Understanding Data URI scheme in Design Automation workflows.

Applicable to all Design Automation Engines.

There is comment in our documentation

> 1. Fail if Autodesk.Das.Shared.Models.Request.Url is a data url, otherwise next step.

This blog tries goes in detail on what data url is and how useful it is wrt to Design Automation.

A Uniform Resource Identifier helps identify a source without ambiguity. Many URI schemes are registered with the IANA; however, Design Automation supports only two kinds of URI schemes for it's request URL.

- http(s)
- data

Scheme	Purpose	Defined by	General format	Notes
https	HTTP connections secured using SSL/TLS	RFC 2817, RFC 7230	`https://<host>:<port>/<path>?<searchpart>`	https://datatracker.ietf.org/doc/html/rfc7230#section-2.7.2 , default Port is 443 if port is not given
http	HTTP resources	RFC 1738, RFC 2616, RFC 7230	`http://<host>:<port>/<path>?<searchpart>`	https://datatracker.ietf.org/doc/html/rfc1738#section-3.3 , default Port is 80 if port is not given
data	Inclusion of small data items inline	RFC 2397	`data:<mediatype>[;base64],<data>`	Only media type `application/json` is support on Design Automation as of today.

The **data URI scheme** is a uniform resource identifier (URI) scheme that provides a way to include data in-line in Web pages as if they were external resources,

```
data:[<mediatype>][;base64],<data>
```

1. mediatype

A **media type** (also known as a **Multipurpose Internet Mail Extensions or MIME type**) indicates the nature and format of a document, file, or assortment of bytes. MIME types are defined and standardized in IETF's RFC 6838.

The Internet Assigned Numbers Authority (IANA) is responsible for all official MIME types, and you can find the most up-to-date and complete list at their Media Types page.

***NOTE Design Automation respects only `application/json` MIME, as defined by RFC 8259: The JavaScript Object Notation (JSON) Data Interchange Format**

2. base64

this is optional, if it is not present in the url, the data should be represented using ASCII encoding for the octets in the range of safe URL characters.

the data should be represented using the standard `%XX hex` encoding of URLS for octets outside that range.

2.0.1. When to URL Encode ?

Before that lets understand what is %XX hex encoding, technically it is called a percent-encoding or URL encoding where an octet encoded as a character triplet, consisting of percent character "%" followed by the two hexadecimal digits representing that octet's numeric value.

Example - `%20` is the percent-encoding for space character.

URI 's includes certain components that are used as delimiting characters, these characters are called reserved characters.

If data for a URI component would conflict with a reserved character's purpose as a delimiter, then the conflicting data must be percent-encoded before the URI is formed.

2.0.1. Reserved Characters

```
[":", "/", "?", "#", "[", "]", "@", "!", "$",
 "&", "'", "(", ")", "*", "+", ",", ";", "="]
```

2.1. Unreserved Characters

Characters that are allowed in a URI but do not have a reserved purpose are called unreserved. These include uppercase and lowercase letters, decimal digits, hyphen, period, underscore, and tilde.

`~_.A-Za-z0-9`

3. Character Encoding Chart

Here is a quick reference chart explaining which characters are “safe” and which characters should be encoded in URLs.

Classification	Included characters	Encoding required?
Safe characters	Alphanumerics `[0-9a-zA-Z]` and unreserved characters. Also reserved characters when used for their reserved purposes (e.g., question mark used to denote a query string)	**NO**
Unreserved characters	`- . _ ~` (does not include blank space)	**NO**
Reserved characters	`: / ? # [ ] @ ! $ & ' ( ) * + , ; =` (does not include blank space)	**YES**
Unsafe characters	Includes the blank/empty space and `" < > % { } | \ ^ ` `	**YES**
ASCII Control characters	Includes the ISO-8859-1 (ISO-Latin) character ranges 00-1F hex (0-31 decimal) and 7F (127 decimal)	**YES**
Non-ASCII characters	Includes the entire “top half” of the ISO-Latin set 80-FF hex (128-255 decimal)	**YES**
All other characters	Any character(s) not mentioned above should be percent-encoded.	**YES**

3.1. What is base64 Encoding

Base64 is an encoding scheme used to represent binary data in an ASCII format. This is useful when binary data needs to be sent over media that are usually designed to handle textual data. Concrete examples would be sending images in an XML file or in an email attachment.

Use this tool -

Free Online Base64 Encoder / Base64 Decoder Tool

***NOTE Design Automation respects both `base64` as well as `URL encode` data**

For all the following examples, below is standard activity is called from different workitem payloads as per the context.

Activity - `dataUrlActivity`

```
{
    "id": "dataUrlActivity",
    "commandLine": [
        "$(engine.path)\\accoreconsole.exe /s $(settings[script].path)"
    ],
    "parameters": {
        "getJson": {
            "verb": "get",
            "description": "inline JSON data in url",
            "required": true,
            "localName": "mad.json"
        },
        "postJson": {
            "verb": "post",
            "description": "post params json data",
            "required": true,
            "localName": "mad.json"
        }
    },
    "engine": "Autodesk.AutoCAD+24_1",
    "description": "DataUrlTest",
    "settings": {
        "script": "(vl-directory-files nil \"*.json\" 0)\n"
    }
}
```

Examples -

1. Basic JSON with safe characters, i.e, data is within the range of safe characters, and not using any reserved or non ascii characters.

The typical workitem to drive the activity is

```
{
    "arguments": {
        "getJson": {
            "url": "data:application/json,{\"filename\": \"test.dwg\"}",
            "verb": "get",
            "localName": "mad.json"
        },
        "postJson": {
            "url": "https://dataurl.requestcatcher.com/",
            "verb": "post",
            "localName": "mad.json"
        }
    },
    "activityId": "madstgacdio.stgexperiments+stg"
}
```

2. JSON data having reserved characters.

   data `{"filename": "!_25_!_test.dwg"}` contains reserved characters `!` which needs to be URL encoded.

```
{
    "arguments": {
        "getJson": {
            "url": "data:application/json,{\"filename\": \"%21_25_%21_test.dwg\"}",
            "verb": "get",
            "localName": "mad.json"
        },
        "postJson": {
            "url": "https://dataurl.requestcatcher.com/",
            "verb": "post",
            "localName": "mad.json"
        }
    },
    "activityId": "madstgacdio.stgexperiments+stg"
}
```

3. JSON data having unicode characters or non-ascii chacters.

   data `{"filename": "M_500_МСК_50 по границе.dwg"}` contains a Russian character, in this case either URL encode or base64 encode the entire JSON data is an accepted format for Design Automation.

   UrlEncode : `{"fileName": "%d0%9c_500_%d0%9c%d0%a1%d0%9a_50%20%d0%bf%d0%be%20%d0%b3%d1%80%d0%b0%d0%bd%d0%b8%d1%86%d0%b5.dwg"}`

   WorkItem

```
{
    "arguments": {
        "getJson": {
            "url": "data:application/json,{\"filename\": \"%D0%9C_500_%D0%9C%D0%A1%D0%9A_50%20%D0%BF%D0%BE%20%D0%B3%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B5.dwg\"}",
            "verb": "get",
            "localName": "mad.json"
        },
        "postJson": {
            "url": "https://dataurl.requestcatcher.com/",
            "verb": "post",
            "localName": "mad.json"
        }
    },
    "activityId": "madstgacdio.stgexperiments+stg"
}
```

The data in the URL `GET` request will first URL decode and written to JSON file on DA server file system.

base64 : `eyJmaWxlbmFtZSI6ICLQnF81MDBf0JzQodCaXzUwINC/0L4g0LPRgNCw0L3QuNGG0LUuZHdnIn0=`

WorkItem

```
{
    "arguments": {
        "getJson": {
            "url": "data:application/json;base64,eyJmaWxlbmFtZSI6ItCcXzUwMF/QnNCh0JpfNTAg0L/QviDQs9GA0LDQvdC40YbQtS5kd2cifQ==",
            "verb": "get",
            "localName": "mad.json"
        },
        "postJson": {
            "url": "https://dataurl.requestcatcher.com",
            "verb": "post",
            "localName": "mad.json"
        }
    },
    "activityId": "madstgacdio.stgexperiments+stg"
}
```

The data in the URL `GET` request will first decoded from base64 and written to JSON file on DA server filesystem.

The `mad.json` data would.

```
{"fileName":"M_500_MCK_50 по границе.dwg"}
```

Exceprt from report.txt

> [06/20/2022 12:40:48] Embedded resource [{"filename":"M_500_MCK_50 по границе.dwg"}] is saved as a file in 'Unicode' at: 'T:\Aces\Jobs\8bb8c2c80fa14e2a885777c863d8f308\mad.json'.

In the codepen Forge-DataUri-Test I have rendered both `bas64` and `urlencode` JSON data.

Node.JS code

```
var json = { filename: "M_500_MCK_50 по границе.dwg" };
var base64 =
  "data:application/json;base64," +
  Buffer.from(JSON.stringify(json)).toString("base64");
var encode = "data:application/json," + encodeURI(JSON.stringify(json));
//Actual Data
console.log(JSON.stringify(json));
//safe url encoded
console.log(encode);
//base64 encoded
console.log(base64);
```

To summarize, data url is helpful scheme when you have small data that you would like embed inline without having the data to store behind any server storage space.