# MarkLogic Server

## Glossary, Copyright, and Support

MarkLogic 9
May, 2017

Last Revised: 9.0-11, November, 2019

# Table of Contents

Glossary, Copyright, and Support

# 1.0 Glossary

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z

**A**

**Access Control**

*Security*. Restricting access based on something other than the identity of the user. See permission, amp, authentication, and Access Control List (ACL). See also Authentication and Access Control in the *Security Guide*.

**Access Control List** (ACL)

*Security*. A list of permissions attached to an Object. The ACL specifies who or what can access an object, or what operations are allowed on that object. See Access Control.

**ACID** (properties)

*Database*. Atomicity (a set of changes either takes place as a whole or doesn't take place at all), Consistency (system rules are enforced, such as that no two documents should have the same identifier), Isolation (uncompleted transactions are not otherwise visible), and Durability (once a commit is made it will not be lost).

**active data**

*MarkLogic*. Active data is data that requires low-latency queries and updates. The "activeness" of a particular document is typically determined by its recency and thus changes over time. See archived data and historical data.

**administration APIs**

The MarkLogic Admin APIs provide a flexible toolkit for creating new and managing existing configurations of MarkLogic Server. The Admin APIs take the following forms:

- XQuery Functions
- Server-side JavaScript Functions
- REST Resources

**administration Interface**

The Administraton Interface (Admin Interface) is graphic user interface for managing MarkLogic Server and is implemented as a web application that runs in your browser.

**aggregate**

*Marklogic*. XML content that includes recurring element names and which can be split into multiple documents with the recurring element as the document root. For more information, see Splitting Large XML Files Into Multiple Documents in the *mlcp User Guide*.

**aggregate function**

*MarkLogic*. Used for tasks like computing a sum or count over an element, attribute, or field range index, or many of these range indexes. Aggregate functions are most useful for analytics that produce a small number of results, such as computing a single numeric value across a set of range index values. MarkLogic predefines many aggregate functions, and you can define your own aggregate UDF. See Introduction to Aggregate Functions in the *Search Developer's Guide*.

**aggregate UDF**

*MarkLogic*. An aggregate UDF (User Defined Function) is a custom function that analyzes values in lexicons and range indexes. An aggregate function is best used for analyses that produce a small number of results, rather than analyses that produce results in proportion to the number of range index values or the number of documents processed. An aggregate UDF is implemented in C++ and deployed as a native plugin. For more information, see Aggregate User-Defined Functions in the *Application Developer's Guide*.

**alerts**

*MarkLogic*. An alerting application is used to notify users when new content is available that matches a predefined (and usually stored) query. An alert is based on a reverse query. Alerts are used with QBFR (Query-Based Flexible Replication). See Configuring Alerting With Flexible Replication in the *Flexible Replication Guide*.

**Amazon Machine Image** (AMI)

*Cloud*. An encrypted machine image that contains all information necessary to boot instances of software. Instances of MarkLogic Server are created from the stock Amazon Linux AMI and have been pre-installed with MarkLogic and the necessary dependencies. A virtual appliance for use with Amazon's Elastic Compute Cloud (EC2).

**Amazon Resource Name** (ARN)

*Cloud*. Amazon Resource Names (ARNs) are used to identify AWS resources across all of AWS. For details, see https://docs.aws.amazon.com/general/latest/gr/aws-arns-and-namespaces.html.

**Amazon Web Services** (AWS)

*Cloud.* The Amazon Cloud Computing service. For details, see http://aws.amazon.com/.

**amp**

*MarkLogic*. An amp provides a user with the additional authorization to execute a specific function by temporarily giving the user additional roles. For details on amps, see Temporarily Increasing Privileges with Amps in the *Security Guide*.

**APIs and communications protocols**

MarkLogic supports the following application programming languages:

- XQuery (native, server-side support)
- JavaSript (native, server-side support)
- XSLT (native, server-side support)
- Java
- Node.js
- C#
- SQL (native, server-side support)
- SPARQL (native, server-side support)
- REST interfaces

In addition, XQuery, JavaScript, XSLT, SQL, and SPARQL are native langauges.

**App Server**

*MarkLogic*. A MarkLogic server, that can be one of the specific types of server; HTTP Server, XDBC Server, ODBC Server, or WebDAV Server. The type of server is defined at the group level when you create a new server.

**archive**

*MarkLogic*. A compressed MarkLogic Server database archive created using the mlcp export command. You can use an archive to restore or copy database content and metadata with the mlcp import command. For details, see Exporting to an Archive in the *mlcp User Guide*.

**archived data**

*MarkLogic*. Data that has aged beyond its useful life in the online storage tiers and is typically taken offline. See active data.

**asserted triples**

*Semantics*. In a system that does inference, asserted triples are those triples that are in the database before inferencing, as opposed to inferred triples which are the result of the inferencing process. See inference, inferred triples, ontology triples, and triple.

**asynchronous**

*General*. Asynchronous I/O, or non-blocking I/O is a form of input/output processing that permits other processing to continue before the transmission has finished. See synchronous.

**Asynchronous Replication**

*Flexible Replication*. A configuration in which the Master does not wait for confirmation that the update has been received by the Replica before committing the transaction and proceeding with additional transactions. Flexible Replication and database replication are asynchronous.

**Attribute-Based Control** (ABAC)

*Security*. A form of security based on attributes assigned to a role or object. ABAC is a logical access control model that controls access to objects by evaluating rules against the attributes of the entities (subject and object) actions and the environment associated with a request. See MarkLogic Security Model in the *Security Guide*.

**authentication**

*Security*. The process of verifying user credentials for a named user, usually based on a username and password. Authentication generally verifies user credentials and associates a session with the authenticated user. It does not grant any access or authority to perform any actions on the system. Authentication can be done internally inside MarkLogic Server, or externally by means of Kerberos.

**authorization**

*Security*. The process of allowing a user to perform some action, such as create, read, update, or delete a document or execute a program, based on the user's identity. Authorization defines what an authenticated user is allowed to do on the server. When an App Server is configured for external authentication, authorization can be done either by MarkLogic Server or by LDAP.

**auxiliary databases**

*MarkLogic*. The default databases that are created during the installation process: Security, Modules, Documents, Schemas, Triggers, and Last-Login.

**availability of a partition or forest**

*MarkLogic*. Refers to the online or offline status of a partition or forest.

## B

**Backup - Typical**

A non-incremental backup without journal archiving. Most of the time when a typical backup is running, all queries and updates proceed as usual. MarkLogic copies stand data from the source directory to the backup target directory, file by file.

**Backup with Journal Archiving**

The backup/restore operations with journal archiving enabled provide a point-in-time recovery option that enables you to restore database changes to a specific point in time between full backups with the input of a wall clock time.

**Backup - Incremental**

An incremental backup stores only the data that has changed since the previous full or incremental backup.

**backward-chaining inference**

*Semantics*. Backward chaining inference is inference done at query time; each query looks at the ontology and expands the query appropriately. Backward chaining inference is more resource intensive at query time, but less costly during ingestion and indexing. See also forward-chaining inference and inference.

**bastion host**

*Cloud*. A bastion host is a server that provides access to a private network from an external network.

**bitemporal**

*MarkLogic*. Bitemporal documents are associated with both a valid time and a system time. Data in a bitemporal format provides the user with a complete history (audit trail) of data in the database. For more information, see Understanding Temporal Documents in the *Temporal Developer's Guide*. See also non-temporal.

**blank node**

*Semantics*. (also known as a bnode or anonymous resource) A subject or object in an RDF graph that has no IRI (Internationalized Resource Identifier) or resource identifier. Typically used to group together other values for a triple. A blank node is represented by a prefixed name with an underscore prefix (`_:mytriple`). See bnode.

**bnode**

*Semantics*. A node in an RDF graph representing a resource for which an IRI (Internationalized Resource Identifier) or literal is not provided. This term can be used interchangeably with blank node.

**BLOB** (Binary Large Object)

*Database*. Binary data stored as a single entity. Typically BLOBs are images, audio, or other multimedia objects.

**BNF** (Backus Normal Form or Backus–Naur Form)

*General*. A notation technique for context-free grammars, often used to describe the syntax of languages used in computing.

**Bootstrap Host**

*MarkLogic*. A MarkLogic Server host machine used by a foreign cluster to initiate communication with the local cluster.

**built-in**

*MarkLogic*. A built-in function (or main module) is one that can be called without using import module in XQuery or JavaScript. MarkLogic built-ins are accessed through XQuery or JavaScript. See module.

**C**

**cache partitions**

*MarkLogic*. Along with setting each cache size, you can also set a cache partition count. Each cache defaults to one or two or sometimes four partitions, depending on your memory size. Increasing the count can improve cache concurrency at the cost of efficiency.

**caching**

*MarkLogic*. On database or group creation, MarkLogic assigns default cache sizes optimized for your hardware, using the assumption that the server will be acting as both E-

node and D-node. You can improve performance in a clustered environment by optimizing each group's cache sizes. With an E-node group, you can increase the size of the caches related to request evaluation at the expense of those related to data management. For a D-node, you can do the opposite.

**capabilities**

*MarkLogic*. The permissions set on a document define access to capabilities for that document. Capabilities include read, insert, update, and execute. Each permission consists of a capability and a role. See permission.

**CBD** (Concise Bounded Definition)

*Semantics*. The Concise Bounded Definition (CBD) specification is used to describe one or more nodes in the RDF graph as a resource. This implementation does not provide any reified statements, and returns a maximum of 9999 triples. See `sem:describe`. For more about CBD in general, see the W3C specification: http://www.w3.org/Submission/CBD/.

**CDH** (Cloudera Distribution Hadoop)

*Hadoop*. Cloudera's Distribution Including Apache Hadoop, an open source data management platform based on Apache Hadoop. One of the Hadoop distributions supported by the MarkLogic Connector for Hadoop. See the *MarkLogic Connector for Hadoop Developer's Guide*.

**CentOS**

*Linux*. A community-supported, free and open source operating system based on Red Hat Enterprise Linux.

**CEP** (Complex Event Processing)

*MarkLogic*. Complex event processing combines data from multiple sources in its tracking and analysis of streams of information about events, and uses that information to infer events or patterns that may suggest more complicated circumstances. The goal of complex event processing is to quickly identify and respond to meaningful events.

**certificate authority** (CA)

*Security*. A trusted third party that certifies the identity of entities, such as users, databases, administrators, clients, and servers. When an entity requests certification, the CA verifies its identity and grants a certificate, which is signed with the CA's private key. If the CA is trusted, then any certificate it issues is trusted unless it has been revoked.

**certificate** or **public key certificate**

*Security*. An electronic document that incorporates a digital signature to bind together a public key with identity information, such as the name of a person or an organization, address, and so on. The certificate can be used to verify that a public key belongs to an individual or organization.

**certificate request**

*Security*. A request data structure containing a subset of the information that will ultimately end up in the certificate. A certificate request is sent to a certificate authority for certification.

**certificate template**

*Security*. A MarkLogic construct that is used to generate certificate requests for the various hosts in a cluster. The template defines the name of the certificate, a description, and identity information about the owner of the certificate.

**cipher**

*Security*. An algorithm for encrypting information so that it's only readable by someone with a key. A cipher can be either symmetric and asymmetric. Symmetric ciphers use the same key for both encryption and decryption. Asymmetric ciphers use a public and private key.

**CLOB** (Character Large Object)

*Database*. Large text document.

**Cloud Capable**

*Cloud*. MarkLogic clusters can be deployed on Amazon Elastic Compute Cloud (EC2) as well as on Microsoft Azure Cloud. An EC2/Azure deployment of MarkLogic enables you to quickly get started with any sized cluster (or clusters) with a minimum upfront investment. You can then scale your clusters up or down, as your needs evolve.

**Cloud Formation** (CF)

*Cloud*. The AWS Cloud Formation service for provisioning startup of AWS resources. For details, see Deploying MarkLogic on EC2 Using CloudFormation in the *MarkLogic Server on Amazon EC2 Guide* and http://aws.amazon.com/cloudformation/. MarkLogic CloudFormation templates are available from http://developer.marklogic.com/products/aws.

**cluster**

*MarkLogic*. Multiple instances of MarkLogic Server configured to run as a cluster. The cluster has multiple machines (hosts), each running an instance of MarkLogic Server. Each host in a cluster is sometimes called a node.

**collection**

*MarkLogic*. A collection is a group of documents that enable queries to efficiently target subsets of content within MarkLogic Server. Collections are described as part of the W3C XQuery specification, but their implementation is undefined. MarkLogic has chosen to emphasize collections as a powerful and high-performance mechanism for selecting sets of documents against which queries can be processed. See Collections in the *Search Developer's Guide* for more information about collections.

**collection - protected**

*Database*. Protected collections enable you to control who can add documents to a collection. A protected collection does not affect access to documents in the collection. Use document permissions to control document access. You can convert a protected collection to an unprotected collection, and vice versa.

**collection - unprotected**

*Database*. An unprotected collection is created implicitly when inserting or updating a document and specifying a collection URI that has not previously been used. An unprotected collection is not stored in the security database. You can convert a protected collection to an unprotected collection, and vice versa.

**collection lexicon**

*MarkLogic*. A lexicon that contains all the collection URIs in a database. See Browsing With Lexicons in the *Search Developer's Guide*.

**column**

*MarkLogic*. A SQL view has a name and a range index reference that identifies a particular document element or attribute. The range index for each column must be created before creating the view.

**combined query**

Search. A query format that enables you to combine different query types and a set of query options into a single bundle. A combined query can include any combination of a string, structured, or cts:query, a QBE, and query options. Combined query is only supported by the REST, Java, and Node.js Client APIs. See Specifying Dynamic Query

Options with Combined Query in the *REST Application Developer's Guide*, or the appropriate client application language developer's guide.

**commit**

*Database*. To end a transaction and make the changes made by the transaction visible in the database. Single-statement transactions are automatically committed upon successful completion of the statement. Multi-statement transactions are explicitly committed, but the commit only occurs if and when the calling statement successfully completes.

**communication between clusters**

*MarkLogic*. Database replication uses inter-cluster communication. Communication between clusters uses the XDQP protocol. Before you can configure database replication, each cluster in the replication scheme must be aware of the configuration of the other clusters. This is accomplished by coupling the local cluster to the foreign cluster.

**compartments**

*Security*. Compartment security allows you to specify more complex security rules on documents. A compartment is a name associated with a role. You specify that a role is part of a compartment by adding the compartment name to each role in the compartment. When a role is compartmented, the compartment name is used as an additional check when determining a user's authority to access or create documents in a database.

**constraint**

*MarkLogic*. A constraint is a name/value pair used to filter search results. A constraint can be used in a search UI to create the facets as part of faceted navigation where every facet is a constraint on the search results. For a constraint to also function as a facet, you have to be able to retrieve all its values. If you search docs.marklogic.com for "xquery", the results will break-down by category of documents that mention XQuery. The categories of documents are the constraints on the search results.

**configuration management**

The MarkLogic Server Configuration Manager allows you to export and import the configuration settings for MarkLogic Server resources. A resource is a MarkLogic Server object, such as a database, forest, App Server, group or host.

**content source**

*MarkLogic*. The database and App Server combination against which your query executes. Query Console automatically builds a list of available content sources for your MarkLogic Server and makes it available in the Content Source list.

**coordinate system**

*Geospatial*. A geospatial coordinate system is a set of mappings that map places on Earth to a set of numbers. The vertical axis is represented by a latitude coordinate, and the horizontal axis is represented by a longitude coordinate, and together they make up a coordinate system that is used to map places on the Earth. For more details, see Understanding Geodetic Coordinates in the *Search Developer's Guide*.

**CPF** (Content Processing Framework)

*Content Processing Framework*. The Content Processing Framework in MarkLogic Server supports multi-step processing as part of content processing applications. Each step in the process performs a particular task or set of tasks. Content processing is a way to programmatically add value to your content. See Overview of the Content Processing Framework in the *Content Processing Framework Guide*. See also pipeline, domain, and trigger.

**CRUD**

*Database*. Create, Read, Update, and Delete: the four basic attributes of persistent storage.

**curated data**

*MarkLogic*. XX

**CTS**

*MarkLogic*. Accronym for Core Text Services. Used in the core search functions.

**CURIE** (Compact URI Expression)

*Semantics*. A shortened version of a URI (Uniform Resource Identifier) signifying a specific resource.

**D**

**data flow management**

Two core concepts of a MarkLogic Data Hub are the ingest data flows and curation data flows. When using MarkLogic Data Hub Service, the repositories, tasks, and plugins are readily available to start creating these data flows. The Data Hub Framework empowers Architects and Developers to leverage the power of MarkLogic to create data flows from a multitude of source systems.

**data harmonization**

Data from different sources should not get discarded. Instead, it gets harmonized so it's consistently queryable. MarkLogic Data Hub does this by leveraging the flexibility of the document model, which allows for an iterative approach.

**data enrichment**

MarkLogic's built-in RDF triple store allows you to enhance your document data with semantic metadata. You can also do Ontology Driven Entity Extraction to make your data more meaningful and discoverable.

**data governance**

When the data is sourced from many different silos, each with their own notions of data governance, you may use MarkLogic to implement, enforce, and verify policies around data security, sharing, provenance, and, retention. See Data Governance in an Unpredictable World.

**database**

*MarkLogic*. MarkLogic databases are made up of one or more forests, and forests are made up of one or more stands. See forest and stand.

**database replication**

*Database*. See Replicate.

**Data Hub**

*MarkLogic*. A Data Hub is a consolidated repository of data that breaks down data silos.

**Data Hub Framework** (DHF)

*MarkLogic*. The Data Hub Framework (or DHF) is a set of tools and libraries that help you build an operational data hub on top of MarkLogic.

**DBPedia**

*Semantics*. A project aiming to extract structured content from the information created as part of the Wikipedia project. Information in Wikipedia is represented as RDF triples, making it an excellent source of facts as triples.

**dc (Dublin Core)**

*Semantics*. A metadata vocabulary containing metadata terms maintained by the Dublin Core Metadata Initiative (DCMI). The current set of the Dublin Core vocabulary includes terms of the Dublin Core Metadata Element Set, along with elements, classes, and DCMI type vocabularies. The Dublin Core vocabulary existed before the Dublin Core Ontology, which is a lightweight RDFS (Resource Description Framework Schema) vocabulary for describing generic metadata.

**declarative rewriter**

*HTTP server*. An implementation of a "rewriter", specified in a declarative syntax (a document) that doesn't require evaluation by a general purpose language. This is an XML document where the implementation is entirely in C++ and requires no evaluation of either XQuery or JavaScript.

**deep time**

*General*. The total time spent evaluating an expression, including time spent evaluating any expressions contained within the specific expression. See elapsed time and shallow time.

**default graph**

*Semantics*. The default graph is the RDF graph where triples are inserted if you don't specify a named graph. In MarkLogic, the default graph has the IRI `http://marklogic.com/semantics#default-graph`. You can specify a different collection during the load process and load triples into a named graph. See graph.

**default partition**

*Database*. A partition with no defined range. Documents that have no partition key, or a partition key value that does not fall into any of the partition ranges, are stored in the default partition.

**directories**

*Database*. Documents are located in directories in a MarkLogic database. Directories are hierarchical in structure (like a filesystem directory structure) and are described by a URI path. Directories are required for WebDAV clients to see documents.

**Disaster Recovery** (DR)

*Database*. Disaster recovery is the ability to rebuild part or all of your data should something happen to your cluster. See Database Replication for Disaster Recovery in the *Database Replication Guide*. See also High Availability (HA).

**distance**

*Geospatial*. The distance between two geospatial objects refers to the geographical closeness of those geospatial objects.

**Distinguished Name (DN)**

*MarkLogic*. A sequence of Relative Distinguished Names (RDNs), which are attributes with associated values expressed by the form `attribute=value`. Each RDN is separated by a comma in a DN.

**DLS** (Document Library Services)

*MarkLogic*. Library Services enable you to create and maintain versions of managed documents in MarkLogic Server. See Understanding Library Services in the *Application Developer's Guide*.

**d-node**

*MarkLogic*. A MarkLogic data (processing) node. See e-node.

**document**

*MarkLogic*. A document is a physical data model in the database. It can be a hierarchy of different types of data organized together, commonly used to represent entities. Document, triple, and lexicon are examples of physical data models (documents) used to represent entities.

**document format**

*MarkLogic*. Refers to how a document is stored in MarkLogic databases; it can be encoded in XML, binary, JSON, or text format.

**documents are like rows**

*Database*. When modeling data for MarkLogic, think of documents more like rows than tables. In other words, if you have a thousand items, model them as a thousand separate documents not as a single document holding a thousand child elements.

Of course MarkLogic documents can be more complex than simple relational rows because XML and JSON are more expressive data formats. One document can often describe an entity (a manifest, a legal contract, an email) completely.

**domain**

*Content Processing Framework*. A domain defines the scope of documents to process. Domains are used to organize and demarcate different sets content in order to process groups of documents in different ways. See Flexible Replication and Understanding and Using Domains in the *Content Processing Framework Guide*.

**dynamic content**

*MarkLogic*. Dynamic content is content generated by your application, such as results returned by XQuery modules.

**E**

**EBS** (Amazon Elastic Block Store)

*Cloud*. See Elastic Block Store (EBS - Amazon Elastic Block Store).

**EC2 Compute Unit (ECU)**

*Cloud*. Provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor.

**elapsed time**

*General*. Both shallow time and deep time are expressed in elapsed wall clock time.

**Elastic Block Store** (EBS - Amazon Elastic Block Store)

*Cloud*. A type of storage designed specifically for Amazon Elastic Compute Cloud (EC2) instances. Amazon EBS allows you to create volumes that can be mounted as devices by Amazon EC2 instances.

**Elastic Compute Cloud** (EC2)

*Cloud*. A commercial web service that enables you to launch and manage server instances and host applications in Amazon's data centers using APIs or available tools and utilities. The Amazon EC2 website is available at: http://aws.amazon.com/ec2/.

**Elastic Load Balancer** (ELB)

*Cloud*. A service that automatically distributes and balances application traffic among multiple EC2 instances. For details, see http://docs.aws.amazon.com/gettingstarted/latest/wah/getting-started-create-lb.html.

**Element Level Security**

*MarkLogic*. Element level security expands role-base authorization to the element level in stored documents using permissions on a protected path. Element level security applies to both XML elements and JSON properties. See Understanding Element Level Security in the *Security Guide*.

**embedded triples**

*Semantics*. See unmanaged triples.

**encryption at rest**

*Security*. Encryption at rest protects your data on disk - at rest as opposed to when that data is being used in a process or in motion.

**endpoint**

*HTTP server*. An XQuery module on MarkLogic Server that is invoked by and responds to an HTTP request for monitoring information.

**e-node**

*MarkLogic*. A MarkLogic evaluator node. See d-node.

**entities**

*MarkLogic*. XML or JSON representations of high-level business objects in your enterprise. Examples of business objects are employee, product, purchase order, and department. See envelope pattern.

**entity modeling**

*MarkLogic*. A model defines logical entity types, their properties, and the relationships between entities. See Why Use Entity Modeling? in *Entity Services Developer's Guide*.

**Entity Services**

*MarkLogic*. An out-of-the-box API and a set of conventions you can use within MarkLogic to quickly set up an application based on entity modeling.

**Enterprise Service Bus** (ESB)

*General*. A software architecture model used for designing and implementing communication between mutually interacting software applications in a service-oriented architecture (SOA). As a software architectural model for distributed computing it is a

specialty variant of the more general client server model and promotes agility and flexibility with regards to communication between applications. Its primary use is in enterprise application integration (EAI) of heterogeneous and complex landscapes. (definition fromWikipedia)

**envelope**

*MarkLogic*. A set of metadata wrapped around the original entity/data, including harmonized parts of the entity. See envelope pattern and Envelope Design Pattern (developer.marklogic.com)

**envelope pattern**

*MarkLogic*. An envelope pattern wraps your data in either XML or JSON metadata. With the envelope pattern, the core content and metadata for that content are kept separate. The envelope pattern is used for both the ingest and harmonize flows in Data Hub Framework (DHF). See "Envelope Pattern" in the Data Hub Framework documentation.

**execute privilege**

*Security*. An Execute Privilege provides the authority to perform a protected action. Examples of protected actions are the ability to execute a specific user-defined function, the ability to execute a built-in function (for example, `xdmp:document-insert`), and so on. For details on execute privileges, see Protecting XQuery and JavaScript Functions With Privileges in the *Security Guide*. See also privilege, amp, role, and URI privilege

**expression**

*XQuery*. The basic parse element of an XQuery program. Expressions can represent literal values, arithmetic operations, functions, function calls, and so on. Expressions can contain other expressions.

**extension**

*XQuery*. An user-defined XQuery module that implements additional resource services that are made available through the MarkLogic REST API. For details, see Extending the REST API in the *REST Application Developer's Guide*.

**External Authentication Configuration Object**

*Security*. An object that specifies which authentication protocol and authorization scheme to use, along with any other parameters necessary for LDAP authentication. After an external authentication configuration object is created, multiple App Servers can use the same configuration object.

**external binary document**

*MarkLogic*. A binary document that is not stored in a MarkLogic database and whose contents are not managed by the server. For details, see Choosing a Binary Format in the *Loading Content Into MarkLogic Server Guide*.

**F**

**facet**

*MarkLogic*. A constraint used for navigation on search results, providing a set of values that exist matching documents. Facets usually include a count of the resulting number of distinct values. Facets returned by a search include the counts and values needed to generate the user interface for the results. For example, a data set of articles could provide facets on author and publication date.

**faceted navigation**

*MarkLogic*. A type of navigation in the user interface that makes use of facets. With faceted navigation, a user can access focused search results by narrowing the search criteria through the use of facets.

**failover - local disk**

*MarkLogic*. Local-Disk Failover uses the intra-cluster forest replication capability introduced with MarkLogic 4.2. With forest replication you can have all writes to one forest be automatically replicated to another forest or set of forests, with each forest held on a different set of disks, generally cheap local disks, for redundancy.

**failover - shared disk**

*MarkLogic*. Shared-Disk Failover uses a clustered filesystem, such as Veritas or GFS. Every Data Manager stores its forest data on a SAN that's potentially accessible by other servers in the cluster. Should one D-node server fail, it will be removed from the cluster and another server in the cluster with access to the SAN will take over for each of its forests.

**fast failover**

*MarkLogic*. If one or more host forests are configured for local-disk or shared-disk failover, you now have the option to failover those forests when you shut down the host. A new option, Immediately fail over forests to replica hosts, has been added to the Host Shutdown confirmation page to enable you to fail over the forests to replica hosts

**field**

*MarkLogic*. A field is typically a small subset of the whole database, a portion of the content that you might want to query as a single unit, such as an abstract. Fields have their own set of indexes, independent of the database indexes. Because the field represents only a relatively small percentage of the content, the relative cost of the extra indexing is small. See Overview of Fields in the *Administrator's Guide*.

**filtered search**

Search. A filtered search is one that is configured to include a filtering step that checks whether each candidate result matches the search criteria. Candidates that do meet the criteria are discarded. A filtered search is accurate, but takes longer than an equivalent unfiltered search. For details, see Understanding the Search Process in the *Query Performance and Tuning Guide*.

**Flexible Replication**

*MarkLogic*. An implementation of replication based on MarkLogic Server CPF (Content Processing Framework). Flexible Replication is single-master, asynchronous, and provides a medium level of throughput and latency. See also QBFR (Query-Based Flexible Replication), domain, pipeline, and Flexible Replication in MarkLogic Server in the *Flexible Replication Guide*.

**flow**

*MarkLogic*. A series of actions that process the data. Flows are implemented using a chain of plugins that perform sequential or concurrent steps in the process. The two types of flows are input and harmonize. See envelope pattern, Data Hub Framework (DHF), and "Envelope Pattern - Flows and Plugins" in the Data Hub Framework documentation.

**flow tracing**

*MarkLogic*. The process that logs information about the flows as they run. Inputs to and outputs from every plugin of every flow are recorded into the JOBS database of the Data Hub Framework. See flow and "Flow Tracing" in the Data Hub Framework documentation.

**FLWOR statement**

*XQuery*. An XQuery programming expression standing for For, Let, Where, Order by, Return. FLWOR (pronounced 'flower') is loosely analogous to SQL's SELECT-FROM-WHERE and can be used to provide join-like functionality to XML documents.

**FName**

*XML*. An element identifier used to avoid naming collisions in a namespace.

**foaf** (friend of a friend)

*Semantics*. An ontology for describing people and social relationships. See http://www.w3.org/wiki/Good_Ontologies.

**Foreign Bind Port**

*MarkLogic*. The port used on each host to handle XDQP communication with foreign clusters.

**Foreign Cluster**

*MarkLogic*. A remote cluster of MarkLogic Server hosts.

**forest**

*MarkLogic*. A forest is a collection of documents that is implemented as a physical directory on disk. A group of 'trees' ('Forests, because we store trees.') A MarkLogic database is made up of 'forests'. See stand.

**forward-chaining inference**

*Semantics*. Forward chaining inference is inference done at index-time - during the ingestion/indexing of triples, creating new, inferred triples and inserting them into the database. Forward-chaining inference is less resource intensive at query-time, but more costly at ingest-time. See also backward-chaining inference and inference.

**fragment**

*MarkLogic*. Fragments are XML documents partitioned into smaller blocks of information for storage. You have the option of specifying how the XMP documents are partitioned. See fragments in the *Administrator's Guide* for more information.

**fully searchable expression**

*Search*. An XPath expression that contains no unsearchable path steps and whose last path step is searchable. Such expressions can be efficiently resolved out of the indexes. Only a fully searchable XPath expression can be used as the first argument of `cts:search`. For more details, see Fully Searchable Paths and cts:search Operations in the *Query Performance and Tuning Guide*.

## G

### graph

*Semantics*. In RDFS (Resource Description Framework Schema), a graph is a collection of triples. In a graph-based RDF model, nodes represent Subject and Object resources, with the Predicate providing the connection between those nodes. Graphs that are assigned a name are referred to as *named graphs*. See node.

### Graph Store

*Semantics*. A mutable repository of RDF graphs managed by one or more services. See the W3C specification for more information about Graph Store.

### graph permissions

*Semantics*. Permissions needed by a user in order to read or modify a graph.

### groups

*MarkLogic*. A set of similarly configured hosts within a cluster. See groups in the *Administrator's Guide*. See also host group.

## H

### Hadoop MapReduce

*Hadoop*. An Apache Software Foundation software framework for reliable, scalable, distributed parallel processing of large data sets across multiple hosts. The Hadoop core framework includes a shared file system (HDFS), a set of common utilities to support distributed processing, and an implementation of the MapReduce programming model. See Apache Hadoop MapReduce Concepts in the *MarkLogic Connector for Hadoop Developer's Guide*.

### harmonization

*MarkLogic*. The DHF process of creating a canonical model of your data using only the parts you need and leaving the rest of the data as-is.

### Harmonize flow

*MarkLogic*. The type of flow that creates a canonical model of your data using only the parts you need and leaving the rest of the data as-is. The harmonize flow is the most common type of flow in DHF and is typically run in batches. See flow and "Envelope Pattern - Harmonize Flows" in the Data Hub Framework documentation.

**hardware security module** (HSM)

*Security*. A hardware security module is a physical device used to safeguard and manage encryption keys. See Encryption at Rest in the *Security Guide* for more information.

**HDFS** (Hadoop Distributed File System)

*Hadoop*. The Hadoop filesystem, which can be used as an input source or an output destination in jobs. HDFS is the default source and destination for Hadoop MapReduce jobs.

**heartbeat**

*MarkLogic*. Each node in a cluster communicates with all of the other nodes in the cluster at periodic intervals. This periodic communication, known as a heartbeat, circulates key information about host status and availability between the nodes in a cluster. Through this mechanism, the cluster determines which nodes are available and communicates configuration changes with other nodes in the cluster. If a node goes down for some reason, it will stop sending heartbeats to the other nodes in the cluster.

**High Availability** (HA)

*Database*. High Availability. Local disk failover or shared disk failover. See High Availability of Data Nodes With Failover in the *Scalability, Availability, and Failover Guide*. See also Disaster Recovery (DR).

**High Performance**

*MarkLogic*. The server is designed to maximize speed and scale. Many MarkLogic applications compose advanced queries across terabytes of data that make up many millions of documents and return answers in less than a second.

**historical data**

*Database*. Data that is less critical for the lowest-latency queries than active data, but still requires online access for queries. Historical data is not typically updated.

**history**

*Query Console*. A record of previously executed versions of a query. Each time you execute a query in Query Console, its query text is saved in the history. Use the history to restore a query to a previous state.

**host**

*MarkLogic*. A host is an instance of MarkLogic Server. A host is not configured individually, but as part of a group. Forests are created on a host and added to a database to interact with HTTP, ODBC, and XDBC Servers running on the same or other hosts. Hosts can be added to a cluster. Each host in a cluster may be referred to as a node, and is usually configured for one of two types of operations; evaluation (e-node) or data management (d-node). See groups, forest, HTTP Server, ODBC Server, and XDBC Server.

**host group**

*MarkLogic*. A group of one or more MarkLogic Server objects. See Groups in the *Administrator's Guide*. See also groups.

**HTML** (Hypertext Markup Language)

*HTML*. A standardized markup language used for tagging text files to format the look and hyperlinks for web pages on the Web.

**HTTP** (Hypertext Transfer Protocol )

*Web*. An application protocol for distributed, collaborative, hypermedia information systems, the underpinning of data communication for the World Wide Web. Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text. HTTP is the protocol to exchange or transfer hypertext.

**HTTP Server**

*MarkLogic*. MarkLogic HTTP servers can access server-side XQuery, JavaScript, and REST (Representational State Transfer) programs via web applications. You can write web applications to connect sets of XML or JSON content, and the applications can return XHTML, XML, or JSON content to a browser or other HTTP-enabled client application. HTTP servers execute code, either from a specified location on the file system or from a Modules database

**I**

**Identity and Access Management (IAM)**

*Cloud*. AWS Identity and Access Management (IAM) is a web service that enables you to manage users and user permissions in AWS. With IAM, you can centrally manage users, security credentials such as access keys, and permissions that control access to AWS resources. For details, see https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html.

**indexes for text and structure**

*Database*. MarkLogic Server search supports a wide range of full-text features. These features include phrase search, boolean search, proximity, stemming, tokenization, decompounding, wildcarded searches, punctuation-sensitive search, diacritic-sensitive/insensitive searches, case-sensitive/insensitive searches, spelling correction functions, thesaurus functions, geospatial searches, advanced language and collation support, document quality settings, numerous relevance algorithms, individual term weighting, topic clustering, faceted navigation, custom-indexed fields, and much more. These features are all designed to build off of each other and work together in an extensible and flexible way.

**inference**

*Semantics*. You can do inference manually as part of an inference query or automatically using an inference engine. Given a set of data as triples and a set of rules (such as a ruleset), you can derive or discover new relationships and new facts. Inference can be done at query time (for each query, look at the data and ontology and ruleset(s), and expand the query appropriately), or at index time (on ingestion/index create new inferred triples). Whether the new relationships are explicitly added to the set of data, are returned at query time, or just influence the results, is an implementation issue. See forward-chaining inference, backward-chaining inference, and inferencing.

**inference engine**

*Semantics*. Software that performs inference. An inference engine may also be referred to as a reasoning engine, a reasoner, a semantic reasoner, or a rules engine.

**inference query**

*Semantics*. An inference query is any query (SPARQL, XQuery, JavaScript) that is affected by inference (that is, affected by some ruleset).

**inference rule**

*Semantics*. A rule that defines a set of triples to be inferred by some set of asserted triples, ontology triples, triples, and rulesets. You must have a ruleset to do inferencing.

**inferencing**

*Semantics*. Inferencing is the process of discovering new facts and relationships in your data using triples, an ontology, and ruleset(s). Inferencing can be done manually as part of an inference query, or automatically using an inference engine. See inference.

**indexable path**

*MarkLogic*. A path expression, limited to the subset of XPath (XML Path language) that can be used to define a path range index. See also Path Field and Path-Based Range Index Configuration in the *XQuery and XSLT Reference Guide*.

**inferred triples**

*Semantics*. In a system that does forward-chaining inference (a system that materializes new triples inferred from existing triples), inferred triples are those triples created by that inferencing. See also asserted triples and backward-chaining inference.

**Ingestion**

*MarkLogic*. The Data Hub Framework (DHF) process that uses an input flow to pull documents into the Data Hub.

**Input flow**

*MarkLogic*. The type of flow that processes each incoming document before it is written into MarkLogic. Input flows are invoked by mlcp (MarkLogic content pump), the Java Client API, or the REST Client API. See envelope pattern and "Envelope Pattern - Input Flows" in the Data Hub Framework documentation.

**input query**

*MarkLogic*. When using MarkLogic Server as an input source, the query that generates input key-value pairs from the fragments/records in the input split.

**input source**

*MarkLogic*. A database, file system, or other system that provides input to a job. For example, a MarkLogic Server instance or HDFS (Hadoop Distributed File System) can be used as an input source.

**input split**

*MarkLogic*. The subset of the input data set assigned to a map task for processing. Split generation is controlled by the InputFormat subclass and configuration properties of a job.

**input split query**

*MarkLogic*. When using MarkLogic Server as an input source, the query that determines which content to include in each split. By default, the split query is built in. In advanced input mode, the split query is part of the job configuration.

**InputFormat**

*Hadoop*. The abstract superclass, org.apache.hadoop.mapreduce.InputFormat, of classes through which input splits and input key-value pairs are created for map tasks. The Apache Hadoop MapReduce API includes InputFormat subclasses for using HDFS as an input source. The MarkLogic Connector for Hadoop API provides InputFormat subclasses for using MarkLogic Server as an input source; see InputFormat Subclasses in the *MarkLogic Connector for Hadoop Developer's Guide*.

**Instance**

*Cloud*. The running system after an Amazon Machine Image (AMI) is launched. Instances remain running unless they fail or are terminated. When this happens, the data on the instance is no longer available. Once launched, an instance looks very much like a traditional host.

**Instance Store** (sometimes referred to as Ephemeral Storage)

*Cloud*. A fixed amount of storage space for an instance. An instance store is not designed to be a permanent storage solution. If an instance reboots, either intentionally or unintentionally, the data on the instance store will survive. If the underlying drive fails or the instance is terminated, the data will be lost.

**Instance Type**

*Cloud*. Defines the size of an Elastic Compute Cloud (EC2) instance. The MarkLogic Server instance types are shown in the table at the end of Step 5 in Creating a CloudFormation Stack using the AWS Console in the *MarkLogic Server on Amazon EC2 Guide*.

**interpretive rewriter**

*HTTP server*. An implementation of a "rewriter" that is specified in an interpreted language, like JavaScript or XQuery.

**IRI** (Internationalized Resource Identifier)

*Web*. An IRI is a compact string that is used for uniquely identifying resources in an RDF triple. IRIs may contain characters from the Universal Character Set (Unicode/ISO 10646), including Chinese or Japanese Kanji, Korean, Cyrillic characters, and so on. See also URI (Uniform Resource Identifier), URL (Uniform Resource Locator), and URN (Uniform Resource Name).

## J

### Java and JavaScript Client APIs

MarkLogic's Java and Node.js Client APIs are built on top of the MarkLogic REST API described in *MarkLogic REST API Reference*.

The Java API enables programmers to use MarkLogic without having to learn XQuery, and can easily take advantage of MarkLogic's advanced capabilities for persistence and search of unstructured documents.

MarkLogic also supports JavaScript middleware in the form of a Node.js process for network programming and server-side request/response processing.

### JavaScript

*Web*. JavaScript is an object-oriented computer programming language commonly used to create interactive effects within web browsers, whose implementations allow client-side scripts to interact with the user. It is also being used in server-side network programming (with Node.js), game development and the creation of desktop and mobile applications.

**JFS** (Journaling File System)

*MarkLogic*. A journaling file system is a file system that keeps track of the changes that will be made in a journal (usually a circular log in a dedicated area of the file system) before committing them to the main file system. In the event of a system crash or power failure, such file systems are quicker to bring back online and less likely to become corrupted.

### job

*Hadoop*. The top level unit of work for a MapReduce system. A job consists of an input data set, a MapReduce program, and configuration properties. Hadoop splits a job into map and reduce tasks which run across a Hadoop cluster. A Job Tracker node in the Hadoop cluster manages MapReduce job requests.

**JPA** (Java Persistence API)

*Java*. The Java Persistence API provides a POJO (Plain Old Java Object) persistence model for object-relational mapping.

**JSON** (JavaScript Object Notation)

*JavaScript*. A text-based open standard language for representing simple data structures and objects.

**JSON-LD** (JavaScript Object Notation-Linked Data)

*JavaScript*. A WC3 recommended format for transporting linked data an RDF graph, or an RDF Dataset in a JSON syntax, allowing data to be serialized in a way that is similar to traditional JSON.

## K

**Kerberos**

*Security*. A ticket-based authentication protocol for trusted hosts on untrusted networks. Kerberos provides users with encrypted tickets that can be used to request access to particular servers. Because Kerberos uses tickets, both the user and the server can verify each other's identity and user passwords do not have to pass through the network.

**key**

*Security*. A piece of information that determines the output of a cipher. SSL/TLS communications begin with a public/private key pair that allow the client and server to securely agree on a session key. The public/private key pair is also used to validate the identity of the server and can optionally be used to verify the identity of the client.

**KMS**

*Security*. AA keystore is a secure location where the actual encryption keys used to encrypt data are stored. The keystore for encryption at rest is a key management system (KMS). This keystore can be either the MarkLogic embedded PKCS #11 secured wallet, an external KMS that conforms to the KMIP-standard interface, or the native AWS KMS (Amazon Web Services Key Management System). The embedded keystore is installed by default when you install MarkLogic 9.0-x or later.

**KMS failover**

*Security*. MarkLogic encryption at rest enables enables KMS failover by allowing you to specify multiple hosts, multiple ports, and multiple KMIP credentials to connect to more than one KMIP server.

## L

**lag limit**

*Database*. A value that specifies the amount of time (in seconds) in which frames being written to the forest's journal can differ from the frames being streamed to the backup journal. For example, if the lag limit is set to 30 seconds, the archived journal can lag behind a maximum of 30 seconds worth of transactions compared to the active journal. If the lag limit is exceeded, transactions are halted until the backup journal has caught up. The decision to set your lag limit time is determined by your RPO (Recovery Point Objective).

**large binary document**

*MarkLogic*. A binary document whose contents are managed by MarkLogic Server and whose size exceeds the large size threshold. For details, see Choosing a Binary Format in the *Loading Content Into MarkLogic Server Guide*.

**Large Data Directory**

*MarkLogic*. The per-forest area where the contents of large binary documents are stored.

**large size threshold**

*MarkLogic*. A database configuration setting defining the upper bound on the size of small binary documents. Binary documents larger than the threshold are automatically classified as large binary documents.

**LDAP**

*Security*. See Lightweight Directory Access Protocol (LDAP).

**lexicon**

*MarkLogic*. A list of unique words or values, either throughout an entire database or within named elements, attributes, or fields. You can also define lexicons that allow quick access to the document and collection URIs in the database. Lexicons are usually backed by a range index. For more information see Querying Lexicons and Range Indexes and Range Indexes and Lexicons, also Browsing With Lexicons in the *Search Developer's Guide*.

**Lightweight Directory Access Protocol** (LDAP)

*Security*. An authentication protocol for accessing server resources over an internet or intranet network. An LDAP server provides a centralized user database where one password can be used to authenticate a user for access to multiple servers in the network. LDAP is supported on Active Directory on Windows Server 2008 and OpenLDAP 2.4 on Linux and other Unix platforms.

**linked data**

*Semantics*. Linked data describes a method of sharing and querying data on the Web, building on Web technologies such as HTTP (Hypertext Transfer Protocol ), RDFS (Resource Description Framework Schema) and IRI (Internationalized Resource Identifier)s, to share information in a way that can be read automatically by computers. Linked data is a term coined by Tim Berners-Lee for a web of data that can be processed by machines. See Linked Open Data and semantic web technologies. See also http://linkeddata.org/.

**Linked Open Data**

*Semantics*. Linked Open Data refers to linked data freely available on the Web like DBPedia. This data can be accessed via REST, a SPARQL endpoint, or downloaded for local use in different formats.

**load balancer**

*MarkLogic*. A load balancer usually spreads incoming requests across the E-nodes. An E-node processes the request and delegates out to the D-nodes for any subexpression of the request involving data retrieval or storage.

**Locking**

*Database*. An update request must use read/write locks to maintain system integrity while making changes. This lock behavior is implicit and not under the control of the user. Read-locks block for write-locks and write-locks block for both read- and write-locks. An update has to obtain a read-lock before reading a document and a write-lock before changing (adding, deleting, modifying) a document. Lock acquisition is ordered, first-come first-served, and locks are released automatically at the end of the update request

**LSQT** (Last Stable Query Time)

*Bitemporal*. An internal database timestamp used to track a point in time. A document with system start time before the LSQT point can be queried but not updated or ingested, and a document after this point can be updated/ingested but not queried. See bitemporal and temporal.

**Local Cluster**

*MarkLogic*. The cluster of MarkLogic Server hosts at your current location. See cluster.

**M**

**Manage App Server**

*MarkLogic*. The App Server on the Monitor Host that is configured to handle monitor requests. The Manage App Server is bound to port 8002 and is the App Server used by the Monitoring Dashboard.

**managed clusters**

*MarkLogic*. A MarkLogic feature that works with AWS features to automatically create and provision the necessary AWS resources and provide MarkLogic with the information needed to manage your cluster. For details see The Managed Cluster Feature in the *MarkLogic Server on Amazon EC2 Guide*.

**managed triples**

*MarkLogic*. Triples with a document root of `sem:triples` are "managed triples" in MarkLogic. Conceptually, managed triples are triples that are not embedded in any document. When you load and query triples as you would with a triple store - without reference to a document - you are generally working with managed triples. These triples can be updated using SPARQL Update. See unmanaged triples and triple.

**map task**

*Hadoop*. A task which contributes to the map step of a job. A map task transforms the data in an input split into a set of output key-value pairs which can be further processed by a reduce task. A map task has no dependence on or awareness of other map tasks in the same job, so all the map tasks can run in parallel.

**mapper**

*Hadoop*. Programatically, a subclass of `org.apache.hadoop.mapreduce.Mapper`. The mapper transforms map input key-value pairs into map output key-value pairs that can be consumed by reduce tasks. An input pair can map to zero, one, or many output pairs.

**MarketPlace**

*Cloud*. The Amazon Web Services (AWS) service for publishing pay-per-use and free (no extra charge) public AMI's on amazon. For details, see https://aws.amazon.com/marketplace.

**MarkLogic Data Hub Service**

MarkLogic Data Hub Service is a fully automated cloud service to integrate data from silos. Based on the MarkLogic Data Hub, the service enables agile teams to immediately start integrating and curating data for both operational and analytical use. Delivered as a cloud service, it provides on-demand capacity, auto-scaling, automated database operations, and proven enterprise data security. Unlike other cloud services, however, it's cost-effective and predictable even as enterprise workloads fluctuate

**MarkLogic Server**

*MarkLogic*. An enterprise grade, scaleable operational database management system designed from the ground up to make massive quantities of heterogenous data easily accessible through search. MarkLogic Server runs on commodity hardware, supports ACID transactions, semantics, and Java, JavaScript, XQuery, and REST APIs. See Overview of MarkLogic Server in the*Concepts Guide*.

**Master**

*Flexible Replication.* The repository that gets updated by the applications. The master, in turn, replicates the updates to other repositories, each known as a Replica.

**Master Cluster**

*MarkLogic*. The cluster on which the replicated documents are updated by the applications; shorthand for the cluster that hosts a Master Database.

**Master Copy**

*MarkLogic*. The content being replicated. For any piece of replicated content there is a Master and at least one copy.

**Master Database**

*MarkLogic*. The database being replicated. In any database replication scheme there is a Master Database and at least one Replica Database.

**Master Forest**

*MarkLogic*. The forest being replicated. In any database replication scheme there is a Master Forest and at least one Replica Forest.

**Metadata Database**

*MarkLogic*. The database that stores and indexes all of the configuration data required to manage a cluster of one or more MarkLogic Servers. For Amazon Web Services (AWS), the SimpleDB service is used to implement the Metadata Database. For details, see http://docs.aws.amazon.com/AmazonSimpleDB/latest/GettingStartedGuide/Welcome.html.

**MIME type**

*Web*. (Multipurpose Internet Mail Extension) A MIME type is a standard identifier used by email clients, web browsers, search engines, and other protocols to indicate the type of data that a file contains. In an HTTP request, a MIME type is referred to as Content-type. Due to their expanded use by other media, MIME types are referred to as Internet Media types.

**mlcp** (MarkLogic content pump)

*MarkLogic*. A command line tool for getting data into and out of a MarkLogic Server database.

**MLJS** (MarkLogic JavaScript API)

*MarkLogic*. A client-side API for using JavaScript with MarkLogic Server.

**module**

*MarkLogic*. The are two types of modules in MarkLogic; main modules and library modules. A main module can be executed as an XQuery or JavaScript program, and must include a query body consisting of an XQuery or JavaScript expression (which in turn can contain other expressions, and so on). A library module has a namespace and is used to define functions. Library modules cannot be evaluated directly; they are imported, either from other library modules or from main modules with an import statement. See built-in.

**Monitor Application**

*MarkLogic*. This can be any application that requests and makes use of monitoring data, such as a Web browser, a plugin for an existing monitoring tool, or the Monitoring Dashboad described in Using the MarkLogic Server Monitoring Dashboard in *Monitoring MarkLogic Guide*.

**Monitor Content**

*MarkLogic*. The XML (Extended Markup Language), HTML (Hypertext Markup Language), or JSON (JavaScript Object Notation) structure that represents the data returned by the Management API.

**Monitor Host**

*MarkLogic*. The host in a MarkLogic Server cluster that uses the Monitor API to respond to requests for monitoring content from the monitor application and returns monitoring information for objects in the cluster.

**Monitoring Sample**

*MarkLogic*. A bit of information captured during a refresh interval on a graph. For example, one of the candlesticks captured in the Query Execution graph is a single sample.

**Monitoring Session**

*MarkLogic*. The timeframe since the dashboard page was last refreshed. For example, if you navigate from the Query Execution page to the Rates and Loads page, you have ended the Query Execution session and started the Rates and Loads session.

**multi-model database**

*MarkLogic*. A multi-model database is designed to support multiple data models in their natural form against a single, integrated backend, and allow data access with a single API to query all the data in their native query language regardless of the data model. Document, graph, relational, and key-value models are examples of data models that may be supported by a multi-model database.

**multi-statement transaction**

*Database*. A transaction created in query or update transaction mode, consisting of one or more statements which commit or rollback together. Changes made by one statement in the transaction are visible to subsequent statements in the same transaction prior to commit. Multi-statement transactions must be explicitly committed by calling `xdmp:commit`.

**multi-tenancy**

*Cloud*. A mode of software operation where multiple independent instances of one or multiple applications operate in a shared environment. The instances (tenants) are logically isolated, but physically integrated. Multi-tenancy is an important feature of cloud computing.

**MVCC** (Multi Version Concurrency Control)

*Database*. A concurrency control method commonly used by database management systems to provide concurrent access to the database, and used in programming languages to implement transactional memory.

**N**

**N3** (Notation 3)

*Semantics*. A serialized format for expressing data in the RDFS (Resource Description Framework Schema) data model with the syntax similar to SPARQL. RDF represents information as a triple consisting of Subject, Predicate, and Object. RDF/XML and Turtle (Terse RDF Triple Language) are other serializations of RDF.

**namespace**

*XML*. In XML (Extended Markup Language), XQuery, and SPARQL, element and attribute nodes are always in a namespace, even if that namespace is the empty namespace (sometimes called *no namespace*). Each namespace has a IRI (Internationalized Resource Identifier) associated with it. An IRI is essentially a unique string that identifies the namespace.

**nascent fragment**

A fragment document that has not yet been commited to a database.

**native plugin**

*MarkLogic*. A native plugin is a plugin that contains user-defined native C++ code, compiled into a dynamically linked library, and packaged and deployed according to the native plugin requires. The library must implement one or more classes conforming to the UDF (User Defined Function) interfaces. For details, see Using Native Plugins in the *Application Developer's Guide*.

**node**

*MarkLogic*. A host in a MarkLogic cluster is sometimes referred to as a node. MarkLogic Server has evaluator nodes (e-node) and data nodes (d-node). Each node in the cluster has its own copy of all of the configuration information for the entire cluster.

**node**

*Semantics*. A Subject or Object in an RDF graph. See RDFS (Resource Description Framework Schema).

**node**

*XML*. A node is part of a larger XML data structure representing a document's contents. Nodes can be document nodes, element nodes, comment nodes, and can contain child nodes, attributes, as well as data.

**Node.js**

*JavaScript*. ("Node") A low-level, open-source scripting environment that allows developers to build network and I/O services with JavaScript.

**non-temporal**

*MarkLogic*. The latest data or information, without any history. See bitemporal.

**NPM** (Node Package Manager)

*JavaScript*. The public repository of libraries for the Node.js ecosystem.

## O

### object

*MarkLogic*. A component of interest in MarkLogic Server, such as a cluster, host, App Server, or database.

### Object

*Semantics*. An RDF resource, which in turn may be the Subject in a triple. An object may be a typed literal. See also Predicate, RDFS (Resource Description Framework Schema), and RDF Datatypes in the *Semantics Developer's Guide*.

### ODBC Server

*MarkLogic*. A host that is an instance of MarkLogic Server designed for connecting to an ODBC content source and supporting SQL queries. See host.

### offline partition or forest

*MarkLogic*. A partition or forest that is not available for queries, but is tracked by the cluster. The benefit of taking data offline is to spare the RAM, CPU, and network resources for the online data.

### online partition

*MarkLogic*. A partition or forest that is available for queries and updates

### ontology

*Semantics*. An ontology provides a semantic model of (a portion of) the world. An ontology is generally expressed as triples (like "a Henley is a shirt") in an ontology language such as RDFS (Resource Description Framework Schema) or OWL (Web Ontology Language). The triples that make up an ontology are ontology triples. Ontologies can include vocabularies, a controlled set of terms used to define concepts and relationships.

### ontology triples

*Semantics*. Triples that make up an ontology are ontology triples. See asserted triples and vocabularies.

### operational data hub

*MarkLogic*. An operational data hub pattern is a particular way of building a Data Hub, which allows for faster, more agile data integration into a single Hub. Because it is operational, an operational data hub allows real-time, interactive access to data. This means that shared services can be run off the Data Hub in addition to analytic reports.

### Ops Director

Ops Director provides you with system monitoring, management, analysis, and support capabilities, as well as license auditing and role-based access control (RBAC) management. The Ops Director web-based interface offers graphical and tabular data representation intended to highlight irregular performance or alerts on a given cluster, host, database, or application server.

### Ops Director Resources

The following resources are created on the host where your Ops Director application runs.

**Table 1: Ops Director Resource Definitions**

| Resource Type | Resource Name and Function |
|---|---|
| *Certificate Authority* | The Certificate Authority (`MarkLogic Ops Director` in this example) is used to generate the secure credentials required for the Ops Director application and Managed Clusters to securely communicate. |
| *Certificate Templates* | `OpsDirector-SSL-Template` for SSL on the Application Cluster.<br><br>`OpsDirector Template` for SSL, if also configured as a Managed Cluster. |
| *Roles* | `opsdir-guest` role is used for a default user of the Ops Director application.<br><br>`opsdir-user` role is required for any user to be able to access the Ops Director application.<br><br>`opsdir-license-admin` role grants a user rights to manage licenses for the Ops Director application.<br><br>`opsdir-admin` role grants a user administrative rights to the Ops Director application. |
| *Execute Privileges* | `opsdir-admin` protects administrative functions.<br><br>`opsdir-license-admin` is required to access license information.<br><br>`opsdir-user` is required to access the browser application. |
| *Database and Forests* | The `OpsDirector` database and forests hold the Ops Director configuration data. |
| *App Servers* | `OpsDirectorApplication` (port 8008) provides the Ops Director browser application and consumes the data stored in the `OpsDirector` database.<br><br>`OpsDirectorSystem` (port 8009) receives data transmitted from Managed Clusters and stores it in the `OpsDirector` database. |

**Table 1: Ops Director Resource Definitions**

| Resource Type | Resource Name and Function |
|---|---|
| *External Security Configuration* | `OpsDirector-External-Security` for authentication of clients.<br><br>`OpsDirectorSystem` for secure communication with `OpsDirectorSystem` server. |
| *Secure Credentials* | `OpsDirector-Credential` to sign `OpsDirector-SSL-Template`.<br><br>`MarkLogic-OpsDirector` for accessing `OpsDirectorSystem` server. |

**Optic API**

The MarkLogic Optic API makes it possible to perform relational operations on indexed values and documents. The Optic API is not a single API, but rather a set of APIs exposed within the XQuery, JavaScript, and Java languages.

**ordering clause**

*MarkLogic*. A part of an ordering specification, as part of the `cts:search()` function, that deals with a single ordering key

**ordering specification**

*MarkLogic*. A way to specify, in a parameter, the ordering of a set of search results from a `cts:search()` function.

**ORM** (Object Relational Mapping)

*Database*. A technique for converting data between incompatible type systems in object-oriented languages, creating a "virtual object database" that can be used from within the language.

**OutputFormat**

*Hadoop*. The abstract superclass, `org.apache.hadoop.mapreduce.OutputFormat`, of classes that store output key-value pairs during the reduce phase. The Apache Hadoop MapReduce API includes OutputFormat subclasses for using HDFS for output. The MarkLogic Connector for Hadoop API provides OutputFormat subclasses for using a MarkLogic Server database as an output destination. See OutputFormat Subclasses in the *MarkLogic Connector for Hadoop Developer's Guide*.

**output options**

With MarkLogic you can generate output in many different formats, such as: XML, HTML, RSS, PDF, Microsoft Office formats, Adobe InDesign, QuarkXPress, and JSON.

**OWL** (Web Ontology Language)

*Semantics*. OWL and OWL2 are knowledge representation languages or ontology languages for authoring ontologies for semantic web technologies. See the W3C specification for OWL (http://www.w3.org/TR/owl-ref/) and OWL 2 (http://www.w3.org/TR/owl2-overview/).

## P

**parameter**

*MarkLogic*. An addition to the end of a resource address to filter and/or format the view returned from MarkLogic Server. Parameters are expressed as query strings in the URL and are described in Specifying Parameters in a Resource Address in the *Monitoring MarkLogic Guide*.

**partially searchable expression**

*Search*. An XPath expression whose first step is searchable, but that contains at least one step that is not fully searchable. A partially searchable expression cannot be fully resolved out of the indexes and cannot be used as the first parameter of cts:search. For more details, see Fully Searchable Paths and cts:search Operations in the *Query Performance and Tuning Guide*.

**partition**

*MarkLogic*. A set of forests sharing the same name prefix and same partition range definition. Typically forests in a partition share the same type of storage and configuration such as updates allowed, availability, and enabled status. Partitions are based on forest naming conventions. A forest's partition name prefix and the rest of the forest name are separated by a dash (-). For example, a forest named 2011-0001 belongs to the 2011 partition.

**partition key**

*MarkLogic*. Defines an element or attribute on which a range index, collection lexicon, or field is set and defines the context for the range set on the partitions in the database. The partition key is a database-level setting.

**partition range**

*MarkLogic*. Defines a range of values for a partition. Documents with a partition key value that fall within the range specified for a partition are stored in that partition.

**path**

*Semantics*. See property path.

**peering**

*Cloud*. Peering is a networking connection between two virtual private clouds that enables you to route traffic between them. For details, see https://docs.aws.amazon.com/vpc/latest/peering/what-is-vpc-peering.html.

**PEP** (Policy Enforcement Point)

*Security*. A Policy Enforcement Point (PEP) receives a directive from the Policy Decision Point (PDP) on what must be carried out before or after an access is approved. If the PEP is unable to comply with the directive, the approved access may or must not be realized and access denied.

**permission**

*MarkLogic*. Permissions are on documents. A permission provides a role with the capability to perform certain actions (read, insert, update, execute) on a document or a protected collection. Permissions consist of a role and capabilities. Permissions are assigned to documents and collections. For details on permissions, see Document Permissions in the *Security Guide*. See privilege.

**pipeline**

*Content Processing Framework*. A pipeline defines document states as a document moves through stages of content processing. You attach pipelines to domains, and the domains determine the documents on which a pipeline acts. In addition to defining document states, a pipeline specifies actions that occur under certain conditions. A pipeline is a core component of the CPF (Content Processing Framework). For more information, see Understanding and Using Pipelines in the *Content Processing Framework Guide*.

**plugin**

*MarkLogic*. An XQuery module that provides extension capabilities using the Plugin framework described in System Plugin Framework in the *Application Developer's Guide*.

**point**

*Geospatial*. A geospatial point is the spot in the geospatial coordinate system representing the intersection of a given latitude and longitude. For more details, see Understanding Points in the *Search Developer's Guide*.

**Point-in-time Queries**

*Database*. Normally a query acquires its timestamp marker automatically based on the time the query started. However, it is also possible for a query to request data at a specific previous timestamp. MarkLogic calls this feature point-in-time queries.

**POJO** (Plain Old Java Object)

*Java*. Name used to emphasize that a given object is an ordinary Java Object, one that does not follow any of the major Java object models, conventions, or frameworks.

**Predicate**

*Semantics*. The relationship between the Subject and the Object of a triple. In graph terms, the predicate is also known as an arc or edge. See also RDFS (Resource Description Framework Schema).

**principal**

*Security*. A unique identity to which Kerberos can assign tickets. For example, in Kerberos, a user is a principal that consists of a user name and a server resource, described as a realm. Each user or service that participates in a Kerberos authentication realm must have a principal defined in the Kerberos database.

**privilege**

*Security*. Roles assigned to a user (or role) give privileges and privileges to the user (or role). There are different types of privileges. An execute privilege specifies a protected action. Only roles associated with the execute privilege can perform the protected action. A URI privilege specifies the right to create a document within a base URI. Only roles associated with the URI privilege can create documents within the base URI. See Role-Based Security Model (Authorization) in the *Security Guide* for more about roles and privileges.

**privilege - granular**

*Security*. Granular privileges extend MarkLogic Server security model by allowing finer granularity access control over configuration and various administration abilities. Granular privileges is a subtype of execute privileges type.

**Promise**

*JavaScript*. A JavaScript interface for interacting with the outcome of an asynchronous event. Every request to the Node.js Client API returns an object with a `result()` method that returns a Promise object. See Promise Result Handling Pattern in the *Node.js Application Developer's Guide*.

**process**

*MarkLogic*. A request or transaction in MarkLogic Server.

**profile report**

*MarkLogic*. An XML report containing statistics for all of the expressions evaluated while profiling was enabled. For a sample profile report, see Simple Enable and Disable XQuery Example in the *Query Performance and Tuning Guide*.

**profiler**

*MarkLogic*. An application which measures the performance characteristics of a running program (in the case MarkLogic Server, of an XQuery program).

**program**

*XQuery*. The expanded version of some XQuery code that is submitted to MarkLogic Server for evaluation, such as a query expression in a .xqy file or XQuery code submitted in an `xdmp:eval` statement. The program consists not only of the code in the calling module, but also any imported modules that are called from the calling module, and any modules they might call, and so on.

**property path**

*Semantics*. A possible route through a graph between two graph nodes, for example, "show me Bill, and all his friends, and all their friends." MarkLogic supports both enumerated paths and unenumerated paths, which are paths that use + or * or ? operators. See Property Path Expressions in the *Semantics Developer's Guide*. See node.

**protected collection**

*MarkLogic*. A collection configured such that only authorized users can associate documents with the collection. A protected collection does not control document access or prevent removal of documents from the collection by unauthorized users. For details, see Protected Collections in the *Administrator's Guide*.

**protected path**

*MarkLogic*. (Element Level Security) A path to an element in a document that has been configured with permissions in Element Level Security is called a *protected path*. Element level security applies to both XML elements and JSON properties. Permissions defined on the element level are honored by both search results and update built-ins. See query roleset and Understanding Element Level Security in the *Security Guide*.

**Provenance and Lineage**

*MarkLogic*. The Data Hub Framework (DHF) process that ensures that the data can be traced back to its origin and that the source data is preserved.

**proximity**

*MarkLogic*. The proximity of search results is how close the results are to each other in a document. Proximity can apply to any type of search terms, including geospatial search terms. For example, you might want to find a search term dog that occurs within 10 words of a point in a given zip code.

**Public Key Infrastructure** (PKI)

*Security*. A security scheme that includes a signature of a certificate authority.

## Q

**QBE** (Query By Example)

*MarkLogic*. QBE is a query whose structure closely models the structure of the documents you want to match. You can use a QBE to search XML and JSON documents with the REST and Java APIs.

**QBFR** (Query-Based Flexible Replication)

*MarkLogic*. Combining alerts (based on reverse queries that can trigger Flexible Replication of data) provides query-based flexible replication of your data. QBFR can be used in a pull or push scenario. See Configuring Alerting With Flexible Replication in the *Flexible Replication Guide*.

**QName** (Qualified name)

*XML*. A QName acts as a valid identifier for elements and attributes. A QName is composed of the namespace name and the local name. Together these uniquely define how the element or attribute is identified and are used to reference particular elements or attributes within XML documents. See also namespace.

**quad**

*Semantics*. A representation of a Subject, Predicate, Object, and an additional resource for the context of the triple. In MarkLogic, the fourth resource is taken to be a named graph.

**queries per second** (QPS)

*MarkLogic*. The number of query requests per second sent to the server.

**query**

*MarkLogic*. Any executable block of XQuery, SQL, or SPARQL. When you run a query in Query Console, you may view the results in your choice of formats.

**Query Console**

*MarkLogic*. Query Console is an interactive web-based query development tool for writing and executing ad-hoc queries in XQuery, JavaScript, SQL, SPARQL, and SPARQL Update. Query Console enables you to quickly test code snippets, debug problems, profile queries, and run administrative XQuery scripts. See Accessing Query Console in the *Query Console User Guide*.

**query roleset**

*Security*. An association between one or more roles and one or more indexes for the purpose of determining privileges for concealed elements or properties using Element Level Security. See also protected path and query rolesets.

**query rolesets**

*Security*. The combination of query rolesets that apply to a path (XML) or property (JSON) in a document. See query roleset.

**query statement**

*MarkLogic*. A statement that contains no update calls. Query statements have a read-consistent view of the database. Since a query statement does not change the state of the database, the server optimizes it to hold no locks or lightweight locks, depending on the type of the containing transaction.

**query transaction**

*MarkLogic*. A transaction which cannot perform any updates; a read-only transaction. A transaction consisting of a single query statement in auto transaction mode, or any transaction created in query transaction mode. Attempting to perform an update in a query transaction raises XDMP-UPDATEFUNCTIONFROMQUERY. Instead of acquiring locks, query transactions run at a particular system timestamp and have a read-consistent view of the database.

**R**

**range index**

*MarkLogic*. MarkLogic Server maintains a universal index for every database to rapidly search the text, structure, and combinations of the text and structure that are found within collections of XML and JSON documents. Queries against these documents may include search conditions based on inequalities (for example, price < 100.00 or date ? thisQtr) XML and JSON documents can incorporate numeric or date information. Specifying range indexes for these elements, attributes, and/or JSON properties will substantially accelerate the evaluation of these queries. See Range Indexes and Lexicons in *Administrator's Guide* for more information.

**RDF** (Resource Description Framework)

*Semantics*. RDF is an abstract data model used to represent facts and relationships, made up of Subject, Predicate, and Object as a triple. It can be serialized as Turtle (Terse RDF Triple Language), N3 (Notation 3), RDF/XML. RDF is a W3C specification with a defined vocabulary. See the W3C specification http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/.

**RDF graph**

*Semantics*. See graph.

**RDF triple**

*Semantics*. An RDF statement made up of a Subject, Predicate, and Object. Each triple represents a single fact or relationship.

**RDF Triple Store**

*Semantics*. A storage tool for the persistent storage, indexing, and query access to RDF graphs (triples). See triple store.

**RDFS** (Resource Description Framework Schema)

*Semantics*. A set of vocabularies, rules about structure and data modeling. In RDF, the RDFS specification lets you describe classes, properties, and metadata about those classes and properties. These rules enable you to infer new facts about your data from the schema. See ruleset.

**RDF/XML**

*Semantics*. A serialized format for expressing data in the RDFS (Resource Description Framework Schema) data model with the syntax similar to SPARQL. RDF represents information as a triple consisting of Subject, Predicate, and Object. N3 (Notation 3) and Turtle (Terse RDF Triple Language) are other serializations for RDF.

**readers/writers locks**

*Database*. A set of read and write locks that lock documents for reading and update at the time the documents are accessed. MarkLogic Server uses readers/writers locks during update statements. Because update transactions only obtain locks as needed, update statements always see the latest version of a document. The view is still consistent for any given document from the time the document is locked. Once a document is locked, any update statements in other transactions wait for the lock to be released before updating the document. For more details, see Update Transactions: Readers/Writers Locks in the *Application Developer's Guide*.

## Rebalancer

*Database.* MarkLogic includes a database rebalancing mechanism that enables it to evenly distribute content among the database forests.

A database rebalancer consists of two parts: an assignment policy for data insert and rebalancing and a rebalancer for data movement. The assignment policies are described in Rebalancer Document Assignment Policies in the Administrator's Guide.

For more information on database rebalancing, see the Database Rebalancing chapter in the Administrator's Guide.

## Redacting Document Content

Redaction is the process of eliminating or obscuring portions of a document as you read it from the database. For example, you can use redaction to eliminate or mask sensitive personal information such as credit card numbers, phone numbers, or email addresses from documents.

**Table 2: Common Redaction Terms**

| Term | Definition |
|---|---|
| *redaction* | The process of modifying a document to obscure or conceal sensitive information. You can redact XML and JSON documents. |
| *redaction rule* | A specification of what portion of a document to redact and what function to use to make the modification. Rules can be defined in XML or JSON. For details, see Defining Redaction Rules. |
| *rule document* | A document containing exactly one redaction rule. Rule documents must be installed in the schema database and be part of a collection before you can use them to redact content. For details, see Installing Redaction Rules. |
| *rule collection* | A database collection that only includes rule documents. A rule must be part of a collection before you can use it to redact documents. |
| *redaction function* | A function used to modify content during redaction. A redaction rule must include a redaction function specification. MarkLogic provides several built-in redaction functions. You can also create user-defined redaction functions. For details, see Built-in Redaction Function Reference and User-Defined Redaction Functions. |
| *source document* | A database document to which you apply one or more redaction rules. Redacting a document creates an in-memory copy. The source document is unmodified. |
| *masking* | A form of redaction in which the original value is replaced by a new value. The new value may be deterministic or random. |

**Table 2: Common Redaction Terms**

| Term | Definition |
|---|---|
| *deterministic masking* | A form of redaction in which the original value is replaced by a new value, and the same input always yields the same output. For an example, see mask-deterministic. |
| *random masking* | A form of redaction in which the original value is replaced by a new, random value. The same input does not result in the same output every time. For an example, see mask-random. |
| *dictionary-based masking* | A specially formatted collection of values that can be used as a source for dictionary-based masking. Redaction dictionaries must be installed in the schemas database. You can define a dictionary using XML or JSON. For details, see Defining a Redaction Dictionary. |
| *redaction dictionary* | A specially formatted collection of values that can be used as a source for dictionary-based masking. Redaction dictionaries must be installed in the schemas database. You can define a dictionary using XML or JSON. For details, see Defining a Redaction Dictionary. |
| *concealment* | A form of redaction in which the original value is completely hidden. The XML element or JSON property containing the redacted value is usually hidden as well, depending on the semantics of the redaction operation. For an example, see conceal. |

**reduce task**

*Hadoop*. A task that contributes to the reduce step of a job. A reduce task takes the results of the map tasks as input, produces a set of final result key-value pairs, and stores these results in a database or file system. A reduce task has no dependence on or awareness of other reduce tasks in the same job, so all the reduce tasks can run in parallel.

**reducer**

*Hadoop*. Programmatically, a subclass of org.apache.hadoop.mapreduce.Reducer. The reducer aggregates map output into final results during the reduce step of a job. The value portion of an input key-value pair for reduce is a list of all values sharing the same key. One input key-value pair can generate zero, one, or many output pairs.

**registered query**

*MarkLogic*. Complex `cts:queries` that are accessed repeatedly and used as unfiltered `cts:query` constructors, can be registered with MarkLogic Server for later use. When you register a `cts:query` expression, a pre-evaluated version of the expression is stored, making subsequent queries using the same expression faster. See Registering cts:query Expressions to Speed Search Performance in the *Search Developer's Guide* for more information.

**regex**

A generic term for a Regular Expression, a sequence of characters used to form a search pattern, used for string pattern matching.

**Replica**

*Flexible Replication*. A repository that receives replicated updates from the Master. See Flexible Replication.

**replica cluster**

*Flexible Replication*. Shorthand for the cluster that hosts a Replica Database. See Flexible Replication.

**Replica Database**

*Flexible Replication*. The database that receives replicated data from the Master Database. See Flexible Replication.

**Replica Forest**

*Flexible Replication*. The forest that receives replicated data from the Master Forest. See Flexible Replication.

**Replicate**

*Flexible Replication*. To create a copy of a document in another database and to keep that copy in sync (possibly with some time-lag/latency) with the original. See Flexible Replication.

**Replication Domain**

*Flexible Replication*. The specification of the set of documents to be replicated. This may be a collection or some other set definition. See Flexible Replication.

**Representation**

*MarkLogic*. A view of a resource in a particular format, such as XML, HTML, or JSON.

**request**

*MarkLogic*. Any invocation of a program, whether through an App Server, through a task server, through , or through any other means. In addition, certain client calls to App Servers (for example, loading an XML document through XCC, downloading an image through HTTP, or locking a document through WebDAV) are also requests.

**resource** (REST)

*MarkLogic*. An abstraction of a MarkLogic REST API service, as presented by the REST architecture.

**Resource Address**

*MarkLogic*. A URL that identifies a MarkLogic Server resource. Resource addresses are described in Resource Addresses in *Administrator's Guide* and Understanding REST Resources in *Monitoring MarkLogic Guide*.

**resource path**

*MarkLogic*. A URL sent to MarkLogic Server to return monitoring information for an object. The resource paths are described in Using the Management API in *Monitoring MarkLogic Guide*.

**REST** (Representational State Transfer)

*MarkLogic*. An architecture style that, in the context of the MarkLogic REST API or monitoring MarkLogic Server, describes the use of HTTP to make calls between a client application and MarkLogic Server to create, update, delete and query content and metadata in the database

**REST API instance**

*MarkLogic*. An instantiation of the MarkLogic REST API against which applications can make RESTful HTTP requests. An instance consists of an HTTP App Server, a URL rewriter, a content database, a modules database, and the modules that implement the API. For details, see Administering REST Client API Instances in the *REST Application Developer's Guide*.

**restricted XPath**

*MarkLogic*. An extended version of indexable path, with support for more constructs like functionCalls, and with some basic level of customization of the restrictions per feature. See also indexable path.

**reverse query**

*MarkLogic*. A reverse query is the opposite of a regular query; it returns all of the queries, that if they were run, would match a document. Reverse queries are use in creating alerts. See Overview of Alerting Applications in MarkLogic Server in the *Search Developer's Guide*.

**rewriter**

*HTTP server*. A method for declaring the rules by which an incoming HTTP request is "rewritten" to target a different path, environment, and/or capability than the default for the server. This is an XQuery module that interprets the URL of an incoming HTTP request and rewrites it to an internal URL that services the request.

**role**

*Security*. A role is a named entity that provides authorization privileges and permissions to other roles or to users. You can assign roles to other roles (which can in turn include assignments to other roles, and so on). Roles are the fundamental building blocks that you use to implement your security policies. See also privilege. For details on roles, see Role-Based Security Model in the *Security Guide*.

**rollback**

*Database*. Immediately terminate a transaction and discard all updates made by the transaction. All transactions are automatically rolled back on error. Multi-statement transactions may also be explicitly rolled back using `xdmp:rollback`, or implicitly rolled back due to timeout or reaching the end of the session without calling .

**RPO** (Recovery Point Objective)

*Database*. The amount of data you can afford to lose in the event of a disaster. A low RPO means that you will restore the most data (at the cost of performance) and a high RPO means you will potentially store less data, but with less impact to performance. See lag limit.

**ruleset**

*Semantics*. A set of inference rules. In MarkLogic, a ruleset may be built up by importing other rulesets. MarkLogic supports several rulesets that map to a common ontology language or subset (RDFS (Resource Description Framework Schema), OWL-Horst).

**S**

**S3** (Amazon Simple Storage Service)

*Cloud*. Amazon S3 is an online file storage web service offered by Amazon Web Services that provides storage through web services interfaces. See Amazon S3 for more information.

**schema**

*Database*. A representation of a SQL schema. A schema is implemented as an XML document in the schemas database and consists of a unique id, a name (which must also be

unique), and a collection of views. During SQL execution, the schema provides the naming context for its views, which enables you to have multiple views of the same name in different schemas. The default schema is called “main.”It is default is the sense that it is always implicitly available and first on the default schema search path for name resolution in SQL. Even though the “main” schema is a default, you must create this schema.

**search APIs**

*Database*. MarkLogic provides search features through a set of layered APIs. The core text search foundations in MarkLogic are the XQuery cts:* and JavaScript cts.* APIs, which are built-in functions that perform full-text search. The XQuery search:*, JavaScript jsearch.*, and REST APIs above this foundation provide a higher level of abstraction that enable rapid development of search applications. The higher-level APIs offer benefits such as the following:

Abstraction of queries from the constraints and indexes that support them.

Built in support for search result snippeting, highlighting, and performance analysis.

An extensible simple string query grammar.

Easy-to-use syntax for query composition.

Built in best practices that optimize performance.

You can use more than one of these APIs in an application. For example, a Java application can include an XQuery extension to perform custom search result transformations on the server. Similarly, an XQuery application can call both search:* and cts:* functions.

**search features**

*Database*. MarkLogic Server search supports a wide range of full-text features. These features include phrase search, boolean search, proximity, stemming, tokenization, decompounding, wildcarded searches, punctuation-sensitive search, diacritic-sensitive/insensitive searches, case-sensitive/insensitive searches, spelling correction functions, thesaurus functions, geospatial searches, advanced language and collation support, document quality settings, numerous relevance algorithms, individual term weighting, topic clustering, faceted navigation, custom-indexed fields, and much more. These features are all designed to build off of each other and work together in an extensible and flexible way.

**searchable expression**

*Search*. An XPath expression that is (a) Rooted in an `fn:doc`, `fn:collection`, or `xdmp:document-properties` call in XQuery, or in an `fn.doc`, `cts.doc`, `fn.collection`, or `xdmp.document-properties` call in Server-Side JavaScript, and (b) uses only forward axes such as “/” and “//”. For details, see Fully Searchable Paths and cts:search Operations in the *Query Performance and Tuning Guide*.

**semantic web technologies**

*Semantics*. A set of standards and best practices for sharing linked data for use by applications over the Web. This includes the "semantics" (context and meaning) of that data. Semantic web techonogies include the RDFS (Resource Description Framework Schema) standard, SPARQL, OWL (Web Ontology Language) and other ontology vocabularies. See also linked data.

**secure credentials**

*Security*. Secure credentials enable a security administrator to manage credentials, making them available to less privileged users for authentication to other systems without giving them access to the credentials themselves.

Secure credentials consist of a PEM encoded x509 certificate and private key and/or a username and password. Secure credentials are stored as secure documents in the Security database on MarkLogic Server, with passwords and private keys encrypted.

**sequence file**

*Hadoop*. A flat file of binary key-value pairs in one of the Apache Hadoop SequenceFile formats. The mlcp (MarkLogic content pump) tool only supports importing Text and BytesWritable values from a sequence file.

**server-side JavaScript**

*JavaScript*. MarkLogic provides a JavaScript environment that runs within an e-node.

**server-side XQuery and XSLT APIs**

*XQuery and XSLT*. MarkLogic includes support for XQuery 1.0 and XSLT 2.0. These are W3C-standard XML-centric languages designed for processing, querying, and transforming XML.

**service**

*MarkLogic*. Describes what to monitor and how to monitor one or more objects in a MarkLogic cluster. Services can define warning and critical thresholds for alerting and can monitor one or more objects in MarkLogic Server.

**service group**

*MarkLogic*. A group of one or more services.

**shallow time**

*General*. The time spent evaluating a specific expression, not including time spent evaluating any expressions contained within the specific expression. See elapsed time and deep time.

**shuffle**

*Hadoop*. The process of sorting all map output values with the same key into a single (key, value-list) reduce input key-value pair. The shuffle happens between map and reduce. Portions of the shuffle can be performed by map tasks and portions by reduce tasks.

**single-statement transaction**

*Database*. Any transaction created in auto transaction mode. Single-statement transactions always contain only one statment and are automatically committed on successful completion or rolled back on error.

**SKOS** (Simple Knowledge Organization System)

*Semantics*. A W3C recommendation designed for representation of thesauri, classification schemes, taxonomies, subject-heading systems, or any other type of structured controlled vocabulary. See also thesaural relationship.

**small binary document**

*MarkLogic*. A binary document stored in a MarkLogic database whose size does not exceed the large size threshold. For more information, see Choosing a Binary Format in the *Loading Content Into MarkLogic Server Guide*.

**smart mastering**

*Data Hub*. Leverage fuzzy logic and AI to match and merge data so you can master data quickly and automatically without having to buy a separate MDM tool.

**snippet**

*MarkLogic*. The result of a search function that returned a portion of the found documents.

**snippeting**

*MarkLogic*. A search function in which portions (snippets) of the found documents are returned in the search results.

**SOAP** (Simple Object Access Protocol)

*Web*. A Web Services protocol for exchanging structured information in computer networks. It relies on Application Layer protocols, most notably Hypertext Transfer Protocol (HTTP) or Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission.

**SPARQL**

*Semantics*. A recursive acronym for SPARQL Protocol and RDF Query Language (SPARQL), a query language designed for querying data in RDFS (Resource Description Framework Schema) format. MarkLogic supports SPARQL1.1.

**SPARQL endpoint**

*Semantics*. A resource that a SPARQL process can contact and use as a service. The endpoint accepts SPARQL queries and updates, and returns the results using SPARQL protocol.

**SPARQL protocol**

*Semantics*. A means of conveying SPARQL queries from query clients to query processors, consisting of an abstract interface with bindings to HTTP (Hypertext Transfer Protocol ) and SOAP (Simple Object Access Protocol).

**SPARQL Update**

*Semantics*. A language for making updates to triples in RDFS (Resource Description Framework Schema) format. MarkLogic suports SPARQL Update 1.1.

**split**

*Hadoop*. The unit of work for one thread in local mode or one MapReduce task in distributed mode.

**SQL support**

*SQL*. The SQL supported by the core SQL engine is SQL92, with the addition of SET, SHOW, and DESCRIBE statements. MarkLogic SQL enables you to connect Bussiness Intelligence (BI) tools, such as Tableau and Qlik.

**SSL** (Secure Sockets Layer)

*Security*. A transaction security standard that provides encrypted protection between browsers and App Servers. When SSL is enabled for an App Server, browsers communicate with the App Server by means of an HTTPS connection, which is HTTP

over an encrypted Secure Sockets Layer. HTTPS connections are widely used by banks and web vendors for secure transactions over the web.

**stand**

*MarkLogic*. Databases are made up of one or more forests, and forests are made up of one or more stands. See forest.

**statement**

*XQuery*. An XQuery main module, as defined by the W3C XQuery standard, to be evaluated by MarkLogic Server. A main module consists of an optional prolog and a complete XQuery expression. Statements are either query statements or update statements, determined statically through static analysis prior to beginning the statement evaluation.

**static content**

*MarkLogic*. Content stored in the modules database of the App Server. MarkLogic Server responds directly to HTTP range requests (partial GETs) of static content. See Downloading Binary Content With HTTP Range Requests in the *Application Developer's Guide*.

**status change**

*MarkLogic*. As part of content processing (create, update, and delete), CPF (Content Processing Framework) automatically handles the document status change events and sets a state (or cleans up in the case of delete) for the document. When you want the result of these changes to move a document in or out of document processing, it is known as a status change. The Status Change Handling pipeline performs these tasks automatically.

**stem**

*MarkLogic*. A word *stem* is the part of a word that is common to all of its inflected variants. For example, in English, "run" is the stem of "run", "runs", "ran", and "running". Words derived from the same meaning and part of speech have the same stem (for example, 'mouse' and 'mice'). Some words can have multiple stems (if the same word can be used as a different part of speech, or if there are two words with the same spelling), and if you use advanced stemming (which can find multiple stems for a word), then stemmed search will find all of the words having the same stem as any of the stems. See Understanding and Using Stemmed Searches in the *Search Developer's Guide*.

**stemmed search**

*MarkLogic*. A stemmed search for a term matches all the terms that have the same stem as the search term. MarkLogic Server can perform stemmed searches in a number of different

languages. See Understanding and Using Stemmed Searches in the *Search Developer's Guide*.

**stemming**

*Search*. When stemming is enabled, MarkLogic Server automatically searches for words that come from the same stem of the word specified in the query, not just the exact string specified in the query. The purpose of stemming is to increase the recall for a search. A stemmed search for a word finds the exact same terms as well as terms that derive from the same meaning and part of speech as the search term. See Stemming in MarkLogic Server in the *Search Developer's Guide*.

**string query**

*Search*. A simple search string constructed using either the default MarkLogic Server search grammar, or a user-defined grammar. For example, 'cat' and 'cat OR dog' are string queries. For details, see Creating a Query From Search Text With cts:parse or Searching Using String Queries in the *Search Developer's Guide*.

**structured query**

*Search*. The pre-parsed representation of a query, expressed as XML or JSON. Structured queries allow you to express complex queries very efficiently. For details, see Querying Documents and Metadata in the *REST Application Developer's Guide* and Searching Using Structured Queries in the *Search Developer's Guide*.

**sub-database**

*Database*. A database contained in a super-database

**Subject**

*Semantics*. A representation of a resource such as a person or an entity. See also Object and Predicate.

**super-database**

*Database*. A database containing other databases (sub-database) so that they can be queried as if they were a single logical database.

**synchronous**

*General*. Synchronous processing happens "at the same time". This form of input/output processing blocks I/O until the operation has finished. Synchronous processing does not permit other processing to continue before/until the transmission has finished. See asynchronous.

**system time**

*Bitemporal*. When the information (captured in a document) was stored in the database. System time may also be called transaction time. See bitemporal and valid time.

**system timestamp**

*MarkLogic*. A number maintained by MarkLogic Server that increases every time a change or a set of changes occurs in any of the databases in a system (including configuration changes from any host in a cluster). Each fragment stored in a database has system timestamps associated with it to determine the range of timestamps during which the fragment is valid.

**T**

**task**

*Hadoop*. An independent subcomponent of a job, performing either map or reduce processing. A task processes a subset of data on a single node of a Hadoop cluster.

**task server**

*MarkLogic*. The Task Server is an App Server that is used to process requests that have been spawned. Spawned requests can originate from the `xdmp:spawn` command or from a post-commit trigger action (for example, as the result of a document running through a pipeline in the Content Processing Framework).

**telemetry**

MarkLogic telemetry provides faster, more complete communication with MarkLogic Support to facilitate the resolution of issues. When telemetry is enabled, it collects, encrypts, and sends diagnostic and anonymized system-level information about a MarkLogic cluster to a secure MarkLogic destination.

**template driven extraction (TDE)**

Template Driven Extraction (TDE) enables you to define a relational lens over your document data, so you can query parts of your data using SQL or the Optic API. Templates let you specify which parts of documents make up rows in a view. You can also use templates to define a semantic lens, specifying which values from a document make up triples in the triple index.

**temporal**

*Bitemporal*. Temporal refers to bitemporal documents and collections. See bitemporal, non-temporal, and LSQT (Last Stable Query Time).

**term**

*MarkLogic*. A term is anything stored in the Universal Index; for example words, stems, two-word phrases, value of a declared field, and so on. What terms that you have in your Universal Index depends on your index settings.

**thesaural relationship**

*Semantics*. A conceptual hierarchy built by terms that are interlinked with a few very generic relationships. A thesaurus is a hierarchical representation of a set of terms related by broader term, narrower term, synonym, and so on. See SKOS (Simple Knowledge Organization System).

**tokenization**

*Search*. The process of splitting a run of text into parts (tokens), without overlap or gaps. Each token is classified as a word, whitespace, or punctuation. For details, see Tokenization and Stemming in the *Search Developer's Guide*.

**transaction**

*Database*. A set of one or more statements which either all fail or all succeed. A transaction is either an update transaction or a query (read-only) transaction, depending on the transaction mode and the kind of statements in the transaction. A transaction is either a single-statement transaction or a multi-statement transaction, depending on the transaction mode at the time it is created.

**transaction mode**

*Database*. Controls the type and the commit semantics of newly created transactions. Mode is one of auto, query, or update. In the default mode, auto, all transactions are single-statement transaction. In query or update mode, all transactions are multi-statement transaction. To learn more about transaction mode, see Transaction Type in the *Application Developer's Guide*.

**transaction-aware**

*Flexible Replication*. A configuration in which all updates that make up a transaction on the Master are applied as a single transaction on the Replica.

**trees**

*XML*. XML documents form a tree structure (a document tree), starting at the "root" (root element) with "branches" containing "leaves" (child elements and attributes). The last line of the document is a the closing tag for the root element. The root element is the parent of the all the other elements. So an XML document tree has a root, branches, and leaves.

**trigger**

*MarkLogic*. Triggers listen for certain events (document create, delete, update, or the database coming online) to occur, and then invoke an XQuery module to run after the event occurs. A trigger is used to as part of CPF (Content Processing Framework). See Overview of Triggers in the *Application Developer's Guide*.

**triple**

*Semantics*. A triple is an instance of a data model representing a fact or relationship, made up Subject, Predicate, Object. Triples can be serialized different formats like Turtle (Terse RDF Triple Language) ,N3 (Notation 3), or RDF/XML. See managed triples and unmanaged triples, RDFS (Resource Description Framework Schema) and graph. See also inference.

**triple index**

*Semantics*. An index that indexes triples ingested into MarkLogic to facilitate the execution of SPARQL queries. Triples can be standalone ("managed") or embedded in a document ("unmanaged").

**triple store**

*Semantics*. See RDF Triple Store.

**Turtle** (Terse RDF Triple Language)

*Semantics*. A format for expressing data in the RDFS (Resource Description Framework Schema) data model with the syntax similar to SPARQL. RDF represents information as a triple consisting of Subject, Predicate, and Object. RDF/XML and N3 (Notation 3) are other serializations of RDF.

**U**

**UDDI** (Universal Description, Discovery and Integration)

*Web*. UDDI is a platform-independent, XML-based registry to register and locate web service applications and businesses on the Internet. UDDI is an open industry initiative, for enabling businesses to publish service listings and discover each other.

**UDF** (User Defined Function)

*MarkLogic*. A C++ function, implemented as a native plugin, that can be used to customize specific MarkLogic operations, such as the computation of aggregates, the tokenization of document content and query text, and the stemming analysis of word tokens. Aggregated user-defined functions or UDFs are functions that analyze values in lexicons and range indexes. Aggregate functions are best used for analyses that produce a small number of results, rather than analyses that produce results in proportion to the

number of range index values or the number of documents processed. For more information, see Aggregate User-Defined Functions in the *Application Developer's Guide*.

**unfiltered search**

*Search*. An unfiltered search is one that is configured to determine results directly from the indexes, without a filtering step to validate whether each candidate result actually meets the search criteria. Unfiltered searches are fast, but can include false positives in the results. For details, see Understanding Unfiltered Searches in the *Query Performance and Tuning Guide*. See also filtered search.

**Universal Index**

*MarkLogic*. One index maintained by MarkLogic Server for every database in order to rapidly search the text, structure, and combinations of the text and structure that are found within collections of XML documents. MarkLogic also has a triple index, collection lexicon (an index), security indexes, and others. See term.

**unmanaged triples**

*MarkLogic*. Triples that are embedded in a document and have an element root of `sem:triple` are "unmanaged" triples. You don't need the outer element of `sem:triples` for an unmanaged triple, even if you have more than one `sem:triple`, but you do need the Subject, Predicate, and Object to be in `sem:triple` elements. Unmanaged triples may also be called embedded triples. See also triple and managed triples.

**update statement**

*Database*. A statement with the potential to perform updates (that is, it contains one or more update calls). A statement may be categorized as an update statement whether or not the statement performs an update at runtime. Update statements run with readers/writers locks, obtaining locks as needed for documents accessed in the statement.

**update transaction**

*Database*. A transaction performs updates (make changes to the database) like a transaction consisting of a single update statement in auto transaction mode, or any transaction created under update transaction mode. Update transactions run with readers/writers locks, obtaining locks as needed for documents accessed in the transaction.

**unsearchable**

*Search*. An XPath expression or path step that cannot be resolved out of the indexes. An XPath expression with an unsearchable path step cannot be used as the first parameter of cts:search. For more details, see Fully Searchable Paths and cts:search Operations in the *Query Performance and Tuning Guide*.

**URI** (Uniform Resource Identifier)

*Web*. A URI is a string of a standardized form to uniquely identify resources. URIs are limited to a subset of the ASCII character set. See also IRI (Internationalized Resource Identifier), URL (Uniform Resource Locator), and URN (Uniform Resource Name).

**URI privilege**

*MarkLogic*. A URI Privilege provides the authority to create documents within a base URI. When a URI privilege exists for a base URI (Uniform Resource Identifier), only users assigned to roles that have the URI privilege can create documents with URIs starting with the base string. See also execute privilege, amp, privilege, privilege, and role. For details on URI privileges, see Protecting Documents in the *Security Guide*.

**URL** (Uniform Resource Locator)

*Web*. A subset of URI that provides the access mechanism and network location of a resource on the World Wide Web. For example, http://www.example.org/news. See also IRI (Internationalized Resource Identifier), URI (Uniform Resource Identifier), and URN (Uniform Resource Name).

**URN** (Uniform Resource Name)

*Web*. A subtype of URI used for defining classes, properties, or individuals. For example, identifying books by their ISBN number. See also IRI (Internationalized Resource Identifier), URI (Uniform Resource Identifier), URL (Uniform Resource Locator).

**user**

*MarkLogic*. A user is a named entity used to authenticate a request to an HTTP, WebDAV, ODBC, or XDBC server. For details on users, see Authenticating Users in the *Security Guide*.

**V**

**valid time**

*Bitemporal*. Valid time marks when the information (captured in a document) was true in the real world. See bitemporal and system time.

**view**

*SQL*. A representation of a SQL view. A view is implemented as an XML document in the schemas database and consists of a unique id, a name (which must be unique in the context of a particular schema), a view scope, and a sequence of column specifications. See Schemas and Views in the *SQL Data Modeling Guide*.

**View**

*Database.* The returned monitoring information about a resource. You can have different views of the same resource. A view can be for a single resource (known as an item view) or a number of resources (known as a "list view").

**view scope**

*Database*. Used to constrain the subset of the database to which the view applies. A view scope can either limit rows in the view to documents with a specific element (local name + namespace) or to documents in a particular collection.

**vocabularies**

*Semantics*. A collection of specific terms used to classify concepts, relationships, and definitions. Vocabularies such as foaf (friend of a friend) and dc (Dublin Core) define concepts and relationships used to describe and represent facts. See ontology.

**W**

**WebDAV Server**

*Web*. WebDAV servers support the WebDAV protocol to allow WebDAV clients to have read and write access (depending on the security configuration) to a database. A WebDAV server only accesses documents and directories in a database; it does not access the file system directly.

**workspace**

*Query Console*. A collection of queries. Use workspaces to organize your queries. You may create multiple workspaces, but only one is active at a time.

**WSDL** (Web Service Definition Language)

*Web*. An XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint.

**X**

**XA**

*Database*. The XA open standard for distributed transaction processing describes the interface between the global transaction manager and the local resource manager.

**XCC** (XML Content Connector)

*MarkLogic*. XCC is an API used to communicate with MarkLogic Server from Java middleware applications. The MarkLogic XCC application is a Java-based connector for accessing content in the server.

**XDBC**

*MarkLogic*. (XML Database Connector) A MarkLogic protocol (analogous to ODBC) for accessing MarkLogic databases. Typically used with Java applications.

**XDBC Server**

*MarkLogic*. XDBC Servers allow XCC (XML Content Connector) applications to communicate with MarkLogic Server. XDBC is analogous to ODBC, a standard middleware protocol used for accessing relational databases, although XDBC is not used for relational data. See XDBC Server Overview in the *Administrator's Guide*.

**XDM** (XQuery and XPath Data Model)

*XML*. The data model shared by XQuery, XPath (XML Path language), and XDM (XQuery and XPath Data Model) programming languages.

**xdmp:**

*MarkLogic*. Accronym for XML Data Management Protocol. Used in the MarkLogic built-in functions.

**XML** (Extended Markup Language)

*XML*. A markup language defining a set of rules for encoding documents in human and machine-readable format. XML was designed to facilitate usability on the Web, to be simple to use, and easily generalized. See also XPath (XML Path language) and XSLT (Extensible Stylesheet Language Transformations).

**XMLns** (XML namespace)

*XML*. Used to provide uniquely named elements in a document and avoid naming collisions.

**XPath** (XML Path language)

*XML*. XPath is used for selecting nodes from an XML (Extended Markup Language) document. Based on a tree representation of the XML document, XPath can navigate around the tree, selecting nodes. It can also be used to compute values from the contents of an XML document. See node.

**XQuery**

*XML*. Language for querying XML (Extended Markup Language) documents. XQuery is a functional programming language for searching and transforming collections of structured and unstructured data.

**XQuery Program**

*XML*. This is the XQuery main module fully expanded, with any XQuery library modules needed for its evaluation. An XQuery program is sometimes referred to as a query, a statement, or a request. For more details on this terminology, see Understanding Transactions in MarkLogic Server in the *Application Developer's Guide*.

**XSD** (XML Schema Definition)

*XML*. XSD specifies how to formally describe the elements in an XML (Extended Markup Language) document. It can be used to verify each piece of item content in a document, to check that the item adheres to the description of its element. XSD can be used to express a set of rules to which an XML document must conform in order to be considered 'valid' according to that schema.

**XSLT** (Extensible Stylesheet Language Transformations)

*XML*. A subset of XPath (XML Path language) used for transforming XML (Extended Markup Language) documents into other XML documents, or other objects like HTML web pages or plain text.

**Y, Z**

**Zero-day Replication**

*Database*. Replicating data from the Master database that existed before replication was configured.

## 2.0 Technical Support

MarkLogic provides technical support according to the terms detailed in your Software License Agreement or End User License Agreement.

We invite you to visit our support website at http://help.marklogic.com to access information on known and fixed issues, knowledge base articles, and more. For licensed customers with an active maintenance contract, see the Support Handbook for instructions on registering support contacts and on working with the MarkLogic Technical Support team.

Complete product documentation, the latest product release downloads, and other useful information is available for all developers at http://developer.marklogic.com. For technical questions, we encourage you to ask your question on Stack Overflow.

# 3.0 Copyright

MarkLogic Server 9.0 and supporting products.
Last updated: August 5, 2020