ENFOCUS

SWITCH

Node.js TypeScript Cheat Sheet

CREATE A SCRIPT FOLDER (CMD/TERMINAL)
```
cd toTheParentFolderOfTheScriptFolder
SwitchScriptTool --create myScriptFolder .
```
OPEN THE SCRIPT FOLDER (VSC)
OPEN SSCRIPT (SWITCH SCRIPTER)
Define the connections and user properties
WRITE CODE (VSC)
```
main.ts
```
INSTALL PACKAGES
```
npm i packagename
npm i @types/packagename --save-dev
```
TRANSPILE
```
SwitchScriptTool --transpile .
```
PLACE SSCRIPT IN FLOW AND RUN FLOW
If the Debug property is set to Yes, attach to the script in VSC and debug
DEPLOY THE SCRIPT
```
SwitchScriptTool --pack . ../production
```
Copy the property values of the script in the flow, change the script to the packed version and paste the property values

STRING FUNCTIONS
```
search(pattern) : Number
includes(pattern, pos) : Boolean
substring(startIndx,endIndx) : String
replace(pattern,newValue) : String
split(pattern) : String[ ]
toLowerCase() : String
toUpperCase() : String
padStart, padEnd, ...
```
STRING PROPERTY
```
length : Number
```
STATIC STRING FUNCTION
```
let tab = String.fromCharCode(9);
```
ESCAPING DOUBLE QUOTES
```
let str = "\""+someTextinQuotes+"\"";
```
USING BACKTICKS
```
let str = `"Quotes' and even ${aVariable}`;
```

NUMBERS
Basic calculations: +, -, *, /
```
Remainder:  let mod = 10 % 5;
Increment:  let i = 0;
            i++; //i is now 1
            i--; //i is now 0 again
```
MATH CLASS FUNCTIONS
```
Math.round(5.2);  //5
Math.pow(5,2);    //25
Math.sqrt(25);    //5
floor, ceil, sin, cos, etc.
```
CONVERSION FROM STRING
```
let integer = parseInt(str);
let float = parseFloat(str);
```
DECIMALS
```
let float = 123.456789;
let floatStr = float.toFixed(2); //"123.46"
let newFloat = parseFloat(float.toFixed(2));
//123.46
floatStr = float.toFixed(0); //"123"
```

REGULAR EXPRESSIONS
```
//global regular expression
let re = /\s/g;
//regular expression with variables
let re = new RegExp("\d{"+nr+"}");
let matches = str.match(re); //returns []
let present = str.test(re); //boolean
```

DATA TYPE DECLARATIONS
```
String:  let str = "";
         let str : string;
Number:  let nr = 1.2;
         let nr : number;
Array:   let arr = [];
         let arr : [];
Boolean: let bool = true;
         let bool : boolean;
Object   let obj : Record<string,any>;
         obj.key1 = value1;
         obj.key2 = value2;
Date:    let now = new Date();
         let date : Date;
RegExp:  let re = /\d{2}$/;
         let re : RegExp;

let abc = "abc";
abc = 123;  //error in TypeScript

let abc : any = "abc";
abc = 123;  //OK

let strValue : string|string[];
strValue = "abc"; //OK
strValue = ["abc","def"]; //OK
strValue = 123; //error

strValue = numValue.toString();
```

BLOCK-SCOPED VARIABLES
```
let abc = 123;
if (abc == 123) {
    abc = 456;
}
//here abc is 456

let abc = 123;
if (abc == 123) {
    let abc = 456;
    //here abc is 456
}
//here abc is 123
```
The same applies for loops and try-catch blocks.

LOOPS
```
let i = 0;
let n = arr.length;
for (i; i<n; i++) {
    //code;
}

for (let i=0; i<arr.length; i++) {
    //code
    continue; //back to loop
    break;    //exit loop
}

while (i<n) {
    //code;
}

do {
   //code;
} while (i<n);
```

OBJECT ACCESS
```
obj.key1 = value1; //key1 is a valid variable name
obj["key name with spaces"] = value2;
obj["123abc"] = value3;
```

COMPARISONS
```
if (a == b) {  //compares value
}

if (a === b) {  //compares value&type
}

if (a !== b) {  //not equal to
}

if (a < b) {  //and >
}

if (a <= b) {  //and >=
}

if (a == 1 && b == 2) { //AND
}

if (a == 1 || b == 2) { //OR
}
```

CONDITIONS
```
if (a === b) {
    //code;
}

if (a === b) {
    //code;
} else {
    //code;
}

if (a === b) {
    //code;
} else if (a === c) {
    //code;
}
```

COMMENTS
```
//comment line

let a = 1;  //line comment

/* commenting out several
lines of code
let a = 1;
let b = 2;
*/
```

SYNCHRONOUS FUNCTIONS
```
function compare(par1, par2) {
  if (par1 == par2) {
    return true;
  } else {
    return false;
  }
}
```

ASYNCHRONOUS FUNCTIONS
```
async function compare(job, p1, p2) {
  if (p1 == p2) {
    await job.log(LogLevel.Info,"=");
  } else {
    await job.log(LogLevel.Info,"!=");
  }
}
```

ERROR CATCHING
```
try {
    //code;
} catch(error) {
    //code
}
```

SCRIPT ENTRY POINTS

Type CTRL+Space Switch to get a list of available code snippets for entry points

```
jobArrived(s: Switch, flowElement: FlowElement, job: Job) : Promise<void>
//must be present, but can be empty
timerFired(s: Switch, flowElement: FlowElement) : Promise<void>
//default interval: 300 seconds
validateProperties(s: Switch, flowElement: FlowElement, tag: string[]) : Promise<{ tag: string, valid: boolean }[]>
validateConnectionProperties(s: Switch, flowElement: FlowElement, c: Connection, tag: string[]) :
        Promise<{ tag: string, valid: boolean }[]>
getLibraryForProperty(s: Switch, flowElement: FlowElement, tag: string) : Promise<string[]>
getLibraryForConnectionProperty(s: Switch, flowElement: FlowElement, c: Connection, tag: string) : Promise<string[]>
flowStartTriggered(s: Switch, flowElement: FlowElement) : Promise<void>
flowStopTriggered(s: Switch, flowElement: FlowElement) : Promise<void>
httpRequestTriggeredSync(request: HttpRequest, args: any[], response: HttpResponse, s: Switch): Promise<void>
//called directly when the webhook arrives so a custom response can be sent
httpRequestTriggeredAsync(request: HttpRequest, args: any[], s: Switch, flowElement: FlowElement): Promise<void>
//called asynchronously to process the payload
```

JOB PROPERTIES

```
job.getName(includeExtension: boolean = true): string
job.getId() : string
job.isFile() : boolean
job.isFolder() : boolean
All other properties use
await job.getPrivateData(tag); //for reading
await job.setPrivateData(tag, value); //for changing
were tag is one of the following enumerators:
EnfocusSwitch.hierarchy(string[])
EnfocusSwitch.emailAddresses (string[])
EnfocusSwitch.emailBody (string)
EnfocusSwitch.userName (string)
EnfocusSwitch.userFullName (string)
EnfocusSwitch.userEmail (string)
EnfocusSwitch.initiated (string)
EnfocusSwitch.submittedTo (string)
EnfocusSwitch.origin (string) //cannot be set of course
```

CREATING A NEW JOB

```
//to be used in timerFired
//when used in jobArrived the new job does NOT inherit the
//properties of the input job
let pathToNewJobFile = tmp.fileSync().name;
let newJob = await flowElement.createJob(pathToNewJobFile);
await newJob.sendToSingle();

//send a new file as a new job inheriting from the input job
let pathToNewJobFile = tmp.fileSync().name; //use tmp package
let newJob = await job.createChild(pathToNewJobFile);
await newJob.sendToSingle("newjobname.txt");
await job.sendToNull();
fs.unlinkSync(pathToNewJobFile); //do NOT forget this
```

SENDING JOBS

```
//single outgoing connection
await job.sendToSingle(newName? : string);

//outgoing data and log connections
//levels: Success, Warning, Error
await job.sendToData(Connection.Level.Success,newName);
await job.sendToLog(Connection.Level.Success, DatasetModel.XML,
   "Log.xml");

//multiple outgoing connections
let outConnections = flowElement.getOutConnections();
for (let i = 0; i < outConnections.length; i++) {
  await job.sendTo(outConnections[i]);
}

//fail a job
job.fail("Some error message (error code %1)", ["404"]);
return;
```

ACCESSING FLOWELEMENT PROPERTIES

```
let a = (await flowElement.getPropertyStringValue("A")) as string;
let b = (await flowElement.getPropertyStringValue("B")) as [];
//for dependent properties make sure they are used!
```

LOGGING MESSAGES

```
//levels: Info, Warning, Error, Debug
await job.log(LogLevel.Info, "Some message");
await flowElement.log(LogLevel.Deubg, "A debug message");
```

CREATING TEMPORARY FILES

```
import * as tmp from "tmp";
import * as rimraf from "rimraf";
import * as fs from "fs";
let pathToNewJobFile = tmp.fileSync({postfix:".pdf"}).name;
let pathToNewJobFolder = tmp.dirSync().name();

//do NOT forget to clean up before exiting the script
fs.unlinkSync(pathToNewJobFile);
rimraf.sync(pathToNewJobFolder);
```

MANIPULATING DATASETS

```
import * as fs from "fs";
import { DOMParser } from "xmldom";
//read an XML dataset
let xmlDsPath = await job.getDataset("XML",
    AccessLevel.ReadOnly);
let parser = new DOMParser();
let xmlStr = fs.readFileSync(xmlDsPath).toString();
let xmlDOM = parser.parseFromString(xmlStr);
//update a JSON dataset
let jsonDsPath = await job.getDataset("JSON",
    AccessLevel.ReadWrite);
let jsonObject = JSON.parse(fs.readFileSync(jsonDsPath).
toString());
//modify jsonObject
fs.writeFileSync(jsonDatasetPath, JSON.
stringify(jsonObject));
```

SWITCH CLASSES

```
Switch
FlowElement
Job
Connection
ImageDocument
PdfDocument
PdfPage
HttpRequest
HttpResponse
XmlDocument
```

ENUMERATIONS

```
AccessLevel.ReadOnly (ReadWrite)
Connection.Level.Success
  (Warning,Error)
DatasetModel.Opaque
  (XML, XMP,JDF, JSON)
LogLevel.Info
  (Warning, Error, Debug)
PropertyType.Literal
  (Number, Date, String, ...)
Scope.FlowElement
  (FlowElements, Element, Flow,
  Global)
EnfocusSwitchPrivateDataTag.hierarchy
  (emailAddress, emailBody, origin,
  userEmail, UserFullName, ...)
ImageDocument.ColorMode.Bitmap
  (Gray, IndexedColor, ...)
ImageDocument.ColorSpace
HttpRequest.Method.POST
  (PUT, DELETE)
```

SCRIPT EXPRESSIONS

```
async function calculateScriptExpression(s: Switch, flowElement: FlowElement, job:Job): Promise<string | number | boolean> {
  const theDefaultAddress = "addr@host.com";
  const theEmailAddresses = await job.getPrivateData('EnfocusSwitch.emailAddresses');
  if (typeof theEmailAddresses === "string" && theEmailAddresses === "") {
    return theDefaultAddress;
  }
    return theEmailAddresses.join(";");
  }
}
```