectedEndOfData = 4096,
jsonErrorParserUnexpectedToken = 4097,
jsonErrorParserInvalidData = 4098,
jsonErrorParserInvalidEscapeSequence = 4099,
jsonErrorParserLast = 4099,
jsonErrorDomFirst = 12288,
jsonErrorDomCannotBeAddedToObject = 12288,
jsonErrorDomCannotBeAddedToArray = 12289,
jsonErrorDomNoReadAccess = 12290,
jsonErrorDomNoWriteAccess = 12291,
jsonErrorDomNodeNotFound = 12292,
jsonErrorDomAlreadyAtRoot = 12293,
jsonErrorDomNodeHasNoName = 12294,
jsonErrorDomNoAssociatedNode = 12295,
jsonErrorDomForeignNode = 12296,
jsonErrorDomLast = 12296,
jsonErrorSchemaFirst = 16384,
    jsonErrorSchemaInvalidUri = 16384,
    jsonErrorSchemaCannotBeResolved = 16385,
    jsonErrorSchemaUnexpectedValue = 16386,
    jsonErrorSchemaUnexpectedValueType = 16387,
    jsonErrorSchemaDuplicatedAssertion = 16388,
    jsonErrorSchemaInvalidAssertionSyntax = 16389,
    jsonErrorSchemaReferenceCycle = 16390,
    jsonErrorSchemaDuplicatedId = 16391,
    jsonErrorSchemaUnexpectedSchema = 16392,
    jsonErrorSchemaUnsupportedSchema = 16393,
    jsonErrorSchemaRegex = 16394,
    jsonErrorSchemaSchemaRequired = 16395,
    jsonErrorSchemaLast = 16395,
    jsonErrorValidationFirst = 20480,
    jsonErrorValidationFalseAssertion = 20480,
    jsonErrorValidationValueType = 20481,
    jsonErrorValidationValue = 20482,
    jsonErrorValidationRequiredProperty = 20483,
    jsonErrorValidationNodeCount = 20484,
    jsonErrorValidationNotUnique = 20485,
    jsonErrorValidationRequiredNode = 20486,
    jsonErrorValidationLast = 20486
} JsonError;

[
  uuid(...),
  helpstring("JSON DOM Document"),
  licensed
```

```
]
coclass JsonDocument {
   [default] interface IJsonDocument;
};

[
  odl,
  uuid(...),
  dual,
  nonextensible,
  oleautomation
]
interface IJsonTools : IDispatch {
   [id(0x00000001), propget]
   HRESULT policy([out, retval] IJsonPolicy** policy);
   [id(0x0000006a)]
   HRESULT createJsonPointer(
                [in] BSTR pointer,
                [out, retval] IJsonSelector** selector);
   [id(0x0000006b)]
   HRESULT createRelativeJsonPointer(
                [in] BSTR pointer,
                [out, retval] IJsonSelector** selector);
};

[
  odl,
  uuid(...),
  dual,
  nonextensible,
  oleautomation
]
interface IJsonPolicy : IDispatch {
    [id(0x00000001), propget]
    HRESULT nullValue([out, retval] VARIANT* nullValue);
    [id(0x00000001), propput]
    HRESULT nullValue([in] VARIANT nullValue);
    [id(0x00000002), propget]
    HRESULT ddaEscape([out, retval] BSTR* key);
    [id(0x00000002), propput]
    HRESULT ddaEscape([in] BSTR key);
};

[
  uuid(...),
  helpstring("JSON Tools"),
  licensed
]
coclass JsonTools {
    [default] interface IJsonTools;
};

[
  odl,
  uuid(...),
  dual,
  nonextensible,
  oleautomation
]
interface IJsonSchemas : IDispatch {
    [id(0xfffffffc), propget, restricted, hidden]
    HRESULT _newEnum([out, retval] IUnknown** punk);
    [id(00000000), propget]
    HRESULT item(
                 [in] long index,
                 [out, retval] BSTR* id);
    [id(0x00000002), propget]
    HRESULT length([out, retval] long* result);
    [id(0x00000064)]
    HRESULT add(
                 [in] IDispatch* schemaNode,
                 [in] BSTR defaultId,
                 [out, retval] IJsonSchema** schema);
    [id(0x00000065)]
    HRESULT find(
                 [in] BSTR id,
                 [out, retval] IJsonSchema** schema);
    [id(0x00000066)]
    HRESULT remove([in] BSTR id);
    [id(0x00000067)]
    HRESULT clear();
    [id(0x00000068)]
    HRESULT annotate(
                 [in] IDispatch* instance,
                [in, optional] VARIANT id,
                [out, retval] IJsonStatus** status);
    [id(0x00000069)]
    HRESULT validate(
                [in] IDispatch* instance,
                [in, optional] VARIANT id,
                [out, retval] IJsonStatus** status);
    [id(0x000000c8)]
    HRESULT onRequestSchema([in] VARIANT handler);
};

[
  odl,
  uuid(...),
  dual,
  nonextensible,
  oleautomation
]
interface IJsonSchema : IDispatch {
    [id(0x00000001), propget]
    HRESULT id([out, retval] BSTR* id);
    [id(0x00000002), propget]
    HRESULT standard([out, retval] JsonSchemaStandard* standard);
    [id(0x00000003), propget]
    HRESULT schemas([out, retval] IJsonSchemas** schemas);
    [id(0x00000004), propget]
    HRESULT root([out, retval] IJsonNode** root);
    [id(0x00000064)]
    HRESULT annotate(
                [in] IDispatch* instance,
                [out, retval] IJsonStatus** status);
    [id(0x00000065)]
    HRESULT validate(
                [in] IDispatch* instance,
                [out, retval] IJsonStatus** status);
};

typedef [uuid(...)]
enum {
    jsonSchemaStandardUndefined = 0,
    jsonSchemaStandardDraft07 = 7
} JsonSchemaStandard;

[
  uuid(...),
  helpstring("JSON Schemas"),
  licensed
]
coclass JsonSchemas {
    [default] interface IJsonSchemas;
};

[
  odl,
  uuid(...),
  dual,
  nonextensible,
  oleautomation
]
interface IJsonEssentials : IDispatch {
    [id(0x00000001), propget]
    HRESULT id([out, retval] long* id);
    [id(0x00000002), propget]
    HRESULT versionMajor([out, retval] long* major);
    [id(0x00000003), propget]
    HRESULT versionMinor([out, retval] long* minor);
    [id(0x00000004), propget]
    HRESULT versionPatch([out, retval] long* patch);
    [id(0x00000005), propget]
    HRESULT licenseText([out, retval] BSTR* license);
    [id(0x00000064)]
    HRESULT unlock(
                [in] BSTR licenseToken,
                [out, retval] VARIANT_BOOL* unlocked);
    [id(0x00000065), propget]
    HRESULT tools([out, retval] IJsonTools** tools);
    [id(0x00000066)]
    HRESULT createDocument([out, retval] IJsonDocument** document);
    [id(0x00000067)]
    HRESULT createSchemas([out, retval] IJsonSchemas** schemas);
};

[
  uuid(...),
       helpstring("JSON Essentials library facade"),
       licensed
     ]
     coclass JsonEssentials {
         [default] interface IJsonEssentials;
     };
};
```

5 References

1. ECMA-404: The JSON data interchange syntax
   https://www.ecma-international.org/publications-and-standards/standards/ecma-404/
2. JSON Schema Specification: Draft7
   https://json-schema.org/specification-links.html#draft-7
3. RFC-6901: JavaScript Object Notation (JSON) Pointer
   https://tools.ietf.org/html/rfc6901
4. Relative JSON Pointer: Internet Draft
   https://datatracker.ietf.org/doc/html/draft-handrews-relative-json-pointer-01
5. RFC-3629: UTF-8, a transformation format of ISO 10646
   https://www.ietf.org/rfc/rfc3629
6. RFC-2781: UTF-16, an encoding of ISO 10646
   https://tools.ietf.org/html/rfc2781
7. RFC-5198: Unicode Format for Network Interchange
   https://tools.ietf.org/html/rfc5198
8. MS-OAUT: OLE Automation Protocol
   https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-oaut/bbb05720-f724-45c7-8d17-f83c3d1a3961
9. RFC-6365: Terminology Used in Internationalization in the IETF
   https://tools.ietf.org/html/rfc6365
10. RFC-7231: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content
    https://tools.ietf.org/html/rfc7231
11. RFC-3986: Uniform Resource Identifier (URI): Generic Syntax
    https://tools.ietf.org/html/rfc3986
12. RFC-1867: Form-based File Upload in HTML
    https://tools.ietf.org/html/rfc1867

6 Credits

The development and distribution of JSON Essentials v1.2.1 would not be possible without the listed below projects and services, which enabled, simplified and accelerated development:

1. JSON Schema Test Suite: The comprehensive JSON Schema tests by http://json-schema.org/, used by the library for development, testing and use cases demonstration, some tests are included and distributed along with the library samples

   https://github.com/json-schema-org/JSON-Schema-Test-Suite

2. Boost C++ libraries: By no means Boost is the most popular C++ framework that provides a countless number of ready solutions for frequently arising needs.

   https://boost.org

3. Vim: The core C++ implementation as well as WSH tests and all helper scripts have been created and debugged in this outstanding cross-platform editor.

   https://www.vim.org

4. Microsoft Visual Studio: No other environments can handle Windows native and managed applications development better then Microsoft Visual Studio.

   https://visualstudio.microsoft.com

5. Visual Studio Code: An extremely handy, multi-functional and highly configurable cross-platform code editor.

   https://code.visualstudio.com

6. Google Workplace: This multi-functional web-service handles PINERY web-site back-end, e-mails and file exchange

   https://workspace.google.com

7. Amazon AWS: This multi-functional web-service handles PINERY domains, web-site and source control hosting.

   https://aws.amazon.com

8. LibreOffice: This document is created in LibreOffice Writer!

   https://www.libreoffice.org

9. Convertio: The web version of this document is converted by this service.

   https://convertio.co

10. Postman: This web-service provides convenient HTTP testing end-points that are used by JSON Essentials tests and samples.

    https://www.postman.com

11. Transfonter: Enabled PINERY web-site to use consistent fonts.

    https://transfonter.org

12. FAR Manager: This file manager helps to navigate and manage file system during development on Windows platform.

    https://www.farmanager.com

13. CMake: The core C++ implementation of JSON Essentials is managed by this build managing system.

    https://cmake.org

14. SlikSVN: A reliable Subversion client utility that keeps all JSON Essentials source files in order and up-to-date.

    https://sliksvn.com

15. Perl Compatible Regular Expressions – PCRE: Handles regular expressions used JSON Schema implementation.

    https://www.pcre.org

16. OpenSSL: Used for code signing certificates for executable modules.

    https://www.openssl.org

17. wk<html>topdf: Enabled automated receipt creation.

    https://wkhtmltopdf.org

7 Third-party Software Licenses

7.1 Perl Compatible Regular Expressions – PCRE

(as take from https://www.pcre.org/licence.txt at the moment of publication)