
DFS Explained

Documentum Foundation Services the SOA face of Documentum. The remote invocation of DFS is implemented using SOAP based web-services. The beauty of DFS is that though DFS is exposed for the remote usages as Web Services it can be called locally using DFS Client libraries. DFS allows the migration easy by allowing services to be developed on existing BOF (Documentum Business Object Framework) also. Another interesting point about this is DFS consolidates the numerous inter depended methods into a Single Service. The DFS data model is expressed in both Java client library class and also in Service XML Schemas. This provides a consistent approach to model data that will be exchanged between the various business processes.

As mentioned above the DFS clients can be of two types.

- Applications (Consumers) written any language that can use WSDL to interact with DFS.

- Clients written on java that uses DFS Java Client library

Data Model of DFS

The data passed to and from the Services are encapsulated into DFS Object Model

These are the few important object types on DFS

- DataPackage**
- DataObject**
- ObjectIdentity**
- Property**
- Content**
- Permissions**
- Relationship**

Lets see what all these does now

DataPackage

According to the DFS Reference Guide from EMC ***The DataPackage class defines the fundamental unit of information that contains data passed to and returned by services operating in the DFS framework.***

That means DataPackage is a collection of DataObject Instances, which is passed back and forth by Object Service Operations. In other words when you call services like Create, Get, Update the Data Object Instances are passed using DataPackage class.

It's like an Envelope for putting various DataObject Instances. Object service operations process all the DataObject instances in the DataPackage sequentially.

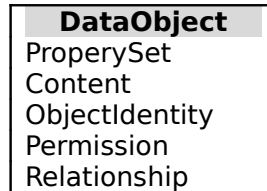
DataObject

A DataObject is the DFS representation of a **Persistent Object** in Documentum Repository. DataObject potentially has all the information related to an Object in repository. This includes the Content, properties (Both Single and Repeated), relation with other Objects in repository, Its permissions etc. Due the nature of these objects, Data Objects are very complex.

Optionally these object instances may have instructions to services about how the parts of Objects have to be processed.

-

The type field of DataObject class represents the underlying typed object type name that corresponds to the Object Instance. For example dm_user, dm_document etc. default type is dm_document. I.e. if no type is specified then service implicitly assign that to type dm_document



The above Figure represents a DataObject.

ObjectIdentity

As the name says this represents a Unique Object in the Repository. An instance of this class must have the repository name and a unique identifier for that object.

The value that represents a repository object may be any of the following

OBJECT_ID – Contains r_object_id of the Object (Represents both Current Object and Non Current Object)

OBJECT_PATH – Contains String expression of the path of the Object in repository

[Cabinet]/[Folder]/[File_Name]

(This Represents only Current Object. Since you can create multiple objects with same name in same directory this does not guarantee uniqueness of the object)

QUALIFICATION – DQL Snippet that qualify an Object uniquely (The DQL Snippet is the Full DQL that follows the keyword **FROM**. E.G. dm_document where r_object_id='09xxxxx')

Note: During the Creation of the object all you need to populate is only repository name.

Property

Each Property represents an Object Attribute (Property) in the Documentum Repository. **Property Set** is a collection, which works as a container of Property Objects

As there are 2 types of Properties for an Object a property can either be a Single property (Single Value Attribute) or Array Property (Repeated Attributes).

Property class has been sub classed to accommodate various data types and they are as follows.

-

StringProperty	StringArrayProperty
NumberProperty	NumberArrayProperty
BooleanProperty	BooleanArrayProperty
DateProperty	DateArrayProperty
ObjectIdProperty	ObjectIdArrayProperty
ArrayProperty (Abstract Class)	

Transient Property

These properties are not the part of persistent properties of a repository object. Transient Property can send custom data fields to a service to be used for a purpose other than setting attributes on repository objects.

Array Property and Value Action

The Repeating Attributes of an Object type is represented as Array Properties. As you can see in the above table The ArrayProperty is an array of the corresponding single property.

Another Interesting aspect of Array Property is Value Action. Now lets see what value action is all about? This is an optional Action – Index mapping pair. These pair has the instruction as to what is the action to be done on a particular element of the ArrayProperty.

The **Index** is the position in the attribute and **Action** is what is the action to be performed on that element of that index.

The possible ActionTypeValues are

Append

When processing ValueAction[p], the value at ArrayProperty[p] is appended to the end of repeating properties list of the persistent repository object. The index of the ValueAction item is ignored.

Insert

When processing ValueAction[p], the value at ArrayProperty[p] is inserted into the repeating attribute list before position index. Note that 1, which must be accounted for in subsequent processing, offsets all items in the list to the right of the insertion point.

Delete

The item at position index of the repeating attribute is deleted. When processing ValueAction[p] the value at ArrayProperty[p] must be set to a empty value.

-

Set

When processing ValueAction[p], the value at ArrayProperty[p] replaces the value in the repeating attribute list at position index. **(From DFS Development Guide)**

Content

Content Class or its Subclass instance represents the actual File Content associated with a DataObject. DataObject can have zero or more Content objects. It can hold Renditions also. Content class object can be configured to hold following

The Whole Document

Page of a Document

Pages (One or More) which has been represented by the characteristic

Permissions

A DataObject has list of Permission Objects, which decides the permission of the object that's represented by the DataObject. The permission list provides read access to the permission on an object in repository by the user who has been logged in currently.

Also have to note an interesting point here that You Cannot change permissions on a repository object by changing the Permission list and saving the DataObject.

For changing permissions of a repository object the real ACL of that object has to be edited or replaced.

As mentioned above multiple Permission Objects creates Permission List.

The Permission Object has a field named permission Type that sets what type of permission is set. The values possible for these fields are BASIC, EXTENDED or CUSTOM

Another important thing about permissions are if you assign one particular permission for one user to one object, all the lower lever permissions below the assigned permissions will be granted to that user. This is known as Compound or Hierarchical Permissions

PermissionType Enum Constants (From DFS Guide)

PermissionType	Permission	Description
Basic	NONE	No access is permitted
	BROWSE	The user can view attribute values of content
	READ	The user can read content but not update.
	RELATE	The user can attach an annotation to object.
	VERSION	The user can version the object.
	WRITE	The user can write and update the object.
	DELETE	The user can delete the object.
Extended	X_CHANGE_LOCATION	The user can change move an object from one folder

-

		to another. All users having at least Browse permission on an object are granted Change Location permission by default for that object.
	X_CHANGE_OWNER	The user can change the owner of the object
	X_CHANGE_PERMIT	The user can change the basic permissions on the object.
	X_CHANGE_STATE	The user can change the document lifecycle state of the object.
	X_DELETE_OBJECT	The user can delete the object. The delete object extended permission is not equivalent to the base Delete permission. Delete Object extended permission does not grant Browse, Read, Relate, Version, or Write permission
	X_EXECUTE_PROC	The user can run the external procedure associated with the object. All users having at least Browse permission on an object are granted Execute Procedure permission by default for that object.