{{# def.definitions }}

{{## def.\_error:\_rule:

{{ 'istanbul ignore else'; }}

{{? it.createErrors !== false }}

{

keyword: '{{= $errorKeyword || \_rule }}'

, dataPath: (dataPath || '') + {{= it.errorPath }}

, schemaPath: {{=it.util.toQuotedString($errSchemaPath)}}

, params: {{# def.\_errorParams[\_rule] }}

{{? it.opts.messages !== false }}

, message: {{# def.\_errorMessages[\_rule] }}

{{?}}

{{? it.opts.verbose }}

, schema: {{# def.\_errorSchemas[\_rule] }}

, parentSchema: validate.schema{{=it.schemaPath}}

, data: {{=$data}}

{{?}}

}

{{??}}

{}

{{?}}

#}}

{{## def.\_addError:\_rule:

if (vErrors === null) vErrors = [err];

else vErrors.push(err);

errors++;

#}}

{{## def.addError:\_rule:

var err = {{# def.\_error:\_rule }};

{{# def.\_addError:\_rule }}

#}}

{{## def.error:\_rule:

{{# def.beginDefOut}}

{{# def.\_error:\_rule }}

{{# def.storeDefOut:\_\_err }}

{{? !it.compositeRule && $breakOnError }}

{{ 'istanbul ignore if'; }}

{{? it.async }}

throw new ValidationError([{{=\_\_err}}]);

{{??}}

validate.errors = [{{=\_\_err}}];

return false;

{{?}}

{{??}}

var err = {{=\_\_err}};

{{# def.\_addError:\_rule }}

{{?}}

#}}

{{## def.extraError:\_rule:

{{# def.addError:\_rule}}

{{? !it.compositeRule && $breakOnError }}

{{ 'istanbul ignore if'; }}

{{? it.async }}

throw new ValidationError(vErrors);

{{??}}

validate.errors = vErrors;

return false;

{{?}}

{{?}}

#}}

{{## def.checkError:\_rule:

if (!{{=$valid}}) {

{{# def.error:\_rule }}

}

#}}

{{## def.resetErrors:

errors = {{=$errs}};

if (vErrors !== null) {

if ({{=$errs}}) vErrors.length = {{=$errs}};

else vErrors = null;

}

#}}

{{## def.concatSchema:{{?$isData}}' + {{=$schemaValue}} + '{{??}}{{=$schema}}{{?}}#}}

{{## def.appendSchema:{{?$isData}}' + {{=$schemaValue}}{{??}}{{=$schemaValue}}'{{?}}#}}

{{## def.concatSchemaEQ:{{?$isData}}' + {{=$schemaValue}} + '{{??}}{{=it.util.escapeQuotes($schema)}}{{?}}#}}

{{## def.\_errorMessages = {

'false schema': "'boolean schema is false'",

$ref: "'can\\\'t resolve reference {{=it.util.escapeQuotes($schema)}}'",

additionalItems: "'should NOT have more than {{=$schema.length}} items'",

additionalProperties: "'{{? it.opts.\_errorDataPathProperty }}is an invalid additional property{{??}}should NOT have additional properties{{?}}'",

anyOf: "'should match some schema in anyOf'",

const: "'should be equal to constant'",

contains: "'should contain a valid item'",

dependencies: "'should have {{? $deps.length == 1 }}property {{= it.util.escapeQuotes($deps[0]) }}{{??}}properties {{= it.util.escapeQuotes($deps.join(\", \")) }}{{?}} when property {{= it.util.escapeQuotes($property) }} is present'",

'enum': "'should be equal to one of the allowed values'",

format: "'should match format \"{{#def.concatSchemaEQ}}\"'",

'if': "'should match \"' + {{=$ifClause}} + '\" schema'",

\_limit: "'should be {{=$opStr}} {{#def.appendSchema}}",

\_exclusiveLimit: "'{{=$exclusiveKeyword}} should be boolean'",

\_limitItems: "'should NOT have {{?$keyword=='maxItems'}}more{{??}}fewer{{?}} than {{#def.concatSchema}} items'",

\_limitLength: "'should NOT be {{?$keyword=='maxLength'}}longer{{??}}shorter{{?}} than {{#def.concatSchema}} characters'",

\_limitProperties:"'should NOT have {{?$keyword=='maxProperties'}}more{{??}}fewer{{?}} than {{#def.concatSchema}} properties'",

multipleOf: "'should be multiple of {{#def.appendSchema}}",

not: "'should NOT be valid'",

oneOf: "'should match exactly one schema in oneOf'",

pattern: "'should match pattern \"{{#def.concatSchemaEQ}}\"'",

propertyNames: "'property name \\'{{=$invalidName}}\\' is invalid'",

required: "'{{? it.opts.\_errorDataPathProperty }}is a required property{{??}}should have required property \\'{{=$missingProperty}}\\'{{?}}'",

type: "'should be {{? $typeIsArray }}{{= $typeSchema.join(\",\") }}{{??}}{{=$typeSchema}}{{?}}'",

uniqueItems: "'should NOT have duplicate items (items ## ' + j + ' and ' + i + ' are identical)'",

custom: "'should pass \"{{=$rule.keyword}}\" keyword validation'",

patternRequired: "'should have property matching pattern \\'{{=$missingPattern}}\\''",

switch: "'should pass \"switch\" keyword validation'",

\_formatLimit: "'should be {{=$opStr}} \"{{#def.concatSchemaEQ}}\"'",

\_formatExclusiveLimit: "'{{=$exclusiveKeyword}} should be boolean'"

} #}}

{{## def.schemaRefOrVal: {{?$isData}}validate.schema{{=$schemaPath}}{{??}}{{=$schema}}{{?}} #}}

{{## def.schemaRefOrQS: {{?$isData}}validate.schema{{=$schemaPath}}{{??}}{{=it.util.toQuotedString($schema)}}{{?}} #}}

{{## def.\_errorSchemas = {

'false schema': "false",

$ref: "{{=it.util.toQuotedString($schema)}}",

additionalItems: "false",

additionalProperties: "false",

anyOf: "validate.schema{{=$schemaPath}}",

const: "validate.schema{{=$schemaPath}}",

contains: "validate.schema{{=$schemaPath}}",

dependencies: "validate.schema{{=$schemaPath}}",

'enum': "validate.schema{{=$schemaPath}}",

format: "{{#def.schemaRefOrQS}}",

'if': "validate.schema{{=$schemaPath}}",

\_limit: "{{#def.schemaRefOrVal}}",

\_exclusiveLimit: "validate.schema{{=$schemaPath}}",

\_limitItems: "{{#def.schemaRefOrVal}}",

\_limitLength: "{{#def.schemaRefOrVal}}",

\_limitProperties:"{{#def.schemaRefOrVal}}",

multipleOf: "{{#def.schemaRefOrVal}}",

not: "validate.schema{{=$schemaPath}}",

oneOf: "validate.schema{{=$schemaPath}}",

pattern: "{{#def.schemaRefOrQS}}",

propertyNames: "validate.schema{{=$schemaPath}}",

required: "validate.schema{{=$schemaPath}}",

type: "validate.schema{{=$schemaPath}}",

uniqueItems: "{{#def.schemaRefOrVal}}",

custom: "validate.schema{{=$schemaPath}}",

patternRequired: "validate.schema{{=$schemaPath}}",

switch: "validate.schema{{=$schemaPath}}",

\_formatLimit: "{{#def.schemaRefOrQS}}",

\_formatExclusiveLimit: "validate.schema{{=$schemaPath}}"

} #}}

{{## def.schemaValueQS: {{?$isData}}{{=$schemaValue}}{{??}}{{=it.util.toQuotedString($schema)}}{{?}} #}}

{{## def.\_errorParams = {

'false schema': "{}",

$ref: "{ ref: '{{=it.util.escapeQuotes($schema)}}' }",

additionalItems: "{ limit: {{=$schema.length}} }",

additionalProperties: "{ additionalProperty: '{{=$additionalProperty}}' }",

anyOf: "{}",

const: "{ allowedValue: schema{{=$lvl}} }",

contains: "{}",

dependencies: "{ property: '{{= it.util.escapeQuotes($property) }}', missingProperty: '{{=$missingProperty}}', depsCount: {{=$deps.length}}, deps: '{{= it.util.escapeQuotes($deps.length==1 ? $deps[0] : $deps.join(\", \")) }}' }",

'enum': "{ allowedValues: schema{{=$lvl}} }",

format: "{ format: {{#def.schemaValueQS}} }",

'if': "{ failingKeyword: {{=$ifClause}} }",

\_limit: "{ comparison: {{=$opExpr}}, limit: {{=$schemaValue}}, exclusive: {{=$exclusive}} }",

\_exclusiveLimit: "{}",

\_limitItems: "{ limit: {{=$schemaValue}} }",

\_limitLength: "{ limit: {{=$schemaValue}} }",

\_limitProperties:"{ limit: {{=$schemaValue}} }",

multipleOf: "{ multipleOf: {{=$schemaValue}} }",

not: "{}",

oneOf: "{ passingSchemas: {{=$passingSchemas}} }",

pattern: "{ pattern: {{#def.schemaValueQS}} }",

propertyNames: "{ propertyName: '{{=$invalidName}}' }",

required: "{ missingProperty: '{{=$missingProperty}}' }",

type: "{ type: '{{? $typeIsArray }}{{= $typeSchema.join(\",\") }}{{??}}{{=$typeSchema}}{{?}}' }",

uniqueItems: "{ i: i, j: j }",

custom: "{ keyword: '{{=$rule.keyword}}' }",

patternRequired: "{ missingPattern: '{{=$missingPattern}}' }",

switch: "{ caseIndex: {{=$caseIndex}} }",

\_formatLimit: "{ comparison: {{=$opExpr}}, limit: {{#def.schemaValueQS}}, exclusive: {{=$exclusive}} }",

\_formatExclusiveLimit: "{}"

} #}}