

Working with Documentum 6 Web Services

Abstract

The EMC Documentum 6 enterprise content management platform provides a fundamentally new service-oriented interface, exposing enterprise content management capabilities as a comprehensive catalog of shared services and web services. These services enable content management functionality to fully participate in an SOA environment, with maximum efficiency and performance. This white paper will provide an introduction to EMC Documentum 6 Enterprise Content Services as well as an overview of the web services offerings of EMC Documentum Foundation Services.

1/17/2008

Copyright © 2008 EMC Corporation. All rights reserved.

EMC believes the information in this publication is accurate as of its publication date. The information is subject to change without notice.

THE INFORMATION IN THIS PUBLICATION IS PROVIDED “AS IS.” EMC CORPORATION MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WITH RESPECT TO THE INFORMATION IN THIS PUBLICATION, AND SPECIFICALLY DISCLAIMS IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Use, copying, and distribution of any EMC software described in this publication requires an applicable software license.

For the most up-to-date listing of EMC product names, see EMC Corporation Trademarks on EMC.com

EMC², EMC, Documentum, and where information lives are registered trademarks of EMC Corporation. SAP is a registered trademark of SAP AG in Germany and in several other countries. All other trademarks used herein are the property of their respective owners.

H3497

Table of Contents

Introduction	4
Industry trends	4
Enterprise content services	4
Service-oriented architecture	4
Web services.....	5
New EMC Documentum Enterprise Content Services	6
EMC Documentum Foundation Services	6
Business value	7
The DFS data model.....	7
The separation of data and services.....	8
Features of DFS	8
Client productivity layer.....	8
Content transfer	9
Exception handling.....	9
Development with EMC Documentum Composer	10
Conclusion	11
About EMC	11
Appendix A—DFS data model and core platform services.....	12
DFS data model	12
DFS core platform services.....	13

Introduction

Industry trends

Enterprise content management is one of the most important information technology innovations available today, largely because of the recognized importance of content in today's ever-changing business environment. Content comes in many forms, but often represents business assets such as contracts, product information, or reports, that need to flow across internal and external business systems and be consumed by multiple different parties. EMC® Documentum® has introduced a new generation of enterprise content management technology that is now being recognized as an integral part of an information infrastructure, providing a unified, service-oriented platform upon which many applications and solutions are built. E-learning systems, contract management applications, and portal applications are just a few of the solutions that can leverage enterprise content management services.

The importance of content and content management underscores the need for a way to easily integrate content and content-rich services across systems; however, today's IT environments are often complex and rigid with limited acceptance of new architectures and technologies. Faced with increased business demands, tight budgets, and inflexible infrastructures, the IT manager requires an approach to effectively integrate content management in a cost-effective and maintainable fashion while maximizing resource reuse. The industry has responded to this problem by defining an architecture that implements common business services in a cost-effective, efficient manner that reduces integration expense and improves business agility. This is called a service-oriented architecture (SOA). The Federal CIOs Council defines service-oriented architecture as:

“An architecture that provides for reuse of existing business services and rapid deployment of new business capabilities based on existing capital assets.”

SOA leverages HTTP-based web functionality to provide key business services, in the form of standards-based web services. In doing so, SOA provides the foundation for a new kind of business agility, cost-effectively allowing IT to be more responsive to business requirements while improving application development, deployment, and on-going support. With the promise of freedom and flexibility, the adoption of SOA is one of the most strategic mandates facing business and IT today.

The EMC Documentum 6 enterprise content management platform provides a fundamentally new service-oriented interface, exposing enterprise content management capabilities as a comprehensive catalog of shared services and web services. These services enable content management functionality to fully participate in an SOA environment, with maximum efficiency and performance. This document will provide an introduction to Documentum 6 Enterprise Content Services (ECS) as well as an overview of the web services offerings of Documentum Foundation Services' (DFS) web services offerings.

EMC Documentum Enterprise Content Services

Service-oriented architecture

The phrase service-oriented architecture represents an architectural style by which loosely coupled, coarse-grained services that represent business processes are integrated together to build business applications. The goal of SOA is to quickly and inexpensively allow the design and implementation of business applications based on new or existing services.

Prior to the advent of service-oriented architectures, applications were implemented independently from one another, often with insufficient thought to future integrations. These heterogeneous business

applications were integrated with tight couplings which tended to create a brittle and inflexible infrastructure. What resulted was a system which provided no room for growth and an inherent predisposition for system failure as the demands of the enterprise outgrew the patchwork of applications.

A service-oriented architecture solves this problem by providing a standards-based interface by which all services interoperate, making them “loosely coupled” to each other. Furthermore, the services are designed and built to be independent of the implementation technology or language, yet function seamlessly since they are built upon the foundation of SOA.

An additional benefit of SOA is in the definition of services. Services can be designed and built as coarse-grained functionality representing business operations. This makes the resulting services more easily understandable and consumable by the business. In a credit application, for example, rather than having separate components to authenticate, look up credit information, and then determine approval, a coarse-grained service can be defined simply as “check credit.” A business consumer assembling an application can leverage the check credit service, thus eliminating the need for a full understanding of the implementation. Additionally, once this service is published and integrated into multiple applications, enhancements to the underlying core service will be reflected in all applications utilizing this service. This not only promotes asset reuse but improves consistency while easing compliance and support. As these loosely coupled services are brought online they can be assembled to create robust and flexible applications without the additional investment typically required when building and integrating custom solutions. Essentially, key business processes can be deployed into the enterprise rather than just the application. The CBDi Forum, an independent analyst firm, identifies an SOA as:

“The policies, practices, and frameworks that enable application functionality to be provided and consumed as sets of services are published at a granularity relevant to the service consumer. Services can be invoked, published, and discovered, and are abstracted away from the implementation using a single, standards-based form of interface.” -CBDi Forum

Web services

In an SOA, loosely-coupled enterprise services form the building blocks of enterprise applications. Once these components are designed and built, a mechanism is required for inter-service communication. Web services, by definition, are a standards-based software system designed to support interactions over a network. By exposing loosely coupled components as services, the IT manager can inexpensively assemble enterprise applications in a dynamic, flexible manner.

For services to provide value they must be interoperable with a wide range of service-requesting client systems. A service may be called on to interact with custom applications, to participate in an Enterprise Services Bus (ESB), or to integrate with large-scale systems such as SAP®. In order to achieve this, the employment of standards is a critical component in the effective delivery of web services.

As a standard, the web services framework defines three stacks, each of which implement a technology for how services are integrated.

- **Wire stack:** Represents technologies that determine how messages are communicated from the service provider to the service requestor. The Simple Object Access Protocol (SOAP) is the standards-based protocol for this communication.
- **Description stack:** Represents technologies that allow a common understanding between the service requestor and service provider. The Web Service Definition Language (WSDL) is the mechanism for describing web services. WSDL is the primary API descriptor for Documentum 6 web services.
- **Discovery stack:** Represents technologies that allow a service requestor to locate a service provider. A variety of service registries are used to provide this function. One example of a service registry is the Universal Description, Discovery, and Integration (UDDI), which provides a standards-based

mechanism for services providers to register and be located by service requestors within and across enterprises.

New EMC Documentum Enterprise Content Services

With its new Enterprise Content Services (ECS), Documentum has the capability to deliver content management functionality for full participation in an enterprise SOA. It is important to recognize that ECS was designed and built from the ground up to leverage shared services and web services. This is more than just wrapping the legacy programming interface into a service. Instead, core Documentum functionality was decomposed and aggregated to fit into the SOA landscape in order to optimally deliver enterprise-ready and business-consumable content management services.

In architecting ECS, the core content management functionality of Documentum is decomposed and then rebuilt as new, higher-level, context-aware services. The functionality required to achieve a specific goal is included in the architecture while care is taken to ensure that exclusive services are isolated. For example, the Object service and the Versioning service are separate, since not every application that could leverage Object services is able to use the Versioning aspect. This allows any application to incorporate services that create, update, and delete documents, without attempting operations that the application cannot perform. This aggregation and exclusion balances the joint goals of simplifying the building of content applications and integrating with non-content management applications.

With enterprise integration as a founding principal, ECS is designed to provide services for maximum reuse and ease of composition into enterprise applications. In addition to cooperating within an SOA, web services can be leveraged to build custom applications as well as integrate with other enterprise applications that support web services standards. With ECS, a wide array of rich services can be leveraged to benefit the enterprise.

EMC Documentum Foundation Services

The new EMC Documentum Foundation Services (DFS) offering is the initial suite of services released under the ECS umbrella. DFS comprises the “core” Enterprise Content Services package. Fundamentally, DFS is a set of out-of-the-box services and design tools that enable the Documentum content management platform to become an integral part of an organization’s information infrastructure. DFS effectively exposes key content management functionality as standards-compliant web services.

With a goal of alignment towards business agility, the DFS communication syntax is designed to consider how end users think about content management operations using natural terms such as ‘Find User’ or ‘Create Object’. The alignment of the technical API with the end user’s Natural language increases the accessibility of the system and its content to both developers and business analysts alike. DFS is designed to provide an interface using language the user will understand without being a content management or Documentum expert. In doing so, content management concepts are easier to understand and utilize in both new and existing enterprise applications.

To accomplish the increased ease of use, hundreds of technical commands are aggregated into simple terms such as, OpenObject, that clearly represent the purpose of the command. DFS provides the following core services (for further details see Appendix A - DFS Data Model and Core Platform Services):

- **Object:** Basic repository object operations without need for version control system
- **Version control:** Operations to enable access and changes to specific object versions
- **Schema:** Operations to retrieve information regarding repository schemas
- **Query:** Primary mechanism for retrieving information from a repository
- **Search:** Provides full-text and structured search capabilities against multiple repositories
- **Workflow:** Operations concerning workflow process templates and instances

With SOA and DFS, system implementation is now ultimately about the composition of available services. Whether you are an application developer using best-of-breed utilities or a business analyst with Business Process Management (BPM) tools and portals, using an SOA with DFS opens the Documentum platform to a larger audience that is closer to the business needs.

Business value

Enterprise Content Services provide significantly more value to the business than systems that do not employ robust web services. The greatest value is due to the lower cost that can be achieved during the development, deployment, and support of applications built upon Documentum. Because these services are course-grained and business user friendly, they are able to be quickly deployed into multiple applications, subsequently improving IT responsiveness to business requirements while reducing overall costs. The services are also intrinsically designed for reuse across enterprise applications. By eliminating the need to re-invent the wheel for every application, ECS efficiently provides yet another way to increase business agility and optimize resources.

Another avenue for recognizing the business value of ECS is through the increased productivity a development staff can realize through reduced ramp-up and training time. The simplified and re-architected, standard-English interface reduces the learning curve for new and skilled developers while also speeding acceptance by the business organization. Additionally, the standards-based web services provide for maximum compatibility with Java, Microsoft .NET, and other development environments. This compatibility greatly improves the ability to leverage skills of existing IT resources.

While these values fundamentally exist even for organizations that have not chosen to adopt a service-oriented architecture, the increased simplicity and business access to the services will facilitate adoption, subsequently reducing the overall cost incurred to implement content management solutions.

The DFS data model

DFS is designed to optimize a number of core platform services. These services cover a broad area and reduce the syntax to a small number of powerful concepts. The interface to these services remains the same regardless of the particular repository details or the context of a given operation. The variability of any given implementation is abstracted through the use of an efficient model that manages data communicated from the service to the client. The sophistication of the DFS data model provides features which enable and encourage service definition and implementation simplicity.

The DFS data model comprises the object model for data passed to and from the platform services. The elements of the data model are as follows (For further details see Appendix A—DFS data model and core platform services):

- **Data package:** A collection of objects that represent data passed to and returned by services operating in the DFS framework
- **Data object:** A representation of an object in an enterprise content management repository including both content and metadata
- **Content:** The representation of content within a data object including documents, pages, or sets of pages
- **Object identity:** Represents the unique identity of a repository object
- **Property:** Represents a collection of attributes regarding an object
- **Permissions:** Represents a list of access privileges on repository objects
- **Relationships:** A representation of the relationship an object has or is intended to have to other objects in the repository

The separation of data and services

DFS defines the data object independent of the service or container, providing an abstraction between the service performing the work and the object containing the data. Once separated, processing and data manipulation can occur geographically independent of each other. This enables content items to be retrieved, manipulated, updated, and even copied independent of the originating content repository. In an enterprise service-oriented architecture this is particularly powerful because the consuming application can retrieve and operate on data with no knowledge of the underlying service implementation or container context. Therefore, the services can be quickly integrated with additional applications without the development or deployment overhead of creating additional services.

To elaborate on the power of this separation, consider a situation in which coarse-grained services are invoked from a geographically distant client. Since data can be managed and manipulated independent of the location or operation of the services, the DFS data and service architecture provides a mechanism for application performance optimizations. The assembler of an application using these coarse-grained services can reduce the frequency of over-the-wire communications to points at which it is mandatory. This allows users to essentially perform the maximum amount of work at any particular location before interacting with another process. Whereas some Web service implementations require substantial increases in overhead, DFS represents a significant boost in efficiency.

Another advantage to the decoupling of data is that it removes the dependence on connectivity to the content server. In previous releases, objects inherently lived in the content server and service calls executed directly against them. Through the power of the DFS object model, you can now take a copy of the object and perform operations in a different location. With DFS you don't have to be connected to the server in order to manipulate an object. The data package represents the entirety of the data, even allowing offline validation.

For example, consider two alternative architectural styles for achieving service orientation: Simple Object Access Protocol (SOAP) and Representational State Transfer (REST). While most web services currently follow the SOAP model, there is a movement in the technical community towards using REST methodology. Although the DFS platform services use SOAP today, the data model architecture enables the same data model provided in Documentum 6 to support SOAP as well as REST. This decoupling of architectural style from the service interfaces and the data representation adds flexibility to DFS that enables clients to efficiently adapt to these changing architectural preferences.

Features of DFS

Client productivity layer

The primary interface to DFS is the standards-based WSDL. All DFS services are accessible using this web services technology. Hence, they can be accessed by consumers using standard tools. A client for a consumer service can connect using WSDL and use a DFS service without proprietary software or libraries.

As a convenience and performance enhancement, language specific client libraries are provided that delivers development and runtime support for both Java and .NET consumers of DFS services. These optional client libraries allow DFS consumers to view the services contract as they would standard, non-remote Java or .NET methods. For example, for a Java developer not yet up to speed with messages, this allows interaction as though DFS were simply a Java API.

A major benefit of using the Java client library is to provide the Java-based consumer with location transparency. Location transparency is not available with the .NET library. Location transparency allows the client runtime to execute either remotely, as a Web service, or locally, as an application running within the Java virtual machine (JVM). When executing remotely, the client library maintains the SOAP-based

procedure invocation behind the scenes, and when executing locally, the client uses a higher-performance direct (in-process) Java call.

This capability solves a number of problems in the testing and debugging of custom web services. Now, you can deeply test an interface locally without incurring the complexities and time delays associated with deploying to an application server. While the primary use for local service invocation may be rapid development and testing, it may also be useful in certain production deployment scenarios. For instance, suppose a business application required a service chain of several distinct services. Three of the services in this chain have an affinity towards a remote geographic location, while others are deployed in the same process space as the consumer. The client library allows you to easily locate these services where they can be run most efficiently, allowing them to interact either locally or remotely without changing the client code.

Content transfer

Inherent in any content management system is the need to efficiently transfer content to the consumer. In a large, geographically distributed system, this can become a daunting task often requiring specialized infrastructure components. The introduction of web services that are implementation- and location-agnostic further complicates this task. To address the issues surrounding content transfer, DFS utilizes the Unified Client Facilities (UCF) services.

DFS provides support for UCF content transfer and includes functionality to address the development issues surrounding content transfer. DFS handles all the complexities of content transfer behind the scenes, the details of which are specified through a content transfer profile. This profile allows a participant in the service chain to easily specify requirements for the transfer including geographic location, caching, and support for UCF. DFS then manages the transfer.

This improved support for UCF and focus on the challenges of content transfer in a distributed network are designed to further enhance the participation of Documentum in an enterprise SOA. In addition, DFC is fully integrated with Documentum distributed content capabilities and supports applications such as Branch Office Caching Services.

Exception handling

Any first-class enterprise technology solution requires a strategy for exception handling. While DFS does not expect service consumers to fail, when it does happen, DFS provides useful ways to determine the cause of the failure. Web services often add a real challenge to effective exception handling as error messages have to be adequately created and packed within a SOAP fault message and returned, across the wire, to the client. When errors occur in some service implementations, adequate information on the error is not passed to the client application. This can make it difficult to identify the problem.

DFS meets the challenges of reporting Web service exception handling by implementing a platform which quickly and efficiently packages the “error” or exception to the client. Overcoming the limitations previously encountered by web-based error reporting, DFS has the functionality of automatically associating exceptions with SOAP faults and delivering the full explanation for the exception to the client in a concise manner. Additionally, if the Java client library is utilized, the consumer has the added benefit of receiving a Java exception object. This exception handling can be disabled for the maximum performance of production applications.

Development with EMC Documentum Composer

A key goal of Documentum 6 is to reduce the cost of developing applications and to facilitate the entire ecosystem of customers, partners, and business analysts. This is accomplished by reducing the learning curve associated with Documentum tools through user-interface consistency and the broad familiarity with the Eclipse framework across the developer community. Eclipse is an open-source, standardized development framework familiar to many developers. The vision for the Documentum toolset is to provide a unified Integrated Development Environment (IDE) supporting the entire application lifecycle, with appropriate functionality and user interface for stakeholders along the entire spectrum of application development and usage.

The Documentum 6 facility that is used to accomplish this is the new Eclipse-based tooling platform called EMC Documentum Composer. By combining the Eclipse Java IDE with the DFS software development kit, it's easy to build DFS web services with Documentum. Our IDE contains a script that can help you automatically generate a WSDL file for your Web service. We also include a script that packages your DFS Java source code into an EAR file, which can be deployed on your preferred, certified Application Server.

Conclusion

EMC Documentum Foundation Services has stepped beyond enabling access to the Documentum platform via web services and is now providing a complete infrastructure for developing applications in the ECM space. The services provided with DFS focus on a small number of powerful concepts that are comprehensible to anyone familiar with application development in general, not just the experienced Documentum developer. These concepts are enhanced by a rich variety of underlying behaviors, based upon the context of the service call.

Through the Documentum Foundation Services, the Documentum platform is equipped to facilitate a wide range of business processes and services. Whether our customers are embracing service orientation as an architectural approach or are simply moving their enterprise towards the power of web services capabilities, DFS is a powerful choice.

About EMC

EMC Corporation (NYSE: EMC) is the world leader in products, services and solutions for information management and storage that help organizations extract the maximum value from their information, at the lowest total cost, across every point in the information lifecycle. Information about EMC's products and services can be found at www.EMC.com.

To learn more about enterprise content management visit us online at <http://www.emc.com/documentum> or call 800.607.9546 (outside the U.S.: +1.925.600.5802).

Appendix A—DFS data model and core platform services

DFS data model

EMC Documentum Foundation Services (DFS) is intentionally designed with only a small number of core platform services. These are course-grained services which reduce the interface to a handful of powerful concepts. Regardless of the particular repository details or context of a given operation, the interface to these services remains the same. The rich capability of the DFS data model provides the design features to enable and encourage simplicity of the services.

The DFS data model comprises the object model for data passed to and from the platform services. The following is a description of the DFS data model.

DataPackage

The DataPackage class defines the fundamental unit of information that contains data passed to and returned by services operating in the DFS framework. A DataPackage is a collection of DataObject instances used to communicate with object service operations such as create, get, and update. Object service operations process all the DataObject instances in the DataPackage sequentially.

DataObject

A DataObject is a representation of an object in an enterprise content management repository including both content and metadata. In the context of EMC Documentum technology, the DataObject functions as a DFS representation of a persistent repository object, such as a dm_sysobject or dm_user. Enterprise content services (such as the object service) consistently process DataObject instances as representations of persistent repository objects.

A DataObject instance is potentially large and complex and much of the work in DFS service consumers will be dedicated to constructing the DataObject instances. A DataObject can contain comprehensive information about the repository object that it represents, including its identity, properties, content, and its relationships to other repository objects. In addition, a DataObject instance may contain settings that instruct the services about how the client wishes the DataObject to be processed.

DataObject instances are consistently passed to and returned by services in simple collections defined by the DataPackage class. The use of these collections permits processing of multiple DataObject instances in a single service interaction.

Content

The content for a DataObject is represented by the Content class and its subtypes. The Content subtypes support multiple types of input to services and multiple content transfer options. A given Content object can represent a complete document, a single page in a document, or a set of pages in a document. A DataObject contains a list of zero or more of these Content instances, allowing representation of both primary content and content renditions.

The ContentProfile object enables a client to set filters that control the content returned by a service. This has important ramifications for service performance because it permits fine control over expensive content transfer operations.

ObjectIdentity

The ObjectIdentity class uniquely identifies a repository object. It contains both the repository name, as well as a unique identifier for the object within that repository. This identifier can take one of several different forms: an object ID, an object path, or a query expression that selects a single object.

Property

A Property object represents a property or attribute of an object. A single Property object can represent a single property, while an array of properties of the same data type is used to represent repeating attributes. Each DataObject optionally contains a set of these Property objects within a PropertySet. Additionally, a Property can be optionally marked as transient. Transient properties are custom Property objects that are not persistent properties of repository objects. They are extremely powerful as they can be used to pass information to a client to serve a purpose other than simply setting attributes on repository objects.

Similar to a ContentProfile, a PropertyProfile defines filters that limit the properties that are returned with an object by a service call. This allows for optimization of a service by returning only those properties that your application requires.

Permissions

A DataObject contains a list of Permission objects. Together, these represent the permissions of the user defined in the service context on the given repository object. The Permission list can serve to provide the calling client with read access to the current user's permissions on a repository object.

The PermissionProfile enables the client to set filters that control the contents of these Permission lists in DataObjects returned by service calls. By default, the platform services will return an empty Permission list. The client must explicitly request in a PermissionProfile for the permissions be returned.

Relationships

The Relationship class defines the relationship a repository object (represented by a DataObject instance) has, or is intended to have, to other objects in the repository. Relationships allow the client to construct a single DataObject that specifies all of its relations to other objects (both existing and new) and to get, update, or create the entire set of objects and their relationships in a single service interaction. For example, this would allow you to create a new folder and a set of new linked documents all in a single service call.

The RelationshipProfile is a client optimization mechanism that provides fine control over the size and complexity of the DataObject instances returned by the services. By default, the platform services will return DataObjects containing no Relationship instances. To alter this behavior, the requestor would provide a RelationshipProfile that specifies the types of Relationship instances to return with the DataObject.

DFS core platform services

The DFS platform services are a set of core web services provided with the Documentum platform to provide a service-oriented interface to content server and associated repositories.

These core services present regularly-performed operations to the user in a clean and straightforward way. The following sections provide an overview of the functionality provided with the DFS platform services.

Object service

The object service provides a set of basic operations on repository objects. These services are responsible for all typical persistent operations on an object such as creating, getting, updating, and deleting repository objects. Additionally, all object services can operate on multiple objects enabling clients to optimize service usage by minimizing the number of service interactions. For example, an object service can create users and associate them with groups in a single transaction.

Note that the object service has been designed to operate independently from version control as not all applications have the requirement to version objects. Each operation within the object service uses default behaviors as they relate to object versions that are appropriate for the given operation.

The object service operations include:

- **create:** Create an object (or objects) in the repository and optionally link into a specific location.
- **createPath:** Create a folder structure in the repository.
- **get:** Retrieve the specified version of a set of objects from the repository.
- **update:** Update properties, content, and relationships of the current version of a set of repository objects, saving the repository objects as the current version. If the objects to be updated do not exist, they are automatically created.
- **delete:** Delete a set of objects from the repository.
- **copy:** Copy a set of objects from one location to another, either within a single repository or from one repository to another. During this operation, the service can optionally make modifications to the objects being copied.
- **move:** Move the current version of a set of objects from one repository to another repository, optionally updating the objects as they are moved.
- **validate:** Validate a set of objects against the repositories' data dictionary rules, testing if they represent valid repository objects with valid repository properties.

Version control service

The version control service provides operations that enable access of and changes to specific versions of objects within the repository. This service provides the following operations:

- **getCheckoutInfo:** Provides checkout information about a set of objects, specifically whether the objects are checked out and the name of the user that has checked them out.
- **Checkout:** Checks out the specified version of a set of repository objects.
- **checkin:** Checks in a set of repository objects.
- **cancelCheckout:** Cancels the checkout of a set repository objects.
- **deleteVersion:** Deletes a specific version of a repository object. If the deleted object is the current version, the previous version in the version tree is made the current version.
- **deleteAllVersions:** Deletes all versions of a repository object.
- **getCurrent:** Exports the current version of a repository object (including any object content).
- **getVersionInfo:** Provides information about the version of a repository object.

Schema service

The schema service provides a mechanism for retrieving information regarding repository schemas—the formal definition of the repository metadata. It exposes a generalized view into the data dictionary, including information regarding types, properties, and relationships. This service is extremely powerful in that it can be used for creating a data structure against which a client can perform offline validation of objects against the repository metadata.

The schema service includes the following operations:

- **getSchemaInfo:** Retrieves schema information for the default schema of the specified repository.
- **getRepositoryInfo:** Retrieves schema information about a repository specified by name, including a list of repository schemas.
- **getTypeInfo:** Retrieves information about a repository type specified by name.
- **getPropertyInfo:** Retrieves information about a repository property specified by type and name.
- **getDynamicAssistValues:** Retrieves information about value assistance for a specified repository property. (Value assistance provides a list of valid values for a property. It is typically used to provide a pick-list of values for an attribute associated with a field on a dialog box.)

Query service

The query service is a primary mechanism for retrieving information from a repository. Query service is general purpose and uses execution semantics similar to the use of queries in an RDMBS. The service returns a data set resulting from the query to the user either directly or through asynchronous caching.

To execute a query against a repository a `PassThroughQuery` object must be created to hold the query statement. The default mode of operation for `PassThroughQuery` returns all the query results directly to the client. When processing larger result sets the client can specify cached mode for the server and then process the results either sequentially or via paging by utilizing the `startIndex` position identifier. The query service shares the same data model as the search service.

The query service provides the following operation:

- **execute:** Runs a query against data in the repository and returns the result

Search service

The search service provides full-text and structured search capabilities against multiple Documentum repositories as well as against external sources. Documentum full-text indexing is required for the search service. The search service shares the same data model as the query service.

The search service provides the following operations:

- **getRepositoryList:** Provides a list of managed and external repositories that are available to the service for searching
- **execute:** Runs a query against data in the repository and returns the result

For Additional information:

EMC Documentum Foundation Services Developer's Guide V.6: A guide to developing with EMC Documentum Foundation Services (DFS) 6.0, a set of technologies that enables service-oriented programmatic access to the EMC Documentum Content Server platform and related products.