

Content Server Core Developer II

Version: 5.5

Last Revised On: November 7, 2003 2:48 pm

Fatwire, corp. PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. In no event shall Fatwire be liable for any loss of profits, loss of business, loss of use of data, interruption of business, or for indirect, special, incidental, or consequential damages of any kind, even if Fatwire has been advised of the possibility of such damages arising from this publication. Fatwire may revise this publication from time to time without notice. Some states or jurisdictions do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

Copyright © 2003 Fatwire, corp. All rights reserved.

This product may be covered under one or more of the following U.S. patents: 4477698, 4540855, 4720853, 4742538, 4742539, 4782510, 4797911, 4894857, 5070525, RE36416, 5309505, 5511112, 5581602, 5594791, 5675637, 5708780, 5715314, 5724424, 5812776, 5828731, 5909492, 5924090, 5963635, 6012071, 6049785, 6055522, 6118763, 6195649, 6199051, 6205437, 6212634, 6279112 and 6314089. Additional patents pending.

Fatwire, Content Server, Content Server Bridge Enterprise, Content Server Bridge XML, Content Server COM Interfaces, Content Server Desktop, Content Server Direct, Content Server Direct Advantage, Content Server DocLink, Content Server Engage, Content Server InSite Editor, Content Server Satellite, and Transact are trademarks or registered trademarks of Fatwire, corp. in the United States and other countries.

iPlanet, Java, J2EE, Solaris, Sun, and other Sun products referenced herein are trademarks or registered trademarks of Sun Microsystems, Inc. *AIX, IBM, WebSphere*, and other IBM products referenced herein are trademarks or registered trademarks of IBM Corporation. *WebLogic* is a registered trademark of BEA Systems, Inc. *Microsoft, Windows* and other Microsoft products referenced herein are trademarks or registered trademarks of Microsoft Corporation. *UNIX* is a registered trademark of The Open Group. Any other trademarks and product names used herein may be the trademarks of their respective owners.

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>) and software developed by Sun Microsystems, Inc. This product contains encryption technology from Phaos Technology Corporation.

You may not download or otherwise export or reexport this Program, its Documentation, or any underlying information or technology except in full compliance with all United States and other applicable laws and regulations, including without limitation the United States Export Administration Act, the Trading with the Enemy Act, the International Emergency Economic Powers Act and any regulations thereunder. Any transfer of technical data outside the United States by any means, including the Internet, is an export control requirement under U.S. law. In particular, but without limitation, none of the Program, its Documentation, or underlying information of technology may be downloaded or otherwise exported or reexported (i) into (or to a national or resident, wherever located, of) Cuba, Libya, North Korea, Iran, Iraq, Sudan, Syria, or any other country to which the U.S. prohibits exports of goods or technical data; or (ii) to anyone on the U.S. Treasury Department's Specially Designated Nationals List or the Table of Denial Orders issued by the Department of Commerce. By downloading or using the Program or its Documentation, you are agreeing to the foregoing and you are representing and warranting that you are not located in, under the control of, or a national or resident of any such country or on any such list or table. In addition, if the Program or Documentation is identified as Domestic Only or Not-for-Export (for example, on the box, media, in the installation process, during the download process, or in the Documentation), then except for export to Canada for use in Canada by Canadian citizens, the Program, Documentation, and any underlying information or technology may not be exported outside the United States or to any foreign entity or "foreign person" as defined by U.S. Government regulations, including without limitation, anyone who is not a citizen, national, or lawful permanent resident of the United States. By using this Program and Documentation, you are agreeing to the foregoing and you are representing and warranting that you are not a "foreign person" or under the control of a "foreign person."

Content Server

Content Server Flexible Assets Course: Exercises and Solutions

Document Revision Date: November 7, 2003 2:48 pm

Product Version: 5.0

Table of Contents

| | |
|---|-----------|
| Module 1: Asset Taxonomy | 5 |
| Module Learning Objectives | 5 |
| Terms to Know | 5 |
| Lesson 1.1: Data Structure of the Flexible Asset Model | 7 |
| Flexible Asset Model | 7 |
| Default Columns and the Flex Asset Type Database Table | 8 |
| The _Mungo Tables | 9 |
| Database Tables | 9 |
| Lesson 1.2: Attributes and Attribute Editors | 11 |
| Attributes | 11 |
| About Attribute Editors | 11 |
| XML Code Examples | 12 |
| Exercise 1.2.1: Attributes and Attribute Editors | 16 |
| Lesson 1.3: Product Parent Definitions and Product Parents | 18 |
| Product Parent Definitions | 18 |
| Exercise 1.3.1: Product Parent Definitions and Product Parents | 21 |
| Lesson 1.4: Product Definitions and Products | 24 |
| Products | 24 |
| Product Definitions | 24 |
| Exercise 1.4.1: Product Definitions and Products | 25 |
| Lesson 1.5: Flex Asset Hierarchy | 27 |
| Attribute Inheritance | 27 |
| Nested Hierarchy | 27 |
| Module Summary | 29 |
| Module 2: Searching for Catalog Data | 31 |
| Module Learning Objectives | 31 |
| Terms to Know | 31 |
| Lesson 2.1: Creating Searchstates and Assetsets | 32 |

| | |
|--|-----------|
| Assetsets | 32 |
| Searchstates | 32 |
| Exercise 2.1.1: Searchstate and Assetset | 34 |
| Lesson 2.2: Displaying Attribute Values. | 35 |
| Displaying Attribute Values | 35 |
| Example Data Set for the Examples in this Chapter | 35 |
| Exercise 2.2.1: Product List Page | 41 |
| Exercise 2.2.2: Product Details Page | 43 |
| Exercise 2.2.3: Product Details Page 2 | 44 |
| Exercise 2.2.4: Creating Product Templates | 46 |
| Exercise 2.2.5: (extra credit) Product List Page 2 | 48 |
| Activity | 49 |
| Module Summary | 50 |
| Activity Answer Key | 51 |
| Module 3: Navigating Through a Product Tree | 53 |
| Module Learning Objectives | 53 |
| Lesson 3.1: Low Level Database Design | 54 |
| Products, Attributes, and Sites | 54 |
| Content Managers vs. Visitors | 54 |
| Developing a Product Tree | 55 |
| Attribute Inheritance and _Mungo Tables | 56 |
| Exercise 3.1.1: Creating a Navigation Bar | 58 |
| Lesson 3.2: Designing a Flex Family | 61 |
| Flex Asset Family Members. | 61 |
| Instructor Demonstration | 63 |
| Creating a Flex Family. | 63 |
| Module Summary | 65 |

Module 1

Asset Taxonomy

In this module you will learn about flexible asset model.

Module Learning Objectives

After completing this module, you will be able to:

- Define and create product attributes.
- Create attribute editors for attributes.
- Build a hierarchy of flexible assets.

Terms to Know

| Terms | Definition |
|--------------------|---|
| Flexible Asset | A flexible (flex) asset is one that can easily change properties, also called attributes. These properties will be stored in a special table as rows and can be easily removed or added. Unlike basic assets, assets of the same asset type may have different set of properties. |
| Attribute | Attribute describes a certain feature of an asset. An attribute is itself an asset. The example of an attribute can be color, description, price, etc. An attribute can have multiple values. A set of attributes defines a flexible asset type. |
| Attribute Editor | An XML based editor interface that allows users to easily enter data for the attribute. |
| Product Definition | A flex asset that is a set of attributes that describes a certain type of products. Example: Monitors, Computers, Printers, etc. |

| Terms | Definition |
|---------------------------|---|
| Product | <p>A flex asset that is an instance of a product definition.</p> <p>Example: ViewSonic Flat 15" monitor is an instance of a Monitors product definition.</p> |
| Product Parent Definition | <p>A flex asset that is a set of attributes that defines a group of products.</p> <p>Example: Discounted Items is a product parent definition that defines a group of products that were discounted due to damage.</p> <p>This product parent definition has a set of attributes, one of which can be discounted price.</p> |
| Product Parent | <p>A flex asset that is an instance of a product parent definition.</p> <p>For example, Office Merchandise is a product parent definition for product parents Office Furniture and Office Supplies.</p> |

Lesson 1.1: Data Structure of the Flexible Asset Model

In this lesson you will learn about the data structure of the flexible asset model.

Flexible Asset Model

A flex asset's attributes are stored in a different table than the attribute values. This differs from the schema for basic assets where attributes and the attribute values are stored in a single table and each attribute corresponds to a column in that asset type's table. Each attribute of a flex asset is itself an asset. This allows instances of flex assets to vary widely.

Flex assets are stored in the FlexAssetType tables. Attribute names and descriptions are stored in FlexAttributes tables. The attribute values are stored in the FlexAssetType_Mungo tables.

For example, for the Products flex asset type, all the fields that do not correspond to attributes are stored in the Products table:

| id | createdby | template | renderid | flextemplateid | externalid |
|----|-----------|----------|----------|----------------|------------|
| A | user01 | MJSTemp | 67845 | 34567 | 92468 |
| B | user02 | MJSTemp | 67845 | 34568 | 92468 |

Figure 1: Products Table

The name and description of each Product attribute is stored in the PAttributes table:

| id | createdby | name | description | updatedby |
|-----|-----------|-------|---------------|-----------|
| 111 | user01 | sku | Product sku | usser01 |
| 112 | user02 | color | Product color | user01 |
| 113 | user03 | price | Product price | user01 |

Figure 2: PAttributes Table

The value or values of each Product attribute are stored in the Products_Mungo table:

| id | ownerid | attrid | floatvalue | moneyvalue | textvalue | stringvalue |
|-------|---------|--------------|------------|------------|-----------|-------------|
| 11111 | | | | | | |
| 22222 | A | 111("sku") | 123789 | | | |
| 33333 | B | 111("sku") | 123790 | | | |
| 44444 | A | 112("color") | | | red | |
| 55555 | B | 112("color") | | | blue | |
| 66666 | A | 113("price") | | 4.5 | | |
| 77777 | B | 113("price") | | 5.25 | | |

Figure 3: Products_Mungo Table

Because of the complexity of the flex asset model, developers should use Content Server tags to retrieve flex asset data, rather than using SQL queries.

Default Columns and the Flex Asset Type Database Table

In general, the default columns in the primary table for a flex asset type are the same as the default columns in the primary table for a basic asset type.

However, there are exceptions to this rule, as described in the following table:

| Columns | Description |
|---------------------|--|
| category | Category is not used in the flex asset model. |
| renderid | The ID of the template asset that is assigned to a flex asset. |
| attributetype | The name of the attribute editor that formats the input style of the attribute when it is displayed in the New and Edit forms. This is an additional column in the primary table for flex attribute types. |
| flextemplateid | The ID of the flex definition that the flex asset was created with. This is an additional column in the primary table for a flex asset type. |
| flexgrouptemplateid | The ID of the parent definition that the flex parent asset was created with. This is an additional column in the primary table for flex parent asset types. |

The _Mungo Tables

The following columns in the _Mungo table always contain a value:

| Column | Description |
|--------------|---|
| id | A unique ID for each attribute value, automatically generated by Content Server when the flex asset is saved and the row is created. This is the table's primary key. |
| ownerid | The ID of the flex asset that the attribute value belongs to. (From the flex asset table: <code>Products</code> , for example.) |
| attrid | The ID of the attribute name. (Found in the <code>FlexAttributes</code> table.) |
| assetgroupid | The ID of the flex parent from which the attribute value is inherited. (From the parent table: <code>ProductGroups</code> , for example.) |

Each row in a _Mungo table also has all the following columns, but has a value (data) in only one of them:

| Column | Description |
|-------------|--|
| floatvalue | The value for each attribute with a data type of <code>float</code> . |
| moneyvalue | The value for each attribute with a data type of <code>money</code> . |
| textvalue | The value for each attribute with a data type of <code>textvalue</code> . |
| datevalue | The value for each attribute with a data type of <code>date</code> . |
| intvalue | The value for each attribute with a data type of <code>int</code> . |
| blobvalue | If the attribute's data type is <code>BLOB</code> , a pointer to the object, which is stored in the <code>MungoBlobs</code> table. |
| urlvalue | A pointer to the data for each attribute with a data type of <code>url</code> . |
| assetvalue | A pointer to the asset ID for each attribute with a data type of <code>asset</code> . |
| stringvalue | The value for each attribute with a data type of <code>float</code> . |

Because the _Mungo tables have URL columns, a default storage directory (`defdir`) must be set for it. Use the `cc.urlattrpath` property in the `gator.ini` file to set the `defdir` for your _Mungo tables.

Database Tables

Just as flex attribute assets compose a flex asset type, multiple flex asset types together compose a flex family.

Each asset type in a flex family has several database tables. For example, the flex asset member has six tables and a flex parent type has five. This data model enables the flex member in a flex family to support more fields than an asset type that uses the basic data model can.

The four most important types of tables in the flex model are as follows:

- The primary table for the asset type
- The `_Mungo` table, which holds attribute values for flex assets and flex parent assets only
- The `MungoBlobs` table, which holds the values of all the flex attributes of type blob
- The `_AMap` table, which holds information about the inheritance of attribute values for flex asset and flex parents only

There are several other tables that store supporting data about the relationships between the flex assets as well as additional configuration information (details about search engines, the location of foreign attributes, publishing information, and, if revision tracking is enabled, version information).

Additionally, certain kinds of site information are held in the same tables that basic assets use. For example, the `AssetPublication` table specifies which Content Server sites the asset type is enabled for.

When you develop the templates that display the flex assets that represent your content, you code elements that extract and display information from the `_Mungo` tables and the `MungoBlobs` table.

Lesson 1.2: Attributes and Attribute Editors

In this lesson you will learn about attributes and attribute editors

Attributes

An attribute is a CS-Direct Advantage asset that describes other CS-Direct Advantage assets. Attributes have unique names. For example, an attribute named `Size` can be the size of a product that is an item of clothing.

There are three attribute-related assets:

- **Product attribute** – an asset that describes a product or product parent. Examples of product attributes are `distributor`, `color`, `sku`, and `style`.
- **Content attribute** – an asset that describes content assets, like advanced article, advanced image, or content group. Examples of content attributes are `headline`, `authorname`, and `wordcount` for articles, and `width`, `height`, and `filetype` for images.
- **Attribute editor** an asset that is an input field for (or presentation object) for defining specific types of product or content attributes. You can use the CS-Direct Advantage Attribute Editor to create a custom input field for defining other product or content attributes. For example, you can create a drop-down list for selecting an attribute value from a set of choices. A `Size` attribute can then be represented as a string that has only certain values, like `Small`, `Medium`, `Large`, or `ExtraLarge`. For more information about attribute editors, please refer to the next section.

An attribute value is one of the following basic types:

- Date
- Float
- Integer
- Money
- String
- Text
- URL

About Attribute Editors

CS-Direct Advantage has a selection of attribute editors that you can use to create a custom input field (or presentation object) for defining a product attribute or content attribute. For example, you can create a drop-down list for selecting an attribute value from a set of choices, such as a set of colors or sizes.

To create a custom input field for defining attributes, click on **New** and then select **Attribute Editor** from the list of asset types. The **Attribute Editor** form appears:

- **Name** – the name of the attribute presentation object. This is a required value (indicated by the red asterisk).
- **Description** – text that describes the attribute presentation object.
- **Status** – refers to workflow status (if applicable). The administrator assigns workflow to asset types.

- **ID** – a unique identifier for the presentation object. This is generated by CS-Direct Advantage.
- **XML** in file – a pointer to the XML code that defines the presentation object. Enter the pathname or click the Browse button to locate the XML file in the file system.
- **XML** – a scrolling text box in which you can type or paste XML code.
- **Created** – identifies your login name as the creator of this presentation object after you save your changes.
- **Modified** – identifies the login name of the individual who subsequently modifies this presentation object.
- **Save Changes** – closes the Attribute Editor and saves the presentation object.
- **Cancel** – closes the Attribute Editor without saving the presentation object.

XML Code Examples

This section includes some XML examples that show how you might use the presentation object. There is a DTD file, named `presentationobject.dtd`, that is fairly simple and supports the following display element types:

- Text field
- Text area
- Pull-down menu
- Radio button
- Check box
- eWebEditPro Active X widget
- Pick-from-tree add button
- Remember button

The following are XML examples for each of the presentation object types.

TextField

You must specify the width (XSIZE); optionally, you can set MAXCHARS, which specifies the maximum number of allowable characters. The default is 256. You can also display a string of asterisks (*) in place of a typed entry, as a password mask for example (BLANKED="YES"; default is NO).

The following TEXTFIELD element defines the XSIZE as 60 and the maximum number of characters as 80:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM
"presentationobject.dtd">
<PRESENTATIONOBJECT NAME="TextFieldTest">
<TEXTFIELD XSIZE="60" MAXCHARS="80">
</TEXTFIELD>
</PRESENTATIONOBJECT>
```

TextArea

You must specify the width (XSIZE) and height (YSIZE) in pixels. Optionally, you can specify the WRAPSTYLE for the text, where the legal values for WRAPSTYLE include the HTML TEXTAREA styles SOFT, HARD, and OFF. The default is SOFT.

The following TEXTAREA element defines the XSIZE as 40 pixels, the YSIZE as 5 pixels, and disables text wrapping by setting WRAPSTYLE to OFF:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM
"presentationobject.dtd">
<PRESENTATIONOBJECT NAME="TextAreaTest">
<TEXTAREA XSIZE="40" YSIZE="5" WRAPSTYLE="OFF">
</TEXTAREA>
</PRESENTATIONOBJECT>
```

Pulldown

You can specify zero or more list items or a query that returns a list of items.

Optionally, you can specify the FONTSIZE for the element. The default is 3 (which is an HTML relative font size).

The following PULLDOWN element defines its FONTSIZE as 7 and adds three items to its list: red, green, and blue:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM
"presentationobject.dtd">
<PRESENTATIONOBJECT NAME="PulldownTest">
<PULLDOWN FONTSIZE="7">
<ITEM>Red</ITEM>
<ITEM>Green</ITEM>
<ITEM>Blue</ITEM>
</PULLDOWN>
</PRESENTATIONOBJECT>
```

RadioButton

You can specify zero or more list items or a query that returns a list of items. Optionally, you can specify the FONTSIZE for the element. The default is 3 (which is an HTML relative font size). You can specify a horizontal or vertical layout. The default is horizontal.

The following RADIOBUTTONS element defines its FONTSIZE as 7, and presents the results of the A Prods query in a vertical layout:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM
"presentationobject.dtd">
<PRESENTATIONOBJECT NAME="RadioButtonTest">
<RADIOBUTTONS FONTSIZE="7" LAYOUT="VERTICAL">
<QUERYASSETNAME>
A Prods
</QUERYASSETNAME>
</RADIOBUTTONS>
</PRESENTATIONOBJECT>
```

CheckBox

You can specify zero or more list items or a query that returns a list of items. Optionally, you can specify the `FONTSIZE` for the element. The default is 3 (which is an HTML relative font size). You can specify a horizontal or vertical layout. The default is orizontal.

The following `CHECKBOXES` element defines its `FONTSIZE` as 7, and presents the results of the `A Prods` query in a vertical layout:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM
"presentationobject.dtd">
<PRESENTATIONOBJECT NAME="CheckBoxTest">
<CHECKBOXES FONTSIZE="7" LAYOUT="VERTICAL">
<QUERYASSETNAME>A Prods</QUERYASSETNAME>
</CHECKBOXES>
</PRESENTATIONOBJECT>
```

eWebEditPro

The `eWebEditPro` attribute editor automatically launches the `eWebEditPro` WYSIWYG HTML editor in the content form, for entering and editing the associated attribute value, typically an HTML text file. The window has various button options for adding style characteristics to the text content. The attribute editor definition determines which buttons are available. Note the following about this attribute editor:

- You must have the `eWebEditPro` application properly installed (available from Ektron, Inc., at www.ektron.com)
- Text entry for an attribute of type `String` is limited to 256 and for type `text`, to 2000. For text entry that exceeds these limits, make the attribute type `url`.
- Encoding characters (`` ``, and so forth) are stored as part of the attribute value and count against character limitations. You must specify the width (`XSIZE`) and height (`YSIZE`) of the edit window in pixels. You must also specify a comma-separated list of style buttons.

Note: See the `presentationobject.dtd` file (“The DTD File”) for button and font options.

The following `EWEBEDITPRO` element defines an edit window 400 pixels wide by 200 pixels high, with buttons for bullets, searching, spell-checking, and tables:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM
"presentationobject.dtd">
<PRESENTATIONOBJECT NAME="eWebEditProTest">
<EWEBEDITPRO XSIZE="400" YSIZE="200"
BUTTONLIST="BULLETSBUTTON,FINDBUTTON,SPELLINGBUTTON,TABLEBU
TTON" FONTSIZE="3">
Chapter 2: Designing Catalog Assets
</EWEBEDITPRO>
</PRESENTATIONOBJECT>
```

PickFromTree

You can include an add button that implies attribute selection from the site tree. NAME is a label for the button.

The following PICKFROMTREE element defines a pick element with the label PickTest:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM
"presentationobject.dtd">
<PRESENTATIONOBJECT NAME="PickTest">
<PICKFROMTREE>
</PICKFROMTREE>
</PRESENTATIONOBJECT>
```

Remember

You can associate a button with attributes of type `asset` that, when clicked on a content form, displays a list of “remembered” assets of the same type from which to select a value for the attribute being created.

The XML for the REMEMBER element is as follows:

```
<?XML VERSION="1.0"?>
<!DOCTYPE PRESENTATIONOBJECT SYSTEM
"presentationobject.dtd">
<PRESENTATIONOBJECT NAME="RememberTest">
<REMEMBER>
</REMEMBER>
</PRESENTATIONOBJECT>
```

Exercise 1.2.1: Attributes and Attribute Editors

Purpose

The objective of this lab exercise is to create a new site and create a few attributes and an attribute editor.

By the end of this exercise you will have several attributes created and one attribute editor. These attributes will be used in the lab exercises that follow in this module for product and product parent definitions.

Directions

Using the step-by-step procedure(s) contained in this lesson, complete the following action(s) in CS-Direct Advantage:

1. Log in to Content Server with the `Coco/hello` user name and password and choose the **Ikea Furniture** site.
2. Create the following single-value product attributes:
 - PName (string)
 - IkeaColor (string)
 - IkeaDescription (string)
 - IkeaPrice (money)
 - IkeaHeight (float)
 - IkeaLength (float)
 - IkeaWidth (float)
 - IkeaProductImage (blob)
 - IkeaFabricType (string)

Note: A good development habit is to prefix attribute names with the abbreviation of your site. For example, if the site is called BestBuy, then Color attribute can be named BBColor. This is useful for product searches when using the CS-Direct Advantage API. All the attributes are shared between various sites in CS-Direct Advantage.

3. Create Attribute Editor called `IkeaColorList` to display a drop-down list box with the following color values (refer to the example of “Pulldown” on page 13):
 - white
 - black
 - green
 - red
 - silver
 - blue
 - brown

4. Assign the new Attribute Editor created in the previous step to the `IkeaColor` product attribute:
 - a. In the **IkeaStoreDesign** section of your screen, right-click on the **IkeaColor** attribute and select **Edit**.
 - b. In the **Attribute Editor** drop-down list box, choose the **IkeaColorList** attribute editor you would like to use for this attribute.
 - c. Click the **Save Changes** button to save the attribute.
5. Choose the already existing **CopyField** attribute editor and assign it to **PName** attribute.

Note

The `CopyField` attribute editor is going to copy the value of the `Name` field for the flex asset into the value of an attribute `PName`. This attribute is going to be used in the exercises in the next module.

Lesson 1.3: Product Parent Definitions and Product Parents

In this lesson you will learn about product parent definitions and product parents.

Some general characteristics of the flex asset model follow:

- Flex assets and flex parents are defined by selecting the flex attributes that describe them.
- The attributes that define the flex assets and flex parents are themselves assets. This means that you can use all of the content management features like workflow, access control, and so on with your individual attributes.
- Flex assets inherit attribute values from their parents. The definition asset types combined with the inheritance of attributes enable you to set up group hierarchies and implement some sort of taxonomy with your data.
- If you ever need to add attributes to your asset types in the future (a common occurrence with products), you just create the new attribute and assign it to the appropriate definitions. In contrast, with the basic asset model, you cannot add more attributes after you have created the asset type.

By using the definition asset types, you can set up multiple “templates” for the same flex asset type.

Product Parent Definitions

Similar to a product definition, a product parent definition serves as a template to define which attributes make up the product parent. In the example above, product parents `Monitors`, `Printers`, and `Scanners` all of have the same Parent Definition called `ComputerProducts`. This product parent definition has only two attributes:

- `Department`
- `ParentCompany`

All products of a product definition that is defined as a child of the `ComputerProducts` product parent definition would automatically inherit these attributes in addition to their own product definition attributes. In the example above, all the following will inherit the attributes:

- `Monitors`
- `15Inch`
- `17Inch`
- `Printers`
- `InkJet`
- `Laser`
- `Scanners`

All the products (monitors) that are listed under the `15Inch` and `17Inch` product parents will also inherit these two attributes on top of the once that product definition monitors have. The following is the complete set of attributes that products of the definition monitors will have:

- `Department` (inherited from the product parent)

- ParentCompany (inherited from the product parent)
- Manufacturer
- ScreenSize
- Resolution
- Price

Often, product parent definitions can be referred to as “levels”. For example, instead of naming product parent definition `ComputerProducts`, you can call the definition `Level2`. Then all of the product parents on the same level could be of the same definition. Thus, product parent definition `Level2` will have the following product parents:

- Headsets
- Televisions
- Monitors
- Printers
- Scanners

The parents of the product parents in the preceding list are `Audio&Video` or `Computers`. They belong to a Parent Definition, which can be called `Level1`.

Another product parent definition is `Level3`. This is the lowest level in this product tree and will have the following product parents:

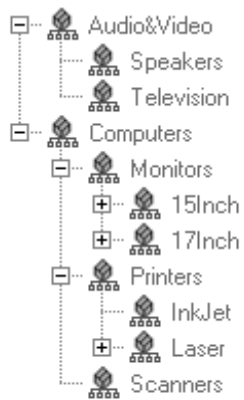
- 15Inch
- 17Inch
- InkJet
- Laser

If, at the same level of the tree, content providers want to have product parents with two different set of attributes, then two distinct product parent definitions must be defined. For example, the company may choose to sell discounted merchandise. In this case, the editor creates two different product parent definitions, `Full Priced Items` and `Discounted Items`, which visually will be on the same level of the product tree. The `Full Priced Items` product parent definition will include the following product parents:

- Monitors
- Printers
- Scanners

The `Discounted Items` will include one or more product parents from group discounted products. The `Discounted Items` product parent definition was one extra attribute called `discount`, which indicates what percentage will be taken off a full price of the product.

The following picture displays this hierarchy:



Though Discounted, Monitors , Printers, and Scanners are visually located in the same level of the hierarchy, they are children of different product parent definitions. The Discounted product parent is a child of a product parent definition called Discounted Items. Similarly, Monitors, Printers, and Scanners are children of the Full Priced Items product parent definition.

Exercise 1.3.1: Product Parent Definitions and Product Parents

Purpose

The objective of this lab exercise is to create new product parent definitions and product parents.

In this exercise, we will create the full product parent tree for the Ikea Furniture store (product parents only). At the end of the exercise, you will have a product tree similar to the following diagram:

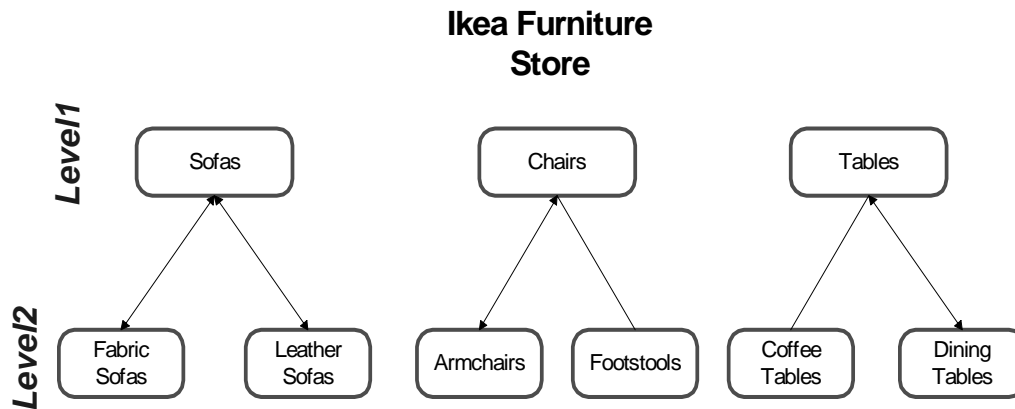


Figure 4: Ikea Furniture Store Product Parent Definitions and Product Parents

Directions

Using the step-by-step procedure(s) contained in this lesson, complete the following action(s) in CS-Direct Advantage:

Creating Product Parent Definitions

1. Create the following product parent definitions at Level1:
 - a. Click Content Server, click on the **New** button and then click on the **New Product Parent Definition** link in the list of asset types.
 - b. In the **Name** and **Description** fields, type Level1.
 - c. Leave the **Product Parents Definitions** unchanged.
 - d. In the **Attributes** field, do not select the attribute(s).
2. Create the following product parent definitions at Level2:
 - a. Click Content Server, click on the **New** button and then click on the **New Product Parent Definition** link in the list of asset types.
 - b. In the **Name** and **Description** fields, type Level2.
 - c. In the **Product Parents Definitions**, select **Level1** as a single required parent of Level2.

- d. In the **Attributes** field, do not select the attribute(s).

Creating Product Parents

1. Create a product parent of the `Level1` definition:
 - a. Click Content Server, click the **New button**, then click the **New Product Parent** link in the list of asset types.
 - b. In the **Name** field, type `Chairs`.
 - c. In the **Product Parent Definition** drop-down list, choose **Level1**, then click the **Continue** button.
2. Repeat the steps for creating product parents using the data supplied in the following table to create more product parents of the `Level1` definition:

| Product Parent | Product Parent Definition | Product Parent's Parent |
|----------------|---------------------------|-------------------------|
| Chairs | Level1 | None |
| Sofas | Level1 | None |
| Tables | Level1 | None |

3. Create a product parent of `Level2` definition:
 - a. Click Content Server, click on the **New button** and then click on the **New Product Parent** link in the list of asset types.
 - b. In the **Name** field, type `Armchairs`.
 - c. In the **Product Parent Definition** drop-down list, choose **Level2** and then click on the **Continue** button.
 - d. In the **Product Parents** section, the product parents will be listed in the drop-down list box (single parent). Select **Chairs** and then click the **Save** button.
CS-Direct will assign your `Armchairs` product parent to be a child of `Chairs` and will place `Armchairs` product parent (node) in the tree.
4. Repeat the steps for creating product parents using the data supplied in the following table to create more product parents of the `Level2` definition:

| Product Parent | Product Parent Definition | Product Parent's Parent |
|----------------|---------------------------|-------------------------|
| Armchairs | Level2 | Chairs |
| Footstools | Level2 | Chairs |
| Fabric Sofas | Level2 | Sofas |
| Leather Sofas | Level2 | Sofas |
| Sofa Beds | Level2 | Sofas |
| Coffee Tables | Level2 | Tables |

| Product Parent | Product Parent Definition | Product Parent's Parent |
|----------------|---------------------------|-------------------------|
| Dining Tables | Level2 | Tables |

Lesson 1.4: Product Definitions and Products

In this lesson you will learn about product definitions and products.

Products

A product is an individual saleable unit with an associated set of attribute values. Each product has a single product definition (see below), which determines how much information (or attributes) about the product appears on your catalog Web pages.

A product can also have one or more optional associated product parent, from which it inherits a set of attributes. A product not only has its own attributes, but can have the attributes of its product parent as well.

Product Definitions

A product definition is a set of attributes that defines one type, or class, of product. You create a named product definition that then serves as a template to create individual products with similar characteristics.

For example, you can create a product definition named monitor with the following attributes:

- Manufacturer
- ScreenSize
- Resolution
- Price

Instances of a product definition are called products. Bellow are three instances of the product definition monitors:

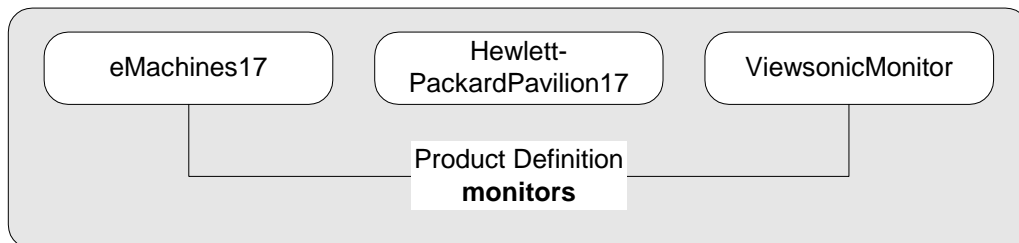


Figure 5: Product Definitions

All products of product definition monitors would then have the same set of named attributes. An individual product of type monitor might have the following values:

- Manufacturer = ViewSonic
- ScreenSize = 17"
- Resolution = 1280 x 1024
- Price = \$399

Exercise 1.4.1: Product Definitions and Products

Purpose

The objective of this lab exercise is to create new product definitions and products.

Directions

In this exercise you will create several product definitions, each of which will have a set of attributes (both required and not required). Then, you will create products for each product definition and assign values to their attributes.

At the end of the exercise, you will have a product tree similar to the following diagram:

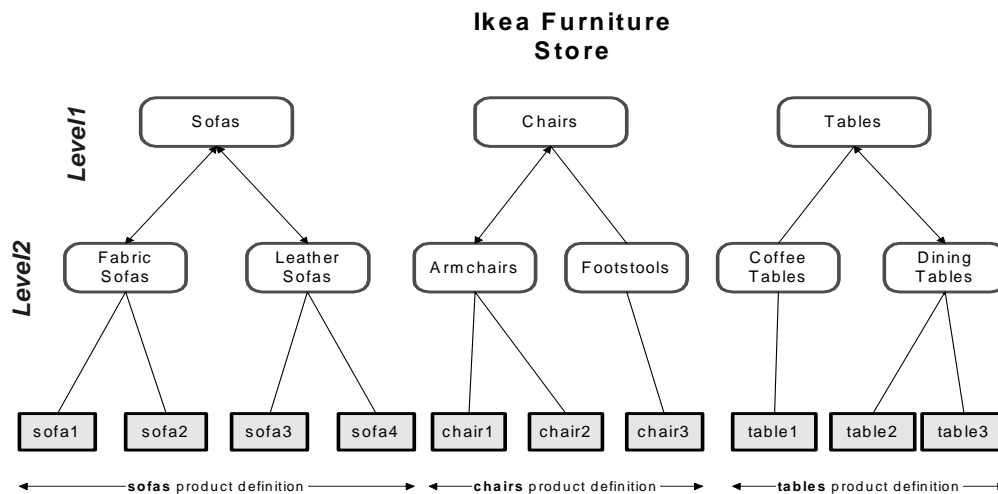


Figure 6: Completed Ikea Furniture Store Product Tree

Directions

Using the step-by-step procedure(s) contained in this lesson, complete the following action(s) in CS-Direct Advantage:

Creating Product Definitions:

1. In the Content Server, click the **New** button and then click the **New Product Definition** link in the list of asset types.
2. In the **Name** and **Description** text fields, type *chairs*.
3. In the **Product Parents Definitions box**, choose **Level2**.

In this field, you indicated that the **Level2** product parent definition will be a parent of the *Chairs* product definition. You can change this field at any time.

4. In the **Attributes** field, select the attribute(s) that will define product parent. Choose the following attributes for the *sofas* definition:
 - *IkeaPrice
 - *IkeaDescription

- *IkeaWidth
- *IkeaLength
- *IkeaHeight
- *IkeaFabricType
- IkeaColor
- IkeaProductImage

5. Using the steps above and Figure 6, add the following product definitions:

- tables (*IkeaPrice, *IkeaDescription, *IkeaWidth, *IkeaLength, *IkeaHeight, IkeaProductImage)
- sofas (*IkeaPrice, *IkeaDescription, *IkeaWidth, *IkeaLength, *IkeaFabricType, IkeaProductImage)

For each product definition, select `Level2` as a single required product parent definition.

If you like, you can make `Level2` to be a multiple required product parent. This allows you to assign multiple parents to a single product.

Creating Products

1. Create the first product (chair1):

- a. In the Content Server, click the **New** button, then click the **New Product** link in the list of asset types.
- b. In the **Name** field, type `chair1`.
- c. In the **Product Definition** drop-down list, choose **chairs** then click the **Continue** button.

Note: Once you assign a product to a particular product definition that cannot be changed. In order to re-assign the product to a different definition, you need to delete the product and re-create it.

- d. In the **Product Parents** section, the product parents (**Armchairs** and **Footstools**) will be listed in the drop-down select box (single parent). Select a product parent (for example, **Armchair**).
- e. Assign values to at least all the required attributes and then click the **Save** button.

2. Using the preceeding steps and the data in Figure 6, add more products.

Note: CS-Direct will assign your new product to a correct product parent (node) in the tree. Sometimes, products can be stored under multiple product parents that belong to multiple product parent definitions. To do this, set the **Product Definition Parent** field to have multiple product parent definitions.

Note: When assigning products to multiple product parents, make sure that the attributes that are inherited from the parents have multiple values. For example, if product parent A and product parent B both have attribute `Attribute1` the attribute will be inherited by the child product C. Since there could be more than one value for the attribute `Attribute1`, the child product needs to be able to inherit both of these values without a conflict.

Lesson 1.5: Flex Asset Hierarchy

In this lesson you will learn about flex assets hierarchy.

Attribute Inheritance

When designing a product tree, one should keep in mind the following rules of attribute inheritance:

- If `product1` inherits `attribute1` from `parent1` and `parent2`, then `attribute1` must be able to accept multiple values.
- If `product1` that inherits the value of an `attribute1` can override this value if product definition that defines `product1` has the same `attribute1`.

The following example demonstrates these rules:

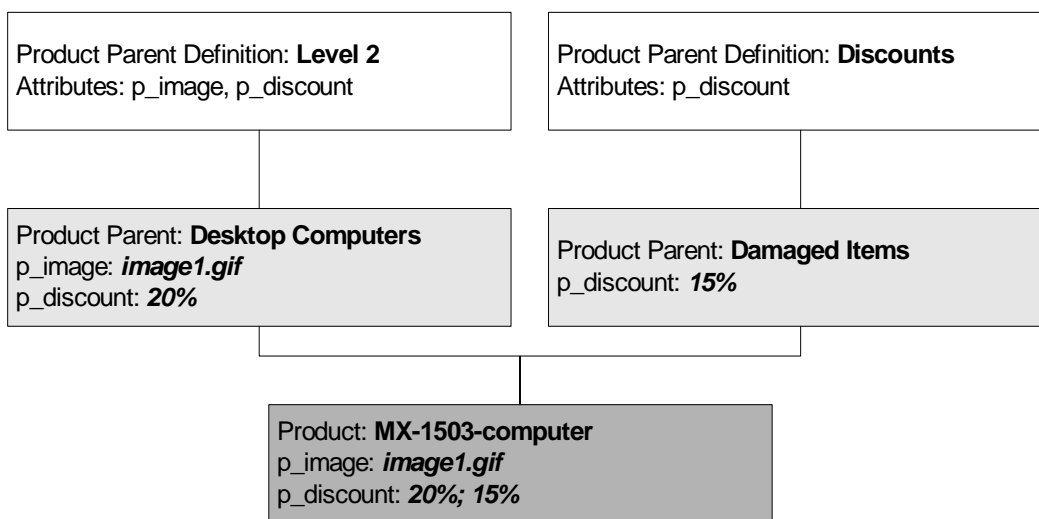


Figure 7: Attribute Inheritance

- If `discount` attribute accepts multiple values, then the product will have **two** values for discount. If `discount` attribute accepts only a single value, then the product will **not** be saved.
- If the product only has one parent, either `Desktop Computers` or `Damaged Items`, then the `discount` attribute can be a single value. If the product definition has also `discount` attribute, then the value inherited from the product parent can be overwritten at the product level.

Nested Hierarchy

When designing a product tree, one has to think in terms of attributes and definitions. You must identify a set of attributes that a particular product should have, and think about some common attributes the products can inherit from the product parent.

The number of attributes that are inherited from the product parent must be limited to a minimum, since each of the attribute values assigned to a Product Parent will be propagated down to the product level of the tree. Each inherited value for each product

parent and product will results in an additional row in the ProductGroup_Mungo and Product_Mungo tables.

Module Summary

- There are two types of assets in the Content Server environment – basic and flex.
- A basic asset has properties that correspond to columns in the database. All the assets of the same asset type have the same set of properties. The properties are defined by the Asset Maker when the asset type is created, and cannot be modified.
- A flexible (flex) asset is one that can easily change properties, also called attributes. These properties are stored in a special table as rows and can be easily removed or added. Unlike basic assets, assets of the same asset type may have different set of properties.
- Flex asset fields are stored in two tables: the *AssetTypeName* (fixed fields only) table and *AssetTypeName_Mungo* (properties/attributes) table. *AssetTypeName_Mungo* table stores the properties for the asset type in rows rather than column; therefore, those properties can be easily added, deleted, and modified.
- An attribute describes a certain feature of an asset. An attribute is itself an asset. The example of an attribute can be *color*, *description*, *price*, etc. An attribute can have multiple values. A set of attributes defines a flexible asset type.
- An attribute describes a certain feature of an asset. The example of an attribute can be *color*, *description*, *price*, etc.
- An attribute editor is an XML based editor interface that allows users to easily enter data for the attribute.
- A product definition is a flex asset, that is composed of a set of attributes which describe a certain type of product.
- A product is a flex asset that is an instance of a product definition. Example: *Reebok Classic White* is an instance of the *Reebok Classics* product definition.
- A product parent definition is a flex asset that is a set of attributes which defines a group of products.
- Product parent definition is a flex asset that is an instance of a product parent definition. For example, *Athletic Shoes* is a product parent definition for product parents *Walking Shoes* and *Running Shoes*.

Module 2

Searching for Catalog Data

In this module you will learn how to use CS-Direct advantage API to search for catalog data.

Module Learning Objectives

After completing this module, you will be able to:

- Create templates for flex assets.
- Use CS-Direct `assetset` tags.
- Use CS-Direct `searchstate` tags to filter the search for flex asset.
- Learn how to display attribute values.
- Navigate and search for products in a CS-Direct Advantage site.

Terms to Know

| Term | Definition |
|--------------------|--|
| searchstate | A searchstate is an object which contains a set of search constraints. You execute a searchstate against Content Server tables to create assetsets. |
| assetset | An assetset is a group of one or more flex assets or flex parent assets. |
| LISTOBJECT | A listobject contains a list of the attributes whose values you want to find. When applied against an assetset, you derive the attribute values for the assets in that assetset. |

Lesson 2.1: Creating Searchstates and Assetsets

In this lesson you will learn how to create a searchstate and use it while searching for flex assets.

Assetsets

An assetset is a group of one or more flex assets or flex parent assets. You use the `assetset` tags to create the set of assets and to extract the attribute values that you want to display.

You can retrieve the following information from an assetset:

- The values for one attribute for each of the flex assets in the assetset.
- The values for multiple attributes for each of the flex assets in the assetset.
- A list of the flex assets in the assetset.
- A count of the flex assets in the assetset.
- A list of unique attribute values for an attribute for all flex assets in the assetset.
- A count of unique attribute values for an attribute for all flex assets in the assetset.

You can create assetsets that include flex assets of more than one type, but only if those flex assets use the same flex attribute type.

The most commonly used `assetset` tags are:

- `assetset:setasset`
- `assetset:setsearchedassets`
- `assetset:getmultiplevalues`
- `assetset:getattributevalues`
- `assetset:getassetlist`

Searchstates

How do you obtain the IDs of the flex assets that you want to display?

A searchstate is a set of search constraints based on the attribute values held in the `_Mungo` table for the flex asset type. You apply searchstates to assetsets.

You build a searchstate by adding or removing constraints to narrow or broaden the list of flex assets that are described by the searchstate. For example, the GE Lighting sample site uses searchstates to create drill-down searching features that visitors use to browse through the product catalog.

An unconstrained searchstate applied to an assetset creates an unfiltered list of all the assets of that type. For example, the following code sample would create an assetset that contains all the products in the GE Lighting catalog:

```
<searchstate:create name="nolimits" />
<assetset:setsearchedassets name="unconstrainedAssetSet"
constraint="nolimits" assettypes="Products" />
```


To narrow the number of products in the assetset, you add constraints. For example, the following code sample would create an assetset that contains only the 40-watt lightbulbs from the catalog:

```
<searchstate:create name="lightbulbs"/>

<searchstate:addsimplestandardconstraint name="lightbulbs"
typename="PAttributes" attribute="wattage" VALUE="40"/>

<assetset:setsearchedassets name="40WattLightbulbs"
constraint="lightbulbs" assettypes="Products"/>
```

A constraint is a filter (restriction) that can be based on the value of an attribute or it can be based on another searchstate, which is called a nested searchstate.

CS-Direct Advantage searchstate can search either the _Mungo table or the attribute indexes created by a search engine. This means that you can mix database and rich-text (full-text through an index) searches in the same query. To apply a constraint against a search engine index, use the `searchstate:addrichtextconstraint` tag.

Because CS-Direct Advantage provides the `searchstate` tag family, which is the most effective way to retrieve assetsets, do not use SQL to query the flex asset database tables.

Exercise 2.1.1: Searchstate and Assetset

The objective of this lab exercise is to learn how to define a searchstate, add constraints to it, and apply the searchstate to an assetset.

Purpose

In this exercise you will first create a searchstate with no constraints and search for all the products in the product tree of GE Lightning. Later, you can choose to add a constraint to your searchstate to only search for products that you created as part of your sample site.

Directions

Complete the following action(s) in Content Server Explorer:

1. Open the `ElementCatalog/IkeaFurniture/JSP/assetset` element.
2. Create a blank searchstate. A blank searchstate has no constraints placed on it.
3. Use the tags `<assetset:setsearchedassets>` and `<assetset:getassetcount>` to get the total number of assets in the database.
4. Use `<ics:getvar>` to show the number of assets in the database on your web page.
The number that will appear will be the total number of assets in your database.
5. Now try using similar code to find the exact number of products in our catalog.
 - a. Create another searchstate with a different name as well as a new assetset.
 - b. Use the `assettypes="Products"` argument to constrain your new assetset to one type of asset.
6. Test your exercise in the browser by calling the `IkeaFurniture/JSP/AssetSet` page. To preview the page in Content Server, complete one of the following actions:

In the browser, type the following URL:

- `http://localhost:7001/servlet/ContentServer?pagename=IkeaFurniture/JSP/AssetSet`

or

In Content Server Explorer, in the SiteCatalog,

- right-click on the page `IkeaFurniture/JSP/AssetSet` and select **Preview page**.
 - Do not provide any arguments. Click the **OK** button.
7. Add a constraint to your searchstate to select only the products that are priced between \$20 and \$1000 dollars.
Use the attribute `IkeaPrice` to find only the products (chairs, tables, or sofas) in the Ikea Furniture site. If you changed the name of the attribute when creating your products, use the appropriate name for your attribute.
 8. Test your exercise again in the browser by calling the `IkeaFurniture/JSP/AssetSet` page.

Lesson 2.2: Displaying Attribute Values

In this lesson you will learn about how to display attribute values for product parents or products.

Displaying Attribute Values

When you code templates for an online site that uses the flex asset model, you are primarily concerned with the values of flex attributes, which are assets themselves.

When you display a basic asset, you use tags that call the asset as a whole, and then display individual components of that asset. In contrast, when you display a flex asset, you do not call the asset as a single unit. Instead, you use tags that call only the attributes of the asset that you want to display.

You use searchstate tags to specify your search criteria. This is roughly analogous to a SQL query. The `<assetset:setsearchedassets>` method applies the searchstate against the database and returns an assetset, which is analogous to a resultset. Assetsets contain the assetids and attribute values of flex assets that match your search criteria. You can then display the attribute values contained in the assetset.

Be sure that you understand the data model of the flex family (or families) that you are using before you begin coding template elements for your flex assets.

Example Data Set for the Examples in this Chapter

The GE Lighting sample site and the CS-Engage extensions to the Burlington Financial sample site illustrate the full power of the flex asset data model and the coding toolset that is delivered with CS-Direct Advantage. The templates and elements in the sample sites illustrate the code for fully functioning online sites that display a nearly real-world amount of data.

The example data set is based on the product flex family, as follows:

| Flex Asset Type | Internal Name (as displayed in the Content Server Interface) | Internal Name (as displayed in the Content Server Database) |
|-----------------|--|---|
| flex attribute | product attribute | PAttributes |
| flex asset | product | Products |
| flex parent | product parent | ProductGroups |

Note

Always use the internal name of the asset type when you use `assettypes`.

The example products in this example data set are pairs of blue jeans that have the following attributes:

| Attribute | Data Type | Number of Values |
|-----------|-----------|------------------|
| sku | string | single |
| color | string | multiple |
| price | integer | single |
| style | text | single |

There are four pairs of blue jeans, defined as follows:

| SKU | Price | Color | Style |
|---------|-------|--------------|----------|
| jeans-1 | 35 | blue | wide |
| jeans-2 | 30 | blue, black | straight |
| jeans-3 | 25 | black, green | straight |
| jeans-4 | 20 | green | flair |

Examples of Assetsets with One Product (Flex Asset)

The code samples in this section do the following:

- Create an assetset that contains one pair of jeans, identified by its sku number
- Log a dependency between the product asset and the rendered page(let)
- Get and display the value for the price attribute and display it
- Get and display the values for the color attribute and display them
- Get and display the values for both the price and color attribute with the same tag (assetset:getmultiplevalues)

Create a Searchstate and Apply It to an Assetset

This line of code creates an unfiltered searchstate named ss:

```
<searchstate:create name="ss"/>
```

Next, we can narrow the unfiltered searchstate named ss so that it finds a specific product in the sample data set, by providing the sku of the product:

```
<searchstate:addrangeconstraint name="ss" typename="PAttributes"
attribute="sku" value="jeans-2"/>
```

Now we can create an assetset named as, applying the searchstate named ss to it:

```
<assetset:setsearchedassets name="as" constraint="ss"
assettypes="Products"/>
```

Since the value of the sku attribute is unique for each product asset, there is only one product in the assetset: the one whose sku value is jeans-2.

Log the Dependency

Because we plan to display information about the jeans-2 product, we need to log the dependency between the jeans-2 asset and the page rendered by this code.

To do that, first we need the ID of the asset. This line of code creates an IList named aslist, and stores all the data for the jeans-2 product-including its ID-in the list:

```
<assetset:getassetlist name="as" listvarname="aslist"/>
```

Now this line of code can log the dependency:

```
<render:logdep cid="aslist.assetid" c="aslist.assettype"/>
```

Get the Price of the Product

Next, let's extract the price of this pair of jeans:

```
<assetset:getattributevalues name="as" attribute="color"
typename="PAttributes" listvarname="pricelist"/>
```

Notice that even though price is a single-value attribute (which means the product only has one price), the `assetset:getattributevalues` tag returns the value of the price attribute as a list variable (`listvarname="pricelist"`).

Display the Price of the Product

Now the following line of code can display the price of the jeans-2 product:

```
Price: <ics:listget listname="pricelist" fieldname="value"/><br/>
```

And this is the result:

```
Price: 30
```

Get the Colors for the Product

Next, let's determine which colors this pair of jeans is available in. As specified above, the color attribute is a multiple-value attribute. Because the `assetset:getattributevalues` tag works the same whether an attribute is a single-value or a multiple-value attribute, we use the tag exactly as we did for single-value price attribute:

```
<assetset:getattributevalues name="as" attribute="color"
typename="PAttributes" listvarname="colorlist"/>
```

Display the Colors of the Product

Now the following code can display the colors for the jeans-2 product, and, because this product can have more than one color, the code loops through the list:

Colors:

```
<ics:listget listname="colorlist" fieldname="#numRows"
output="rows"/>
<ics:if condition="<%= ics.GetVar("rows").compareTo("0") == 0 %>">
  <ics:then>
    No values available for this attribute
  </ics:then>
<ics:else>
  <!-- list all the values for an attribute here -->
  <ics:listloop listname="colorlist" maxrows='<%=
ics.GetVar("rows") %>'>
    <ics:listget listname="colorlist" fieldname="value"/><br/>
  </ics:listloop>
```

And this is the result:

Colors: **black blue**

Create a List Object for the <assetset:getmultiplevalues> tag

In general, you should not use the `assetset:getattributevalues` tag when you want to get the value for more than one attribute.

The `<assetset:getmultiplevalues>` tag gets and scatters the values from more than one attribute, for all the assets in an assetset. Because the tag makes only one call to the database for all the attribute values, it performs the query more efficiently than using multiple `<assetset:getattributevalues>` tags.

Before you can use this tag, however, you must use the `listobject` tags to create a list object, which contains the attributes whose values you would like to retrieve. The list object is passed into the `<assetset:getmultiplevalues>` tag, which then returns the values of the attributes named in the list object. The list object needs one row for each attribute that you want to get.

This next line of code creates a list object named `lo` that has columns named `attributetypename`, `attributename`, and `direction`.

```
<listobject:create name="lo"
columns="attributetypename,attributename,direction"/>
```

Then, this line adds a row to the list object for each attribute, color and price:

```
<listobject:addrow name="lo">
  <listobject:argument name="attributetypename"
value="PAttributes"/>
  <listobject:argument name="attributename" value="price"/>
  <listobject:argument name="direction" value="none"/>
</listobject:addrow>

<listobject:addrow name="lo">
  <listobject:argument name="attributetypename"
value="PAttributes"/>
```

```

    <listobject:argument name="attributename" value="color"/>
    <listobject:argument name="direction" value="none"/>
</listobject:addrow>

```

The next line of code converts the list object to a list variable named lolist:

```
<listobject:tolist name="lo" listvarname="lolist"/>
```

Get and display values for price and color attributes with <assetset:getmultiplevalues> tag for a single product

And now we can get the values for both the price and the color attribute from our original assetset, named as:

```

<assetset:getmultiplevalues name="as" list="lolist"
byasset="false" prefix="multi"/>

```

Display the Value of Price and Color for the jeans-2 Product

Now that the values are stored in the list variable (lolist), the following code can display all the values for all the attributes:

```

<br><b>Price:</b>
<ics:if condition='<%= ics.GetErrno() != 0%>'>
  <ics:then>
    No values available for this attribute
  </ics:then>
  <ics:else>
    <!-- list all the values for an attribute here -->
    <ics:listloop listname="multi:price">
      <ics:listget listname="multi:price" fieldname="value"/>
    </ics:listloop>
  </ics:else>
</ics:if>

<br><b>Color:</b>
<ics:if condition='<%= ics.GetErrno() != 0%>'>
  <ics:then>
    No values available for this attribute
  </ics:then>
  <ics:else>
    <!-- list all the values for an attribute here -->
    <ics:listloop listname="multi:color">
      <ics:listget listname="multi:color" fieldname="value"/>
    </ics:listloop>
  </ics:else>
</ics:if>

```

This code sets up a nested loop that loops through all the attributes in the lolist variable, and then loops through all the distinct attribute values for each of the attributes in the lolist list variable.

The resulting page will look like this:

```

price is: 30
color is: blue black

```

Get and display values for price and color attributes with <assetset:getmultiplevalues> tag for multiple products

And now we can get the values for both the price and the color attribute from our original assetset, named as:

```
<assetset:getassetlist name="as" listvarname="asset_list"/>
<assetset:getmultiplevalues name="as" list="lolist"
byasset="false" prefix="multi"/>
```

Display the Value of Price and Color for multiple products

```
<%
    IList resultList = ics.GetList("asset_list");

    for (int j=1; j <= resultList.numRows(); j++) {
        resultList.moveTo(j);

        out.write("<br>Asset ID: " + resultList.getValue("assetid"));

        IList plist = ics.GetList("multi:" +
resultList.getValue("assetid") + ":price");
        IList clist = ics.GetList("multi:" +
resultList.getValue("assetid") + ":color");

        if (plist != null && plist.hasData()) {
            for (int i = 1; i <= plist.numRows(); i++) {
                plist.moveTo(i);
                out.write("<br>Price: " + plist.getValue("value"));
            } //for
        } else {
            out.write("<br>Price: attribute has no data");
        }

        if (clist != null && clist.hasData()) {
            for (int i = 1; i <= clist.numRows(); i++) {
                clist.moveTo(i);
                out.write("<br>Color: " + clist.getValue("value"));
            } //for
        } else {
            out.write("<br>Color: attribute has no data");
        }
    }
%>
```


Exercise 2.2.1: Product List Page

The objective of this lab exercise is to learn how to create a listobject. A listobject is used to create a resultset using information from an assetset.

Purpose

In this exercise you will first create a listobject, which will consist of several rows. Each row will correspond to a particular attribute. You will then apply the listobject to an assetset to filter the assetset to only those assets that have the values for all of the attributes in the listobject. As the result of your search you will receive a resultset in the form of an IList. You will then traverse the resultset to display all of the values of the attributes for your assets.

Directions

Complete the following action(s) in Content Server Explorer:

1. Open the `ElementCatalog/IkeaFurniture/JSP/productlist` element.
2. Using the assetset you created in your last exercise, create a list object that will find the values of the `IkeaPrice` and `IkeaColor` attributes for all the products in the assetset.
 - a. Use the `<listobject:create>`, `<listobject:addrrow>` and `<listobject:tolist>` tags to create your listobject.
 - b. You will need to use one `<listobject:addrrow>` tag for each `IkeaPrice` and `IkeaColor` attribute.

Note

All the attribute names are case sensitive. You can select any set of attributes that you want to search for. Make sure that at least one product in your product tree has values for all of the attributes that you include in the listobject.

3. Using the `<assetset:getassetlist>` tag, apply the listobject you created to an assetset. This will return a resultset *somelist* with sorted values of attributes.
4. Create an HTML table by looping through all the values of `IkeaPrice` and `IkeaColor` for your products.
 - a. Use the Content Server `<ics:listloop>` and `<ics:listget>` tag to traverse the resultset returned in the previous step.
 - b. For each row in the resultset, using the `<ics:getvar>` tag, display the asset ID and the values for `PName` and `IkeaPrice` attributes.

Use the format `SORT_attributename` to reference the attribute values returned in the list *somelist*. For example:

```
<ics:listget listname="somelist" fieldname="SORT_IkeaPrice"/>
```

Use `assetid` to reference the ID of the asset. For example:

```
<ics:listget listname="somelist" fieldname="assetid"/>
```

5. Test your exercise in the browser by calling the `IkeaFurniture/JSP/ProductList` page. To preview the page in Content Server, complete one of the following actions:
 - Type the following URL in your browser:
`http://localhost:7001/servlet/ContentServer?pagename=IkeaFurniture/JSP/ProductList`
or
 - Using Content Server Explorer, in the SiteCatalog, right-click on the page `IkeaFurniture/JSP/ProductList` and select **Preview page**.
Do not provide any arguments and click **OK**.

Exercise 2.2.2: Product Details Page

The objective of this lab exercise is to learn how to use the tag `<assetset:getattributevalues>` to display the values for attributes for a single product.

Purpose

In this exercise, unlike in the previous ones, where you were dealing with multiple assets (products), you will be only working with the single asset (product). You will first learn how to set an assetset to a single product id. You will then use `<assetset:getattributevalues>` tag to get the value(s) of an individual attribute for the product in the assetset.

Directions

Complete the following action(s) in Content Server Explorer:

1. Using the Content Server Explorer, open the `IkeaFurniture/JSP/productdetails` element.
2. To accept, evaluate, and print the product ID passed to this page, add the following code to your element :
Product ID: `<ics:getvar name="cid">`
3. Use the `<assetset:setasset>` tag to create a single-asset assetset. Set the assetset to the value of `cid` (product ID).

4. Use the `<assetset:getattributevalues>` tag to get the attribute value for product name (PName) product description (IkeaDescription) and the value for the Color attribute (IkeaColor). The resultset returned by the `<assetset:getattributevalues>` will contain the values for the attributes.

Using `<assetset:getattributevalues>`, you can display the values for other attributes. Use the correct names for your attributes.

5. Test your exercise in the browser by calling the `IkeaFurniture/JSP/ProductDetails` page. To preview the page in Content Server, complete one of the following actions:
 - Type the following URL in your browser:
`http://localhost:7001/servlet/ContentServer?pagename=IkeaFurniture/JSP/ProductDetails&cid=ProductID`
6. In the previous exercise you have created a product list page. In the `IkeaFurniture/JSP/productlist` element, create a hypertext link that will link this page to your `IkeaFurniture/JSP/ProductDetails` page and pass the variable `assetid` with the URL. Your `ProductDetails` page should accept `assetid` as a product id and display its details.
7. Test your exercise again in the browser by calling the `IkeaFurniture/JSP/ProductList` page first and then selecting one of the products on the page.

Exercise 2.2.3: Product Details Page 2

The objective of this lab exercise is to learn how to use the tag `<assetset:getmultiplevalues>` to display the values for multiple attributes for a single product.

Purpose

In this exercise you will use the `<assetset:getmultiplevalues>` tag to get the values of multiple attributes for a single product in the `assetset`. You will first need to create a `listobject` with all the attributes that you want to view in a product and then apply this `listobject` to an `assetset`.

Directions

Complete the following action(s) in Content Server Explorer:

1. Using Content Server Explorer, open the `IkeaFurniture/JSP/productlist` element. Change the hyperlink created in the previous exercise to `IkeaFurniture/JSP/ProductDetails2` page. Leave the rest of the code unchanged.
2. Open the `IkeaFurniture/JSP/productdetails2` element.
3. Similar to the previous exercise, use the `<assetset:setasset>` tag to create a single-asset `assetset`. Set the `assetset` to the value of `cid` (product ID).
4. Create a `listobject` with the following attributes as rows:
 - PName
 - IkeaPrice
 - IkeaColor
 - IkeaHeight
 - IkeaLength
 - IkeaWidth

Note

For the future use of the `listobject` with the `<assetset:getmultiplevalues>` tag in this version of the product, set the `direction` argument in your `listobject` to **none**.

5. Use the `<assetset:getmultiplevalues>` tag to apply the `listobject` created in the previous step to an `assetset`.
6. Use `<assetset:getmultiplevalues>` to return separate lists of values for each attribute in the `listobject`. The name of the list for each attribute can be referred to as following: `prefix:attributename` (the prefix should be specified as argument in the `<assetset:getmultiplevalues>` tag). Please refer to the tag syntax and the examples in the “Create a List Object for the `<assetset:getmultiplevalues>` tag” on page 38 of this lesson.

Note

When using `<assetset:getmultiplevalues>` with a single asset, set the `byasset` argument to **false**. If you set it to `true`, you will have to know the ID of the asset in order to access the attribute values.

7. For each attribute complete all of the following steps:
 - a. Check whether the list is not empty by using the conditional statement with the `IList` predicate. For example, to check whether the product has at least one value for an attribute refer to code examples in “Display the Value of Price and Color for the jeans-2 Product” on page 39 in this lesson)
 - b. If the list is not empty, display the values in the list.
 - c. If the list is empty, indicate that no attribute value exists.
8. Test your exercise in the browser by calling the `IkeaFurniture/JSP/ProductList` page first and then selecting one of the products on the page.

Exercise 2.2.4: Creating Product Templates

The objectives of this exercise to learn how to create different Product templates for different Product definitions.

Purpose

In this exercise you will use create several templates that will help content users to preview the products in the CS-Direct Advantage UI.

Directions

Complete the following action(s) in Content Server Direct Advantage Interface:

Creating a Product Template for the Chairs Product Definition

1. If you are not already logged in, log in to the **IkeaFurniture** site with the `Coco/hello` user name and password.
2. In the top left corner of the screen, click the **New** tab. The **New Screen** appears with the list of asset types enabled in the **Ikea Furniture** site.
3. Click the **New Template** link. The Template form appears.
4. In the **Name** field, type `ChairProductTemplate`.
5. From the **For the Asset Type** drop-down list box, select **Products** and then click the **Continue** button.
6. In the **Description** field, type the following:
`Template to display attribute values for Products of chairs definition`
7. From the **Source** drop-down menu, select **IkeaFurniture**.
8. From the **Applies to subtypes:** (Product Definitions) box select **chairs**.
You can select multiple Product Definitions that this template could be used for if you think that they share the same attributes.
9. In the **Create Template Element?** section, click **JSP**.
10. In the **Template Element Description** field, enter a description of the template.
When you save this template asset, the information in this field is written to the description column for the element entry in the `ElementCatalog` table.
11. In the Template Element Logic entry area, copy code from the `productdetails2` element that you coded in the exercise “Product Details Page 2” on page 44.
12. Modify your template code if necessary to include the attributes that `chairs` Product Definition has.
13. Click the **Save** button to save your template.

Creating Product Templates for other Product Definition

1. Using the steps in the “Creating a Product Template for the Chairs Product Definition”, create templates for other Product Definitions.
2. Modify the code for your template where you display the attributes that are appropriate for these definitions.

Testing a Product Template in the CS-Direct Advantage interface

1. Assign your template to one of the chair products assets:
 - a. Click the **Edit** button to edit a chair asset.
 - b. In the **Template** field, choose **ChairProductTemplate** as the default template.
 - c. Save your changes.
2. Preview your chair product asset by clicking the **Preview** button.
3. Using the steps above, test your other templates.

Exercise 2.2.5: (extra credit) Product List Page 2

The objective of this lab exercise is to learn how to use the `<assetset:getmultiplevalues>` tag to display multiple attribute values for multiple products.

Purpose

In this exercise you will use the `<assetset:getmultiplevalues>` tag to get the values of multiple attributes for multiple products in the assetset. You will first need to create a listobject with all the attributes that you want to view, and then apply this listobject to an assetset.

Directions

Complete the following action(s) in Content Server Explorer:

1. Open the `IkeaFurniture/JSP/productlist2` element.
2. Create a listobject with the following attributes as rows:
 - `IkeaPrice`
 - `IkeaColor`

Note

For the future use of the listobject with the `<assetset:getmultiplevalues>` tag in this version of the product, set the `direction` argument in your listobject to **none**.

3. Use `<assetset:getassetlist>` to get the list of assets that have values for all the attributes in the listobject.
4. Use `<assetset:getmultiplevalues>` to return separate lists of values for each attribute in the listobject. The lists will be sorted by asset.

Note

When using `<assetset:getmultiplevalues>` with multiple assets, set the `byasset` argument to **true**.

5. Display the values for each list and each asset in the list. Please refer to the tag syntax and the examples in the “Display the Value of Price and Color for multiple products” on page 40 of this lesson.
6. Test your exercise in the browser by calling the `IkeaFurniture/JSP/ProductList2` page.

Activity

Review the previous lesson and answer the questions below. The answer key is found on page 35.

Based on your understanding of CS-Direct Advantage API, answer the following questions:

1. If the `listobject` contains the following attributes:

- `attr1`
- `attr2`
- `attr3`

Can the product be displayed if it has 2 values for `attr1` and one value for `attr2`?

2. What will the search return if the `listobject` is applied to an `assetset`?
3. Can the asset id be displayed when using the `<assetset:getattributevalues/>` tag on an `assetset` with multiple assets?
4. Can the asset id be displayed when using `<assetset:getassetlist/>` tag on an `assetset` with multiple assets?

True or False

Mark each statement true or false.

1. _____ `<assetset:getattributevalues>` is mainly used when displaying attributes for a single asset.
2. _____ `<assetset:getattributevalues>` returns a list of assets.
3. _____ `<assetset:getmultiplevalues>` can be used for either displaying multiple attribute values for a single asset or multiple attribute values for multiple assets in the `assetset`.

Module Summary

- An assetset is a group of one or more flex assets or flex parent assets. You use the `assetset` tags to create the set of assets and to extract the attribute values that you want to display.
- An assetset may include any type of flexible assets, and non-homogeneous types of flex assets (flex assets from more than one flex asset type). For example, an assetset may consist of several products and product parents.
- An assetset is a set of id(s) of assets and not a set of attribute values.
- A searchstate is a set of search constraints based on the attribute values held in the `_Mungo` table for the flex asset type. You apply searchstates to assetsets.
- A listobject is used when searching for multiple attributes in a set of assets.
- `<assetset:getassetlist>` will apply a listobject to an assetset and will return an `IList` of attribute values for all the attributes in the listobject.
- `<assetset:getassetlist>` will not return a value for an attribute for an asset that does not have at least one value for each attribute in the listobject.
- `<assetset:getattributevalues>` returns value(s) for a single attribute for each asset in an assetset. This tag is useful when used on an assetset that consists of a single flex asset.
- `<assetset:getmultiplevalues>` can return values for multiple attributes on either a single asset or multiple assets.

Activity Answer Key

Below are the answers to the activities in this training guide.

Lesson 3.2

1. [T]
2. [F]
3. [T]

Module 3

Navigating Through a Product Tree

In this module you will learn about attribute inheritance and how the attribute values are stored in the database. Students will also learn how to design attributes for products and product parents to create various content taxonomies.

Module Learning Objectives

After completing this module, you will be able to:

- Design attributes for products and product parents.
- Describe a low level database design, _Mongoose tables, attribute inheritance, and database performance.
- Explain how the attribute values get inherited and stored in the database.

Lesson 3.1: Low Level Database Design

In this lesson you will learn about attributes and attribute inheritance for products and product parents.

Products, Attributes, and Sites

Unlike the basic asset types, flex assets are not divided by site. This means that you can use a single template to display data entered from any CS-Direct Advantage site. For example, if you create a template that searches for all assets with a price attribute equal to \$50, it is possible to retrieve both a chair from the IKEA site and a fund from Burlington Financial in the same assetset (i.e. resultset).

The search in the CS-Direct Advantage API revolves around attribute names and not asset names. CS-Direct Advantage API tags extract attributes from the `_Mungo` table. In order to search for products that belong to a specific site, a developer can:

- Create a special attribute called `sitename`. This attribute can be set to the name of the site and assigned to the top-level product parent and thus inherited by all of the children product parents and products. The value of the attribute should not be modified.
- Name all attributes with a prefix that is unique to each site. For example, all the attributes belonging to the GE Lighting site can start with the prefix `GE` (`GEprice`, `GEwattage`, etc).

Content Managers vs. Visitors

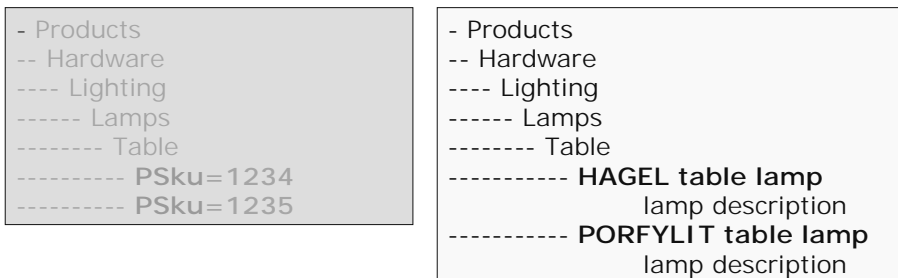
To understand attribute inheritance and design, consider the differences between the needs of the content manager and needs of the visitor.

A **content manager** is a person within the company whose responsibility is to create hierarchy and organize products (content) in such a way that it makes sense for this particular organization.

A **visitor** is someone who views the product (content) on a web site.

A content manager and a visitor might have different expectations. A content manager is more concerned about inheritance and the right structural organization of the product line. A visitor is more concerned about presentation and simplicity.

Thus, a content manager may think that a table lamp product should be a part of the following taxonomy presented on the left of the image below. A visitor may expect a more simplified taxonomy for easier navigation, as shown in the picture on the right:



CS-Direct Advantage is designed to support multiple taxonomies of the same product (content) tree. Because the CS-Direct Advantage API always revolves around attribute values, it is easy to maintain single records for multiple taxonomies. The challenge is to come up with the right set of attributes.

Developing a Product Tree

To navigate through the content manager's presentation of products and product parents, each product parent must have the additional attributes created:

| Attribute Name | Description |
|----------------|--|
| PParent | The name of the product parent's parent. |
| PPName | The name of the product parent. |

All the product parent definitions should have these two attributes required, so that content managers will have to assign value to both of them for each product parent. For example, the top-level product parent Products has no parent (PParent=noparent and the PPName=Products). Though these attributes seem like redundant information, they are necessary because CS-Direct Advantage tags only view the attributes and their values—not the asset field values. Similarly, other product parents in the content manager tree will have the following values for the two attributes:

| Product Parent Name | Attribute Name | Attribute Values |
|---------------------|----------------|------------------|
| Products | PParent | noparent |
| | PName | Products |
| Hardware | PParent | Products |
| | PName | Hardware |
| Lighting | PParent | Hardware |
| | PName | Lighting |
| Lamps | PParent | Lighting |
| | PName | Lamps |
| Table | PParent | Lamps |
| | PName | Table |

The following is a code sample that the developers can use when searching for all the product parents whose parent is Hardware:

```
<searchstate:create name="ss"/>
<searchstate:addsimplestandardconstraint name="ss"
typename="PAttributes" attribute="PParent" value="Lighting"
immediateonly="true"/>
<assetset:setsearchedassets name="ProductParentSet"
constraint="ss" assettypes="ProductsGroups" />
<assetset:getattributevalues name=" ProductParentSet "
typename="PAttributes" attribute="PName" listvarname="nameList"/>

<ics:listget listname="nameList " fieldname="#numRows"
output="rows"/>
<ics:listloop listname ="nameList " maxrows='<%=
ics.GetVar("rows") %>'>
    <ics:listget listname ="nameList " fieldname="value"/>
</ics:listloop>
```

The search will return a product parent named **Lamps**.

In order to create a visitor hierarchy, one should design a different set of attributes that displays product parents and products in the way visitors expect them. A content manager should not create too many attributes, especially if they are inherited on multiple levels of the tree. Refer to the next section for a discussion of a lower-level database design.

Attribute Inheritance and _Mungo Tables

In the preceeding example, two attributes, PParent and PPName, will be inherited on each level of the tree and reassigned a different value for each product parent. This process, while valuable for site design, has a great impact of performance.

Each attribute value is a row in the ProductGroups_Mungo table, and every inherited value of an attribute is an additional row in the ProductGroups_Mungo table. Thus, in the example with the Products, Hardware, Lighting, and Table product parents, the PParent attribute will be inherited by all of the product parents listed here and reassigned a different value for each one of them. This results in the following rows being added to the ProductGroups_Mungo table:

| id | ownerid | attrid | | moneyvalue | | stringvalue | islegal |
|------|----------|---------|--|------------|--|-------------|---------|
| 1234 | Products | PParent | | | | | |
| 1235 | Hardware | PParent | | | | noparent | |
| 1236 | Lighting | PParent | | | | noparent | F |
| 1237 | Table | PParent | | | | noparent | F |
| 1238 | Hardware | PParent | | | | noparent | F |
| 1239 | Lighting | PParent | | | | Products | |
| 1240 | Table | PParent | | | | Products | F |
| 1241 | Lighting | PParent | | | | Products | F |

| id | ownerid | attrid | | moneyvalue | | stringvalue | islegal |
|------|---------|---------|--|------------|--|-------------|---------|
| 1242 | Table | PParent | | | | Hardware | |
| 1243 | Table | PParent | | | | Lighting | F |

Because the `PParent` attribute of the `Products` product parent has been set to `noparent`, four rows have been added to the `ProductGroups_Mungo` table—one for the `Products` product parent, and one for each product parent who is a child of `Products`. When the `Hardware` product parent gets its own value (`Products`) for `PParent` attribute, CS-Direct Advantage overwrites the value `noparent` inherited from `Products` with the new value, by simply adding an extra row to the table. The old value becomes an illegal value. CS-Direct Advantage will not delete this value from the database, but rather keep the row and mark it as illegal in the `islegal` column (F).

This database design allows content managers to keep history for all the values of all the attributes values ever inherited. This gives the product a great flexibility, allowing users to delete and change values of inherited attributes without affecting the hierarchy. However, this flexibility has a great effect on the product performance.

`_Mungo` tables can grow exponentially, keeping all the values as rows and accumulating large amounts of data. When using the CS-Direct Advantage API, the developers should limit their search as much as possible to restrict the number of tables accessed. Also, when configuring the database, it is recommended that `_Mungo` tables have their own table space in the database.

CS-Direct Advantage relies heavily on Content Server resultset caching. Since CS-Direct Advantage is built on top of Content Server, it caches the resultsets generated by database into memory. Implementing resultset caching on a site improves the performance and reduces the load on Content Server and the database. Outdated resultset are removed from the cache in one of the three ways:

- Resultsets are deleted from the cache automatically when a table changes.
- Resultsets are deleted from the cache using `CatalogManager's flushcatalog` command.
- Resultsets time out and are deleted based on values set in the property file, `futuretense.ini`.

For more information about how to configure `futuretense.ini` file to use the Content Server resultset caching, please refer to documentation.

Exercise 3.1.1: Creating a Navigation Bar

The objective of this lab exercise is to learn how to display product navigation bar. CS-Direct Advantage allows you to be flexible and not to display the tree on your finished Web site exactly the way it was built by content managers. However, in this exercise, we will display product parents exactly as they are created in the product tree, with only product parents of two levels.

Purpose

In this exercise, you will search for all the product parents of the Level1 definition in the product tree of Home Office furniture store. Later, you will find the children of each parent in the set by creating a new searchstate.

Directions

Using the step-by-step procedure(s) contained in this lesson, complete the following action(s) in CS-Direct Advantage:

Adding PParent and PName Attributes

1. Add the following attributes:

- PParent

Make them both of type String and single-valued. These attributes will be used in the Level1 and Level2 product parent definitions.

2. Change the Level1 and Level2 product parent definitions to include PName and PParent and PName attributes. Make both of these attributes required. Place the PName at the end of the list of attributes (known bug).
3. Edit all of your product parents to assign the values to the PParent and PName attributes. The following tables should help you to assign the correct values to the attributes:

| Product Parent | Attribute Values |
|----------------|-------------------------------------|
| Chairs | PName=Chairs PParent=noparent |
| Sofas | PName=Sofas PParent=noparent |
| Tables | PName=Tables PParent=noparent |
| Armchairs | PName=Armchairs PParent=Chairs |
| Footstools | PName=Footstools PParent=Chairs |
| Fabric Sofas | PName=Fabric Sofas PParent=Sofas |

| Product Parent | Attribute Values |
|----------------|---------------------------------------|
| Leather Sofas | PName=Leather Sofas PParent=Sofas |
| Sofa Beds | PName=Sofa Beds PParent=Sofas |
| Coffee Tables | PName=Coffee Tables PParent=Tables |
| Dining Tables | PName=Dining Tables PParent=Tables |

Adding Code to Render Product Navigation Tree

1. Open the `ElementCatalog/IkeaFurniture/JSP/navbar` element.
2. Using the following steps, display products parents of `Level1` definition:
 - a. Create a searchstate called `ss1`. Add a constraint to the `ss1` searchstate to search for flex assets where the attribute `PParent="noparent"`.

Note

Make sure that the `immediateonly` argument is set to `true` within the `<searchstate:addsimplesearchconstraint>` tag. This ensures that you will only search for those product parents where the `PParent` attribute value is not inherited.

- b. Create an assetset called `parentset1` by applying the `ss1` constraint in your search within the `<assetset:setsearchedassets>` tag. Search only for product parents (`assettypes="ProductGroups"`).
- c. Use the `<assetset:getattributevalues>` tag to filter the `parentset1` assetset to only those product parents that have the attribute named `PName`.

Note

Since this is a required, single-valued attribute for all the product parents, the `<assetset:getattributevalues>` tag will return the list of values where the number of values are the same as the number of product parents in the assetset. For example, if there are only 5 product parents in your assetset (where `PParent="noparent"`), then the list returned will also have 5 values for the `PName` attribute (one for each product parent).

- d. Use the `<ics:listloop>` and the `<ics:listget>` tags to display the name of each product parent in `Level1`.
3. Add more code to display the children of each parent of the `Level1` definition.

4. Test your exercise in the browser by calling the `IkeaFurniture/JSP/NavBar` page by typing the following URL in the browser:

```
http://localhost:7001/servlet/  
ContentServer?pagename=IkeaFurniture/JSP/NavBar
```

Lesson 3.2: Designing a Flex Family

In this lesson you will learn about various aspects of designing a catalog product tree and flex assets.

Flex Asset Family Members

Most management systems utilize multiple flex asset families. How you design these families and the assets that compose them affects both your database and the usability of the management system.

As you design flex asset families you must create a balance:

- Limit the number flex parent definitions associated with a given flex definition, so that content providers and editors are not inundated with too many choices.
- Create enough flex parent definitions and flex definitions so that content providers and editors can find a definition that is appropriate for the data they want to enter.

Designing the flex asset families for your project is a process consisting of the following steps:

1. Determine all of the attributes you will need.
2. Determine which items have attributes with unique values.
3. Determine the number of flex definitions you need.
4. Determine the number of flex families you need

Determine the Attributes You Need

The first step in designing a flex asset family is to list all of the attributes that you need for your site.

Note that this means more than determining the attributes you need for your business requirements or the attributes that you want to display to web site visitors.

You must also determine how you want content to be displayed on the finished web site and how you want web site visitors to “drill down” to items that they want.

For example, if you want to display a list of the ten most recent stories submitted to a newspaper, you must include an attribute that holds the date and time that the story was submitted, allowing your developers to create logic which retrieves the most recent submissions. Similarly, if you would like web site visitors to search on articles based on the section they fall under—Sports, for example—you must include a section attribute.

Determine Which Attribute Values are Unique

Children in the asset inheritance tree inherit attributes and their values from their parents and grandparents. Attributes where the values must be unique for each instance of an item-SKU, for example—are included in the flex definition, near the bottom of the asset inheritance tree. Conversely, attributes with common values are candidates for being item parents and item parent definitions, so that those values can be inherited by the individual items that need them.

Determine the Number of Flex Definitions You Need

The number of flex definitions you create affects the number of data fields that appear in the asset forms on the management system. The number of definitions you must create is determined by how different the individual flex assets at the bottom of the asset inheritance tree are.

Note that the number of attributes that compose a Flex Definition impacts the amount of time it takes for that Flex Definition's form to load. It takes between 50 and 350 milliseconds to display one attribute field (depending upon your attribute editor), so that displaying many attribute fields leads to slow forms.

In an online catalog, for example, you could create one flex definition called item which would act as the template for all items that the editors enter into the system.

If, however, the catalog contains very different items, like sheets and toasters, a universal product definition is not the best choice; sheets require fields that toasters do not, forcing editors to leave many fields empty. A better solution is to create two flex definitions, one for toasters and one for sheets, where each flex definition contains only the fields necessary for that type of item.

Determine the Number of Flex Families You Need

The database schema for Content Server Direct Advantage includes `_Mungo` tables. A `_Mungo` table contains attribute values for an associated Flex Family. `_Mungo` tables can grow very large, often containing more than a million records.

Creating multiple Flex Families is a good way to separate your data into several `_Mungo` tables in your database, thus differentiating your data and allowing you to control security and archiving on a Flex Family-by-Flex Family basis.

Note, however, that searching for content across multiple `_Mungo` tables is slower than searching for content in a single `_Mungo` table.

Flex Asset Design Tips

Use a multi-value field if an item is in more than one category—for instance, if a news story can be classified as both “business” and “international,” or if a movie can be classified as both “romance” and “musical.”

Instructor Demonstration

Follow along as the instructor demonstrates how to create a product parent definition and a new product parent.

Creating a Flex Family

Complete the following steps to create a new flex family:

1. In the Content Server **Admin** tab, expand **Flex Family Maker** and select **Add New** to add a new flex family of assets.
2. In the **Flex Attribute** field, select either **Product Attribute** or **Content Attribute** if you want to use one of the existing flex attribute types. If you would like to create a new flex attribute type, enter the name for your attribute in the **Flex Attribute** field. For example, you can create an `eAttribute` flex asset type.
3. In the **Flex Parent Definition** field, select either **Product Parent Definition** or **Content Parent Definition** if you want to use one of the existing flex parent definition asset types. If you would like to create a new flex parent definition, enter the name for your parent definition in the **Flex Parent Definition** field. For example, you can create an `eParentDefinition` flex parent definition asset type.
4. In the **Flex Definition** field, select either **Product Definition** or **Content Definition** if you want to use one of the already existing flex definition asset types. If you would like to create a new flex definition, enter the name for your flex definition in the **Flex Definition** field. For example, you can create an `eContentDefinition` flex definition asset type.
5. In the **Flex Parent** field, select either **Product Parent** or **Content Parent** if you want to use one of the already existing flex parent asset types. If you would like to create a new flex parent, enter the name for your parent in the **Flex Parent** field. For example, you can create an `eParent` flex asset type.
6. In the **Flex Asset** field, select one of the following asset types: **Products**, **Article (Flex)**, **Image (Flex)**, **PDF**, or **Drill Hierarchy**. If you do not want to use the existing asset type and would like to create your own flex asset type instead, enter the name for your asset type in the **Flex Asset** field. For example, you can create an `eDocument` flex asset type.
7. In the **Attributes** field, select the attribute(s) that will define your product parents. Choose whether you want each attribute to be required or optional and whether you will allow this attribute to have multiple values.
8. On the next screen, click the **Continue** button.
9. For the `eAttribute`, in the **Plural** field, type **eAttributes**. This will be the plural form of the flex asset type named `eAttribute`.
10. In the **Desc** field, type **eAttribute**. This is how the flex asset type name will be displayed on the screen.
11. Repeat step 8 for all the flex assets listed on the screen.
12. Click the **Go** button. Your new flex family of assets should now be created.

13. Right-click on a Flex Family Maker node on the **Admin** tree and select **Refresh**. You should find your new asset types under each of the following categories:

- Flex Attribute Types
- Flex Parent Definition Types
- Flex Definition Types
- Flex Parent Types
- Flex Asset Types

CS-Direct Advantage does not have a utility to delete a family of flex assets.

Module Summary

- Attribute values are recorded in the `ProductGroups_Mungo` table for a product parent and all of its children. Values that products inherit from their product parents are also recorded in this table.
- Inherited attribute values attributes are not removed from the database if they are overwritten by new values. Instead, they are marked void (F).
- When designing products and product parent definitions, consider limiting the number of attributes to a minimum, especially if those attributes are inherited by multiple products and product parents.
- When designing flex assets consider the following:
 - Determine the attributes you need
 - Determine which attributes values are unique
 - Determine the number of flex definitions you need
 - Determine the number of flex families you need

