upstate.edu

Lesson 10: Structured Data Nodes

31-39 minutes

Introducing Node Paths

- To work efficiently with pages and data definition blocks as Cascade API objects, we need to be able to handle structured data nodes
- To access structured data nodes and the data within, we need to use node path expressions
- The methods `getStructuredDataNode` and `getStructuredDataNodes` discussed in the next section both take a node path expression as argument
- A Node path expression is a String
- Such an expression is the result of concatenating field identifiers with slash delimiters
- These field identifiers come from the corresponding data definition
- A node path expression is different than an XPath expression
- To fully understand the concept of node path, let us look at the page data definition dd_page.xml

used at Upstate

- Here is the XML structure:

```
system-data-structure
   group    identifier="pre-main-group"
      asset identifier="mul-pre-main-chooser"
   group    identifier="main-group"
      asset identifier="mul-pre-h1-chooser"
      text  identifier="h1"
      asset identifier="mul-post-h1-chooser"
      text  identifier="float-pre-content-blocks-
around-wysiwyg-content"
         checkbox-item
      text  identifier="wysiwyg"
      asset identifier="mul-post-wysiwyg-chooser"
   group    identifier="post-main-group"
      asset identifier="mul-post-main-chooser"
   group    identifier="top-group"
      asset identifier="mul-top-group-chooser"
   group    identifier="bottom-group"
      asset identifier="mul-bottom-group-chooser"
   group    identifier="admin-group"
      asset identifier="master-level-override"
      asset identifier="page-level-override"
```

- In the XML of a data definition, the root element is always `system-data-structure`

- The root element can only have three types of child elements: `group`, `text` and `asset`
- A `group` element can contain other `group`, `text` and `asset` children
- A `text` can contain `X-item` elements (X being one of the following: checkbox, dropdown, radio, and selector)
- Each `group`, `text` or `asset` element must have an identifier
- Identifiers among siblings must be unique
- `X-item` elements do not have identifiers
- A `group` element must have at least one child element
- Besides the element names `group`, `text` and `asset`, I also include the field identifiers, because these are the parts of node path expressions
- The children of `system-data-structure`, namely, the six `group` elements, are top-level elements
- Also note that I consistently use the prefix `mul-` in identifiers of multiple fields; for example:

```
<asset identifier="mul-pre-h1-chooser"
label="Above-the-H1 Blocks" multiple="true"
render-content-depth="5" type="block"/>
```

- When the page data definition is associated with a content type, which is then associated with a page, the page will have a set of structured data nodes with the following structure:

```
group     "pre-main-group"
   +asset "pre-main-group/mul-pre-main-chooser"
group     "main-group"
   +asset "main-group/mul-pre-h1-chooser"
   text   "main-group/h1"
   +asset "main-group/mul-post-h1-chooser"
   text   "main-group/float-pre-content-blocks-
around-wysiwyg-content"
   text   "main-group/wysiwyg"
   +asset "main-group/mul-post-wysiwyg-chooser"
group     "post-main-group"
   +asset "post-main-group/mul-post-main-chooser"
group     "top-group"
   +asset "top-group/mul-top-group-chooser"
group     "bottom-group"
   +asset "bottom-group/mul-bottom-group-chooser"
```

```
group     "admin-group"
   asset  "admin-group/master-level-override"
   asset  "admin-group/page-level-override"
```

- Each `group`, `text`, or `asset` element in the data definition is instantiated as a `group`, `text`, or an `asset` node respectively
- A multiple field can correspond to a set of nodes bearing the same identifier
- Multiple node sets are marked with the + characters
- To access a node, we need to start with an identifier of a top-level element in the data definition
- Derivatively, the identifier of a top-level field becomes the identifier of the corresponding top-level node
- If the node we want to access is a descendant of a top-level group node, then we need to concatenate the identifier of the parent node of the node we want to access with the identifier of the corresponding field of the node, delimited by slashes (`/`)
- These are what I call node path expressions
- For example, to access the `h1` node, we need to use the node path expression `main-group/h1`
- This is the result of concatenating the identifier of the node's parent, i.e., `main-group` and the field identifier of the node, i.e., `h1`, with a slash in between
- Note that:
- A node path expression is not an XPath expression because identifiers are not element names
- In a node path expression, there should be no leading nor trailing slashes
- Only field identifiers and slash delimiters can appear in a node path expression; substrings like `[0]` and `//` cannot
- All instances of the same multiple field share the same node path expression
- That is to say, as far as node path expressions are concerned, there is no way to distinguish two instances of the same field
- To access a specific instance of a multiple field, we need to use its position in the set

Structured Data Nodes, Data Definition Blocks, and Pages

- When working with a `com.hannonhill.cascade.api.adapters.XHTMLDataDefinitionBlockAPIAdap` or `com.hannonhill.cascade.api.adapters.PageAPIAdapter` object, we will need to use methods defined in `com.hannonhill.cascade.api.adapters.StructuredDataNodeAPIAdapter` to handle structured data nodes
- These `com.hannonhill.cascade.api.adapters.StructuredDataNodeAPIAdapter` objects are the actual data containers of the hosting block or page
- Both `com.hannonhill.cascade.api.adapters.XHTMLDataDefinitionBlockAPIAdap` and `com.hannonhill.cascade.api.adapters.StructuredDataNodeAPIAdapter` classes provide three methods to retrieve structured data nodes:
- `getStructuredData()`
- `getStructuredDataNode( java.lang.String )`
- `getStructuredDataNodes( java.lang.String )`
- Example:

```
## a data definition block
#set( $block = $_.locateBlock( "_cascade/blocks
/data/latin-wysiwyg", "formats" ) )
#set( $nodes = $block.StructuredData )

## a page
#set( $nodes = $currentPage.StructuredData )
```

- The value associated with the String parameter in both `getStructuredDataNodes` and `getStructuredDataNode` is a node path expression
- `getStructuredData()` returns an array of `com.hannonhill.cascade.api.adapters.StructuredDataNodeAPIAdapter` objects
- These objects are the top-level nodes, corresponding to top-level fields (direct children of the `system-data-structure` element) defined in the data definition
- Since there must be at least one top-level element in a data definition, `getStructuredData()` always returns a non-empty array
- `getStructuredDataNode( java.lang.String )` returns the

first instance of a multiple field, or the only instance of a non-multiple field, filtered by the node path expression

- If the node path expression is invalid, then the method returns `null`
- `getStructuredDataNodes( java.lang.String )` returns an array of multiple instances of a multiple field, filtered by the node path expression
- If the node path expression is valid, then there will be at least one instance in the array; if the node path expression is invalid, then the array will be empty

Data Definitions without Multiple Fields

- To have a better understanding on how to work with structured data nodes, let us look at two more data definitions
- The first one we want to look at, named `Wysiwyg`, is a simple one:

```
<system-data-structure>
    <text default="yes" help-text="Select yes to
display" identifier="display" label="Display"
required="true" type="radiobutton">
        <radio-item value="yes"/>
        <radio-item value="no"/>
    </text>
    <!-- wysiwyg -->
    <group identifier="wysiwyg-group"
label="Wysiwyg Group">
        <text identifier="wysiwyg-content"
label="Content" wysiwyg="true"/>
        <text identifier="admin-options"
label="admin-options" restrict-to-
groups="Administrators,CWT-Designers"
type="checkbox">
            <checkbox-item value="local-view-
only"/>
        </text>
    </group>
</system-data-structure>
```

- The XML structure of the data definition:

```
system-data-structure
   text identifier="display"
      radio-item
```

```
      radio-item
   group identifier="wysiwyg-group"
      text identifier="wysiwyg-content"
      text identifier="admin-options"
         checkbox-item
```

- There are two top-level fields: `text` whose identifier is `display` and `group` whose identifier is `wysiwyg-group`
- Before we start retrieving structured data nodes, we need to keep the following in mind: only `group`, `text` and `asset` fields in a data definition have corresponding structured data nodes
- Elements like `radio-item` and `checkbox-item` do not have corresponding structured data nodes
- When a block associated with this data definition is located, and when the property `StructuredData` is used, we will get the two top-level nodes:

```
#set( $block = $_.locateBlock( "_cascade/blocks
/data/latin-wysiwyg", "formats" ) )
#set( $nodes = $block.StructuredData )
$nodes.Class.Name ##=>
[Lcom.hannonhill.cascade.api.asset.common.StructuredDataNode;
$nodes.size()     ##=> 2
```

- To retrieve the top-level group directly, we can call `getStructuredDataNode`:

```
#set( $group = $block.getStructuredDataNode(
"wysiwyg-group" ) )
$group.Identifier ##=> wysiwyg-group
```

- To get the second text node in the group, we will need a longer node path expression:

```
#set( $checkbox = $block.getStructuredDataNode(
"wysiwyg-group/admin-options" ) )
$checkbox.Identifier ##=> admin-options
```

  Note how the identifiers are concatenated to create the node path expression.

- Since there are no multiple fields in this data definition, we may not want to call `getStructuredDataNodes`, though it still can be used to retrieve a single node:

```
#set( $checkboxes = $block.getStructuredDataNodes(
"wysiwyg-group/admin-options" ) )
$checkboxes[ 0 ].Identifier ##=> admin-options
```

- Note that because `getStructuredDataNodes` returns an array, we need to use an index to access a node in the array

Data Definitions with Multiple Fields

- Next, we want to look at a data definition containing multiple fields
- Here is a simple one named `Accordion`:

```
<system-data-structure>
    <text default="yes" help-text="Select yes to
display" identifier="display" label="Display"
type="radiobutton">
        <radio-item value="yes"/>
        <radio-item value="no"/>
    </text>
    <!-- accordion -->
    <group identifier="h2-wysiwyg-group" label="H2
WYSIWYG Group" multiple="true">
        <text help-text="Text to be clicked to
expand the hidden content" identifier="accordion-
h2" label="H2 Text"/>
        <text identifier="wysiwyg" label="Content"
wysiwyg="true"/>
    </group>
</system-data-structure>
```

- We can retrieve all instances of `h2-wysiwyg-group` by doing this:

```
#set( $groups = $block.getStructuredDataNodes(
"h2-wysiwyg-group" ) )
$groups.size() ##=> 3
```

- Note that, again, the String passed into the method can contain only identifiers delimited by slashes; expressions like `h2-wysiwyg-group[1]` won't work
- There is no easy way to find out if a node is an instance of a multiple field, unless we are willing to query the database and examine the data definition
- This is precisely the reason why I consistently use the `mul-` prefix in identifiers of multiple fields
- As we will see later, the library code used to process structured data nodes relies on this prefix to identify multiple instances

Categories and Types of Structured Data Nodes

- When we create a data definition, we know that there are fifteen

different types of fields: group, text, multiline, wysiwyg, radio, checkbox, dropdown, multiselect, calendar, datetime, page chooser, file chooser, block chooser, symlink chooser, and linkable chooser

- However, if we look at the XML of a data definition, we find that only three types of fields can have corresponding structured data nodes: group, text, and asset
- There are nine types of text nodes and five types of asset choosers
- This is the entire picture:
- Group
- Text:
- Textbox
- Multiline
- Wysiwyg
- Radio
- Checkbox
- Dropdown
- Multiselect
- Calendar
- Datetime
- Asset:
- Page chooser
- File chooser
- Block chooser
- Symlink chooser
- Linkable chooser

Getting Category and Type Information

- Given a `com.hannonhill.cascade.api.adapters.StructuredDataNodeAPIAdapter` object, we can use the three associated `isX` methods to find its category:

```
#if( $node.isGroup() )
Type: Group
#elseif( $node.isAsset() )
Type: Asset
#elseif( $node.isText() )
Type: Text
```

```
#end
```

- Given a group node, we can call the `getChildren` method or use the `Children` property to retrieve its children:

```
#if( $node.isGroup() )
    Number of Children: $node_.Children.size()
#end
```

- Note that `getChildren` (or `Children`) is meaningful only for group nodes; text nodes and asset nodes do not have child nodes
- To get a child bearing a specific identifier, call `getChild( java.lang.String )`; if the child is an instance of a multiple node set, then the method returns the first instance
- Given a text node, we can call the `getTextNodeOptions()` method to get a `com.hannonhill.cascade.api.asset.common.TextNodeOptions` object
- Through the `com.hannonhill.cascade.api.asset.common.TextNodeOptions` object, we can call one of the eight `isX` (X being one of the following: Radio, PlainText, Checkbox, Multiselect, Wysiwyg, Dropdown, Calendar, and Datetime) methods to find out the subtype of the node:

```
#if( $node.isText() )
    #set( $textOptions = $node.TextNodeOptions )

    #if( $textOptions.isRadio() )
        Subtype: Radio
    #elseif( $textOptions.isWysiwyg() )
        ...
    #end
#end
```

- The `isPlainText` method returns true when the text node is either a text box or a multiline text area
- Given an asset node, we can call the `getAssetIdentifier` method to get a `com.hannonhill.cascade.api.asset.common.PathIdentifier` object if the chooser in question is attached with an asset; this is the first way to get a `PathIdentifier` object
- When an asset node is attached with an asset, the `getAsset` method or the `Asset` property returns a `com.hannonhill.cascade.api.asset.home.FolderContainedAsset`

object, which represents the attached asset (a block, a file, a page, or a symlink)

- Through the `com.hannonhill.cascade.api.asset.home.FolderContainedAsset` object, we can call the `getIdentifier` method or use the `Identifier` property to get a `com.hannonhill.cascade.api.asset.common.PathIdentifier` object; this is the second way to get a `PathIdentifier` object
- Through this `PathIdentifier` object, we can get the ID String or path information of the attached asset
- When an asset node is not attached with anything, there is no simple way of finding out what type of assets can be attached to the node; we have to consult the data definition instead

Getting the Data: Group

- The only data a group node can store is a set of child nodes
- Call `getChildren()` (or `Children`) to retrieve all child nodes
- Call `getChildren( java.lang.String )` to retrieve all instances of a child multiple field; the String should be the identifier of the multiple field
- Call `getChild( java.lang.String )` to retrieve the first instance of a child multiple field or the single instance of a non-multiple field

Three Methods for Text Nodes

- For text nodes, there are three methods (and associated properties) we can use to work with the data a node stores:
- `getTextValue()`
- `getTextValues()`
- `hasTextValue( java.lang.String )`
- `getTextValue()` returns either `null` or a single String
- `getTextValues()` returns an array of Strings
- `hasTextValue( java.lang.String )` returns a boolean

Getting the Data: Plain Text

- There are two types of plain text nodes: single-line text box and multiline text area

- When a plain text node does not store any String, `getTextValue` returns `null`; otherwise it returns the stored String
- `getTextValues` returns an empty array if the node has no data, or an array containing the single String the node stores
- `hasTextValue( java.lang.String )` returns true only if the argument String passed in is equal to the stored String

Getting the Data: Calendar

- When a calendar node does not store any String, `getTextValue` returns `null`; otherwise it returns the stored String (e.g. `02-07-2018`)
- `getTextValues` returns an empty array if the node has no data, or an array containing the single String the node stores
- `hasTextValue( java.lang.String )` returns true only if the argument String passed in is equal to the stored String
- The String can be turned into a `Date` object or a `Long` object; see Lesson 3: Numbers, Strings, Dates, XML, and XSLT

Getting the Data: Checkbox

- A checkbox node can store zero, one, or more Strings
- `getTextValue` always returns `null`
- `getTextValues` returns an empty array if the node has no data, or an array containing all selected items
- `hasTextValue( java.lang.String )` returns true only if the argument String passed in is in the String array returned by `getTextValues`

Getting the Data: Datetime

- When a datetime node does not store any String, `getTextValue` returns `null`; otherwise it returns the stored String (e.g. `1519560000000`)
- `getTextValues` returns an empty array if the node has no data, or an array containing the single String the node stores
- `hasTextValue( java.lang.String )` returns true only if the argument String passed in is equal to the stored String

Getting the Data: Dropdown

- A dropdown node can store zero or one Strings

- When a dropdown node does not store any String, `getTextValue` returns `null`; otherwise it returns the stored String (the selected item)
- `getTextValues` returns an array containing all items if no item is selected, or an array containing the single String the node stores (the selected item)
- `hasTextValue( java.lang.String )` returns true only if the argument String passed in is equal to the stored String

Getting the Data: Multiselect

- A multiselect node can store zero, one, or more Strings
- `getTextValue` always returns `null`
- `getTextValues` returns an empty array if the node has no data, or an array containing all selected items
- `hasTextValue( java.lang.String )` returns true only if the argument String passed in is in the String array returned by `getTextValues`

Getting the Data: Radio

- A radio node can store zero or one Strings
- When a radio node does not store any String, `getTextValue` returns `null`; otherwise it returns the stored String (the selected item)
- `getTextValues` returns an array containing all items if no item is selected, or an array containing the single String the node stores (the selected item)
- `hasTextValue( java.lang.String )` returns true only if the argument String passed in is equal to the stored String

Getting the Data: Wysiwyg

- When a wysiwyg node does not store any String, `getTextValue` returns `null`; otherwise it returns the stored String (markups and so on)
- `getTextValues` returns an empty array if the node has no data, or an array containing the single String the node stores
- `hasTextValue( java.lang.String )` returns true only if the argument String passed in is equal to the stored String
- There is an extra method named `getTextValueAsXMLElement`

for Wysiwyg nodes

- This method returns the contents of the node as an `org.jdom.Element` object
- It acts like the reverse of serialization: turning the markup String back to an `org.jdom.Element` object
- This may be useful when we want to analyse the contents as XML

Getting the Assets

- For any asset node, use `$node.Asset.Class.Name` to test if there is an asset attached to the node
- If `$node.Asset.Class.Name` is true, then `$node.Asset` returns the attached asset
- The attached asset, if there is one, can be a page, a file, a block, or a symlink
- Use methods defined in classes like `com.hannonhill.cascade.api.adapters.PageAPIAdapter` and `com.hannonhill.cascade.api.adapters.FileAPIAdapter` to work with these assets
- As pointed out above, there are two ways to retrieve a `com.hannonhill.cascade.api.asset.common.PathIdentifier` object, storing information about the attached asset
- The object can be used to retrieve more information:
```
Asset ID: $node.AssetIdentifier.Id
Asset Path:
$node.Asset.Identifier.Path.PathAsString
```

A Library Macro to Dump Node Data

- In the library format named `chanw-process-cascade-api`, there is a macro named `#chanwDisplayStructuredDataNode` that can be invoked to dump the information of a node and the data it stores:
```
#import( "site://_brisk/core/library/velocity
/chanw/chanw-library-import" )

#set( $block = $_.locateBlock( "_cascade/blocks
/data/all-fields", "formats" ) )
#set( $node = $block.getStructuredDataNode(
"group" ) )
```

```
#chanwDisplayStructuredDataNode( $node )
```

This code snippet outputs information of the following type:

```
Identifier: group
Type: Group
Number of Children: 14
```

- If we pass in a text node, the output will be different:

```
Identifier: text
Type: Text
Subtype: PlainText
Text Value: Hello
Text Values: [ Hello ]
```

- Here is the output of all fifteen types of nodes:

```
Identifier: group
Type: Group
Number of Children: 14

Identifier: text
Type: Text
Subtype: PlainText
Text Value: Hello
Text Values: [ Hello ]

Identifier: multiline
Type: Text
Subtype: PlainText
Text Values: [  ]

Identifier: wysiwyg
Type: Text
Subtype: Wysiwyg
Text Value:
<p>Some contents.</p>
Text Values: [
<p>Some contents.</p> ]

Identifier: checkbox
Type: Text
Subtype: Checkbox
Text Values: [  ]

Identifier: dropdown
```

```
Type: Text
Subtype: Dropdown
Text Value: Deux
Text Values: [ Deux ]

Identifier: radio
Type: Text
Subtype: Radio
Text Value: No
Text Values: [ No ]

Identifier: multiselect
Type: Text
Subtype: Multiselect
Text Values: [  ]

Identifier: calendar
Type: Text
Subtype: Calendar
Text Value: 02-07-2018
Text Values: [ 02-07-2018 ]

Identifier: datetime
Type: Text
Subtype: Datetime
Text Value: 1519560000000
Text Values: [ 1519560000000 ]

Identifier: page-chooser
Type: Asset
Asset ID: 9a13b3198b7f08ee5d439b31fa8691c5
Asset Path: velocity/courses/introductory-course
/v-introductory-lesson07
Asset Type:
com.hannonhill.cascade.api.adapters.PageAPIAdapter
Site: formats

Identifier: file-chooser
Type: Asset
Asset ID: 43efb8fb8b7f08ee0faa9686c2b8e6e1
Asset Path: images/ambassadors.jpg
Asset Type:
```

```
com.hannonhill.cascade.api.adapters.FileAPIAdapter
Site: hospital

Identifier: block-chooser
Type: Asset
Asset ID: c317f0858b7f08ee7a0ca8de9327f681
Asset Path: _cascade/blocks/data/xhtml
Asset Type:
com.hannonhill.cascade.api.adapters.XHTMLDataDefinitionBlockAPIAdap
Site: formats

Identifier: symlink-chooser
Type: Asset
Asset ID: d16471738b7f08ee25883c8e39976848
Asset Path: formats
Asset Type:
com.hannonhill.cascade.api.adapters.SymlinkAPIAdapter
Site: cascade-admin

Identifier: linkable-chooser
Type: Asset
```

Problems Associated with Node Path Expressions

- There are two major problems associated with node path expressions:
- Multiple instances of the same multiple field share the same node path expression
- The nature of the delimiter `/`
- When we are dealing with nodes instantiated from a data definition without multiple fields, node path expressions work fine
- But once we start introducing multiple fields, node path expressions alone won't work because instances of the same field share the same node path expression
- In order to retrieve a node from a multiple set, we need to take two steps:

1. We need to call `getStructuredDataNodes( java.lang.String )` to retrieve the set
2. From the returned array, we need to use an index to retrieve a node from the set

- Example:

```
#set( $block = $_.locateBlock( "_cascade/blocks
/data/thread_directors-clinical_science", "com" )
)
#set( $nodes = $block.StructuredData )
#set( $groups = $block.getStructuredDataNodes(
"h2-wysiwyg-group" ) )
$groups.Class.Name      ## an array
$groups[ 0 ].Class.Name ## the first node
```

- Note that `[ 0 ]` cannot be part of the node path expression
- Since a multiple node can have instances of multiple groups as ancestors, the problem becomes more complicated
- The `/` characters are used as delimiters in node path expressions
- A node path expression is a String
- String methods like `match` and `replaceAll` involve regular expressions
- The `/` characters are used as delimiters in regular expressions
- The two uses of `/` can interfere with each other, especially when we need to use a node path expression as a regular expression

Introducing Fully Qualified Identifiers

- You need to understand the concept of fully qualified identifiers (FQIs) only if you want to use the Upstate Velocity library; if you don't, you can go to the next lesson
- This concept is borrowed from web services; see `DataDefinition` for more discussion
- A fully qualified identifier (FQI) of a structured data node is a variant of its node path expression
- Two modifications to a node path expression have been introduced:
- The absolute position of a node within a multiple set is built into the FQI of the node
- the `;` characters are used as delimiters
- The absolute position of a node within a multiple set is 0-based
- The first instance of a set has an index `0`
- The last instance of the set has an index `$set.size() - 1`
- Assuming that we are working with a block associated with the following data definition:
  `<system-data-structure>`

```
    <text default="yes" help-text="Select yes to
display" identifier="display" label="Display"
type="radiobutton">
        <radio-item value="yes"/>
        <radio-item value="no"/>
    </text>
    <!-- accordion -->
    <group identifier="h2-wysiwyg-group" label="H2
WYSIWYG Group" multiple="true">
        <text help-text="Text to be clicked to
expand the hidden content" identifier="accordion-
h2" label="H2 Text"/>
        <text identifier="wysiwyg" label="Content"
wysiwyg="true"/>
    </group>
</system-data-structure>
```

To access the first instance of `h2-wysiwyg-group`, we use the FQI `h2-wysiwyg-group;0`; to access the second instance, if there is one, we use `h2-wysiwyg-group;1`.

- When dealing with multiple nodes with multiple ancestors, we can have FQIs like `mul-group;1;mul-subgroup;0;mul-text;2`
- FQIs of nodes are always unique
- They can be used to identify nodes without ambiguity
- They can be hard-coded in formats because they never change: either an FQI does not identify a node, or if it does, it always points to the same node

A Library Macro to Process Structured Data Nodes

- In the library format named `chanw-process-cascade-api`, there is a macro named `chanwGetFQINodeMap` that can be invoked to process the structured data nodes of a page or data definition block
- This macro accepts one argument, which can either be a node or an array of nodes returned by `getStructuredData`, `getStructuredDataNode` and `getStructuredDataNodes`
- The macro processes the node(s) passed in recursively and populates a map named `$chanwGetFQINodeMap` with data
- Here is a code snippet invoking the macro:
  `#import( "site://_brisk/core/library/velocity`

```
/chanw/chanw-library-import" )

#set( $block = $_.locateBlock( "_cascade/blocks
/data/hospital-home-slideshow", "hospital" ) )
#set( $group = $block.getStructuredDataNode(
"slide-show-group" ) )
#chanwGetFQINodeMap( $group )
$chanwGetFQINodeMap
```

- Keys in the map `$chanwGetFQINodeMap` are fully qualified identifiers of the node(s) and associated descendants
- Each key, corresponding to a unique node, points to an inner map
- An example of an asset node entry:

```
    slide-show-group;mul-image;0;selectimage={

node=com.hannonhill.cascade.api.adapters.StructuredDataNodeAPIAdapt
        data=images/stroke-home-image.jpg,
        type=asset,
        size=1,
        subtype=file,
        content=null,

site=com.hannonhill.cascade.api.adapters.SiteAPIAdapter@481161b8
    },
```

- `slide-show-group;mul-image;0;selectimage` is a FQI of a node found in the block
- The inner map has seven entries:
- node: the structured data node object itself
- data: the data stored in the node; this is relevant only when the node is either a text node or an asset node
- type: the type String ("asset", "group", or "text")
- size: the size of the set; for a non-multiple node, the size is 1; for multiple nodes, this is the size of the set
- subtype: for a text node, the subtype String can be "wysiwyg" or "plaintext"; for an asset node, the subtype String can be "file" or "page"
- content: can store an `org.jdom.Element` object if this is recoverable from block choosers
- site: the `SiteAPIAdapter` object associated with an attached asset

- An example of a text node entry:

```
slide-show-group;mul-image;0;altText={

node=com.hannonhill.cascade.api.adapters.StructuredDataNodeAPIAdapt
        data=Upstate Comprehensive Stroke Center,
        type=text,
        size=1,
        subtype=plaintext,
        content=null,
        site=null
},
```

- An example of a group node entry:

```
slide-show-group;mul-image;2={

node=com.hannonhill.cascade.api.adapters.StructuredDataNodeAPIAdapt
        data=null,
        type=group,
        size=5,
        subtype=null,
        content=null,
        site=null
},
```

  For this entry, we can tell that this is the third group node of a set of 5.

- By outputting the keys of the map `$chanwGetFQINodeMap`, we can get all the FQIs in the map:
  `$chanwGetFQINodeMap.keySet()`

  and this is the output:

```
[
slide-show-group,
slide-show-group;style,
slide-show-group;caption-yes,
slide-show-group;mul-image;0,
slide-show-group;mul-image;1,
slide-show-group;mul-image;2,
slide-show-group;mul-image;3,
slide-show-group;mul-image;4,
slide-show-group;mul-image;0;selectimage,
slide-show-group;mul-image;0;altText,
slide-show-group;mul-image;0;subheadline,
slide-show-group;mul-image;0;image-link,
```

```
slide-show-group;mul-image;1;selectimage,
slide-show-group;mul-image;1;altText,
slide-show-group;mul-image;1;subheadline,
slide-show-group;mul-image;1;image-link,
slide-show-group;mul-image;2;selectimage,
slide-show-group;mul-image;2;altText,
slide-show-group;mul-image;2;subheadline,
slide-show-group;mul-image;2;image-link,
slide-show-group;mul-image;3;selectimage,
slide-show-group;mul-image;3;altText,
slide-show-group;mul-image;3;subheadline,
slide-show-group;mul-image;3;image-link,
slide-show-group;mul-image;4;selectimage,
slide-show-group;mul-image;4;altText,
slide-show-group;mul-image;4;subheadline,
slide-show-group;mul-image;4;image-link
]
```

- Any node entry can be retrieved by using the FQI of the node:
  `$chanwGetFQINodeMap[ "slide-show-group;mul-image;0;altText" ][ "data" ]`

  This outputs "Upstate Comprehensive Stroke Center".

How is the `$chanwGetFQINodeMap` Object Used in Brisk?

- I am jumping ahead to the intermediate and advanced courses when I bring in the Brisk site
- I want to provide some brief discussion here on how the `$chanwGetFQINodeMap` map is used in Brisk
- At Upstate, we design our data definitions in a specific way; see Data Definition
- When a data definition block is attached to a block chooser on a page, a macro, whose name is computed from the block, will be invoked to process the block; see Lesson 8: Automatic Macro Invocations and Block Processing
- The invoked macro is a presentational macro, whose responsibility is to present the data in the block
- An example:

```
#macro( processSlideShowGroup $map )
##
    #drulykgInvokeMacro(
```

```
"${design_namespace}ProcessSlideShowGroup" [ $map
] )

    #macro( rwd4ProcessSlideShowGroup $map )
        ## code to produce an rwd4 slideshow
<div class="slideshow" id="slideshow">
#foreach( $count in [ 1..$map[ "slide-show-
group;mul-image;0" ][ $KEY_SIZE ] ] )
    <div class="slide">
    <img src="${map[ "slide-show-group;mul-
image;$foreach.index;selectimage" ][ $KEY_DATA ]}"
         alt="${map[ "slide-show-group;mul-
image;$foreach.index;altText" ][ $KEY_DATA ]}" />
    </div>
#end
</div>
    #end

    #macro( rwd4TreeProcessSlideShowGroup $map )
        ## code to produce an rwd4Tree slideshow
    #end
#end
```

The macro `#rwd4ProcessSlideShowGroup` produces the following:

```
<div class="slideshow" id="slideshow">
    <div class="slide">
        <img src="images/stroke-home-image.jpg"
alt="Upstate Comprehensive Stroke Center" />
    </div>
    <div class="slide">
        <img src="images/home-bratslavsky-
cancer.jpg" alt="Upstate Cancer Center" />
    </div>
    <div class="slide">
        <img src="images/uh-downtown-night-
pano.jpg" alt="Upstate University Hospital" />
    </div>
    <div class="slide">
        <img src="images/upstate-transport-
helicopter.jpg" alt="Upstate Trauma Center" />
    </div>
    <div class="slide">
```

```
        <img src="images/home-slider-gch.jpg"
alt="Golisano Children's Hospital" />
    </div>
</div>
```

- The `$chanwGetFQINodeMap` object, created by library code and containing all data from the block, will be handed to this macro as an argument (stored in the `$map` parameter)
- When a data definition block is attached to a block chooser on a page, all data in the block is processed automatically by library code, and the resulting map is handed to the presentational macro
- You should appreciated the fact that there is absolutely no code in the presentational macro to process the block; in fact, the macro does not know the existence of a block
- Without the need to process a block, the presentational logic becomes extremely simple
- With the help of FQIs, accessing data stored in the map is trivial
- FQIs embedded with digits can be computed in loops
- The presentational macro should just add XHTML markups around the data provided
- Between the action of attaching a block to a page and the presentational macro creating the desired markups, there is just the library code in the middle, and no more actions are required