Determine how to use configuration files in Magento. Which configuration files correspond to different features and functionality?

List of existed *.xml configs

- `acl.xml` - resource title, sort
- `adminhtml/rules/payment_{country}.xml` - paypal
- `address_formats.xml`
- `address_types.xml` - format code and title only
- `cache.xml` - name, instance - e.g. full_page=Page Cache
- `catalog_attributes.xml` - catalog_category, catalog_product, unassignable, used_in_autogeneration, quote_item *
- `communication.xml`
- `config.xml` - defaults
- `crontab.xml` - group[], job instance, method, schedule*
- `cron_groups.xml` *
- `di.xml` - preference, plugins, virtual type*
- `eav_attributes.xml` - locked entity attributes (global, unique etc.)
- `email_templates.xml` - id label file type module – view/frontend/email/name.html
- `events.xml` - observers, shared, disabled*
- `export.xml`
- `extension_attributes.xml` - for, attribute code, attribute type
- `fieldset.xml`
- `import.xml`
- `indexer.xml` - class, view_id, title, description
- `integration.xml`
- `integration/api.xml`
- `integration/config.xml`
- `menu.xml` - admin menu
- `module.xml` - version, sequence
- `mview.xml` - scheduled updates, subscribe to table changes, indexer model
- `page_types.xml`
- `payment.xml` - groups, method allow_multiple_address
- `pdf.xml` - renders by type (invoice, shipment, creditmemo) and product type
- `product_types.xml` - label, model instance, index priority, (?) custom attributes, (!) composable types
- `product_options.xml`
- `resources.xml`
- `routes.xml`
- `sales.xml` - collectors (quote, order, invoice, credit memo)
- `search_engine.xml`
- `search_request.xml` - index, dimensions, queries, filters, aggregations, buckets
- `sections.xml` - action route placeholder -> invalidate customer sections
- `system.xml` - adminhtml config
- `validation.xml` - entity, rules, constraints -> class
- `view.xml` - vars by module
- `webapi.xml` - route, method, service class and method, resources
- `widget.xml` - class, email compatible, image, ttl (?), label, description, parameters
- `zip_codes.xml`

Interfaces for work with configs

\Magento\Framework\Config\Reader\Filesystem -> \Magento\Framework\Config\ReaderInterface

Gets .xsd names from schema locator, gets full .xml file list from file resolver, merges all files, validates, runs converter to get resulting array.

- read(scope)
  - `fileResolver->get(_filename)`
  - merge and validate each file (if mode developer)
  - validate merged DOM
  - converter->convert(dom) => array
- `_idAttributes`, `_fileName`, `_schemaFile` (from schemaLocator), `_perFileSchema` (from schemaLocator), filename (menu.xml)
- schemaFile from schemaLocator

\Magento\Framework\Config\ConverterInterface

Convert an array to any format

- `convert(\DOMDocument $source)`

\Magento\Framework\Config\SchemaLocatorInterface - full path to .xsd

- `getPerFileSchema` - per file before merge
- `getSchema` - merged file

\Magento\Framework\Config\ValidationStateInterface

This interface retrieves the validation state.

- `isValidationRequired()`

\Magento\Framework\App\Arguments\ValidationState is default implementation, that require validation only in developer mode.

\Magento\Framework\Config\ScopeListInterface

This interface the list of all scopes.

- `getAllScopes()`

\Magento\Framework\Config\Data -> \Magento\Framework\Config\DataInterface

Helps to get the configuration data in a specified scope.

- `merge(array $config);`
- `get($key, $default = null)`

Links and examples:

- Product types configs model to read data fromproduct_types.xml: \Magento\Catalog\Model\ProductTypes
- Implementation via virtual types to read data from layout.xml:Magento/Theme/etc/di.xml
- https://www.atwix.com/magento-2/working-with-custom-configuration-files/

Configuration load and merge flow

Loading order

1. First step is a `app/etc/di.xml` loading
2. Next step is collecting configuration files from all modules and merging them: `vendor_name/component_name/etc/*.xml`
3. At the last step Magento will collect all configs from `vendor_name/component_name/etc/<area>/*.xml`

Merge flow

> Nodes in configuration files are merged based on their fully qualified XPaths, which has a special attribute defined in $idAttributes array declared as its identifier. This identifier must be unique for all nodes nested under the same parent node.
>
> - If node identifiers are equal (or if there is no identifier defined), all underlying content in the node (attributes, child nodes, and scalar content) is overridden.
> - If node identifiers are not equal, the node is a new child of the parent node.
> - If the original document has multiple nodes with the same identifier, an error is triggered because the identifiers cannot be distinguished.
> - After configuration files are merged, the resulting document contains all nodes from the original files.
>
> – Magento DevDocs - Module configuration files

config merger = \Magento\Framework\Config\Dom

- when merging each file
  - createConfigMerger|merge
  - \Magento\Framework\Config\Dom::_initDom(dom, perFileSchema)
  - \Magento\Framework\Config\Dom::validateDomDocument
  - $dom->schemaValidate
- after all merged
  - \Magento\Framework\Config\Dom::validate(mergedSchema)
  - \Magento\Framework\Config\Dom::validateDomDocument

Sensitive and environment settings

This scope is a very huge part which includes a lot of things and there is a short list of useful links to the official Magento DevDocs documentation:

- Set configuration values
- Sensitive and system-specific
- Magento Enterprise B2B Extension configuration paths reference
- Other configuration paths reference

Example of how to set sensitive settings

- shared config app/etc/config.php
- sensitive or system-specific app/etc/env.php:

```
    <type name="Magento\Config\Model\Config\TypePool">
       <arguments>
          <!-- sensitive config items -->
          <argument name="sensitive" xsi:type="array">
             <item name="payment/paypal_express/merchant_id"
xsi:type="string">1</item>
             <!-- keys, password, emails, personally identifiable information -->
          </argument>
          <!-- environment specific config items -->
          <argument name="environment" xsi:type="array">
             <item name="payment/paypal_express/debug" xsi:type="string">1</item>
             <!-- URLs, IPs, hosts, modes sandbox/live, email recipients -->
          </argument>
       </arguments>
    </type>
```

Sensitive info doesn't get exported with `bin/magento app:config:dump`. use env. params, e.g. `CONFIG__DEFAULT__PAYMENT__TEST__PASWORD` for `payment/test/password`

`bin/magento app:config:dump`:

- system-specific > app/etc/env.php
- shared > app/etc/config.php
- sensitive - skipped

`bin/magento config:sensitive:set`:

- writes to app/etc/env.php