Title	File Reader API
Document ID	A000000021 +++
Short description	An overview of an API to read server-side files.
Status	WIP
Created	29-SEPT-2020
Modified	15-MAR-2021
Current version	1.9
Audience type/s	Internal
Audience level/s	Intermediate
Document type	Technical guide
Project ID/s	PD000000018
JIRA ID/s	JD000001218
Author user ID/s	MM-091824
Author name/s	Mark Mehmet
Additional information where applicable	Related documents Revision history Periodic review history Sign-off history

+++ This table is an example of an optional, automatically-generated document information page for inclusion in internal documents only. The information shown is sourced from a documentation repository, created and maintained as part of a documentation development and review lifecycle. The hyperlinks in this table are for illustration only and do not link to any internal or external resources.

File Reader API

An overview of an API to read server-side files

Date

Topics

Diagrams

File Reader API tables

Code Examples

File Reader API overview

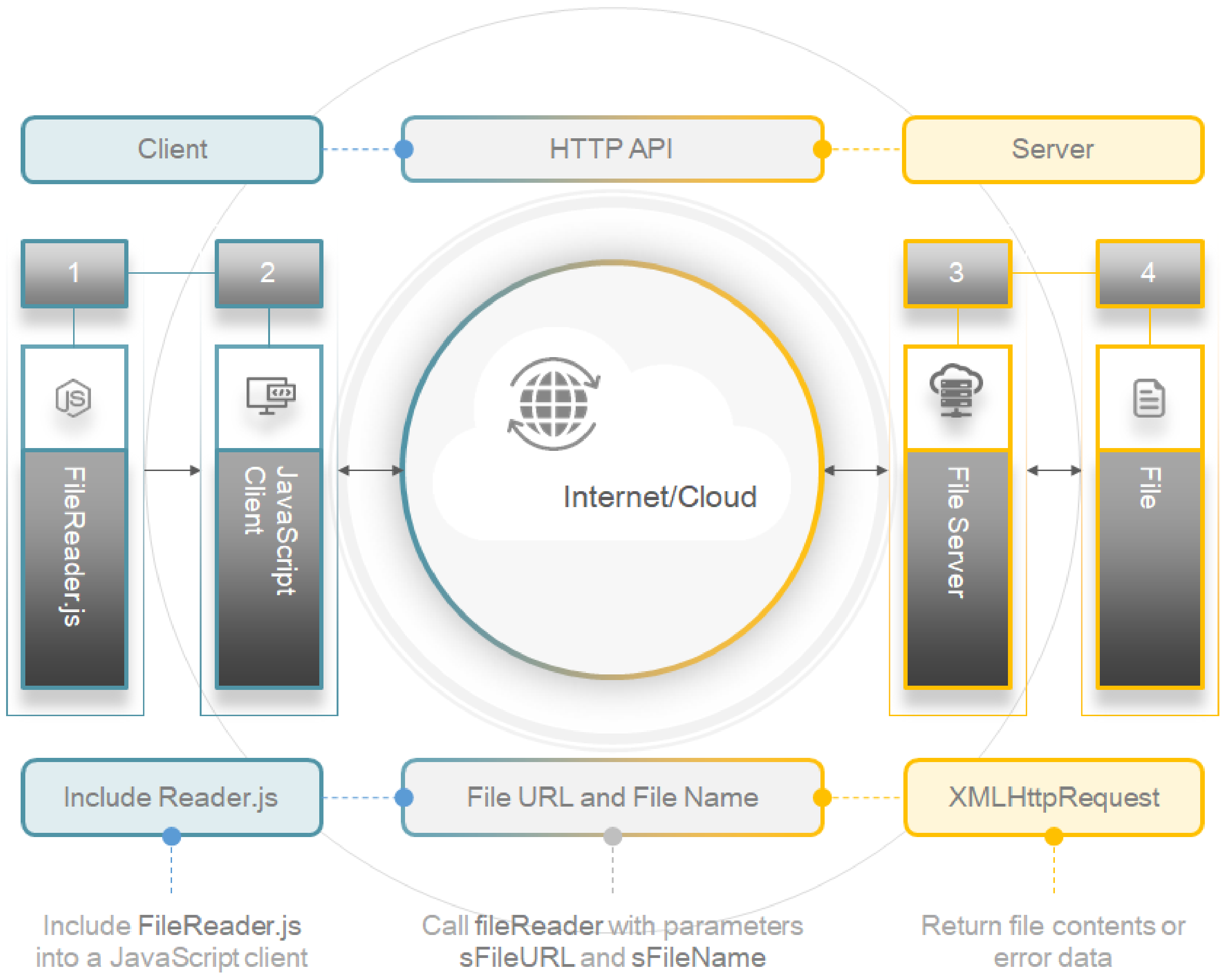
Step-by-step usage instructions are available in File Reader API example use.

File Reader is a simple API for reading server-side text files. It uses an additional, embedded API - the XMLHttpRequest. The high-level process flow for the File Reader, and the placement of File Reader components in this process, are illustrated and introduced in File-Reader-API-diagram-1. Visual overview.

This article contains the following key sections.

- File Reader API logic overview - flowchart of the program logic.
- File Reader API structured code design - for readability and ease of maintenance.
- File Reader API optimised code design - for byte-size reduction and execution efficiency.
- File Reader API example use - a step-by-step guide for use with a web client.

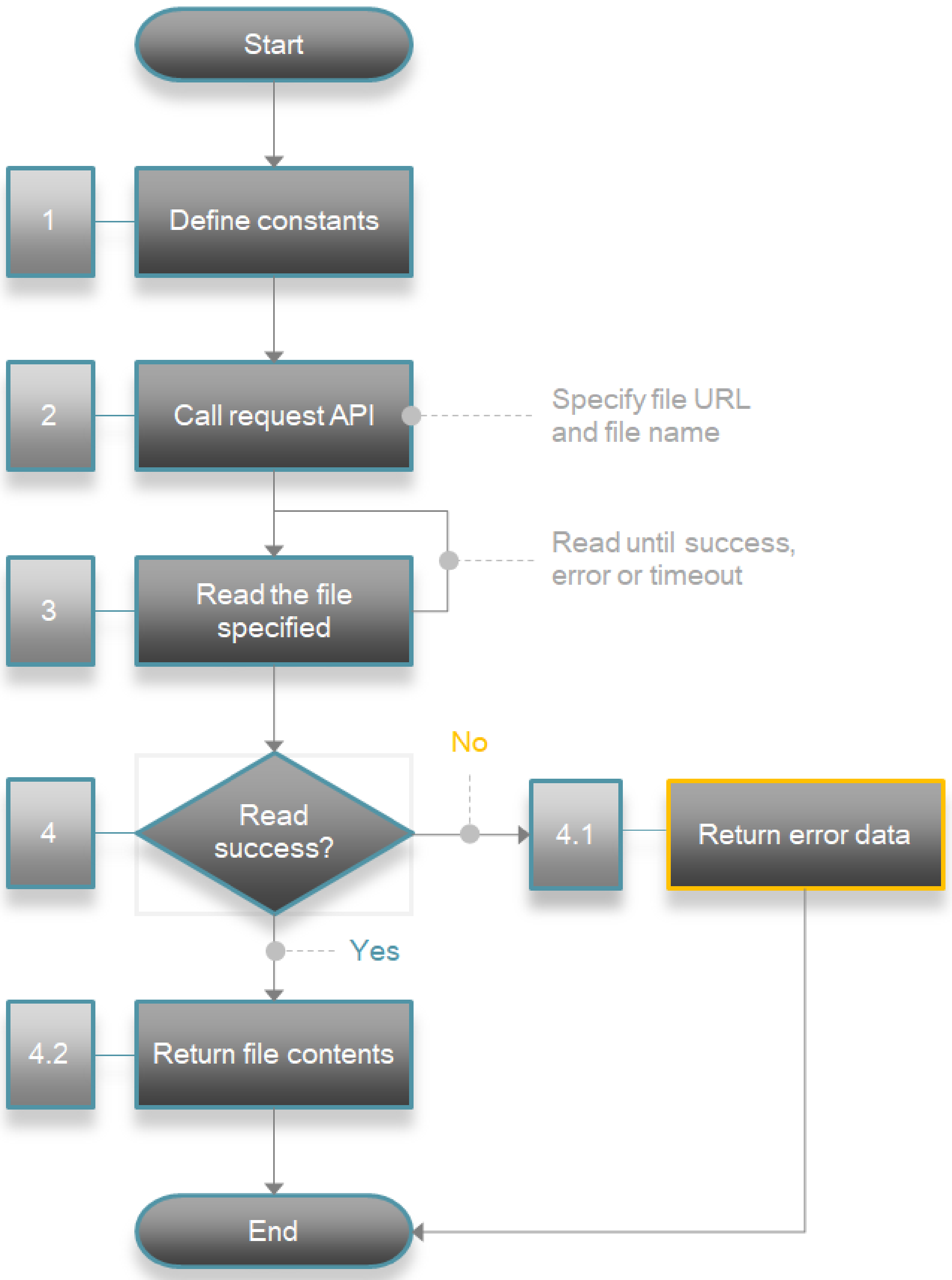
File-Reader-API-diagram-1. Visual overview



File Reader API logic overview

File-Reader-API-diagram-2. Logic overview illustrates the logic in File Reader. Two versions of File Reader are available. These versions are introduced and compared in File-Reader-API-table-1. Comparing structured and optimised code versions. A **structured code version**, designed to be modular and easier to read, and an **optimised code version**, designed for byte and execution efficiency, but with less readable code. The logic presented in File-Reader-API-diagram-2. Logic overview is the same for both versions. All components of the structured code are listed and described in File Reader API supplement 1. Source code component list.

File-Reader-API-diagram-2. Logic overview



File-Reader-API-table-1. Comparing structured and optimised code versions

	Criteria	Structured code version	Optimised code version
01	GitHub URLs ###	FileReader.js	FileReader.js
02	Main design objectives	Structure, modularity, readability and ease of maintenance	Byte-efficiency, quicker execution, smaller footprint
	Byte/line counts	`1947/103`	`574/27` (~70% decrease)
03	Behaviour summary	Functionality is distributed across two functions – the entry function `fileReader`, and `fileStatus` called from `fileReader`.	All functionality in one function - `fileR`, the equivalent of `fileReader`.
04	Entry function	`fileReader`	`fileR`

05	Function prototypes	`function fileReader`
		`(`
		`    string,`
		`    string`
		`)`
		`{`
		
		`/*`
		`Comments`
		`*/`
		`    return`
		`        fileStatus(object);`
		`}`
		
		`function fileStatus(object)`
		`{`
		
		`/*`
		`Comments`
		`*/`
		`    return`
		`        object.property;`
		`}`
			`function fileR`
			`(`
			`    string,`
			`    string`
			`)`
			`{`
			`    return`
			`        object.property;`
			`}`

[###] These are not actual source code URLs, and have been included for example only. Clicking on the adjacent links will open the home page of the GitHub website.

	Criteria	Structured code version	Optimised code version
06	**Internal comments**	Structured and consistent use of internal comments. An example below.	All internal comments removed
		`/* Function: ... Behaviour: ... Parameter/s: ... Called by: ... Calls: ... Return/s: ... */`	N/A
07	**Naming conventions**	Meaningful constant, variable and function names. Some examples below.	Shorter, less meaningful constant, variable and function names
		`CN_FILE_OK`	`F_OK`
		`fileReader`	`fileR`
		`nStatus`	`nS`
08	**Evaluation of conditions**	Traditional evaluation of conditions. An example below.	Streamlined evaluation of conditions
		`if (nStatus == CN_FILE_OK) { return oFileObj.responseText; } else { return CN_FILE_ERR + nStatus.toString(); }`	`sR = (nS == F_OK ? oF.responseText : F_ERR + nS.toString());`

File Reader API structured code design

The structured version of File Reader consists of the three parts listed below. Each part contains detailed internal comments, documenting function parameters and return values where relevant.

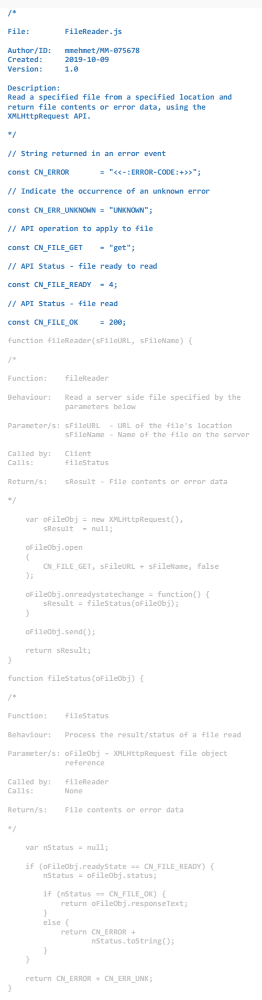
1. File-Reader-API-code-1. A file header consisting of a file description and the definition of constants.
2. File-Reader-API-code-2. The main, or entry function - `fileReader`.
3. File-Reader-API-code-3. The second function - `fileStatus`, called from within `fileReader`.

File-Reader-API-code-1. File Header and Constants Definition

Code (colourisation guide)

```
/*

File:         FileReader.js

Author/ID:    Mark Mehmet/MM-091824
Created:      2019-10-09
Version:      1.0

Description:
Read a specified file from a specified location and
return file contents or error data, using the
XMLHttpRequest API.

*/

// String returned in an error event

const CN_ERROR       = "<<-:ERROR-CODE:+>>";

// Indicate the occurrence of an unknown error

const CN_ERR_UNKNOWN = "UNKNOWN";

// API operation to apply to file

const CN_FILE_GET    = "get";

// API Status - file ready to read

const CN_FILE_READY  = 4;

// API Status - file read

const CN_FILE_OK     = 200;
```

Code pattern



```
/*

File:         FileReader.js

Author/ID:    mmehmet/MM-075678
Created:      2019-10-09
Version:      1.0

Description:
Read a specified file from a specified location and
return file contents or error data, using the
XMLHttpRequest API.

*/

// String returned in an error event

const CN_ERROR       = "<<-:ERROR-CODE:+>>";

// Indicate the occurrence of an unknown error

const CN_ERR_UNKNOWN = "UNKNOWN";

// API operation to apply to file

const CN_FILE_GET    = "get";

// API Status - file ready to read

const CN_FILE_READY  = 4;

// API Status - file read

const CN_FILE_OK     = 200;

function fileReader(sFileURL, sFileName) {

/*

Function:     fileReader

Behaviour:    Read a server side file specified by the
              parameters below

Parameter/s: sFileURL  - URL of the file's location
              sFileName - Name of the file on the server

Called by:    Client
Calls:        fileStatus

Return/s:     sResult - File contents or error data

*/

    var oFileObj = new XMLHttpRequest(),
        sResult  = null;

    oFileObj.open
    (
        CN_FILE_GET, sFileURL + sFileName, false
    );

    oFileObj.onreadystatechange = function() {
        sResult = fileStatus(oFileObj);
    }

    oFileObj.send();

    return sResult;
}

function fileStatus(oFileObj) {

/*

Function:     fileStatus

Behaviour:    Process the result/status of a file read

Parameter/s: oFileObj - XMLHttpRequest file object
              reference

Called by:    fileReader
Calls:        None

Return/s:     File contents or error data

*/

    var nStatus = null;

    if (oFileObj.readyState == CN_FILE_READY) {
        nStatus = oFileObj.status;

        if (nStatus == CN_FILE_OK) {
            return oFileObj.responseText;
        }
        else {
            return CN_ERROR +
                   nStatus.toString();
        }
    }

    return CN_ERROR + CN_ERR_UNK;
}
```

File-Reader-API-code-2. Structured method – fileReader.

Code (colourisation guide)

```
function fileReader(sFileURL, sFileName) {

/*

Function:     fileReader

Behaviour:    Read a server side file specified by the
              parameters below

Parameter/s:  sFileURL  - URL of the file's location
              sFileName - Name of the file on the server

Called by:    Client
Calls:        fileStatus

Return/s:     sResult - File contents or error data

*/

    var oFileObj = new XMLHttpRequest(),
        sResult  = null;

    oFileObj.open
    (
        CN_FILE_GET, sFileURL + sFileName, false
    );

    oFileObj.onreadystatechange = function() {
        sResult = fileStatus(oFileObj);
    }

    oFileObj.send();

    return sResult;
}
```

Code pattern

```
/*

File:        FileReader.js

Author/ID:   mmehmet/MM-075678
Created:     2019-10-09
Version:     1.0

Description:
Read a specified file from a specified location and
return file contents or error data, using the
XMLHttpRequest API.

*/

// String returned in an error event

const CN_ERROR       = "<<-:ERROR-CODE:+>>";

// Indicate the occurrence of an unknown error

const CN_ERR_UNKNOWN = "UNKNOWN";

// API operation to apply to file

const CN_FILE_GET    = "get";

// API Status - file ready to read

const CN_FILE_READY  = 4;

// API Status - file read

const CN_FILE_OK     = 200;

function fileReader(sFileURL, sFileName) {

/*

Function:     fileReader

Behaviour:    Read a server side file specified by the
              parameters below

Parameter/s: sFileURL  - URL of the file's location
             sFileName - Name of the file on the server

Called by:   Client
Calls:       fileStatus

Return/s:    sResult - File contents or error data

*/

    var oFileObj = new XMLHttpRequest(),
        sResult  = null;

    oFileObj.open
    (
        CN_FILE_GET, sFileURL + sFileName, false
    );

    oFileObj.onreadystatechange = function() {
        sResult = fileStatus(oFileObj);
    }

    oFileObj.send();

    return sResult;
}

function fileStatus(oFileObj) {

/*

Function:    fileStatus

Behaviour:   Process the result/status of a file read

Parameter/s: oFileObj – XMLHttpRequest file object
             reference

Called by:   fileReader
Calls:       None

Return/s:    File contents or error data

*/

    var nStatus = null;

    if (oFileObj.readyState == CN_FILE_READY) {
        nStatus = oFileObj.status;

        if (nStatus == CN_FILE_OK) {
            return oFileObj.responseText;
        }
        else {
            return CN_ERROR +
                   nStatus.toString();
        }
    }

    return CN_ERROR + CN_ERR_UNK;
}
```

File-Reader-API-code-3. Structured method – fileStatus

Code (colourisation guide)

```
function fileStatus(oFileObj) {

/*

Function:     fileStatus

Behaviour:    Process the result/status of a file read

Parameter/s: oFileObj – XMLHttpRequest file object
             reference

Called by:    fileReader
Calls:        None

Return/s:     File contents or error data

*/

    var nStatus = null;

    if (oFileObj.readyState == CN_FILE_READY) {
        nStatus = oFileObj.status;

        if (nStatus == CN_FILE_OK) {
            return oFileObj.responseText;
        }
        else {
            return CN_ERROR +
                   nStatus.toString();
        }
    }

    return CN_ERROR + CN_ERR_UNK;
}
```

Code pattern

```
/*

File:        FileReader.js

Author/ID:   mmehmet/MM-075678
Created:     2019-10-09
Version:     1.0

Description:
Read a specified file from a specified location and
return file contents or error data, using the
XMLHttpRequest API.

*/

// String returned in an error event

const CN_ERROR       = "<<-:ERROR-CODE:+>>";

// Indicate the occurrence of an unknown error

const CN_ERR_UNKNOWN = "UNKNOWN";

// API operation to apply to file

const CN_FILE_GET    = "get";

// API Status - file ready to read

const CN_FILE_READY  = 4;

// API Status - file read

const CN_FILE_OK     = 200;

function fileReader(sFileURL, sFileName) {

/*

Function:    fileReader

Behaviour:   Read a server side file specified by the
             parameters below

Parameter/s: sFileURL  - URL of the file's location
             sFileName - Name of the file on the server

Called by:   Client
Calls:       fileStatus

Return/s:    sResult - File contents or error data

*/

    var oFileObj = new XMLHttpRequest(),
        sResult  = null;

    oFileObj.open
    (
        CN_FILE_GET, sFileURL + sFileName, false
    );

    oFileObj.onreadystatechange = function() {
        sResult = fileStatus(oFileObj);
    }

    oFileObj.send();

    return sResult;
}

function fileStatus(oFileObj) {

/*

Function:    fileStatus

Behaviour:   Process the result/status of a file read

Parameter/s: oFileObj – XMLHttpRequest file object
             reference

Called by:   fileReader
Calls:       None

Return/s:    File contents or error data

*/

    var nStatus = null;

    if (oFileObj.readyState == CN_FILE_READY) {
        nStatus = oFileObj.status;

        if (nStatus == CN_FILE_OK) {
            return oFileObj.responseText;
        }
        else {
            return CN_ERROR +
                   nStatus.toString();
        }
    }

    return CN_ERROR + CN_ERR_UNK;
}
```

File Reader API optimised code design

The optimised version of File Reader, shown in its entirety in File-Reader-API-code-4. Single optimised function has the characteristics enumerated further below. A single function `fileR` combines `fileReader` and `fileStatus` into an optimised form. More byte-count reducing optimisations are possible, such as the removal of unnecessary spaces and indentations. However, these result in further reductions in readability and achieve no material performance gains, especially given the small footprint of code, even in its structured form.

1. Internal comments removed.
2. Constant and variable names shortened.
3. Streamlined evaluation of conditions.
4. Significantly reduced readability.

File-Reader-API-code-4. Single optimised function

Code (colourisation guide)

```
const F_ERR   = "<<-:ERROR-CODE:+>>";
const F_ERR_U = "UNKNOWN ";
const F_GET   = "get";
const F_RS    = 4;
const F_OK    = 200;

function fileR(sU, sF) {
    var oF = new XMLHttpRequest(),
        sR = null,
        nS = null;

    oF.open(F_GET, sU + sF, false);

    oF.onreadystatechange = function() {
        if (oF.readyState == F_RS) {
            nS = oF.status;

            sR = (nS == F_OK ?
                  oF.responseText : F_ERR +
                  nS.toString());
        }
    }

    oF.send();

    return (nS != null ? sR : F_ERR + F_ERR_U);
}
```

Code pattern

```
const F_ERR   = "<<-:ERROR-CODE:+>>";
const F_ERR_U = "UNKNOWN ";
const F_GET   = "get";
const F_RS    = 4;
const F_OK    = 200;

function fileR(sU, sF) {
    var oF = new XMLHttpRequest(),
        sR = null,
        nS = null;

    oF.open(F_GET, sU + sF, false);

    oF.onreadystatechange = function() {
        if (oF.readyState == F_RS) {
            nS = oF.status;

            sR = (nS == F_OK ?
                  oF.responseText : F_ERR +
                  nS.toString());
        }
    }

    oF.send();

    return (nS != null ? sR : F_ERR + F_ERR_U);
}
```

File Reader API example use

Refer to File-Reader-API-table-2. A step-by-step usage guide for using File Reader within a HTML file. For brevity, this example uses the optimised code.

File-Reader-API-table-2. A step-by-step usage guide

Step	HTML File Section and Requirement	Example

01	`<head>`
	Include the JavaScript file `FileReader.js`
		`<head>`
		`    <script type = "text/javascript" \`
		`            src  = "../js/FileReader.js">`
		`    </script>`
		`</head>`

02	`<script>`
	If multiple files are to be read from the same site, define a constant for the website.
		`const CN_URL  =`
		`      "https://www.w3.org/TR/PNG/";`

03	`<script>` Initialise a string to hold the return value of the `fileR` function	`var sFileData = new String();`

04	`<script>`
	Call `fileR`, passing the site and name of file to read.
		`sFileData = fileR`
		`(`
		`    CN_URL, "iso_8859-1.txt"`
		`);`

05	`<script>`
	Process results. Detect an error by checking for the string defined in `F_ERR`.
		`if (sFileData.indexOf(F_ERR) == -1) {`
		`    // Process file contents`
		`}`

06	`<script>`
	Read/process another file if required.
		`sFileData = null;`
		
		`sFileData = fileR`
		`(`
		`    CN_URL, "iso_8859-2.txt"`
		`);`

File-Reader-API-code-5. Example usage of the File Reader API file in a client

Code (colourisation guide)

```
<html>

    <head>

        <script type = "text/javascript" \
                src  = "FileReader.js">
        </script>

    </head>

    <body>

        <script>

            const CN_URL =
                  "https://www.w3.org/TR/PNG/";

            var sFileData = new String();

            sFileData = fileR
            (
                CN_URL, "iso_8859-1.txt"
            );

            if (sFileData.indexOf(F_ERR) == -1) {
                // Process file contents
            }
            else {
                // Process error
            }

        </script>

    </body>

</html>
```

Code Pattern

```
<html>
    <head>
        <script type = "text/javascript" \
                src  = "FileReader.js">
        </script>
    </head>
    <body>
        <script>
            const CN_URL =
                  "https://www.w3.org/TR/PNG/";
            var sFileData = new String();
            sFileData = fileR
            (
                CN_URL, "iso_8859-1.txt"
            );
            if (sFileData.indexOf(F_ERR) == -1) {
                // Process file contents
            }
            else {
                // Process error
            }
        </script>
    </body>
</html>
```

File Reader API supplement 1. Source code component list

All components of the File Reader API, including files, functions, properties, and variables, are listed in File-Reader-API-table-3. Source code component list.

File-Reader-API-table-3. Source code component list

Component	Name	Type	Short description
Files (01)	`FileReader.js`	File	JavaScript file – GitHub URL [***]
API (07)	`XMLHttpRequest`	Constructor	XMLHttpRequest constructor
	`open`	Function	Description on MDN web docs
	`status`	Function	Description on MDN web docs
	`send`	Function	Description on MDN web docs
	`onreadystatechange`	Property	Description on MDN web docs
	`readyState`	Property	Description on MDN web docs
	`responseText`	Property	Description on MDN web docs
Constants (05)	`CN_ERROR`	String	String returned in an error event
	`CN_ERR_UNKNOWN`	String	Indicate the occurrence of an unknown error
	`CN_FILE_GET`	String	API operation to apply to the file
	`CN_FILE_READY`	Integer	API Status - file ready to read
	`CN_FILE_OK`	Integer	API Status - file read

[***] Not an actual source code URL - included for example only. Clicking the link will open the home page of the GitHub website

Component	Name	Type	Short Description
Functions (02)	`fileReader`	Function	Entry function
	`fileStatus`	Function	File processing function
Parameters (03)	`sFileURL`	String	Location of file to read
	`sFileName`	String	Name of file to read
	`oFileObj`	Object	XMLHttpRequest object instance
Local Variables (03)	`oFileObj`	Object	XMLHttpRequest object instance
	`sResult`	String	File contents or error
	`nStatus`	Integer	Status of a file read

File Reader API supplement 2. Program code colour conventions

File-Reader-API-table-4. Program code colour conventions lists the colour conventions used for highlighting various aspects of the code segments presented. This automatically generated colourisation uses CSS. Refer to File Reader API supplement 3. Alternative CSS colour scheme for an example of an alternative colour scheme.

File-Reader-API-table-4. Program code colour conventions

	Convention	Example/s
01	`Constants`	`CN_FILE_READY`
02	`Developer defined functions`	`fileReader` `fileR`
03	`Developer function parameters`	`sFileURL` `oFileObj`
04	`Developer function variables`	`sResult`
05	`Developer function return`	`return oFileObj.responseText;`
06	`Objects in remarks`	`Function: fileReader`
07	`Remarks`	`// Process file contents`
08	`Standard reserved words and symbols`	`; + var new null`
09	`API functions and properties`	`oFileObj.onreadystatechange`
10	`Static strings and numbers`	`"http://www." 200`

File Reader API supplement 3. Alternative CSS colour scheme

Refer to File-Reader-API-table-5. Program code colour conventions using alternative colours for an alternative CSS colour scheme (the style is unchanged.) File-Reader-API-code-6. Example code colourisation using alternative colours shows the optimised code in these colours.

File-Reader-API-table-5. Program code colour conventions using alternative colours

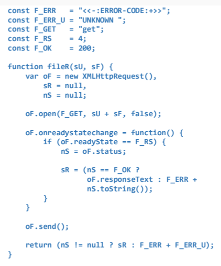
	Convention	Example/s
01	`Constants`	`CN_FILE_READY`
02	`Developer defined functions`	`fileReader` `fileR`
03	`Developer function parameters`	`sFileURL` `oFileObj`
04	`Developer function variables`	`sResult`
05	`Developer function return`	`return oFileObj.responseText;`
06	`Objects in remarks`	`Function: fileReader`
07	`Remarks`	`// File location prefix`
08	`Standard reserved words and symbols`	`; + var new null`
09	`API functions and properties`	`oFileObj.onreadystatechange`
10	`Static strings and numbers`	`"http://www." 200`

File-Reader-API-code-6. Example code colourisation using alternative colours

Code (colourisation guide)

```
const F_ERR   = "<<-:ERROR-CODE:+>>";
const F_ERR_U = "UNKNOWN";
const F_GET   = "get";
const F_RS    = 4;
const F_OK    = 200;

function fileR(sU, sF) {
    var oF = new XMLHttpRequest(),
        sR = null,
        nS = null;

    oF.open(F_GET, sU + sF, false);

    oF.onreadystatechange = function() {
        if (oF.readyState == F_RS) {
            nS = oF.status;

            sR = (nS == F_OK ?
                  oF.responseText : F_ERR +
                  nS.toString());
        }
    }

    oF.send();

    return (nS != null ? sR : F_ERR + F_ERR_U);
}
```

Code Pattern



```
const F_ERR   = "<<-:ERROR-CODE:+>>";
const F_ERR_U = "UNKNOWN ";
const F_GET   = "get";
const F_RS    = 4;
const F_OK    = 200;

function fileR(sU, sF) {
    var oF = new XMLHttpRequest(),
        sR = null,
        nS = null;

    oF.open(F_GET, sU + sF, false);

    oF.onreadystatechange = function() {
        if (oF.readyState == F_RS) {
            nS = oF.status;

            sR = (nS == F_OK ?
                  oF.responseText : F_ERR +
                  nS.toString());
        }
    }

    oF.send();

    return (nS != null ? sR : F_ERR + F_ERR_U);
}
```