VSCode JSON Language Server

The JSON Language server provides language-specific smarts for editing, validating and understanding JSON documents. It runs as a separate executable and implements the language server protocol to be connected by any code editor or IDE.

Capabilities

Server capabilities

The JSON language server supports requests on documents of language id `json` and `jsonc`.

- `json` documents are parsed and validated following the JSON specification.
- `jsonc` documents additionally accept single line (`//`) and multi-line comments (`/* ... */`). JSONC is a VSCode specific file format, intended for VSCode configuration files, without any aspirations to define a new common file format.

The server implements the following capabilities of the language server protocol:

- Code completion for JSON properties and values based on the document's JSON schema or based on existing properties and values used at other places in the document. JSON schemas are configured through the server configuration options.
- Hover for values based on descriptions in the document's JSON schema.
- Document Symbols for quick navigation to properties in the document.
- Document Colors for showing color decorators on values representing colors and Color Presentation for color presentation information to support color pickers. The location of colors is defined by the document's JSON schema. All values marked with `"format": "color-hex"` (VSCode specific, non-standard JSON Schema extension) are considered color values. The supported color formats are `#rgb[a]` and `#rrggbb[aa]`.
- Code Formatting supporting ranges and formatting the whole document.
- Folding Ranges for all folding ranges in the document.
- Semantic Selection for semantic selection for one or multiple cursor positions.
- Goto Definition for $ref references in JSON schemas
- Diagnostics (Validation) are pushed for all open documents
  - syntax errors
  - structural validation based on the document's JSON schema.

In order to load JSON schemas, the JSON server uses NodeJS `http` and `fs` modules. For all other features, the JSON server only relies on the documents and settings provided by the client through the LSP.

Client requirements:

The JSON language server expects the client to only send requests and notifications for documents of language id `json` and `jsonc`.

The JSON language server has the following dependencies on the client's capabilities:

- Code completion requires that the client capability has *snippetSupport*. If not supported by the client, the server will not offer the completion capability.

- Formatting support requires the client to support *dynamicRegistration* for *rangeFormatting*. If not supported by the client, the server will not offer the format capability.

Configuration

Initialization options

The client can send the following initialization options to the server:

- `provideFormatter: boolean | undefined`. If defined, the value defines whether the server provides the `documentRangeFormattingProvider` capability on initialization. If undefined, the setting `json.format.enable` is used to determine whether formatting is provided. The formatter will then be registered through dynamic registration. If the client does not support dynamic registration, no formatter will be available.
- `handledSchemaProtocols`: The URI schemas handles by the server. See section `Schema configuration` below.
- `customCapabilities`: Additional non-LSP client capabilities:
  - `rangeFormatting: { editLimit: x } }`: For performance reasons, limit the number of edits returned by the range formatter to `x`.

Settings

Clients may send a `workspace/didChangeConfiguration` notification to notify the server of settings changes. The server supports the following settings:

- http
  - `proxy`: The URL of the proxy server to use when fetching schema. When undefined or empty, no proxy is used.
  - `proxyStrictSSL`: Whether the proxy server certificate should be verified against the list of supplied CAs.
- json
  - `format`
    - `enable`: Whether the server should register the formatting support. This option is only applicable if the client supports *dynamicRegistration* for *rangeFormatting* and `initializationOptions.provideFormatter` is not defined.
  - `schemas`: Configures association of file names to schema URL or schemas and/or associations of schema URL to schema content.
    - `fileMatch`: an array of file names or paths (separated by `/`). `*` can be used as a wildcard. Exclusion patterns can also be defined and start with '!'. A file matches when there is at least one matching pattern and the last matching pattern is not an exclusion pattern.
    - `url`: The URL of the schema, optional when also a schema is provided.
    - `schema`: The schema content.
  - `resultLimit`: The max number folding ranges and outline symbols to be computed (for performance reasons)

```
{
    "http": {
```

```
            "proxy": "",
            "proxyStrictSSL": true
        },
        "json": {
            "format": {
                "enable": true
            },
            "schemas": [
                {
                    "fileMatch": [
                        "foo.json",
                        "*.superfoo.json"
                    ],
                    "url": "http://json.schemastore.org/foo",
                    "schema": {
                        "type": "array"
                    }
                }
            ]
        }
    }
```

Schema configuration and custom schema content delivery

JSON schemas are essential for code assist, hovers, color decorators to work and are required for structural validation.

To find the schema for a given JSON document, the server uses the following mechanisms:

- JSON documents can define the schema URL using a `$schema` property
- The settings define a schema association based on the documents URL. Settings can either associate a schema URL to a file or path pattern, and they can directly provide a schema.
- Additionally, schema associations can also be provided by a custom 'schemaAssociations' configuration call.

Schemas are identified by URLs. To load the content of a schema, the JSON language server either tries to load from that URI or path itself or delegates to the client.

The `initializationOptions.handledSchemaProtocols` initialization option defines which URLs are handled by the server. Requests for all other URIs are sent to the client.

`handledSchemaProtocols` is part of the initialization options and can't be changed while the server is running.

```
let clientOptions: LanguageClientOptions = {
        initializationOptions: {
            handledSchemaProtocols: ['file'] // language server should only try to
load file URLs
        }
        ...
}
```

If `handledSchemaProtocols` is not set, the JSON language server will load the following URLs itself:

- `http` , `https` : Loaded using NodeJS's HTTP support. Proxies can be configured through the settings.

- `file` : Loaded using NodeJS's `fs` support.

Schema content request

Requests for schemas with URLs not handled by the server are forwarded to the client through an LSP request. This request is a JSON language server-specific, non-standardized, extension to the LSP.

Request:

- method: 'vscode/content'
- params: `string` - The schema URL to request.
- response: `string` - The content of the schema with the given URL

Schema content change notification

When the client is aware that a schema content has changed, it will notify the server through a notification. This notification is a JSON language server-specific, non-standardized, extension to the LSP. The server will, as a response, clear the schema content from the cache and reload the schema content when required again.

Schema associations notification

In addition to the settings, schemas associations can also be provided through a notification from the client to the server. This notification is a JSON language server-specific, non-standardized, extension to the LSP.

Notification:

- method: 'json/schemaAssociations'
- params: `ISchemaAssociations` or `ISchemaAssociation[]` defined as follows

```
interface ISchemaAssociations {
  /**
   * An object where:
   *   - keys are file names or file paths (using `/` as path separator). `*` can be
used as a wildcard.
   *   - values are an arrays of schema URIs
   */
  [pattern: string]: string[];
}

interface ISchemaAssociation {
  /**
   * The URI of the schema, which is also the identifier of the schema.
   */
  uri: string;

  /**
   * A list of file path patterns that are associated to the schema. The '*'
wildcard can be used. Exclusion patterns starting with '!'.
   * For example '*.schema.json', 'package.json', '!foo*.schema.json'.
   * A match succeeds when there is at least one pattern matching and last matching
pattern does not start with '!'.
   */
  fileMatch: string[];
  /*
   * The schema for the given URI.
```

```
    * If no schema is provided, the schema will be fetched with the schema request
service (if available).
    */
   schema?: JSONSchema;
}
```

`ISchemaAssociations`

- keys: a file names or file path (separated by `/`). `*` can be used as a wildcard.
- values: An array of schema URLs

Notification:

- method: 'json/schemaContent'
- params: `string` the URL of the schema that has changed.

Item Limit

If the setting `resultLimit` is set, the JSON language server will limit the number of folding ranges and document symbols computed. When the limit is reached, a notification `json/resultLimitReached` is sent that can be shown that can be shown to the user.

Notification:

- method: 'json/resultLimitReached'
- params: a human readable string to show to the user.

Try

The JSON language server is shipped with Visual Studio Code as part of the built-in VSCode extension `json-language-features`. The server is started when the first JSON file is opened. The VSCode JSON documentation for detailed information on the user experience and has more information on how to configure the language support.

Integrate

If you plan to integrate the JSON language server into an editor and IDE, check out this page if there's already an LSP client integration available.

You can also launch the language server as a command and connect to it. For that, install the `vscode-json-languageserver` npm module:

```
npm install -g vscode-json-languageserver
```

Start the language server with the `vscode-json-languageserver` command. Use a command line argument to specify the preferred communication channel:

```
vscode-json-languageserver --node-ipc
vscode-json-languageserver --stdio
vscode-json-languageserver --socket=<port>
```

To connect to the server from NodeJS, see Remy Suen's great write-up on how to communicate with the server through the available communication channels.

Participate

The source code of the JSON language server can be found in the VSCode repository at extensions/json-language-features/server.

File issues and pull requests in the VSCode GitHub Issues. See the document How to Contribute on how to build and run from source.

Most of the functionality of the server is located in libraries:

- jsonc-parser contains the JSON parser and scanner.
- vscode-json-languageservice contains the implementation of all features as a re-usable library.
- vscode-languageserver-node contains the implementation of language server for NodeJS.

Help on any of these projects is very welcome.

Code of Conduct

This project has adopted the Microsoft Open Source Code of Conduct. For more information see the Code of Conduct FAQ or contact opencode@microsoft.com with any additional questions or comments.

License