declare var ajv: {

(options?: ajv.Options): ajv.Ajv;

new(options?: ajv.Options): ajv.Ajv;

ValidationError: typeof AjvErrors.ValidationError;

MissingRefError: typeof AjvErrors.MissingRefError;

$dataMetaSchema: object;

}

declare namespace AjvErrors {

class ValidationError extends Error {

constructor(errors: Array<ajv.ErrorObject>);

message: string;

errors: Array<ajv.ErrorObject>;

ajv: true;

validation: true;

}

class MissingRefError extends Error {

constructor(baseId: string, ref: string, message?: string);

static message: (baseId: string, ref: string) => string;

message: string;

missingRef: string;

missingSchema: string;

}

}

declare namespace ajv {

type ValidationError = AjvErrors.ValidationError;

type MissingRefError = AjvErrors.MissingRefError;

interface Ajv {

/\*\*

\* Validate data using schema

\* Schema will be compiled and cached (using serialized JSON as key, [fast-json-stable-stringify](https://github.com/epoberezkin/fast-json-stable-stringify) is used to serialize by default).

\* @param {string|object|Boolean} schemaKeyRef key, ref or schema object

\* @param {Any} data to be validated

\* @return {Boolean} validation result. Errors from the last validation will be available in `ajv.errors` (and also in compiled schema: `schema.errors`).

\*/

validate(schemaKeyRef: object | string | boolean, data: any): boolean | PromiseLike<any>;

/\*\*

\* Create validating function for passed schema.

\* @param {object|Boolean} schema schema object

\* @return {Function} validating function

\*/

compile(schema: object | boolean): ValidateFunction;

/\*\*

\* Creates validating function for passed schema with asynchronous loading of missing schemas.

\* `loadSchema` option should be a function that accepts schema uri and node-style callback.

\* @this Ajv

\* @param {object|Boolean} schema schema object

\* @param {Boolean} meta optional true to compile meta-schema; this parameter can be skipped

\* @param {Function} callback optional node-style callback, it is always called with 2 parameters: error (or null) and validating function.

\* @return {PromiseLike<ValidateFunction>} validating function

\*/

compileAsync(schema: object | boolean, meta?: Boolean, callback?: (err: Error, validate: ValidateFunction) => any): PromiseLike<ValidateFunction>;

/\*\*

\* Adds schema to the instance.

\* @param {object|Array} schema schema or array of schemas. If array is passed, `key` and other parameters will be ignored.

\* @param {string} key Optional schema key. Can be passed to `validate` method instead of schema object or id/ref. One schema per instance can have empty `id` and `key`.

\* @return {Ajv} this for method chaining

\*/

addSchema(schema: Array<object> | object, key?: string): Ajv;

/\*\*

\* Add schema that will be used to validate other schemas

\* options in META\_IGNORE\_OPTIONS are alway set to false

\* @param {object} schema schema object

\* @param {string} key optional schema key

\* @return {Ajv} this for method chaining

\*/

addMetaSchema(schema: object, key?: string): Ajv;

/\*\*

\* Validate schema

\* @param {object|Boolean} schema schema to validate

\* @return {Boolean} true if schema is valid

\*/

validateSchema(schema: object | boolean): boolean;

/\*\*

\* Get compiled schema from the instance by `key` or `ref`.

\* @param {string} keyRef `key` that was passed to `addSchema` or full schema reference (`schema.id` or resolved id).

\* @return {Function} schema validating function (with property `schema`). Returns undefined if keyRef can't be resolved to an existing schema.

\*/

getSchema(keyRef: string): ValidateFunction | undefined;

/\*\*

\* Remove cached schema(s).

\* If no parameter is passed all schemas but meta-schemas are removed.

\* If RegExp is passed all schemas with key/id matching pattern but meta-schemas are removed.

\* Even if schema is referenced by other schemas it still can be removed as other schemas have local references.

\* @param {string|object|RegExp|Boolean} schemaKeyRef key, ref, pattern to match key/ref or schema object

\* @return {Ajv} this for method chaining

\*/

removeSchema(schemaKeyRef?: object | string | RegExp | boolean): Ajv;

/\*\*

\* Add custom format

\* @param {string} name format name

\* @param {string|RegExp|Function} format string is converted to RegExp; function should return boolean (true when valid)

\* @return {Ajv} this for method chaining

\*/

addFormat(name: string, format: FormatValidator | FormatDefinition): Ajv;

/\*\*

\* Define custom keyword

\* @this Ajv

\* @param {string} keyword custom keyword, should be a valid identifier, should be different from all standard, custom and macro keywords.

\* @param {object} definition keyword definition object with properties `type` (type(s) which the keyword applies to), `validate` or `compile`.

\* @return {Ajv} this for method chaining

\*/

addKeyword(keyword: string, definition: KeywordDefinition): Ajv;

/\*\*

\* Get keyword definition

\* @this Ajv

\* @param {string} keyword pre-defined or custom keyword.

\* @return {object|Boolean} custom keyword definition, `true` if it is a predefined keyword, `false` otherwise.

\*/

getKeyword(keyword: string): object | boolean;

/\*\*

\* Remove keyword

\* @this Ajv

\* @param {string} keyword pre-defined or custom keyword.

\* @return {Ajv} this for method chaining

\*/

removeKeyword(keyword: string): Ajv;

/\*\*

\* Validate keyword

\* @this Ajv

\* @param {object} definition keyword definition object

\* @param {boolean} throwError true to throw exception if definition is invalid

\* @return {boolean} validation result

\*/

validateKeyword(definition: KeywordDefinition, throwError: boolean): boolean;

/\*\*

\* Convert array of error message objects to string

\* @param {Array<object>} errors optional array of validation errors, if not passed errors from the instance are used.

\* @param {object} options optional options with properties `separator` and `dataVar`.

\* @return {string} human readable string with all errors descriptions

\*/

errorsText(errors?: Array<ErrorObject> | null, options?: ErrorsTextOptions): string;

errors?: Array<ErrorObject> | null;

\_opts: Options;

}

interface CustomLogger {

log(...args: any[]): any;

warn(...args: any[]): any;

error(...args: any[]): any;

}

interface ValidateFunction {

(

data: any,

dataPath?: string,

parentData?: object | Array<any>,

parentDataProperty?: string | number,

rootData?: object | Array<any>

): boolean | PromiseLike<any>;

schema?: object | boolean;

errors?: null | Array<ErrorObject>;

refs?: object;

refVal?: Array<any>;

root?: ValidateFunction | object;

$async?: true;

source?: object;

}

interface Options {

$data?: boolean;

allErrors?: boolean;

verbose?: boolean;

jsonPointers?: boolean;

uniqueItems?: boolean;

unicode?: boolean;

format?: false | string;

formats?: object;

keywords?: object;

unknownFormats?: true | string[] | 'ignore';

schemas?: Array<object> | object;

schemaId?: '$id' | 'id' | 'auto';

missingRefs?: true | 'ignore' | 'fail';

extendRefs?: true | 'ignore' | 'fail';

loadSchema?: (uri: string, cb?: (err: Error, schema: object) => void) => PromiseLike<object | boolean>;

removeAdditional?: boolean | 'all' | 'failing';

useDefaults?: boolean | 'empty' | 'shared';

coerceTypes?: boolean | 'array';

strictDefaults?: boolean | 'log';

strictKeywords?: boolean | 'log';

strictNumbers?: boolean;

async?: boolean | string;

transpile?: string | ((code: string) => string);

meta?: boolean | object;

validateSchema?: boolean | 'log';

addUsedSchema?: boolean;

inlineRefs?: boolean | number;

passContext?: boolean;

loopRequired?: number;

ownProperties?: boolean;

multipleOfPrecision?: boolean | number;

errorDataPath?: string,

messages?: boolean;

sourceCode?: boolean;

processCode?: (code: string, schema: object) => string;

cache?: object;

logger?: CustomLogger | false;

nullable?: boolean;

serialize?: ((schema: object | boolean) => any) | false;

}

type FormatValidator = string | RegExp | ((data: string) => boolean | PromiseLike<any>);

type NumberFormatValidator = ((data: number) => boolean | PromiseLike<any>);

interface NumberFormatDefinition {

type: "number",

validate: NumberFormatValidator;

compare?: (data1: number, data2: number) => number;

async?: boolean;

}

interface StringFormatDefinition {

type?: "string",

validate: FormatValidator;

compare?: (data1: string, data2: string) => number;

async?: boolean;

}

type FormatDefinition = NumberFormatDefinition | StringFormatDefinition;

interface KeywordDefinition {

type?: string | Array<string>;

async?: boolean;

$data?: boolean;

errors?: boolean | string;

metaSchema?: object;

// schema: false makes validate not to expect schema (ValidateFunction)

schema?: boolean;

statements?: boolean;

dependencies?: Array<string>;

modifying?: boolean;

valid?: boolean;

// one and only one of the following properties should be present

validate?: SchemaValidateFunction | ValidateFunction;

compile?: (schema: any, parentSchema: object, it: CompilationContext) => ValidateFunction;

macro?: (schema: any, parentSchema: object, it: CompilationContext) => object | boolean;

inline?: (it: CompilationContext, keyword: string, schema: any, parentSchema: object) => string;

}

interface CompilationContext {

level: number;

dataLevel: number;

dataPathArr: string[];

schema: any;

schemaPath: string;

baseId: string;

async: boolean;

opts: Options;

formats: {

[index: string]: FormatDefinition | undefined;

};

keywords: {

[index: string]: KeywordDefinition | undefined;

};

compositeRule: boolean;

validate: (schema: object) => boolean;

util: {

copy(obj: any, target?: any): any;

toHash(source: string[]): { [index: string]: true | undefined };

equal(obj: any, target: any): boolean;

getProperty(str: string): string;

schemaHasRules(schema: object, rules: any): string;

escapeQuotes(str: string): string;

toQuotedString(str: string): string;

getData(jsonPointer: string, dataLevel: number, paths: string[]): string;

escapeJsonPointer(str: string): string;

unescapeJsonPointer(str: string): string;

escapeFragment(str: string): string;

unescapeFragment(str: string): string;

};

self: Ajv;

}

interface SchemaValidateFunction {

(

schema: any,

data: any,

parentSchema?: object,

dataPath?: string,

parentData?: object | Array<any>,

parentDataProperty?: string | number,

rootData?: object | Array<any>

): boolean | PromiseLike<any>;

errors?: Array<ErrorObject>;

}

interface ErrorsTextOptions {

separator?: string;

dataVar?: string;

}

interface ErrorObject {

keyword: string;

dataPath: string;

schemaPath: string;

params: ErrorParameters;

// Added to validation errors of propertyNames keyword schema

propertyName?: string;

// Excluded if messages set to false.

message?: string;

// These are added with the `verbose` option.

schema?: any;

parentSchema?: object;

data?: any;

}

type ErrorParameters = RefParams | LimitParams | AdditionalPropertiesParams |

DependenciesParams | FormatParams | ComparisonParams |

MultipleOfParams | PatternParams | RequiredParams |

TypeParams | UniqueItemsParams | CustomParams |

PatternRequiredParams | PropertyNamesParams |

IfParams | SwitchParams | NoParams | EnumParams;

interface RefParams {

ref: string;

}

interface LimitParams {

limit: number;

}

interface AdditionalPropertiesParams {

additionalProperty: string;

}

interface DependenciesParams {

property: string;

missingProperty: string;

depsCount: number;

deps: string;

}

interface FormatParams {

format: string

}

interface ComparisonParams {

comparison: string;

limit: number | string;

exclusive: boolean;

}

interface MultipleOfParams {

multipleOf: number;

}

interface PatternParams {

pattern: string;

}

interface RequiredParams {

missingProperty: string;

}

interface TypeParams {

type: string;

}

interface UniqueItemsParams {

i: number;

j: number;

}

interface CustomParams {

keyword: string;

}

interface PatternRequiredParams {

missingPattern: string;

}

interface PropertyNamesParams {

propertyName: string;

}

interface IfParams {

failingKeyword: string;

}

interface SwitchParams {

caseIndex: number;

}

interface NoParams { }

interface EnumParams {

allowedValues: Array<any>;

}

}

export = ajv;