DATA DICTIONARY BUILDING BLOCK 9.5
TECHNICAL GUIDE
Contents

Chapter 1: Data Dictionary building block basics
Features of the Data Dictionary building block ......................................................... 1
Building block structure ......................................................................................... 1

Chapter 2: How the Data Dictionary building block works
Data dictionary metadata ....................................................................................... 4
Difference between data dictionary and typical Adobe application modeling technology .............................................................................................................. 4
Data dictionary elements ......................................................................................... 6
Data Dictionary building block architecture ......................................................... 9
Developing a Data Dictionary Solution .................................................................. 11

Chapter 3: Understanding the Data Dictionary building block assets
Tools ....................................................................................................................... 17
Data Dictionary Services ....................................................................................... 21
Default endpoints .................................................................................................. 21
Dependence of the Data Dictionary building block on Expression Evaluator Service (EXM) ........................................................................................................... 21
Security authentication and authorization ............................................................. 22
Data Dictionary run time workflow ...................................................................... 22
Samples .................................................................................................................. 24

Chapter 4: Implementing the Data Dictionary user stories
Creating a data dictionary ...................................................................................... 33
Chapter 1: Data Dictionary building block basics

The Data Dictionary Building Block 9.5 provides a framework for representing the underlying data model in business vocabulary, such as name, age, eligibility, and so on. It allows non-technical users to easily create correspondence, content assets, and processes using the vocabulary that they understand, without having to worry about how the actual data will be sourced. This building block is based on the Adobe application modeling technology, which allows for an independent representation of metadata and back-end integration. Independent representation allows business users, with no knowledge of back-end system integration to create and use models at a solution level.

Features of the Data Dictionary building block

Following are the key features provided by the Data Dictionary building block:

- Provides spring-based services for data dictionary creation and management.
- Provides flex based user interface components that can be used for data dictionary creation and management.
- Provides a Flex® client that facilitates solution level flex components to call Data Dictionary services.
- Provides a Java™ client that enables solution developer to call Data Dictionary services.

Building block structure

The Data Dictionary building block includes all the components, assets, tools, and APIs for building a solution requiring a data dictionary based on Adobe® LiveCycle® Enterprise Suite 2.5 (ES2.5) software. The directory structure is designed to make it easy to find files and assets when you need them.

The Data Dictionary building block components are installed in the following folder: c:\Adobe\Adobe LiveCycle ES2\sa_resources\SA_SDK_9.5\BuildingBlocks\datadictionary referenced in this document as [dct_root]

<table>
<thead>
<tr>
<th>Directory</th>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[dct_root]/client/flex</td>
<td>adobe-dct-flex-services.swc</td>
<td>Contains ActionScript® Value Objects, Data Dictionary building block service flex APIs, events, and fault handling. The Data Dictionary building block service flex APIs expose the Data Dictionary server-side services via remoting.</td>
</tr>
<tr>
<td></td>
<td>adobe-dct-widgets.swc</td>
<td>Contains various user interface widget SWCs that can be assembled within a reference application to create a custom unified end user interface. The Data Dictionary Browser and Data Dictionary Editor are components of this SWC.</td>
</tr>
<tr>
<td></td>
<td>adobe-dct-widgets_rb.swc</td>
<td>Contains the resource bundles corresponding to supported locales for dct-widgets.</td>
</tr>
<tr>
<td></td>
<td>adobe-dct-widgets_styles.swc</td>
<td>Contains the style resources such as icons, images, and the CSS style sheet corresponding to dct-widgets.</td>
</tr>
<tr>
<td>Directory</td>
<td>Filename</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>src/adobe-dct-widgets_rb-src.zip</td>
<td>Contains the source of adobe-dct-widgets_rb.swc.</td>
<td></td>
</tr>
<tr>
<td>src/adobe-dct-widgets_styles-src.zip</td>
<td>Contains the source of adobe-dct-widgets_styles.swc.</td>
<td></td>
</tr>
<tr>
<td>{dct_root}/client/java</td>
<td>adobe-dct-client.jar</td>
<td>Contains server-side value objects, service APIs, and exception classes which are required for compilation while writing any custom code for the Data Dictionary building block. It also enables invocation of the DCT services from Java clients (via the configured remote mechanism, such as Web-Services or RMI).</td>
</tr>
<tr>
<td>{dct_root}/service</td>
<td>adobe-dct-services.jar</td>
<td>Contains the service classes (service beans) and entity objects for the Data Dictionary building block.</td>
</tr>
</tbody>
</table>
Chapter 2: How the Data Dictionary building block works

The Data Dictionary building block provides a way for the application to hide the data source associated with the elements used in it. Client applications are able to gain a level of abstraction between real world objects and the actual data source of the elements. For example, in the Correspondence Management Solution Accelerator 9.5, there is a requirement for users building templates (for letters or packages) to easily include any data into their solutions. The Data Dictionary building block provides a means to insert data placeholders having no particular ties to any specific data source. For example, a document can be created with friendly names, such as "Dear {First Name}", "Mr. {LastName}". Any change of back-end data source does not force the user to change the bindings of placeholders. For example, the following diagram details a sample data dictionary representation for a company:

Sample data dictionary representation
Data dictionary metadata

The Data Dictionary building block is based on the Adobe application modeling technology, which allows for an independent representation of metadata and back-end integration. Metadata, in the solution context, represents descriptions of the data structures and their associated attributes. Therefore, a data dictionary is a catalog of domain-specific objects and their associated definitions. The Data Dictionary building block allows business users to use models at a solution level. They can also create a domain-specific data dictionaries or enhance the artifacts without requiring back-end data integration. For example, an insurance domain data dictionary can contain data dictionary elements (DDEs) such as person, age, eligibility, loan amount, and so on. Data dictionary models can be developed and seamlessly exported into an extended application model, for import into production systems.

The following table details the metadata attributes associated with a data dictionary:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
<td>Unique ID of the data dictionary. Once created the ID cannot be changed.</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td>Unique user provided name of the data dictionary.</td>
</tr>
<tr>
<td>description</td>
<td>String</td>
<td>The description of the data dictionary.</td>
</tr>
<tr>
<td>dictionaryType</td>
<td>Boolean</td>
<td>Indicates whether the dictionary is a system data dictionary.</td>
</tr>
<tr>
<td>extendedProperties</td>
<td>Object</td>
<td>A map of custom properties added at the data dictionary level, including user interface specific properties or other key-value pair of properties.</td>
</tr>
</tbody>
</table>

The data dictionary created or updated using the Data Dictionary Editor is stored in the Data Dictionary system. The dictionary is available for create, update, and delete operations through the editor and relevant client-server side APIs. The solution accesses the data dictionary definition and metadata to carry out its operations. For example, a Data Dictionary based solution accesses the data dictionary to get the list of DDEs that it requires. With each DDE, extended properties are stored and used for rendering, validation, or any business logic that the solution requires.

Difference between data dictionary and typical Adobe application modeling technology

The Data Dictionary building block leverages the Adobe application modeling technology for capturing the domain model. Data dictionaries created using the Data Dictionary tools are extended and annotated models with custom annotations. They capture the structure, metadata, and back-end data bindings. A data dictionary can stand on its own without Adobe® LiveCycle® Data Services 3.1 destination support during development. In this scenario, the data dictionary still supports end-to-end solution-level use cases without having real data source integration in place. The solution developer creates artifacts in terms of the data dictionary only. When moving to production a Data Services destination can then be added.

Supported XSD constructs for data dictionary creation

The following table details the constructs and data types supported by the Data Dictionary building block schema definition:
The XSD import to create a data dictionary does not support all XSD elements, the following table details some of the limitations:

<table>
<thead>
<tr>
<th>XSD Constructs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>complexType</td>
<td>Inline and “ref” complex types. Sequence within complex types. Type names are honored to fill the compositeType attribute of a DDE. These data types are used during the FML export and subsequent import. They are internal and are not shown in user interface.</td>
</tr>
<tr>
<td>Simple Types</td>
<td>String, Number, Date &amp; Boolean primitives</td>
</tr>
<tr>
<td>xs:ID</td>
<td>Any attribute with type xs:ID is marked as key element</td>
</tr>
<tr>
<td>maxOccurs</td>
<td>Honored to mark the collection type data dictionary element</td>
</tr>
<tr>
<td>minOccurs</td>
<td>minOccurs == 0 marks corresponding data dictionary element as optional/non-required</td>
</tr>
<tr>
<td>Enumerations</td>
<td>The Data Dictionary building block only supports enumerations which define a list of acceptable values.</td>
</tr>
<tr>
<td>References</td>
<td>Circular references/Recursive references in XSD Schema are handled.</td>
</tr>
<tr>
<td>xs:Restrictions</td>
<td>Restrictions such as length, maxExclusive, maxInclusive, maxLength, minExclusive, minInclusive, minLength, pattern, totalDigits, and so on, are not supported.</td>
</tr>
<tr>
<td>xs:include and xs:import</td>
<td>An XML Schema spanning over multiple files is not supported</td>
</tr>
<tr>
<td>xs:any</td>
<td>The &lt;any&gt; Element is a wildcard that allows the insertion of any element belonging to a list of namespaces.</td>
</tr>
<tr>
<td>xs:Substitution</td>
<td>Substituting one element with another element is not supported.</td>
</tr>
</tbody>
</table>

**Limitations of data model export for data dictionary creation**

A data dictionary can be expressed into and created from an annotated and extended data model. The annotations in question are data dictionary specific and capture all data dictionary-specific information in the model. The following table details model level annotations:

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDS</td>
<td>name</td>
<td>Required. The name of the data dictionary.</td>
</tr>
<tr>
<td>DDS</td>
<td>displayName</td>
<td>Optional. The user-friendly name of the data dictionary.</td>
</tr>
<tr>
<td>DDS</td>
<td>version</td>
<td>Optional. The version of the data dictionary.</td>
</tr>
<tr>
<td>DDS</td>
<td>type</td>
<td>Optional. The type of data dictionary can be SYSTEM or NONSYSTEM. Refer to the &quot;DataDictionaryType&quot; enum class in Java Docs for more information.</td>
</tr>
<tr>
<td>General</td>
<td>description</td>
<td>Optional. Description of the data dictionary.</td>
</tr>
</tbody>
</table>

The following table details entity level annotations:
How the Data Dictionary building block works

Data dictionary elements

A data dictionary is composed of data dictionary elements (DDEs) of which there are three types: Simple, Composite, and Collection elements. Simple DDEs are primitive elements such as strings, numbers, dates, and Boolean values that hold information such as a city name. A Composite DDE contains other DDEs, which can be of type primitive, composite or collection. For example, an address, which consists of a street address, city, province, country, and postal code. A Collection is a list of similar Simple or Composite DDEs. For example, a customer with multiple locations, or different billing and shipping addresses.

Data Dictionary Elements must adhere to a few rules with respect to the schema structure.

- Only composite type is allowed as top-level DDE in a data dictionary.
- An annotated FML with multiple top-level entities, cannot be imported.
- Name, reference name and element type are mandatory fields for a data dictionaries and DDEs.
- The reference name must be unique.
  - A parent DDE (composite) can’t have two children with the same name.
- Enums only contain primitive String types.

Computed data dictionary elements

A data dictionary can also include computed elements. A computed data dictionary element must be associated with an expression. This expression is evaluated to get the value of a data dictionary element at run time. A computed DDE value is a function of other DDE values or literals. By default JSP Expression Language (EL) expressions are supported. The EL expressions use the ${ } characters and valid expressions can include literals, operators, variables (data dictionary element references/DDE path), and function calls. While referencing a data dictionary element in the expression, the path of the DDE is used. The path is the absolute path of a DDE in dotted notation that corresponds to the location of DDE in the data dictionary definition hierarchy. The path is unique for every data dictionary element within a data dictionary.

A computed DDE Person.fullName can be associated with an EL concatenation expression such as

\${Person.firstName} \${Person.lastName}
Data dictionary element types

The following diagram shows the relationships among the above DDE types:

The following table details the common attributes associated with a DDE:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Required. Name of the DDE. It must be unique.</td>
</tr>
<tr>
<td>reference Name</td>
<td>String</td>
<td>Required. Unique Reference name for the DDE allowing for references to the DDE that are independent of changes to the hierarchy or structure of the data dictionary. Text modules are mapped using this name.</td>
</tr>
<tr>
<td>displayname</td>
<td>String</td>
<td>An optional user friendly name of the DDE.</td>
</tr>
<tr>
<td>description</td>
<td>String</td>
<td>Description of the DDE.</td>
</tr>
<tr>
<td>elementType</td>
<td>String</td>
<td>Required. The type of DDE: STRING, NUMBER, DATE, BOOLEAN, COMPOSITE, COLLECTION.</td>
</tr>
<tr>
<td>elementSubType</td>
<td>String</td>
<td>The subtype for DDE: ENUM. Only allowed for STRING and NUMBER elementType.</td>
</tr>
<tr>
<td>key</td>
<td>Boolean</td>
<td>A Boolean field to indicate if a DDE is key element. A key element’s value must be specified such that a data dictionary instance (DDI) can be fetched from the back-end system. Keys are not required if the data dictionary is created from fully filled input data and not by querying any back-end source.</td>
</tr>
<tr>
<td>computed</td>
<td>Boolean</td>
<td>A Boolean field to indicate if a DDE is computed. A computed DDE value is a function of other DDE values. By default, jsp el expressions are supported.</td>
</tr>
</tbody>
</table>
How the Data Dictionary building block works

**XSD data dictionary data type mapping**

Exporting an XSD to the model's annotated FML requires specific data mapping, which is detailed in the following table. The DDI column indicates the type of the DDE value as available in the DDI.

<table>
<thead>
<tr>
<th>XSD</th>
<th>Data Dictionary</th>
<th>model</th>
<th>DDI (Instance Value Data Type)</th>
</tr>
</thead>
<tbody>
<tr>
<td>xs:element of type - Composite Type</td>
<td>DDE of type - COMPOSITE</td>
<td>Entity</td>
<td>java.util.Map</td>
</tr>
<tr>
<td>xs:element where maxOccurs &gt; 1</td>
<td>DDE of type - COLLECTION-A DDE node is created next to the COLLECTION DDE which captures information from the parent COLLECTION node. The same gets created for both collection of simple/composite data types. Whenever you have a COLLECTION of the type composite, the Data Dictionary tree captures the constituent fields in the children of the DDE created for capturing type information. - DDE (COLLECTION) - DDE(COMPOSITE for type info) - DDE(STRING) field1 - DDE(STRING) field2</td>
<td>Collection property in FML Simple - Mapped as a collection Composite - Mapped as a relationship of cardinality one-to-many with the corresponding entity.</td>
<td>java.util.List</td>
</tr>
<tr>
<td>Attribute of type - xs:id</td>
<td>DDE of type - STRING</td>
<td>id (property)</td>
<td>java.lang.String</td>
</tr>
</tbody>
</table>
## XML Binding

In previous releases all XML bindings had relatives paths, now XML binding is based on the type of DDE. Collection nodes do not have root nodes for XML binding and all other XML bindings must have the same root node. The following table details the binding conventions:

<table>
<thead>
<tr>
<th>DDE Type</th>
<th>Binding</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Composte</td>
<td>No XML Binding</td>
<td></td>
</tr>
<tr>
<td>Computed</td>
<td>No XML Binding</td>
<td></td>
</tr>
<tr>
<td>Collection</td>
<td>Absolute path</td>
<td>/Person/name</td>
</tr>
<tr>
<td>Primitive</td>
<td>Absolute path</td>
<td>/Person/name</td>
</tr>
<tr>
<td>Primitive, in a</td>
<td>Relative path (does not</td>
<td>Person/name</td>
</tr>
<tr>
<td>collection</td>
<td>start with '/')</td>
<td></td>
</tr>
<tr>
<td>Attribute</td>
<td>Absolute path (requires</td>
<td>/purchaseOrder/shipTo/@country</td>
</tr>
<tr>
<td></td>
<td>'@')</td>
<td></td>
</tr>
</tbody>
</table>

### Data Dictionary building block architecture

The Data Dictionary building block consists of clients and services that connect to the back-end system using the Asset Manager Building Block 9.5.
How the Data Dictionary building block works

Data Dictionary building block architecture

**Data Dictionary building block clients**

The Data Dictionary clients include:

- Data Dictionary Java clients (adobe-dct-client.jar). The Java clients include server-side value objects, service APIs, and exception classes required for any custom code for the building block.

- Data Dictionary Flex client (adobe-dct-flex-services.swc): The Flex client includes various ActionScript Value Objects, Data Dictionary service flex APIs, events, and fault handling.

- Data Dictionary building block user interface components. The Data Dictionary building block user interface components (Data Dictionary Editor, Data Dictionary Browser) use the Data Dictionary Services for interacting with the back-end repository.
  
  - Data Dictionary Editor: The Administrator user interface for creating and modifying a data dictionary, its properties, and elements. In the Definition view and Properties view, you can associate the value of various properties with the data dictionary elements. You can associate metadata with a data dictionary and each of its elements by adding extended properties.
  
  - Data Dictionary Browser. Allows you to browse a data dictionary and its elements. Use the Data Dictionary Browser to examine data dictionary elements when you create content, and define rules and expressions around the content. Using the Data Dictionary Browser you can select primitive elements (leaf nodes in the tree), but not composite or collection elements (branch nodes).
Data Dictionary building block services
The Data Dictionary building block services are exposed as spring beans. Following is the list of beans that are provided in Data Dictionary building block:

- DataDictionaryRegistryService: This service provides Create, Edit, and Delete operations for the Data Dictionary building block.
- DataDictionaryUtilService: This service provides data dictionary utility functions. These functions allow the following actions:
  - Import an XSD to create a data dictionary.
  - Import or export an existing data dictionary as an annotated FML.
  - Create sample XML data for the data dictionary.
- Expression Evaluation Service: This service is as a reusable component that uses standard EL syntax and libraries for expression evaluation. The component has server-side APIs for expression evaluation on the server and a client-side counterpart for evaluation of expressions on the client. The createDDInstanceFromXml() API on the server calls this component for resolution of expressions within computed data dictionary elements.

Back-end integration layer
The server-side component of the Asset Manager building block exposes operations/APIs to interact with the underlying repository that stores the assets. The services component for the Asset Composer Building Block 9.5 encapsulates APIs for the individual assets that interact with the Asset Manager building block server-side APIs. The server-side component of the Asset Manager building block facilitates operations and other utility APIs, such as list of dependencies on an asset and downloading the content bytes for an asset.

Developing a Data Dictionary Solution
The Data Dictionary building block allows middleware developers to create and modify data dictionaries programmatically or by using a simple user interface.

Creating a data dictionary
You use the Data Dictionary Editor to create a data dictionary. You can then extend the data dictionary by adding additional required information, including fields. The Data Dictionary Editor supports two methods for creating data dictionaries. You can create data dictionary elements manually, or you can import a data dictionary from an existing FML model or XSD schema. Regardless of how the data dictionary was created, the business process owner does not need knowledge of the back-end systems. The business process owner only needs knowledge of the domain objects, and their definitions, for their process. For more information on the Data Dictionary Editor, see “Data Dictionary Editor” on page 19.

System data dictionary
You can configure the Asset Manager building block user interface related preferences using Asset Type Definition. Asset Type Definition is a System Data Dictionary representing the asset type. For example, the Correspondence Management Solution Accelerator delivers preconfigured data dictionaries for specific asset types in form of FML files. These files can be repackaged and redeployed. A middleware developer must have the following permission assigned to create a System Data Dictionary:
permission name="CM System Data Dictionary Create"
Other permissions that may be required include: CM System Data Dictionary Delete, CM System Data Dictionary Edit, and CM System Data Dictionary View. You must also add the user to the System Data Dictionary Administrator group as follows:

1. In the LiveCycle Administration Console, click Settings > User Management > Users and Groups.
2. Edit the group System Data Dictionary Administrator.
3. Click the Child Principals tab and add the user to the group.

A user with appropriate permissions and privileges can create a System Data Dictionary by marking the data dictionary as System in the Properties view/tab.

Creating a data dictionary manually

When creating the data dictionary manually you must enter each element one by one into the Data Dictionary Editor. You then manually enter any Extended Properties that correspond to the data dictionary itself, which are stored as metadata attributes.
How the Data Dictionary building block works

Data dictionary extended properties

Then add the data dictionary elements with their associated properties. For information on data dictionary element attributes see “Data dictionary elements” on page 6.

Data dictionary elements
Creating a data dictionary from XSD

When creating the data dictionary from an XSD schema you import the schema from the Data Dictionary Editor. Click Import Data Dictionary (XSD) on the Definition pane of the Data Dictionary Editor.

There are a few important items that you must consider when creating a data dictionary:

- An XSD can only be created from one top-level node. If a schema is imported with multiple top-level nodes, the Data Dictionary Editor prompts you to select a top-level node.
- Data dictionary element definitions or extended properties for a data dictionary do not support empty keys or empty strings. Empty values are ignored. Do not use the colon(;) or vertical bar(|) characters within the key or value of an extended property. If you do, the key can be lost or split into multiple keys.
- The creation of sample XML data from a data dictionary containing attributes is not supported. Sample-XML data generation does not produce the correct XML. The error is due to the fact that both attributes and elements are treated as elements in the sample-XML-data generation.

You can modify the properties for your data dictionary, as well as any properties for your elements. Extended properties can be defined at this time. For an example of a properly formed schema, see “Sample Schema that can be imported into the Data Dictionary” on page 25.

Creating a data dictionary from an existing model

When creating the data dictionary from an existing model (FML), import the model using the Data Dictionary Editor. Click Import Data Dictionary (FML) on the Definition pane of the Data Dictionary Editor. A data dictionary created using the Data Dictionary Editor has been extended and annotated using custom annotations. Only FML files exported using the Data Dictionary Editor can be imported. For more information see “Difference between data dictionary and typical Adobe application modeling technology” on page 4.
The data dictionary created this way has both the Properties and Definition for the data dictionary.

Migrating the data dictionary

Once you have created a data dictionary, you can migrate it to another environment. For example, a data dictionary created in a staging environment can be migrated into a production environment. The data dictionary is exported into an annotated model (FML), and then imported on the new system. You migrate the data dictionary using the Data Dictionary Editor. Open the data dictionary in the Data Dictionary Editor and click the Export FML icon to save the annotated FML file.

You can copy the FML file to another system. On the new system launch the Data Dictionary Editor and use Import FML to load the Data Dictionary onto the new system.

Note: A data dictionary created using the Data Dictionary Editor has been extended and annotated using custom annotations. Therefore only FML files exported using the Data Dictionary Editor can be imported. For more information see “Difference between data dictionary and typical Adobe application modeling technology” on page 4.

Migrating to Data Dictionary 1.5

If you are applying a patch to the Data Dictionary building block 9.5.0, due to changes in the XML schema you must run the migration utility to migrate your data dictionary from version 1.0 to 1.5. The utility can be invoked by navigating to http://<hostname>:<port>/cmsa/Migrate in an authenticated session.

The Migration utility performs the following tasks:

1. The Data Dictionary version field value is updated from blank/null/1.0 to 1.5. This field tracks whether the Data Dictionary has been migrated. New Data Dictionaries created in this release have the version set to 1.5. In previous versions the version field is populated in the exported FML and is not present in the other Data Dictionaries.
2. All DDE display name fields are assigned a non null unique value. This value is calculated from the path by removing all "." characters from the path. If the value is still not unique then an incrementing number is appended.
3. The XML binding of all of the DDEs are updated to a newer value. In previous releases all XML bindings had relatives paths, now XML binding is based on the type of DDE, as follows:
   - The binding for primitive DDE is an absolute XPath.
   - Composite DDEs do not have any binding.
   - The binding for collection DDE is an absolute XPath. The binding for all the child DDEs of collection DDE is relative to the parent collection DDE.
Generating sample XML data for the data dictionary

You can generate sample XML Data to use as test data. In the Data Dictionary Editor, use the Get Sample XML Data option on the Definition pane to export the sample XML data for the data dictionary.

Solution development

A typical solution contains user interface components, back-end services such as spring beans, forms, domain-specific artifacts such as text and conditional modules, configurations, and relevant packaging. You can also create java classes for capturing the domain-specific run time data. For example, a Person class that encapsulates all the information of a newly created Person such as a name, social security number, and other personal information. It is also possible that data capture for core data is external to the solution and the solution just reads it. The solution does not need to capture the person information in any of its workflows since artifacts contain references to DDEs within them. For example, a text module could contain a reference to a name DDE. At run time, the values from the corresponding DDIs would fill them in, subject to the context.

A customer targeted solution requires custom development and custom back-end integration with the legacy systems, such as HCMs/CRMs. The integration can be at the initiation point of the workflows, where the driving data is passed in form of input XML data. Here the driving data is already fetched from the legacy system before being passed into the solution. The Data Dictionary building block provides APIs to create DDI out of the passed in XML. The DDE bindings in the application can then be resolved on this DDI.

The DDI created at run time is the available to flex clients through the Data Dictionary Flex client API. The solution artifacts, along with the data dictionary created on the development system, are exported and imported into the production setup. The data dictionary is exported as an annotated FML file, through the Data Dictionary Editor. Any data dictionary created is included in the Solution WAR/EAR/jar and taken to the production setup in its native FML format.
Chapter 3: Understanding the Data Dictionary building block assets

The data dictionary is a catalog of data elements, and their descriptions, used in a business domain using a business vocabulary. The data dictionary also contains metadata for managing the data model to provide descriptions of the data structures and their associated attributes. Typically, business users do not require knowledge of metadata representations like XSD (xml schema), FML (Adobe application modeling language), Java classes, and ActionScript classes, used by developers. However, they usually require access to these data structures and attributes to build solutions. The Data Dictionary building block provides an independent representation of metadata and back-end integration. Business users, with no knowledge of back-end system integration, can use this building block to create and use models at a solution level. Within LiveCycle, business users are able to express conditions or use data in the solution’s workflows using the data dictionary.

Tools

Developing a Solution Accelerator that uses a data dictionary typically involves the use of the following tools:

1. Data Dictionary Browser. The user interface that enables browsing a data dictionary and its elements.
2. Data Dictionary Editor. The user interface that allows creation and maintenance of the data dictionary that is shipped with this building block. It is accessed through the Manage Assets page in your portal:
   - Enables import of domain-specific XSD schema to create a data dictionary.
   - Allows addition/deletion of data dictionary elements (DDEs) over/from a data dictionary
   - Allows association of attributes at data dictionary level.
   - Allows association of attributes at DDE level.
   - Allows exporting of a data dictionary into annotated FML.
   - Allows importing of an annotated FML to create a data dictionary.
   - Allows creation of sample data xml for a data dictionary. The sample xml can be used to enable the data dictionary instance-dependent use cases, quickly, without having back-end support in place.
3. Java Development Tools:
   - IDE for creating Java/Spring projects
   - ANT build tools
4. LiveCycle Tools
   - Adobe® LiveCycle® Designer 9
   - Flash® Builder™

For additional information about the Data Dictionary building block services, see these resources:

- For information about the Flex APIs, see the ActionScript 3.0 Reference for the Adobe Flash Platform.
- For information about the Java APIs, see the Solution Accelerators API Reference.
Data Dictionary Browser

The Data Dictionary Browser presents elements in a tree structure. The browser allows selection of DDEs so they can be referenced.
Data Dictionary Editor

The Data Dictionary Editor is used for creating and managing data dictionaries. A middleware developer can create, modify, and delete metadata properties for both the data dictionary and its elements using the Data Dictionary Editor. The Data Dictionary Editor enforces validations while creating or updating a data dictionary. For more information on the rules for creating DDEs for a valid schema, see Data dictionary elements.

The Data Dictionary Editor allows business process owners to create data dictionaries by importing them from a valid XML schema or FML model. Business process owners can also export their data dictionaries to FML or generate a sample xml data from the Data Dictionary Editor.

If the middleware developer attempts to import a schema with multiple top-level elements, the Data Dictionary Editor prompts for a single top-level element. The developer selects one of the top elements for use in creating a data dictionary tree. The data dictionary represents a cohesive instance of data for a given context, rather than a set of disjointed data. An XSD with a forest of elements represents multiple trees of elements - with each tree being an unrelated data set. A data dictionary is created over one such tree. It does not allow import of annotated FMLs with multiple top-level entities. Both the name, and element type for a data dictionary/DDE are required parameters and must be specified. A parent DDE (composite) can’t have two children with the same name.

Data Dictionary Editor validations

The Data Dictionary Editor enforces following validations when creating or updating a data dictionary.

- Only composite type is allowed as Top-level Element in a data dictionary.
- Composite and Collection elements are not allowed at leaf level. Only Primitive (String, Date, Number, Boolean) elements are allowed at leaf level. This validation ensures that there is no composite and collection element without a child DDE.
• While importing XSD to create a data dictionary, the Data Dictionary Editor Prompts for a top-level element, if multiple exist, to create the data dictionary.
• Does not allow import of annotated FMLs with multiple top-level entities.
• The name and element type for a data dictionary/DDE are required parameters.
• A parent DDE (composite) can’t have two children with the same name.
• Ensures a DDE is marked computed, only if it is not a required parameter. A required element cannot be computed and a computed element cannot be required. Also, Collection and Composite Element cannot be computed elements.
• Ensures a DDE is marked required, only when it is not computed. It also ensures that it is not the "collectionElement" denoting the type of Collection (that is the only children of a collection Element).
• Empty keys or Duplicate keys are not allowed in extendedProperties for a data dictionary or DDE.
• Do not use the colon(:) or vertical bar(|) characters within the key or value of an extended property. If you do, the key can be lost or split into multiple keys. There is no validation for the use of these prohibited characters.

Validations that are applied at the Data Dictionary Level
• The Data Dictionary name must not be null.
• The Data Dictionary name may only contain alphanumeric characters.
• The child element list in the Data Dictionary must not be null or empty.
• The Data Dictionary must not contain more than one top level Data Dictionary Element.
• Only composite type is allowed as Top-level Element in a Data Dictionary.

Validations that are applied at the Data Dictionary Element Level
• All DDE names must not be null.
• All DDEs must not have a null Element Type.
• All DDE display names must not be null.
• All DDE display names must be unique.
• All DDE names may only contain alphanumeric characters and “_”.
• All DDE display names may only contain alphanumeric characters and “_”.
• Composite and Collection elements are not allowed at leaf level. Only Primitive (String, Date, Number, Boolean) elements are allowed at leaf level. This validation ensures that there is no composite and collection element without a child DDE.
• A composite parent DDE must not have two child elements with the same name.
• The ENUM subtype may only be used for String and Number elements.
• Collection and Composite elements cannot be computed.
• A DDE cannot be both computed and required.
• Computed DDEs must contain a valid non-null epression.
• Computed DDEs must not have XML binding.
• Collection DDEs that denote its type must not be computed or required.
• DDEs of subtype ENUM must not contain null or empty value sets.
• The XML binding of a collection DDE must not map to an attribute.
Understanding the Data Dictionary building block assets

- The XML binding syntax must be valid, such as, only one @ appears, the @ should only be followed by an attribute name, and so on.

Data Dictionary Services

The Data Dictionary building block services are exposed as spring beans. Following is the list of beans that are provided in Data Dictionary building block:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataDictionaryRegistryService</td>
<td>Provides operations creating, retrieving, updating and deleting (CRUD operations) for a data dictionary.</td>
</tr>
<tr>
<td>DataDictionaryUtilService</td>
<td>Provides data dictionary utility functions to:</td>
</tr>
<tr>
<td></td>
<td>• Import an XSD to create a data dictionary.</td>
</tr>
<tr>
<td></td>
<td>• Import/export of a data dictionary as annotated FML.</td>
</tr>
<tr>
<td></td>
<td>• Create a data dictionary object from XSD/FML. The data dictionary object can then be persisted through the APIs in the DataDictionaryRegistryService.</td>
</tr>
<tr>
<td></td>
<td>• Create sample xml data.</td>
</tr>
</tbody>
</table>

Default endpoints

The Data Dictionary building block APIs are exposed by default over Flex remoting. The APIs are not exposed for java clients. The solution integrator can expose the spring beans over their chosen endpoints with desired level of security. The configuration is property of a solution and not of the building block.

Dependence of the Data Dictionary building block on Expression Evaluator Service (EXM)

The Expression evaluator (EXM) service is exposed as a building block. The Data Dictionary building block is dependent on the EXM, which is a pluggable component. The Expression evaluator supports JSP EL evaluation. The expression evaluator service is used for evaluating computed DDEs for the data dictionary. The following points detail the DDE evaluation capabilities that the Data Dictionary building block supports using the EXM.

1. A DDE can be marked as Computed and the corresponding JSP EL expression can be provided for it through the Data Dictionary Editor user interface.

2. The computing expression for a DDE can be based on the values of other DDEs. For example, a DDE with name 'fullName' can bear an expression - ${firstName}$ {lastName} which evaluates full name as concatenation of first and last name.

3. The Data Dictionary building block does not support nesting of computed expressions. An operand in the expression cannot be another computed expression. The computed DDE evaluation is done on the server side when a DDI is being created.

4. The computed DDEs are created as non-Required, which means null values are permitted in a DDI.
Security authentication and authorization

No security is configured by default for the Data Dictionary building block service operations. The configuration is left to the system integrator to expose the Data Dictionary APIs at desired level of security.

Data Dictionary run time workflow

The Data Dictionary run time is responsible for returning the data dictionary instances (DDI) corresponding to a data dictionary. The Data Dictionary run time supports creation of DDI from Input XML Data (IXD) only. The solution has an initiating input XML data file that is used to run off its processes. For example, a letter generation process that is triggered with an input XML file containing all the relevant details. The input XML data must confirm to the schema of the XSD from which a data dictionary was created.

If a data dictionary was created manually, there would not have been any default XSD schema. When a data dictionary is created from an XSD schema, the import was modified, it's possible for its structure to decouple with the starting XSD schema. To prevent these errors, the Data Dictionary building block provides a tool to generate a sample XML corresponding to the data dictionary. The driving system ensures that IXD is provided to the system in the format consistent with the sample XML. The Data Dictionary run time creates a DDI out of the input XML based on a given data dictionary reference. The DDI can then be used to fill in the values in the application. The DDI is a read only artifact, which means there is no support of writing back data dictionary instances in the system. However, a data dictionary instance can be serialized and passed around in the workflows.

End-to-end flow definition

The following diagram details the end-to-end data dictionary definition. It includes the integration process including the personas involved, tools used, metadata storage, and its access at run time:
1 The Business Process Owner can use the Data Dictionary Editor to create a data dictionary manually. For information on creating the data dictionary using the Data Dictionary Editor, see “Creating a data dictionary” on page 11.

2 The Business Process Owner can also use the Data Dictionary Editor to create data dictionary by importing a schema.

3 On saving the data dictionary, the data dictionary metadata definition gets stored in the back end ie in the asset manager.

4 The business process owner can export the data dictionary using the Data Dictionary Editor, in form of an annotated FML model. This annotated model can then be imported to create a data dictionary.

5 A LiveCycle developer can use the domain XSD to create both the data dictionary and the artifacts such as the forms for the application. Designer can be used for importing an XSD and creating relevant forms from it.

6 The Solution Developer creates the solution in terms of artifacts, such as forms required by the application, corresponding Java classes, flex components, and so on.

7 The Solution Developer assembles them into a deployable archive (for example, a solution ear/war or just a set of solution beans). The solution is then deployed. The solution can be tested with sample data dictionary XML generation capability exposed through the Data Dictionary Editor.

8 The deployed customer solution accesses both the Data Dictionary metadata and Data Dictionary run time to accomplish its flows. For example, a document composer application accesses a letter data dictionary to render the required fields.

9 The Data Dictionary run time accesses the Data Dictionary metadata to create the Data Dictionary Instances (DDI). For more information on the Data Dictionary run time see “Data Dictionary run time workflow” on page 22.

10 The Data Dictionary run time does not integrate directly with back-end legacy systems.
Samples

The following details sample models and code samples which show implementation details for the Data Dictionary building block.

Mapping Simple (Primitive) Elements

A primitive DDE represents a field or attribute which is atomic in nature. Primitive data dictionary elements defined outside the scope of a complex type (composite DDE) or a repeating element (collection DDE) can be stored in any location within the XML Schema. The location of the data corresponding to a primitive DDE is not dependent on the mapping of its parent DDE. Primitive DDE uses the mapping information from the XML Binding field to determine its value and the mappings translate into one of the following:

- an attribute
- an element
- a text context
- nothing (an ignored DDE)

The following example shows a simple schema.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name='age' type='integer'/>
  <xs:element name='price' type='decimal'/>
</xs:schema>
```

Mapping Composite Elements

Allowing absolute mapping for child elements of a Composite DDE provides more flexibility in terms of Xpath Binding. Mapping a composite DDE to a particular complex type element in XML schema limits the scope of binding its child elements.

The following example shows the schema for a note.

```xml
<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Mapping Collection Elements

A collection element must be mapped only to another collection element (which has cardinality > 1). The child DDEs of a collection DDE have relative(local) XML Binding with respect to its parent’s XML Binding. As the child DDEs of a collection element must have the same cardinality as that of parent, the relative binding is mandated to ensure the cardinality constraint so that the child DDEs don’t point to a non repeating XML Schema element. In the example below, the cardinality of "TokenID" must be same as that of "Tokens" which is its parent collection DDE.
The XML Schema below declares an element with the name Tokens and a maxOccurs attribute of “unbounded”. Thus, Tokens is a collection element.

```xml
<?xml version="1.0" encoding="utf-8"?>
<Root>
  <Tokens>
    <TokenID>string</TokenID>
    <TokenText>
      <TextHeading>string</TextHeading>
      <TextBody>string</TextBody>
    </TokenText>
  </Tokens>
</Root>

Sample Schema that can be imported into the Data Dictionary

The following example shows the schema for a company.

```xml
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns="DCT" targetNamespace="DCT" xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="Company">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Name" type="xs:string"/>
        <xs:element name="Type" type="xs:anySimpleType"/>
        <xs:element name="HeadOfficeAddress" type="Address"/>
        <xs:element name="SalesOfficeAddress" type="Address" minOccurs="0"/>
        <xs:element name="HeadCount" type="xs:integer"/>
        <xs:element name="CEO" type="Employee"/>
        <xs:element name="Workers" type="Employee" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
Sample data model

The following code example details the FML export of a Company data dictionary, the data dictionary specific annotations use the DDS annotation. This example also contains two annotation groups: General and XmlBinding. The description and xpath are two respective annotations which are captured as part of these two annotation groups.

```xml
<model xmlns="http://ns.adobe.com/Fiber/1.0">  
  <annotation name="DDS">  
    <item name="name">Company</item>  
    <item name="displayName">Company</item>  
    <item name="version">1.0</item>  
    <item name="type">NONSYSTEM</item>  
    <item name="extended_k1">v1</item>  
    <item name="extended_k2">v2</item>  
  </annotation>  
  <annotation name="General">  
    <item name="description"/>  
  </annotation>  
  <entity name="Company">  
    <annotation name="DDS">  
      <item name="setAsTopLevelWithName">Company</item>  
      <item name="displayName">Company</item>  
      <item name="version">1.0</item>  
      <item name="type">NONSYSTEM</item>  
      <item name="extended_k3">v3</item>  
      <item name="extended_k4">v4</item>  
    </annotation>  
    <annotation name="General">  
      <item name="description"/>  
    </annotation>  
    <annotation name="XMLBinding">  
      <item name="xpath">/Company</item>  
    </annotation>  
  </entity>  
</model>
```
<annotation name="LiveCycleES">
  <item name="generate_type">true</item>
</annotation>

<property name="Name" required="true" type="string">
  <annotation name="DDS">
    <item name="displayName">Name</item>
  </annotation>
  <annotation name="General">
    <item name="description"/>
  </annotation>
  <annotation name="XMLBinding">
    <item name="xpath">/Name</item>
  </annotation>
</property>

<property name="Type" required="true" type="string">
  <annotation name="DDS">
    <item name="displayName">Type</item>
  </annotation>
  <annotation name="General">
    <item name="description"/>
  </annotation>
  <annotation name="XMLBinding">
    <item name="xpath">/Type</item>
  </annotation>
</property>

<property cardinality="one-to-one" name="HeadOfficeAddress" required="true" type="Address">
  <annotation name="DDS">
    <item name="displayName">HeadOfficeAddress</item>
  </annotation>
  <annotation name="General">
    <item name="description"/>
  </annotation>
  <annotation name="XMLBinding">
    <item name="xpath">/HeadOfficeAddress</item>
  </annotation>
</property>

<property cardinality="one-to-one" name="SalesOfficeAddress" required="false" type="Address">
  <annotation name="DDS">
    <item name="displayName">SalesOfficeAddress</item>
  </annotation>
  <annotation name="General">
    <item name="description"/>
  </annotation>
  <annotation name="XMLBinding">
    <item name="xpath">/SalesOfficeAddress</item>
  </annotation>
</property>

<property name="HeadCount" required="true" type="double">
  <annotation name="DDS">
    <item name="displayName">HeadCount</item>
  </annotation>
  <annotation name="General">
    <item name="description"/>
  </annotation>
  <annotation name="XMLBinding">
    <item name="xpath"/>
  </annotation>
</property>
<item name="xpath">/HeadCount</item>
</annotation>
</property>
<property cardinality="one-to-one" name="CEO" required="true" type="Employee">
  <annotation name="DDS">
    <item name="displayName">CEO</item>
  </annotation>
  <annotation name="General">
    <item name="description"/>
  </annotation>
  <annotation name="XMLBinding">
    <item name="xpath">/CEO</item>
  </annotation>
</property>
<property cardinality="one-to-many" name="Workers" required="true" type="Employee[]">
  <annotation name="DDS">
    <item name="displayName">Workers</item>
  </annotation>
  <annotation name="General">
    <item name="description"/>
  </annotation>
  <annotation name="XMLBinding">
    <item name="xpath">/Workers</item>
  </annotation>
</property>
</entity>
<entity name="Address">
  <annotation name="LiveCycleES">
    <item name="generate_type">true</item>
  </annotation>
  <property name="Street" required="true" type="string">
    <annotation name="DDS">
      <item name="displayName">Street</item>
    </annotation>
    <annotation name="General">
      <item name="description"/>
    </annotation>
    <annotation name="XMLBinding">
      <item name="xpath">/Street</item>
    </annotation>
  </property>
  <property name="City" required="true" type="string">
    <annotation name="DDS">
      <item name="displayName">City</item>
    </annotation>
    <annotation name="General">
      <item name="description"/>
    </annotation>
    <annotation name="XMLBinding">
      <item name="xpath">/City</item>
    </annotation>
  </property>
  <property name="State" required="true" type="string">
    <annotation name="DDS">
      <item name="displayName">State</item>
    </annotation>
    <annotation name="General">
      <item name="description"/>
    </annotation>
    <annotation name="XMLBinding">
      <item name="xpath">/State</item>
    </annotation>
  </property>
</entity>
<entity name="State">
  <property name="State" required="true" type="string">
    <annotation name="XMLBinding">
      <item name="xpath">/State</item>
    </annotation>
  </property>
</entity>

<entity name="Zip">
  <property name="Zip" required="true" type="string">
    <annotation name="XMLBinding">
      <item name="xpath">/Zip</item>
    </annotation>
  </property>
</entity>

<entity name="Employee">
  <property cardinality="one-to-one" name="PersonName" required="true" type="Name">
    <annotation name="XMLBinding">
      <item name="xpath">/PersonName</item>
    </annotation>
  </property>
  <property name="DOB" required="true" type="string">
    <annotation name="XMLBinding">
      <item name="xpath">/DOB</item>
    </annotation>
  </property>
  <property cardinality="one-to-one" name="CurrAddress" required="true" type="Address">
    <annotation name="XMLBinding">
      <item name="xpath">/CurrAddress</item>
    </annotation>
  </property>
  <property name="DOJ" required="true" type="date">
    <annotation name="XMLBinding">
      <item name="xpath">/DOJ</item>
    </annotation>
  </property>
</entity>
<item name="displayName">DOJ</item>
</annotation>
<annotation name="General">
  <item name="description"/>
</annotation>
<annotation name="XMLBinding">
  <item name="xpath">/DOJ</item>
</annotation>
</property>

<property name="Phone" required="true" type="double">
  <annotation name="DDS">
    <item name="displayName">Phone</item>
  </annotation>
  <annotation name="General">
    <item name="description"/>
  </annotation>
  <annotation name="XMLBinding">
    <item name="xpath">/Phone</item>
  </annotation>
</property>

<entity name="Name">
  <annotation name="LiveCycleES">
    <item name="generate_type">true</item>
  </annotation>
  <property name="FirstName" required="true" type="string">
    <annotation name="DDS">
      <item name="displayName">FirstName</item>
    </annotation>
    <annotation name="General">
      <item name="description"/>
    </annotation>
    <annotation name="XMLBinding">
      <item name="xpath">/FirstName</item>
    </annotation>
  </property>
  <property name="MiddleName" required="true" type="string">
    <annotation name="DDS">
      <item name="displayName">MiddleName</item>
    </annotation>
</entity>
Invoke data dictionary APIs through HTTP

In order to use this sample you will need to add the &lt;dct_root&gt;/client/java/adobe-dct-client.jar to your project's classpath.

The data dictionary services are exposed using Spring Remoting over HTTP and the following configuration is loaded in the application context.

```xml
<bean name="/service/DataDictionaryRegistryService" class="org.springframework.remoting.httpinvoker.HttpInvokerServiceExporter">
  <property name="service" ref="lc.dct.dataDictionaryRegistryService" />
  <property name="serviceInterface" value="com.adobe.dct.service.DataDictionaryRegistryService" />
</bean>

<bean name="/service/DataDictionaryUtilService" class="org.springframework.remoting.httpinvoker.HttpInvokerServiceExporter">
  <property name="service" ref="lc.dct.dataDictionaryUtilService" />
  <property name="serviceInterface" value="com.adobe.dct.service.DataDictionaryUtilService" />
</bean>
```

Invoke the data dictionary Util Service, and pass the data dictionary Id as parameter to get sample XML data for data dictionary.
HttpInvokerProxyFactoryBean httpInvokerProxyFactoryBean = new HttpInvokerProxyFactoryBean();
httpInvokerProxyFactoryBean.setServiceInterface(DataDictionaryUtilService.class);
httpInvokerProxyFactoryBean.setServiceUrl(soapEndpoint + 
"/cmsa/remoting/service/DataDictionaryUtilService*");
UsernamePasswordAuthenticationToken userAuthToken = new 
UsernamePasswordAuthenticationToken(userName, password);
SecurityContextHolder.getContext().setAuthentication(userAuthToken);
httpInvokerProxyFactoryBean.setHttpInvokerRequestExecutor(new 
AuthenticationSimpleHttpInvokerRequestExecutor());
httpInvokerProxyFactoryBean.afterPropertiesSet();
DataDictionaryUtilService ddUtilService = (DataDictionaryUtilService) 
httpInvokerProxyFactoryBean.getObject();
ddUtilService.getSampleXMLDataForDD(dataDictionaryId);

**Invoke data dictionary through the Flex client**

In order to use this sample you will have to add <dct_root>/client/flex/adobe-dct-flex-services.swc to your Flex project's classpath.

Get the singleton instance of data dictionary service.

DataDictionaryUtilServiceImpl ddUtilService = DataDictionaryUtilServiceImpl.getInstance();

Invoke the data dictionary Util Service, and pass the data dictionary Id as parameter to export a data dictionary as annotated FML.

```public
/**
 * Exporting Data Dictionary
 */
private function exportAsFML(event:*):void
{
    var ddId:String = //The id of an existing data dictionary.
    ddUtilService.exportDataDictionaryAsFML(ddId, handleSuccess, handleFault);
}
/* Success Handler */
private function handleSuccess(event:DCTSServiceEvent):void
{
    var ddFML:String = event.data as String;
}
/* Fault Handler */
private function handleFault(event:FaultEvent):void
{
    trace("Error exporting DataDictionary"+ (event.message as ErrorMessage).faultCode );
}
```

Last updated 4/18/2011
Chapter 4: Implementing the Data Dictionary user stories

Before you do any customization work, you must set up your development environment. For information, see Setting up your development environment in the Correspondence Management Solution Accelerator Solution Guide.

Creating a data dictionary

A middleware developer typically creates the data dictionary. This person is knowledgeable about a particular domain and its objects, and understands domain entities and relevant relationships between them. For example, a developer in the financial sector is aware of domain objects such as account, balance, and so on.

For details on creating a data dictionary, see Story: Creating a data dictionary in the Correspondence Management Solution Accelerator Solution Guide.