Cartridge Path Substitution

The cartridge path controls file lookup for your site and defines what code runs. B2C Commerce searches the cartridge from left to right when a specific controller, script, model, or ISML template is called via URL or code.

You can substitute the cartridge path file lookup with the require() function in these situations.

Relative to the current file: require('./shipping') or require('../../util')

Relative to the current cartridge: require ('~/cartridge/scripts/cart')

From the beginning of the cartridge path: require('\*/cartridge/scripts/util/array')

Specific B2C Commerce API: require('dw/catalog/CatalogMgr')

Specific module or cartridge: require('server')

controller : https://share.vidyard.com/watch/TFjkeSxUKR9Mh1cDSVF4qz?&assignmentId=a5c3m0000022R4nAAE

customize b2c sfra

------------------

SFRA comes with an app\_storefront\_base cartridge and a server module. The base cartridge contains functionality that’s common to most sites.

You can layer additional functionality over it with plug-in cartridges, LINK cartridges, and custom code cartridges.

B2C Commerce provides plug-in cartridges for wish lists, gift registries, Apple Pay, and product comparison, as well as middleware capability.

LINK partners, such as PayPal and Bazaarvoice, provide LINK cartridges. So there’s all kinds of apps to choose from.

\*\*This means that you don’t edit or rename the app\_storefront\_base cartridge and the other plugins, such as plugin\_applepay. It also means that you need to keep your version of the app\_storefront\_base cartridge and other plugins up to date to get all the changes.

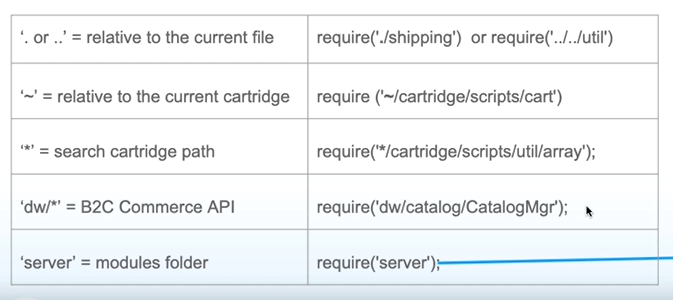
\*\*\*Keep in mind that the base cartridge and module are not intended to be customized

--Because SFRA is backward compatible between point releases (except when required for urgent security fixes), you can just download a new version and run automated tests. No more searching for code changes so you can port it to the customized code.  
  
!!!!High imp

-------

Each site must have at least one custom cartridge. If you want to create multiple sites, we suggest you create multiple custom cartridges that separate functionality specific to a brand or locale. This lets you reuse most of the cartridge stack for a new site or microsite.

|  |  |
| --- | --- |
| **Components** | **Example** |
| Hooks | Hooks let you configure functionality that’s called at a specific time in the application flow or for a specific event. This means using OCAPI hooks or custom hooks via the B2C Commerce script *System* package *HookMgr* class methods.  The application can use them, for example, to calculate or validate the basket, or to invoke a payment processor. |
| Modules | Modules are an industry standard way to group code for functionality that’s shared across multiple resources. Modules make it easy to add storefront functionality and then reuse it wherever it applies.  SFRA supports JavaScript/B2C Commerce script modules that conform to the Modules 1.1.1 CommonJS specification. The functionality in a CommonJS module can be reused by multiple controllers. A module’s .ds or .js file is typically stored in a cartridge in the *script* folder or in a *modules* folder at the same level as a cartridge. The application can access modules in the cartridge, other cartridges, and the modules folder. |
| Templates | Templates (same as with SiteGenesis) determine how information displays on the storefront. SFRA includes two decorator templates:   * **page.isml**—Contains navigation information. * **checkout.isml**—Doesn't contain navigation information. Removing navigation information, for example, usually improves the percentage of cart abandonment. Shoppers see no easy way out. |
| Models | SFRA models provide a JSON object layer for the application. Models convert objects returned by the B2C Commerce script APIs into pure JSON objects designed for the storefront. Models also apply business logic for the storefront.  Controllers create and update models. To customize a model, you create the model and then add data to it that can be used for rendering. |
| Controllers and Routes | Middleware lets you to execute code before and after a controller is invoked. Route lets you to add a PSR-7 compatible callable to the stack that’s invoked every time the app runs.  Whether you extend or override a controller can impact functionality and performance. If you extend a controller, the application executes the original middleware, and then the extension. If the original middleware includes a third-party interaction, the interaction still executes. If the extension also includes the interaction, the interaction executes twice. |
| Forms | Create HTML forms using templates and controllers. Using form definitions, you can also persist form data during a session and store it in system objects or custom objects. |

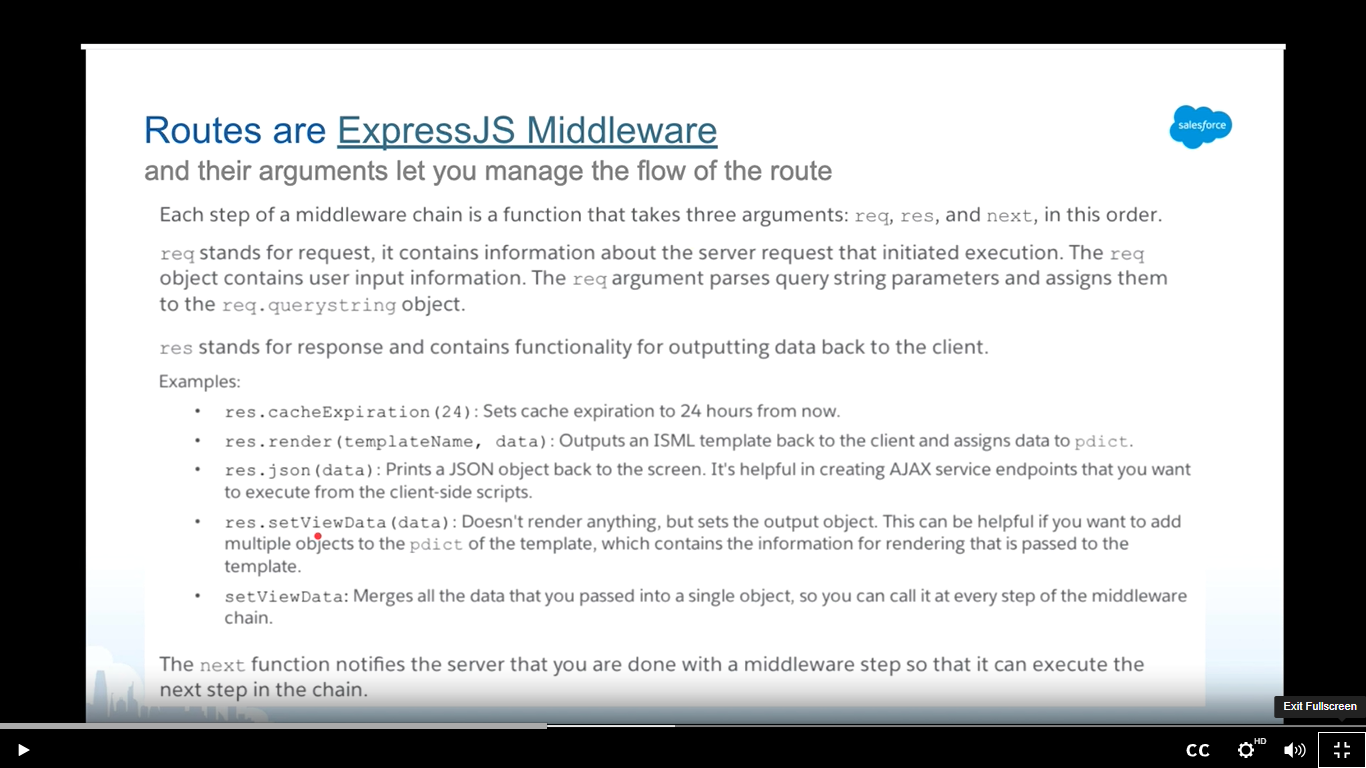


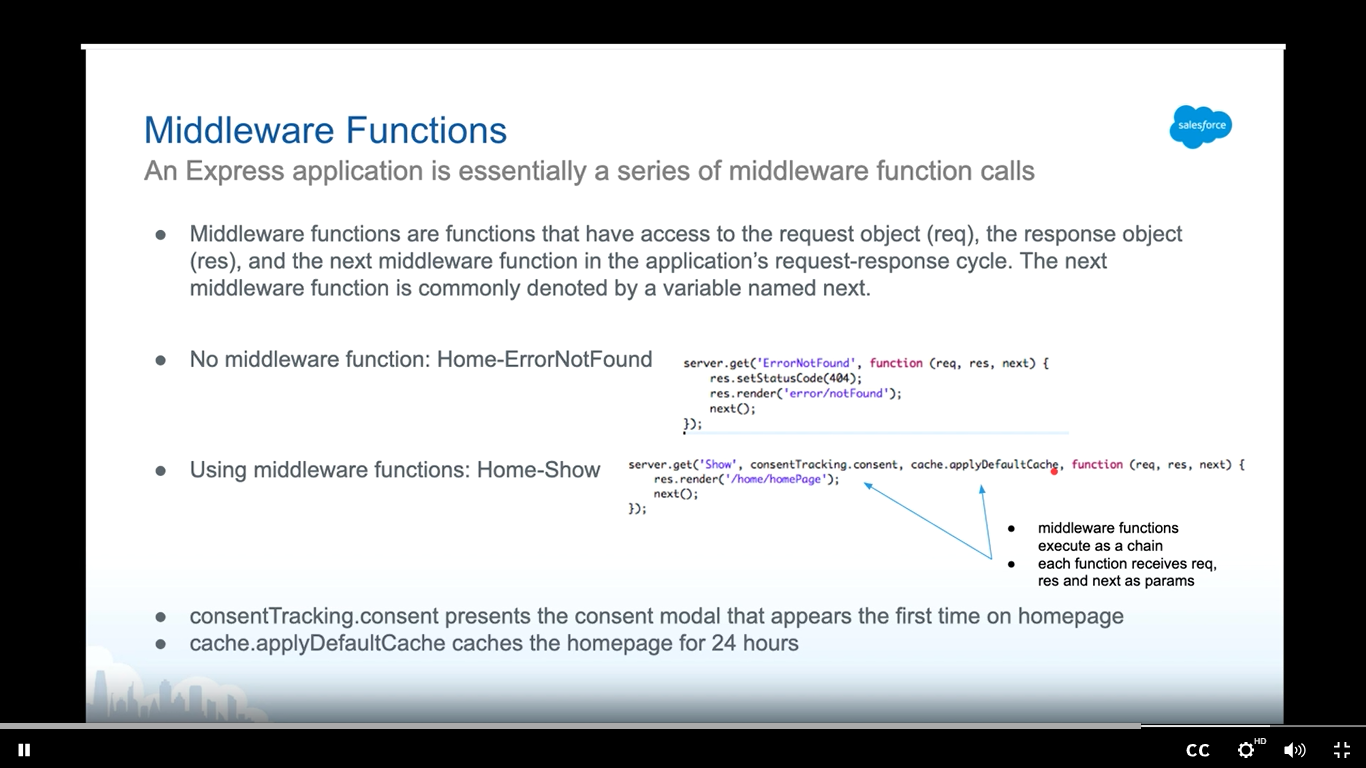
TOP level classes:  
------------------------

<https://documentation.b2c.commercecloud.salesforce.com/DOC1/index.jsp?topic=%2Fcom.demandware.dochelp%2FDWAPI%2Fscriptapi%2Fhtml%2Fapi%2Fpackage_TopLevel.html>

Global class

<https://documentation.b2c.commercecloud.salesforce.com/DOC1/index.jsp?topic=%2Fcom.demandware.dochelp%2FDWAPI%2Fscriptapi%2Fhtml%2Fapi%2Fclass_TopLevel_global.html>





Server side js classes:

<https://documentation.b2c.commercecloud.salesforce.com/DOC1/index.jsp?topic=%2Fcom.demandware.dochelp%2Fcontent%2Fb2c_commerce%2Ftopics%2Fsfra%2Fb2c_sfra_overview.html&assignmentId=a5c3m0000022R4oAAE>

caching:

<https://share.vidyard.com/watch/n2wW1yH9opoqJ2kNGis2qD?&assignmentId=a5c3m0000022R4xAAE>