happiest minds
AI FIRST. AGILE ALWAYS.

How to Create and Export Design Tokens in Figma for Scalable UI - Pipeline Details

The previous post established the foundation for this article which analyzes the token extraction pipeline architecture through comprehensive research. It demonstrates how this method differs from Token Studio which serves as a widely used Figma plugin for design token administration. The pipeline serves as the primary system which handles Figma design token extraction, transformation procedures and final token bundling. The two main components include:

Figma API Integration

The codebase establishes a connection with Figma's API through authentication procedures to retrieve design tokens from original source files. The system enables extraction of the latest design information without requiring any manual export processes. The system needs a Figma personal access token combined with the FigmaAPI class which manages both authentication procedures and interaction with Figma's REST endpoints. The Figma API documentation is accessible through this link

Sample code snippet:

```
export default class FigmaApi {
  private baseUrl = 'https://api.figma.com'
  private token: string

  constructor(token: string) {
    this.token = token
  }

  async getAllFigmaVariables(fileKey: string) {
    const resp = await axios.request<GetFigmaVariablesResponse>({
      url: `\({this.baseUrl}/v1/files/\){fileKey}/variables/local`,
      headers: {
        Accept: '*/*',
        'X-Figma-Token': this.token,
      },
    })
    return resp.data
  }
}
```

Key considerations:

Authentication: The system protects personal access tokens by maintaining their storage in environment variables.

File identification: Each Figma file possesses a distinct FILE_KEY which serves to determine the source of its design system.

API versioning: The v1 endpoint provides a stable foundation which supports Figma's ongoing development of its API interface.

Token Transformation: From Figma to CSS

The transformation pipeline serves as the principal component which transforms Figma's REST API results into CSS variables that are ready for production use. The system uses this multi-stage process to manage aliases, multi-mode tokens, color transformations, and special character sanitization. The library Style Dictionary serves as both a vital component and an essential library for this project. The system provides the following advantages

- Custom Format Extensibility - The custom formatter demonstrates Style Dictionary's most powerful feature—complete control over output format while leveraging its core token infrastructure. The system produces mode-specific CSS which maintains alias preservation—this function remains unavailable through standard formatters.
- Concentrates its efforts on producing output format logic through its processing of mode blocks and alias handling operations.
- Enables users to build Android and iOS applications through its free multi-platform support feature.
- Supports the industry-standard token format which matches the Design Tokens Community Group specification requirements.

Stage 1 - Figma Variables to JSON Tokens. The first transformation stage converts Figma's variable format into Style Dictionary-compatible JSON after fetching the variables from Figma.

```
async function main() {
  const fileKey = process.env.FILE_KEY
  const api = new FigmaApi(process.env.PERSONAL_ACCESS_TOKEN)

  // Fetch all figma variables from Figma
  const figmaVariables = await api.getAllFigmaVariables(fileKey)

  // Transform to token files grouped by collection
  const tokensFiles = tokenFilesFromFigmaVariables(figmaVariables)

  // Write JSON files to disk inside the outputDir
  let outputDir = 'tokens_new'
  if (!fs.existsSync(outputDir)) {
    fs.mkdirSync(outputDir)
  }
  // More code
}
```

Depending on the JSON structure, it is not a direct conversion to a compatible JSON. The common challenges that are usually encountered include:

Special characters in names

Figma supports spaces, dots, and hyphens in variablenames. These need to be handled for file system compatibility and for CSS variable naming (spaces to hyphens, dots to delimiters)

Handling modes

When there are multiple modes in a collection (light and dark modes), there is a need for special grouping logic to split tokens by mode while still maintaining a hierarchical structure

Alias preservation

References between variables need to be preserved as aliases (such as {Global.white. primary}) instead of being resolved directly, so that the decision to resolve them is left to downstream tools

Type mapping

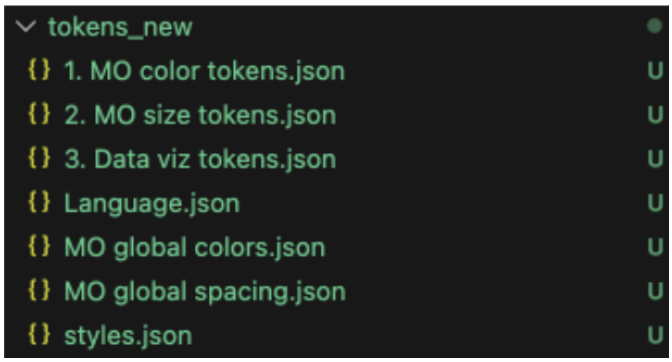
Figma's FLOAT type needs to map to number, BOOLEAN to boolean, and so on, to ensure semantic correctness of the output

A custom method such as **tokenFilesFromFigmaVariables** can be used to reduce the challenges and convert Figma variable format to Style Dictionary-compatible JSON

Sample code snippet:

```
function tokenFilesFromFigmaVariables(figmaVariables: LocalVariable[
  const tokenFiles: { [fileName: string]: any } = {};

  figmaVariables.forEach((variable) => {
    const collection = /* find collection for this variable */;
    const hasMultipleModes = collection.modes.length > 1;

    collection.modes.forEach((mode) => {
      // Sanitize collection name for filesystem
      const collectionName = collection.name.replace(/[/\\?%*:|"<>]/
      const fileName = `${collectionName}.json`;

      // Build hierarchical path
      let path: string[];
      if (hasMultipleModes) {
        // Prefix with mode name for multi-mode collections
        const modeName = mode.name.replace(/\s/g, '');
        path = [modeName, ...variable.name.split('/')];
      } else {
        path = variable.name.split('/');
      }

      // Navigate/create nested structure
      let parent: any = tokenFiles[fileName];
      const tokenName = path.pop()!;

      path.forEach((groupName) => {
        parent[groupName] = parent[groupName] || {};
        parent = parent[groupName];
      });

      // Create token with metadata
      parent[tokenName] = {
        type: figmaTokenType(variable),
        value: figmaTokenValue(variable, mode.modeId, figmaVariables
        description: variable.description
      }
    });
  });

  return tokenFiles;
}
```

By the end of Stage 1, we should be able to locate the newly created token files in the destination that has been specified



Stage 2 - Once the figma design tokens have been extracted into the JSON files, we establish a config for our StyleDictionary. Here, we specify the source files, multi-platform capabilities, and the custom formatter that produces mode-aware CSS with sophisticated alias handling. The code will differ based on the structure of the JSON files and the type of output that is required. In my scenario, it maintains semantic links by outputting original alias notation in mode regions and resolving global primitives. The custom formatter produces five different regions within the output CSS:

Global Color Primitives (:root**) -** Outputs the resolved hex values for the base color palette tokens that lack mode variation.

```
:root {
 --global-white-primary: #ffffff;
 --global-red-90: #ff000f;
 --opacity-black-16: #00000029;
}
```

Mode-Specific Semantic Colors (.mode-light / .mode-dark**) -** Outputs original alias notation ({...} braces) for semantic references, to be converted to var() by postprocessor. Outputs nested elevation classes with box-shadow values resolved.

```
:root {
 --global-white-primary: #ffffff;
 --global-red-90: #ff000f;
 --opacity-black-16: #00000029;
}
```

Spacing & Sizing Tokens (:root**) -** Outputs resolved pixel values for spacing tokens; outputs original aliases for MO size tokens to reference spacing scale.

```
:root {
 --spacing-15: 48px;
 --text-font-size-heading-1: {spacing-15};
}
```

Typography Utility Classes - Breaks down composite typography values and produces CSS utility classes with decomposed properties (font-family, size, weight, line-height)

```
.heading-display-h1 {
  font-family: 'ABCvoice Display';
  font-size: 48px;
  line-height: 64px;
  font-weight: 600;
}
```

Language Mode Overrides (.language-*) - Organizes language-specific font families by locale, allowing **i18n font switching** through class selectors.

```
.language-jp {
  --jp-abcvoice: 'ABCvoice JP';
  --jp-abcvoice-display: 'ABCvoice Display';
}
```

Stage 3 - Postprocessor Normalization. In the final stage, the postprocessor applies normalization. It does the following:

Alias conversion : {Global.white.primary} -> var(--global-white-primary)

Spacing normalization : {spacing-15} -> var(--spacing-15)

Brand token mapping : Maps --brand-white to var(--global-white-primary)

Status deduplication : Eliminates duplicate declarations

Sample code snippet:

```
function aliasToVarName(alias) {
  const m = String(alias).trim().match(/^\{([^}]+)\}$/);
  if (!m) return null;
  // Convert {Global.white.primary} → --global-white-primary
  const pathStr = normalizePrefixes(m[1]).replace(/[.]/g, '-').toLo
  return `--${pathStr}`;
}

function rewriteModeBlockResolveAliases(inner, rootNameToVal, status
  const declRe = /(\s*--([a-z0-9_-]+)\s*:\s*)([^;]+)(;)/gi;

  return inner.replace(declRe, (full, pre, name, val, suf) => {
    const valTrimmed = val.trim();

    // Handle brace aliases: {spacing-15} → var(--spacing-15)
    if (/^\{[^}]+\}$/.test(valTrimmed)) {
      const varName = aliasToVarName(valTrimmed);
      if (varName) {
        return `\({pre}var(\){varName})${suf}`;
      }
    }

    // Handle brand token mapping: --brand-white → var(--global-wh
    if (name === 'brand-white' && rootNameToVal.has('--global-white-
      return `\({pre}var(--global-white-primary)\){suf}`;
    }

    // Pass through literal values unchanged
    return full;
  });
}
```

The final output is a Production-Ready CSS. Mode tokens reference globals through var(), which allows runtime theme switching. Browsers cache var() lookups efficiently. It also Aliases document semantic relationships (--bg-primary -> --global-white). The final CSS output shows the multi-stage process:

```
/* Global colors tokens */
:root {
  --global-white-primary: #ffffff;
  --global-black-primary: #000000;
}

/* Semantic color tokens and elevation styles by mode */
.mode-light {
  --brand-white: var(--global-white-primary);
  --background-base: var(--global-white-primary);

.elevation-4 {
    box-shadow: 0px 0px 1px 0px rgba(0, 0, 0, 0.08), 0px 2px 4px 0p
  }
}

/* Global spacing tokens */
:root {
  --spacing-15: 48px;
}

/* size tokens */
:root {
  --text-font-size-heading-1: var(--spacing-15);
}

/* Language Modes */
.language-jp {
  --jp-abcvoice: 'ABCvoice JP';
  --jp-abcvoice-display: 'ABCvoice Display';
}
```

All the above-mentioned stages can be initiated through the npm script one by one

```
npm run sync-figma-to-tokens (Stage1)

npm run build:modes (Stage 2)

npm run postprocess:modes (Stage 3)
```

Alternative Approach

Instead of the approach that we have discussed, we can also make use of the Token Studio plugin (previously known as Figma Tokens plugin) which helps in the export of design tokens to CSS with variable mappings. It supports:

JSON Token Export: Exports tokens in Style Dictionary-compatible JSON format with alias references

```
{
  "color": {
    "global": {
      "white": {
        "primary": { "value": "#ffffff", "type": "color" }
      }
    },
    "background": {
      "base": {
        "value": "{color.global.white.primary}",
        "type": "color"
      }
    }
  }
}
```

Multi-Mode/Theme Support: Tokens are grouped into theme sets (Light, Dark, and so on)

Style Dictionary Integration: Token Studio JSON can be integrated with Style Dictionary to produce CSS with var() mappings

However, we must consider the advantages and disadvantages before making a choice on which one to use

Advantages

Version Control: JSON tokens are version-controlled with Git, even before Figma sync

Designer Self-Service: No coding required for designers

Cross-Platform: No dependency on Figma; can edit JSON directly

Flexibility: More control over token management

Advantages

Manual Sync: Requires a plugin to push/pull tokens

Plugin Stability: Depends on third-party plugin stability

Paid Service: The essential plan is paid

Not Native Variables: Stays at plugin layer, does not generate native variables

Conclusion

As a recommendation, the pipeline with Figma Variables >>API>> JSON>> Style Dictionary>> CSS>> Postprocessor, etc., appears to be more robust and maintainable, especially if the team can manage the technical complexity; however, Token Studio provides designer self-service, which has its costs and limitations

ABOUT THE AUTHOR

HOMI CHOUDHURY

SENIOR ARCHITECT, PDES

Homi Choudhury is a result-driven Senior Architect with deep experience designing and building enterprise applications. He currently leads AI initiatives and LLM-based architectures, while also driving Model Context Protocol (MCP) adoption for interoperable, enterprise-grade conversational systems. Alongside strong frontend and backend expertise, Homi is known for mentoring teams, optimizing performance, and translating complex technical ideas into clear, practical outcomes for both technical and non-technical audiences.

About Happiest Minds Technologies

Happiest Minds Technologies Limited (BSE, NSE: HAPPSTMNDS) is an AI First, customer-centric digital engineering company committed to delivering 'Happiest People . Happiest Customers'. With an integrated approach that spans from chip to cloud, Happiest Minds delivers secure and scalable solutions across product engineering, cybersecurity, analytics , and automation platforms. Happiest Minds brings purpose and precision to every engagement, helping enterprises solve complex business challenges and fast-track their digital evolution across industry sectors such as Banking, Financial Services & Insurance (BFSI), EdTech, Healthcare & Life Sciences, Hi-Tech and Media & Entertainment, Industrial, Manufacturing, Energy & Utilities, and Retail, CPG & Logistics.

Happiest Minds has been honored by both the Golden Peacock Awards and the Institute of Company Secretaries of India (ICSI) for its exemplary Corporate Governance practices. Guided by its mission of 'Happiest People . Happiest Customers' and consistently recognized as a great place to work, Happiest Minds is headquartered in Bengaluru, India, with a global presence across the Americas, UK, Europe, Australia, the Middle East, Africa, and Asia.

For more information, please write to us at business@happiestminds.com

happiest minds
AI FIRST. AGILE ALWAYS.

www.happiestminds.com