THE HYBRID ARCHITECTURE OF ADOBE EXPERIENCE MANAGER

HOW TO AUTHOR AND DELIVER MULTICHANNEL CONTENT

Executive summary

The ways in which people digitally connect with the world continue to expand, ranging from traditional web browsers and mobile apps to the newest wearable tech, virtual reality, and IoT devices. This technological growth challenges marketers to produce consistent, engaging experiences across multiple channels. IT organizations must not only support this landscape of connected devices but also prepare for the future.

Adobe Experience Manager, the leading solution for content management, features a decoupled, modular architecture and provides extensible capabilities that empower marketers to build fantastic experiences for any channel.

Content management in Experience Manager is built around the concept of fluid experiences. Fluid experiences decouple content and its management from its delivery channels, enabling you to reuse content across any channel quickly and easily.

Content delivery in Experience Manager leverages a platform approach, which Experience Manager Sites, Screens, Assets, Forms, and content services all build upon. This approach creates a standardized, reusable and extensible content delivery architecture that can target any channel.

The channel-agnostic content management and content delivery framework of Experience Manager combine to form a hybrid content management system (CMS) that is capable of owning one channel or supporting multiple channels.

This white paper explains the hybrid architecture of Adobe Experience Manager and how brands can leverage its powerful authoring and delivery mechanisms to engage customers with relevant content on the channel and device of their choice.

CONTENTS

Why a hybrid CMS?

Many CMSs fall into the category of either a traditional or headless CMS. A hybrid CMS combines the concepts of traditional and headless CMSs into a single architecture, resulting in the best of both worlds while mitigating their disadvantages.

Traditional CMS architecture

A traditional CMS manages and delivers content on a single technology stack, minimizing total cost of ownership with efficient system maintenance and training. It controls all of the templating and presentation logic and outputs fully formatted HTML. A traditional CMS is often tightly coupled with a single channel, resulting in a monolithic software architecture that makes multichannel content delivery difficult and costly.

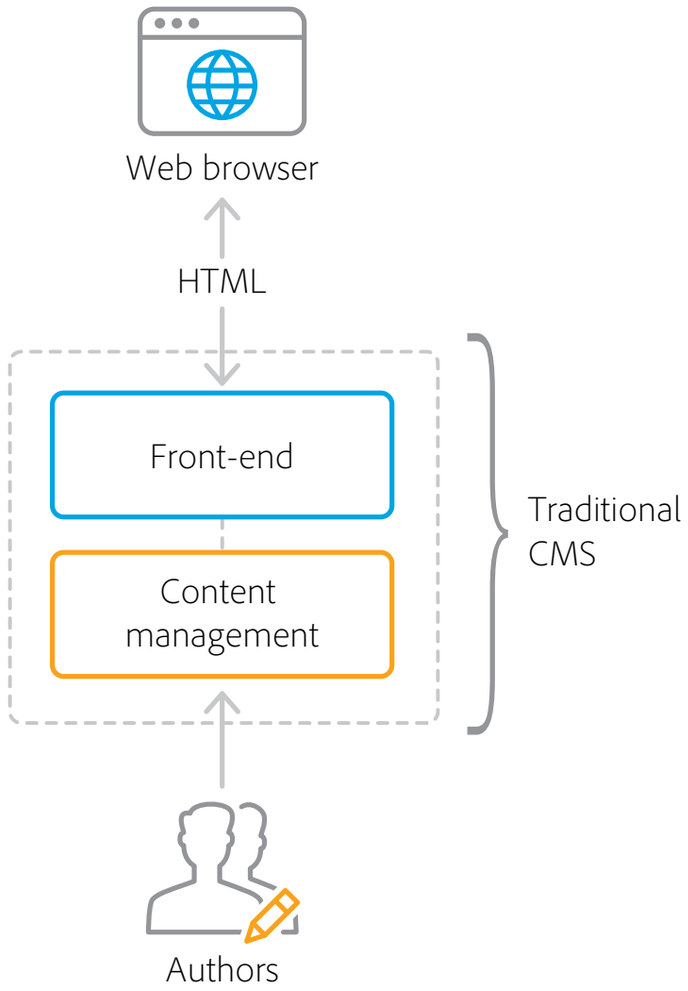


Figure 1. Traditional CMS architecture

Traditional CMS advantages:

- *Enables marketers to offer a consistent message and personalized experiences*
- *Empowers marketers to update presentation and layout with in-context previews*
- *Lowers total cost of ownership initially with a single architecture stack and efficiencies in system maintenance and training*

Traditional CMS disadvantages:

- *Limits development to a single channel, resulting in content silos that are expensive to break through*
- *Slows IT with limited available technology, disempowers front-end developers, and creates a bottleneck between teams*
- *Adds complexity and expense over time as new systems are introduced to address new channels*

Headless CMS architecture

The term ***headless*** originates from the idea that the front-end presentation layer is the "head" of the application. Rather than delivering HTML or formatted content directly, a headless CMS decouples content from presentation, enabling content to be used by a variety of front-end technologies. A headless CMS exposes content through well-defined HTTP APIs.

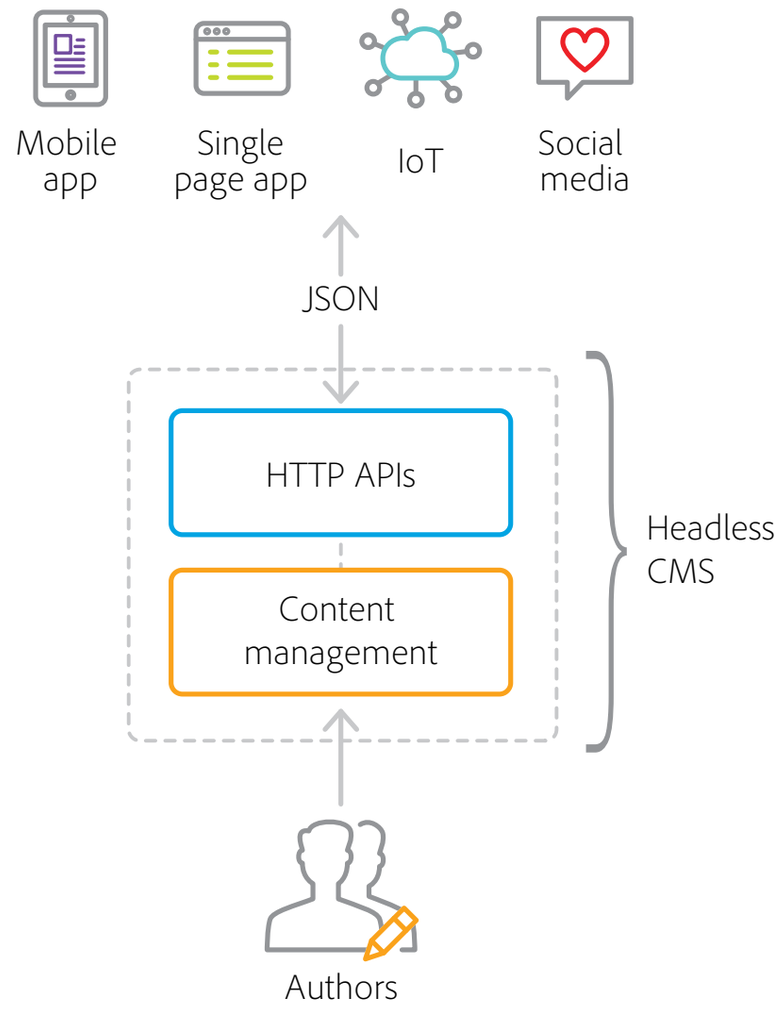


Figure 2. Typical headless architecture

Headless CMS advantages:

- *Scales efficiently to multiple channels and unlocks content for use by any consumer*
- *Empowers IT to use the best technology for the job and to scale work across multiple development teams*
- *Supports new channels*

Headless CMS disadvantages:

- *Increases complexity for security, access control, and personalization*
- *Slows time to market, as presentation changes require developer support, and increases back-and-forth between IT and marketing*
- *Increases total cost of ownership, as IT must develop and maintain a variety of systems and applications*

Experience Manager: A hybrid CMS

Experience Manager takes a hybrid approach that offers the best of both worlds: the efficiency and ease of use of a traditional CMS combined with the flexibility and scalability of a headless development framework. Experience Manager provides a central hub to house and distribute content and experiences to any channel and delivers content either as fully formatted HTML or as JavaScript Object Notation (JSON) over HTTP APIs. With Experience Manager, brands can choose between traditional or headless delivery and can mix and match capabilities to meet their business needs.

Experience Manager offers a suite of capabilities that help brands deliver great digital experiences. With the following capabilities, Experience Manager acts like a traditional CMS and provides channel-centric experiences from end to end:

- Experience Manager Sites is the preeminent leader in experience and web content management (WCM), enabling enterprise and midmarket brands to design, author, manage, publish, and optimize their websites.
- Experience Manager Screens is a powerful tool for the digital signage market, allowing marketers to design, author, manage, schedule, deliver, and optimize dynamic content across digital displays.
- Experience Manager Forms transforms paper-based organizations through digital forms and customer communication solutions.

Recognizing that many other channels exist, Experience Manager includes features that express content in a variety of formats through API endpoints:

- **Content services** exposes content via HTTP APIs using a standard JSON schema, enabling brands to expose content to any channel without coding.
- **Sling Model Exporter** allows developers to quickly render any Experience Manager content into JSON using custom business logic.

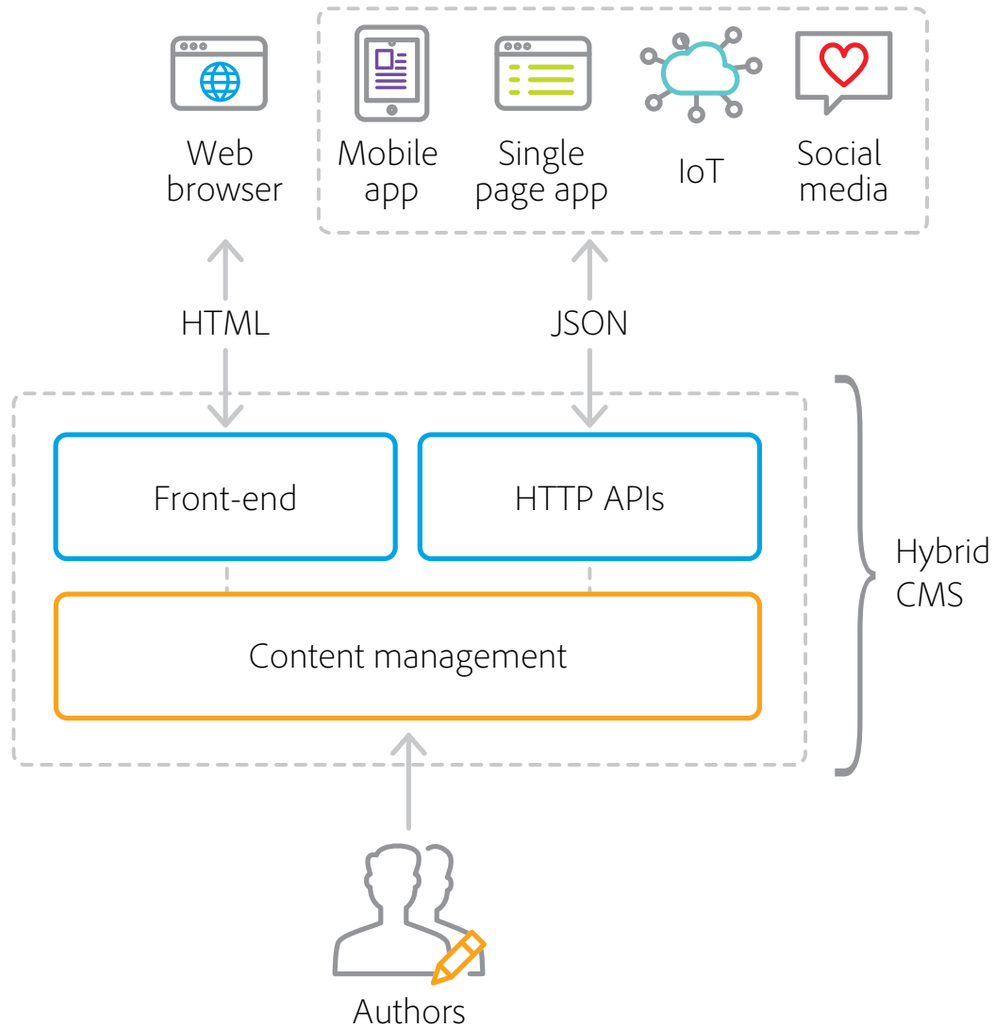


Figure 3. Adobe Experience Manager hybrid architecture

Multichannel authoring with Experience Manager fluid experiences

The hybrid architecture of Experience Manager enables fluid experiences—brands can create content once and deliver to multiple channels. Authors create and manage content in a centralized location, improving efficiency and message consistency while maintaining the flexibility to optimize experiences per channel. Content can be an image, text-based editorial snippets, or a combination of several pieces of content that creates a reusable experience.

The following Experience Manager capabilities and features support fluid experiences:

- **Experience Manager Assets** offers seamless access to image, video, and document assets across organizations, partners, teams, and channels. Marketers and creative teams can work side by side to generate and deliver engaging content.
- **Content fragments** are text-based editorial content that may include some structured data elements, but no design or layout information.
- **Experience fragments** combine several pieces of content, such as text and images, to form an experience that makes sense on its own. They include design and layout information.

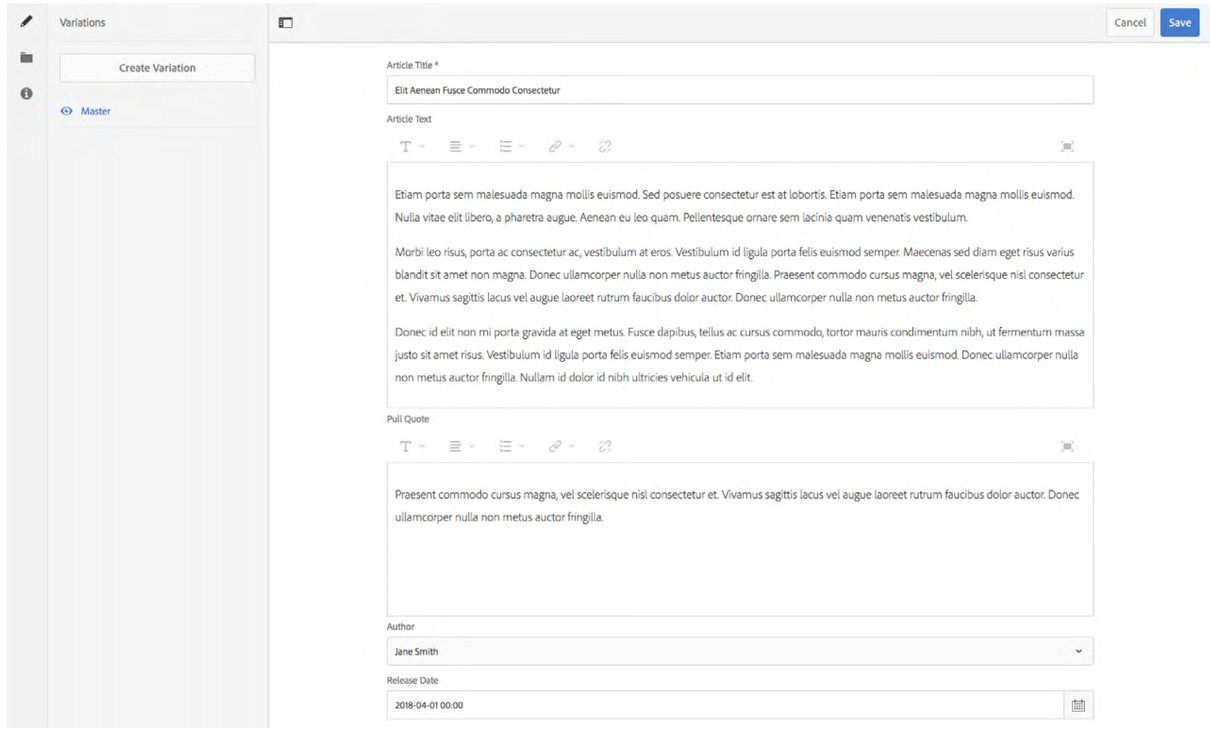


Figure 4. Authoring a content fragment in Experience Manager

Content fragments

A content fragment is a design- and presentation-agnostic set of content. Content fragments can contain unstructured data, for example, text and images, or structured data elements based on a data model. Unstructured content fragments are ideal for articles or other long forms of text, as authors can focus on writing and rely on downstream channels to manage layout and formatting. Structured content fragments are ideal for business-specific data. A text summarization feature enables authors to create variations of content optimized for downstream channels. Experience Manager Assets stores content fragments, so they can take advantage of version control, approval workflows, and translation services.

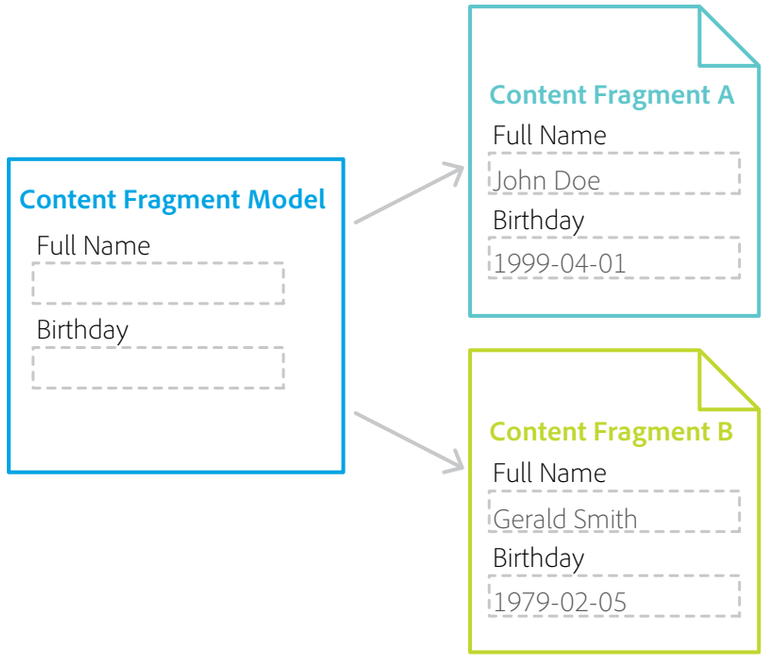


Figure 5. Example of a content fragment model

Content fragment models define the data schema for a content fragment and contain the following fields:

- Text
- Multiline text
- Number
- Boolean
- Date and time
- Enumeration
- Tags
- Reference

A drag-and-drop user interface empowers authors to generate content fragment models quickly and easily without coding, specifying data types, and adding data entry rules.

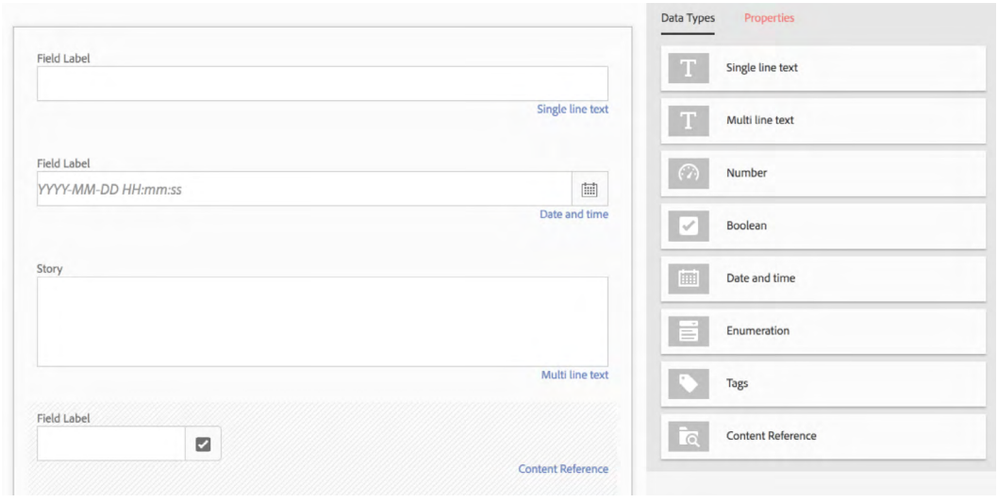


Figure 6. Screen showing how to create a content fragment

From a single content fragment, you can generate one or more variants that adhere to the model and use the variants for different channels.

Use content fragments when the content:

- Is channel agnostic
- Adheres to a structure
- Does not require a specific layout or design
- Is not directly connected to the channels or experiences that expose it

Use content fragments for HTML delivery in:

- An Experience Manager Sites page to transform the content it contains into HTML that can be styled for display on a web page
- An experience fragment to transform the content it contains into HTML that can be styled for a social post—for more information, read about experience fragments

Delivery mechanisms for content fragments

- Combine content fragments with formatting and templating and deliver as fully formatted HTML with out-of-the-box integration with Experience Manager Sites or Screens.
- Deliver in JSON with Experience Manager content services.

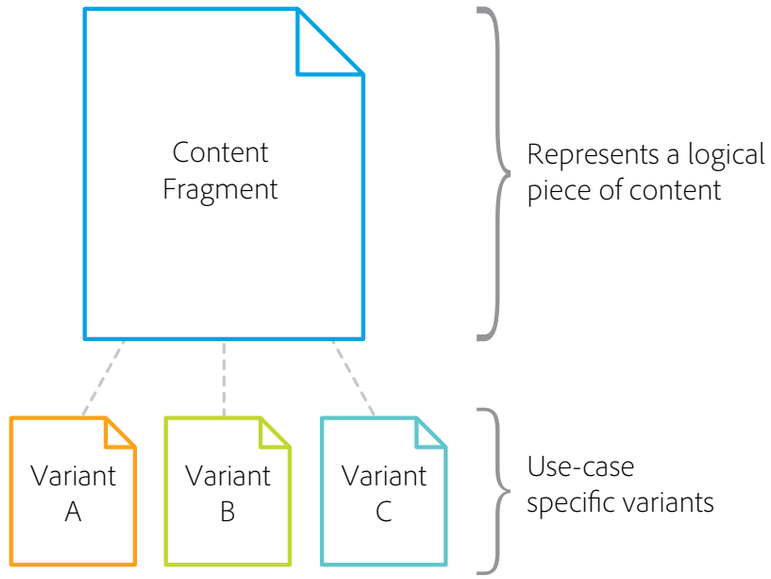


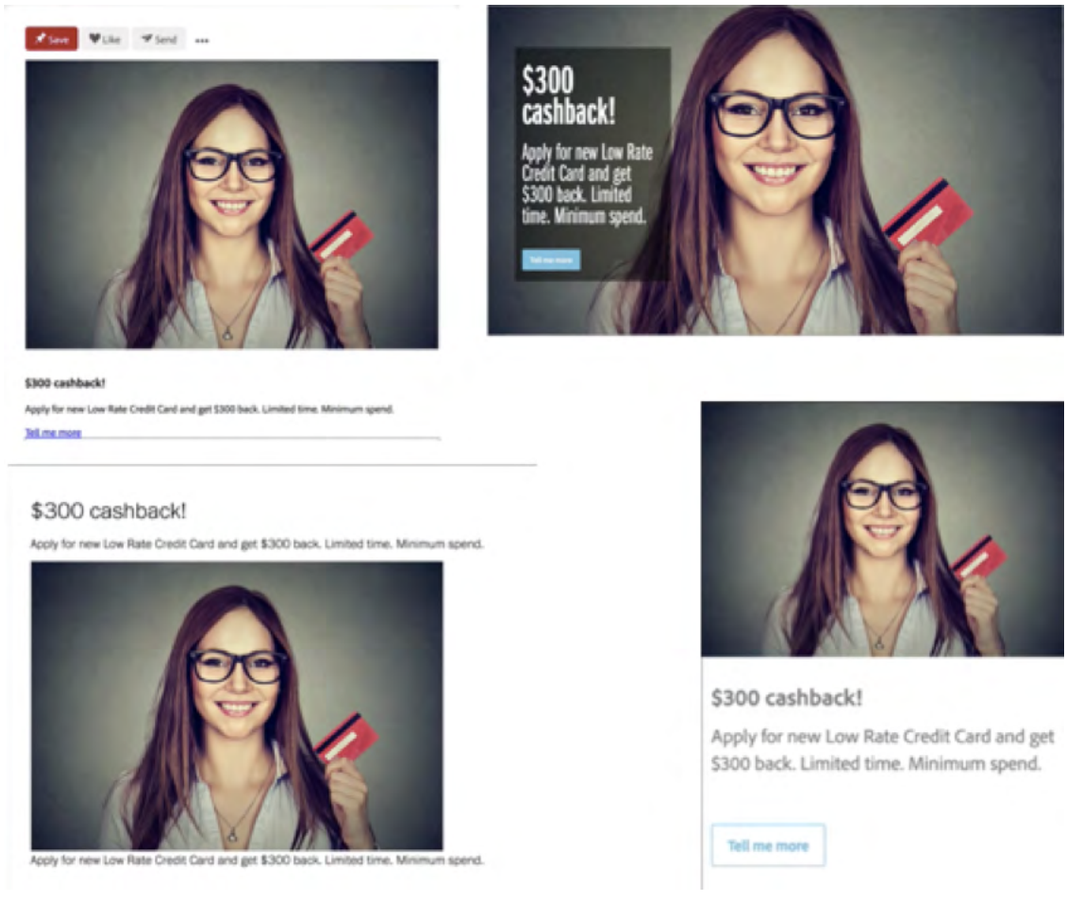
Figure 7. Use-case specific varients

Key considerations—Content fragments:

- *Leverage the features of Experience Manager Assets, including taxonomy, metadata, workflow, versioning, discoverability, security, collaboration, and content intelligence*
- *Leverage localization capabilities in Experience Manager if they are organized by locale in Assets*
- *Adhere to a defined data model, ensuring data consistency; data models can change over time, but changes affect all content fragments that use them*
- *Support long-form editorial text, such as an article, where the text is an element of the article's data model*
- *Are not directly accessed, but rather provide content to channels, which are responsible for managing the presentation of the content fragment*
- *In some cases, content fragment models can create relationships with other pieces of content—for example, a content fragment can reference an image or even another content fragment; however, the reference is path-based, and if the referenced content moves, the path will no longer point to the correct location*

Experience fragments

An experience fragment combines one or more pieces of content with design and layout. A good example of an experience fragment is a promotional experience composed of a banner image, text, and a call to action button. Experience fragments allow marketers to manage experiences from a central location and ensure a consistent message while delivering contextually optimized content to each channel. While experience fragments define standalone experiences, they are designed to optimize display for different channels, such as a web page, social feed, mobile app, or IoT device.



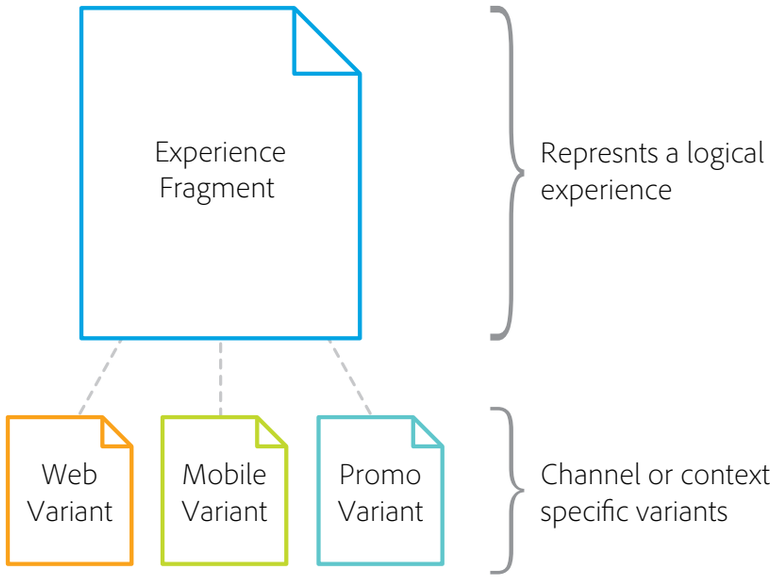


Figure 8. Example of an experience fragment viewed on different channels

Experience fragments, like content fragments, are composed of one or more variations, each addressing a different context or channel, optimizing the presentation of the core experience to best align with the channel and its audience.

Authors create variations using a predefined experience fragment template that supports any channel. Templates can be reused across experience fragments or even across variations of a single experience fragment, accelerating the rate of creation and publishing of channel-specific content. Most experience fragment variations are web based and intended for a browser. Experience Manager provides variations for social posting to Facebook and Pinterest out of the box.

Use experience fragments when the experience:

- Can stand on its own
- Displays differently across channels
- Includes layout or design

Use cases for experience fragments include:

- A website product promotion that is syndicated to third-party websites for cross-promotion
- A campaign synchronized across a brand's website, mobile app, and social media feeds
- A/B testing and audience targeting, enabled by out-of-the-box integration with Adobe Target, that optimizes experience fragment variations by channel or audience

Delivery mechanisms for experience fragments

- Embed server-side with out-of-the-box components into other Experience Manager features like Sites or Screens and deliver as HTML.
- Embed variations into third-party applications over HTTP endpoints and integrate as HTML.
- Post variations to social media channels via respective social media APIs.
- Send variations to Adobe Target for dynamic embedding in a Sites or third-party website.

Key considerations—Experience fragments:

- *Are stored as Experience Manager pages, enabling rich experience curation*
- *Are built on Experience Manager templates and components, simplifying self-service configuration by authors*
- *Support content reuse between variations using the live copy feature, which provides intelligent content synchronization*
- *Support content reuse between variations as well as other experience fragments using build blocks*
- *Provide in-context experience creation, allowing authors to visualize how the experience will appear in other channels*
- *Contain presentation elements, so third-party integration is via HTML snippets*
- *Provide out-of-the box social media templates for Facebook and Pinterest platforms—support for additional social media platforms requires custom code*

Content fragment and experience fragment comparison

	Content fragment	Experience fragment
Definition	Design-agnostic content based on a structured data model	Composition of one or more pieces of content with design and layout applied
Core tenets	Channel agnostic	Experience specific
	Structured data based on content fragment models	Unstructured data composed of Experience Manager components.
	Sets of raw data, stored as primitive data types: text, numbers, date and times, Booleans, or references	Rich experiences consisting of design, layout, and behavior, composed of one or more disparate pieces of data
Variations	Use-case specific	Channel/context specific
	Sets of raw data, stored as primitive data types: text, numbers, date and times, Booleans, or references.	Rich experiences consisting of design, layout, and behavior, composed of one or more disparate pieces of data
Technical implementation	Experience Manager digital asset management (DAM): Asset JCR node type	Experience Manager CQ: Page JCR node type

Multichannel content delivery in Experience Manager

To support all channels, CMSs must expose the content they manage in a multiformat and scalable way. There are two standard formats for exposing web content: HTML and JSON. HTML has long been the language of the web browser, while JSON is a lightweight delivery format that powers modern connected applications.

Experience Manager uses templates, pages, and components to compose content and deliver it via both HTML and JSON. Components can reference and reuse content abstraction features, such as content fragments, experience fragments, and assets, within the same channel or across channels. The following sections will explore in detail how Experience Manager delivers content via HTML and JSON.

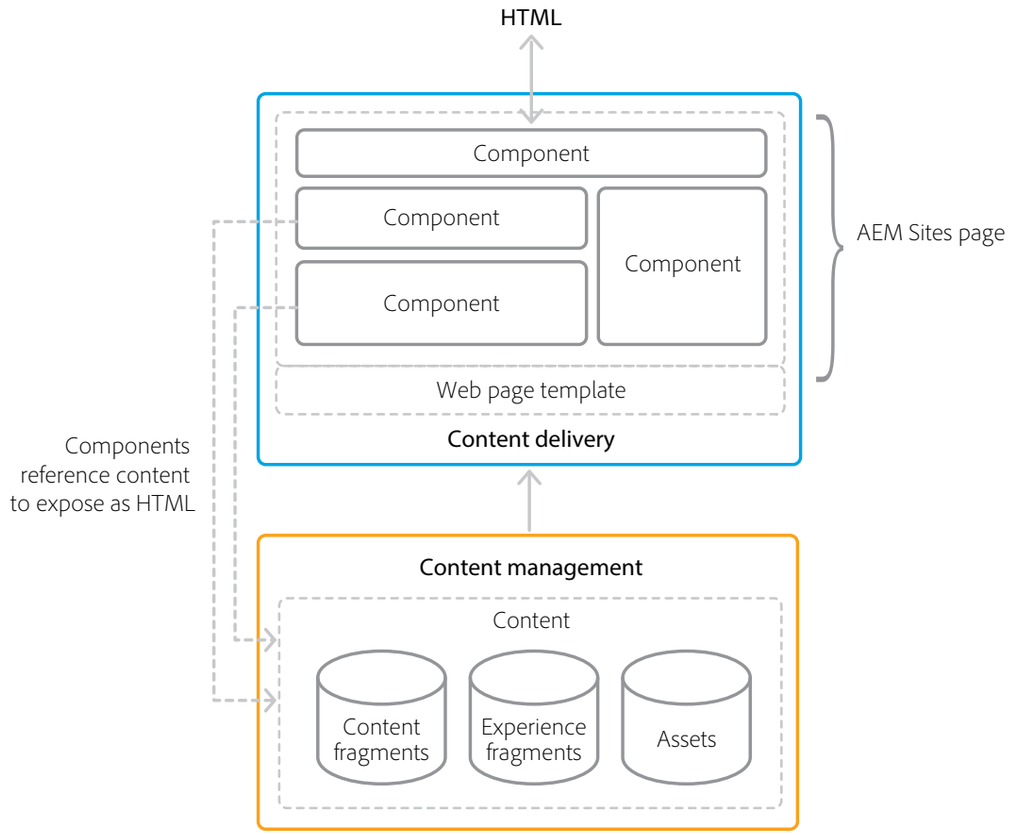


Figure 9. Delivering HTML with Experience Manager

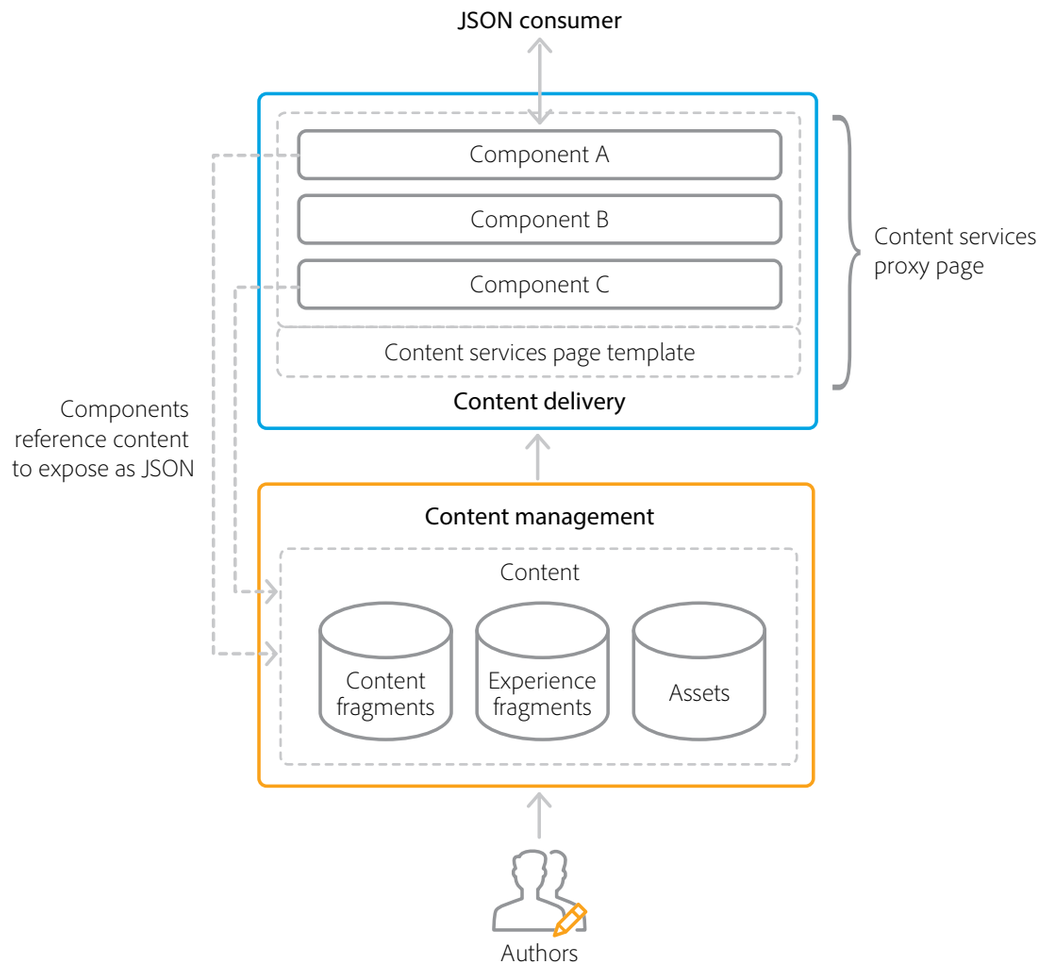


Figure 10. JSON output in Experience Manager

Delivering HTML content in Sites, Screens, and Forms

Experience Manager can operate as a traditional web CMS for delivering HTML in Sites, Screens, and Forms. Templates define layout, content, and configurations and are responsible for including and supporting assets like CSS, JavaScript, and fonts. When used for web pages, templates also dictate which components authors can use on pages. A WYSIWYG, in-context authoring experience grants authors complete control over content layout and formatting. Components render in HTML and allow authors to configure, aggregate, or reference Experience Manager content or create new content inline.

WCM components

Using components, Experience Manager enables brands to reuse content across pages and channels and render the content as HTML at runtime. This structure promotes consistency across channels and offers a single, centralized location for updating content. However, authors also have the flexibility to modify components for a particular page. For example, authors can reference an image component stored in the DAM system, include it on a web page, and modify its properties for that page only.

Experience Manager includes a set of core components—such as breadcrumbs, lists, navigation, text, images, and more—that speed up website development. New projects should use core components as a starting point. To meet business requirements, developers can create custom components that extend core component functionality or implement entirely new functionality.

Authors drag and drop components and combine them with other components on a web page. A component on a page is an instance of the component and contains a **`sling:resourceType`** property that refers to the component. Dialog boxes enable authors to modify the properties associated with the component instance, so they can reuse components across pages and channels. HTML Template Language (HTL) dynamically reads component properties and renders HTML. One or more HTL scripts are typically associated with a component. HTL scripts often rely on a Sling Model to handle complex business logic.

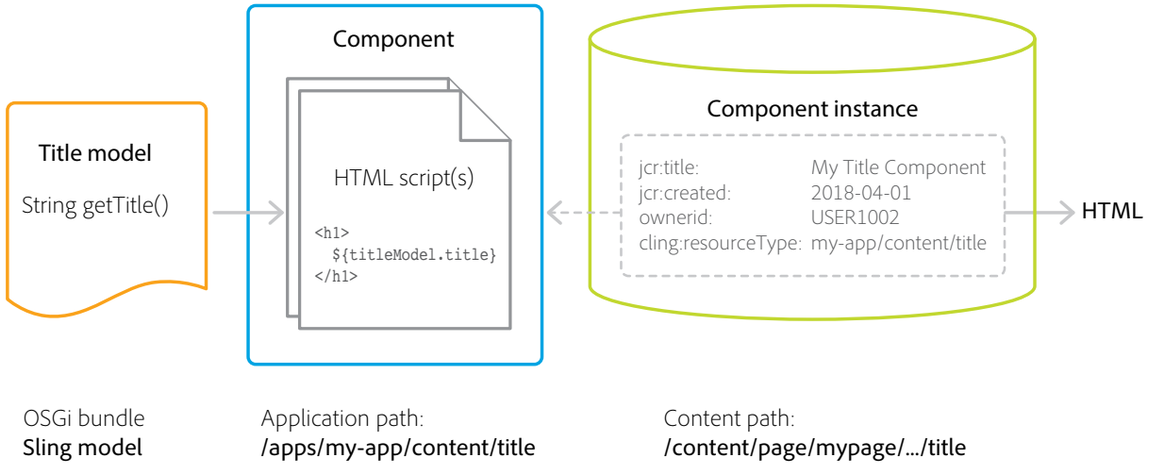


Figure 11. One or more HTL scripts reference a WCM component and dynamically render content as HTML.

Content fragment and experience fragment components

Experience Manager extends the same component structure and flexibility to content fragments and experience fragments. Authors can reference a content fragment or experience fragment variation for a particular page and render the content as HTML. Using the content fragment or experience fragment component, authors can modify components to point to a different fragment variation. They can also modify the layout and position fragments on the page. Since the content of the fragment itself is read-only when referenced via a component, authors create and update content directly in the fragment, promoting content consistency and reuse.

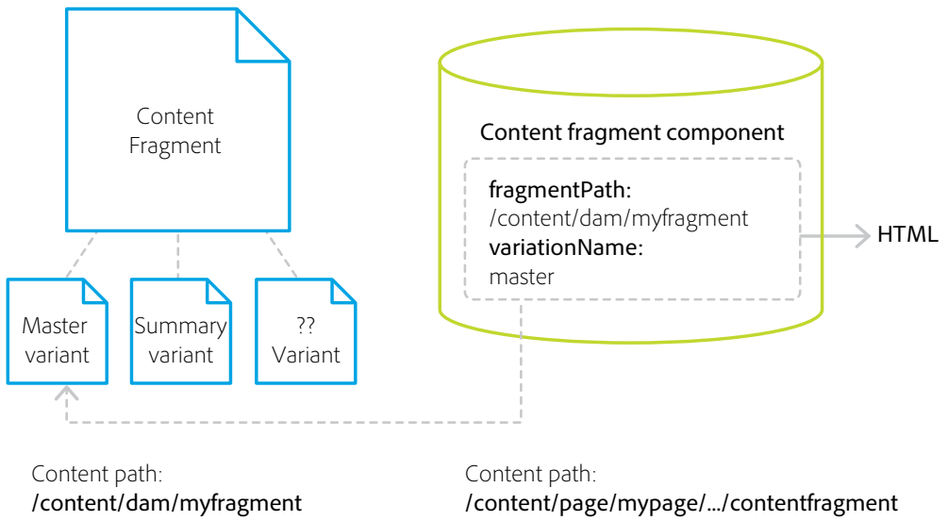


Figure 12. A content fragment component references the master variation of a content fragment titled "My Content Fragment." The content of the master variation renders as HTML on the page based on the HTTP request with an .html extension.

Key considerations—Content fragment components for HTML:

- Are a subset of Experience Manager core components available on GitHub
  - Support two options for rendering a content fragment: single text element and multiple elements:
  - Single text element renders multiline text of a content fragment, such as an article body. This mode automatically creates drop zones between paragraphs, allowing content authors to insert other media into the body of the content.
- Multiple elements render each data element of the content fragment, including field labels derived from the content fragment model. This mode is most commonly used in conjunction with Experience Manager content services.
- "Stash" a copy of the content fragment text beneath the component on the page, allowing keyword search queries to find the page, even though all of the content is maintained in the DAM system with content fragments; background processes ensure that as authors update content fragments, any references are also updated
- Can be extended using content fragment APIs

Key considerations—Experience fragment components for HTML:

- Support only web-based variations
- Reference and render only the experience fragment markup on a page; to provide an in-context preview to authors, developers must include any CSS or JavaScript used by the experience fragment web template in the target page template
- Do not support search indexing

Underlying technology: Apache Sling HTTP web framework

Experience Manager is based on Apache Sling, a RESTful web application framework that manages HTTP requests and delivers content. Apache Sling transforms Experience Manager content—such as pages, experience fragments, and components—into HTML. Sling resource resolution stitches together multiple components and their corresponding scripts and delivers a fully formatted HTML page.

While Apache Sling is capable of exposing content in a variety of formats, Adobe recommends it only be used to serve HTML (and supporting assets, JavaScript and CSS) and binaries (images, documents, videos). For JSON formats, Experience Manager provides more robust and extensible tools, as described in the next section.

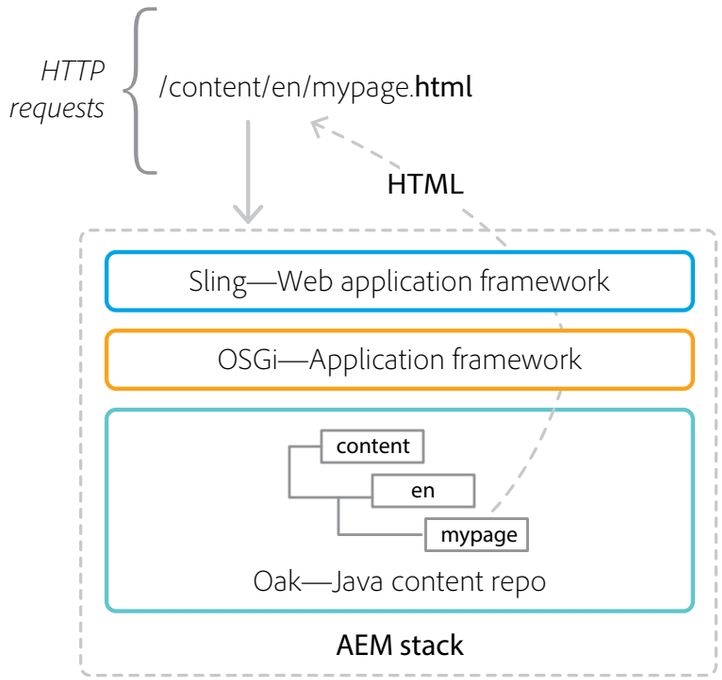


Figure 13. Relationship between Apache Sling and Experience Manager for HTML content delivery

Delivering content in JSON with Experience Manager content services

Experience Manager content services is a zero-code framework that exposes content in a standard JSON format. Content services allow brands to create dedicated HTTP endpoints separate from traditional Experience Manager channels. It serializes content via a JSON exporter, automatically exposing Experience Manager content, such as content fragments, in JSON. This promotes the reuse of Experience Manager content while decoupling the JSON HTTP endpoint from the rest of the application, giving front-end developers a consistent API to code against.

Experience Manager content services builds on top of Apache Sling Model Exporter and provides its own JSON schema for components that implement its interfaces. Content services still leverages Experience Manager templates, pages, and components, but outputs to JSON instead of HTML.

- **Templates** define the top-level JSON schema and structure for downstream consumers by enforcing which components must be on a page and which components authors can add.
- **Pages** define the HTTP API endpoints that deliver JSON. Developers can define HTTP APIs using one or more pages, with each page acting as an HTTP endpoint that delivers a logical set of JSON data.
- **Components** are units of data that allow authors to choose and configure which data to expose. Similar to the HTML use case, authors can configure components to aggregate and reference other content in Experience Manager.

Using the content fragment component to output to JSON format

One of the more powerful aspects of Experience Manager content services is that it exposes both the content fragment and the underlying content fragment model. Brands can render content fragments in HTML and also expose them in JSON format. Content and data elements (label, data type) of a content fragment are exposed as JSON, allowing downstream applications to make intelligent decisions on how best to render the content.

The Experience Manager content services endpoint requests the Experience Manager content services endpoint page using the model.json selector and extension. Any components added to the page are included in the resulting JSON. For a code example of the HTTP request and JSON response, see Appendix A.

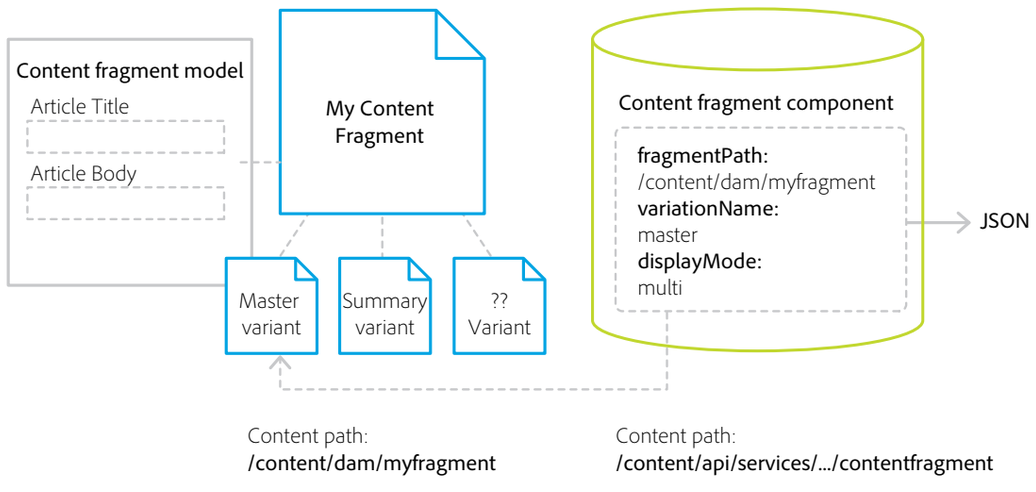


Figure 14. Content services exposes both the content fragment and the underlying content fragment model.

Use content fragment components when:

- You will reuse content across HTML and JSON channels
- Developers need to manage APIs outside Experience Manager deployments
- You require APIs to be rapidly defined and deployed
- APIs could benefit from localization/translation, as Experience Manager content services endpoints can leverage language copy and multisite manager features

Key considerations— Content fragment components for JSON:

- Are a subset of Experience Manager core components available on GitHub
- Automatically expose content as JSON as a part of content services
- Enable marketers to add and update them on Experience Manager pages; content services exposes these pages as API endpoints
- Require strict governance for page templates—a page structure that clearly delineates fixed versus flexible components ensures that marketers, during the course of authoring, cannot introduce changes that break the API endpoints
- Offer limited capabilities to add dynamic business logic without custom code

Developing custom components

Developers can create custom components by implementing the ComponentExporter interface, making them compatible with Experience Manager content services. For more information, see the code example in Appendix B.

Underlying Technology: Apache Sling Model Exporter

The underlying technology of Experience Manager content services is Apache Sling Model Exporter, which builds on the Sling HTTP web framework and provides a mechanism for serializing Sling Model Java objects into any JSON format. Sling Models are developed in Java as annotated POJOs and represent some logical set of content or data. The Sling Model Exporter framework is an extension of Sling Models and facilitates the automatic serialization of the Sling Model object into JSON, using the FasterXML Jackson library.

The Sling Model Exporter binds to content using the content's **`sling:resourceType`** and a selector, allowing developers to attach distinct JSON representations to different content in Experience Manager. HTTP requests to the content resource retrieve the content in its JSON format. The content resource has the matching sling:resourceType set and a customs selector and extension:

HTTP GET /path/to/content.<selector>.json, for example: HTTP GET /content/site/en/products.details.json

Use the Apache Sling Model Exporter when:

- You require a custom JSON schema (different from content services) to expose Experience Manager content
- Exposing existing Experience Manager content that is incompatible with content services, for example, content without content fragments or Sites components not using Sling Models
- You need to dynamically inject complex business logic into the JSON output

Key considerations—Apache Sling Model Exporter:

- Requires custom development that you must maintain
- Should use a different selector than .model to avoid conflicts with Experience Manager content services

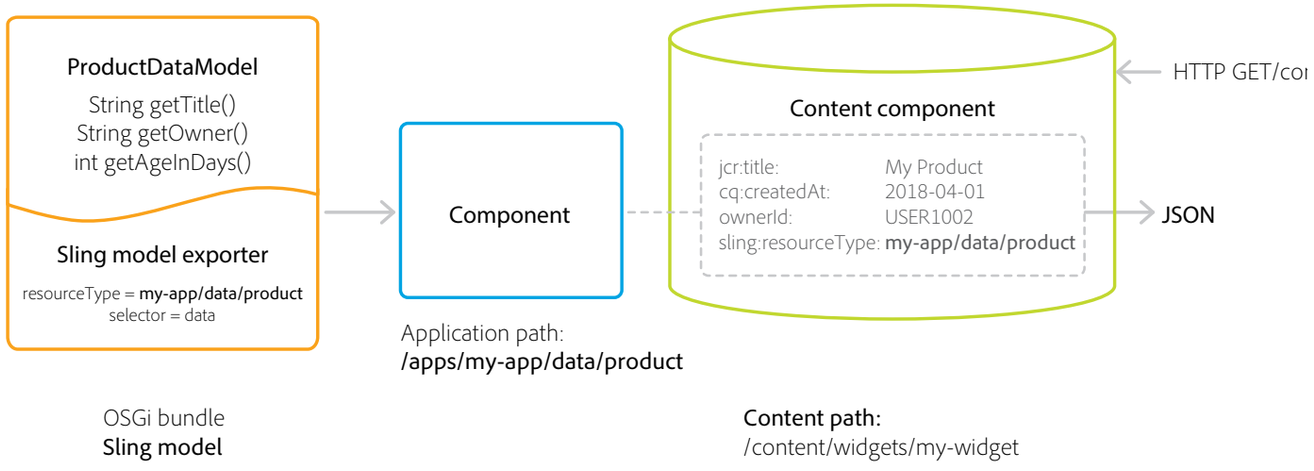


Figure 15. Adding Sling Model Exporter annotation to a Sling Model returns serialized JSON based on a data selector for a content component.

Enabling single-page applications with Experience Manager Sites SPA Editor

Single-page applications (SPAs) are growing in popularity as the performance and responsiveness of web experiences become critical. Developers can build these experiences rapidly with popular JavaScript frameworks such as React and Angular, typically using decoupled or "headless" CMSs. However, this approach cuts marketers and authors out of presentation or layout decisions. Time to market and total cost of ownership increase, as any presentation changes require development support.

Sites solves these problems while enabling developers to use their familiar frameworks and development tools. The Sites SPA Editor offers a WYSIWYG user interface that enables marketers and authors to make in-context changes to content, layout, and presentation, just as they would with traditional web pages. Any changes or updates are immediately reflected in the SPA.

Key considerations—SPA Editor:

- Is intended for use with Sites
- Is powered by content services with JSON content from Sites
- Supports integration with React and Angular frameworks via an SDK
- Enables developers to continue to use familiar front-end technologies to develop the SPA
- Enables marketers to leverage familiar Experience Manager authoring tools to edit the SPA

Summary: HTML Delivery vs. JSON Delivery in Experience Manager

	HTML delivery	JSON delivery
Capability or feature	Sites, Screens, Forms	Content services
Underlying technology	Sling HTTP web framework	Sling Model Exporter
Templates	Responsive layouts that define the structure and appearance of a web page	JSON schema definitions
Pages	Traditional pages forming an Experience Manager website, based on templates and components	HTTP API JSON endpoints adhering to the JSON schema definition that includes templates and components
Components	Modules that collect and present content on a page	Modules that collect and transform content into normalized JSON
Referenced content	Content fragments, experience fragments, Assets	Content fragments, Assets

Best practices for multichannel content delivery

Implementing a CMS that serves all audiences and channels can seem like a daunting task. The following tips will help maximize success.

- **Start small and be agile.** Take a phased approach to building out your content management strategy, and measure your success before implementing larger projects. Remember to react and evolve—Rome wasn't built in a day.
- **Consider future channels today.** Don't try to predict the future. Instead, stay as flexible and extensible as possible.
- **Content is only as good as the creator, so consider the authoring experience.** When focusing on technical strategies for exposing content to a myriad of devices, it is easy to forget about creators. Work with creators and authors to understand how to make their experience great.
- **Define your one voice and one message.** Cross-channel marketing experiences require your brand to have a cohesive, unified, and consistent voice. Look upon this journey in content consolidation and multichannel delivery as an opportunity to affect organizational alignment and centralize content creation.
- **One size doesn't fit all.** You will have different experiences requiring various authoring approaches, delivered to an assortment of channels. The hybrid paradigm is about using the right option for a given use case. Embrace the options.

Appendix A: Code example showing HTTP request and JSON response

```
HTTP GET /content/api/articles.model.json

{
    :type: "my-app/components/structure/api-page"
    title: "Data API",
    lastModifiedDate: 1467202845038,
    templateName: "api-page",
    language: "en-US",
    :items: {
        root: {
            :items: {
                contentfragment: {
                    :type: "my-app/components/content/
content-fragment",
                    title: "My Article",
                    model: "my-app/models/article",
                    elements: {
                        title: {
                            value: "My Article",
                            dataType: "string",
                            title: "Article Title"
                        },
                        body: {
                            value: "... the article
text... ",
                            dataType: "string",
                            title: "Article Text"
                        },
                        ... other Content Fragment fields ...
                    },
                    ... Other components that expose content
to this API ...
                },
                :itemsOrder: [
                    "title",
                    "body"
                ]
            }
        },
        :itemsOrder: [
            "root",
        ]
}
```

Appendix B. Code example showing custom component development

```
import com.adobe.cq.export.json.ComponentExporter;


@SlingModel(
    adaptables = SlingHttpServletRequest.class,
    adapters = { MyComponent.class, ComponentExporter.class }
)
@Exporter(
    name = ExporterConstants.SLING_MODEL_EXPORTER_NAME,
    extensions = ExporterConstants.SLING_MODEL_EXTENSION
)
@JsonSerialize(as = MyComponent.class)
public class MyComponentImpl implements ComponentExporter {


    String getHelloWorld() {
        return "Hello World";
    }


    @Override
    String    getExportedType() {
        return "my-app/components/content/my-component;
    }
```