

CAStor® SDK

HTTP Interface and Software Development Kit



Integrate Caringo®
Object Storage powered
by CAStor



Our object storage software was developed with guaranteed accessibility in mind regardless of application or device. To achieve this we based the CAStor Simple Content Storage Protocol (SCSP) on HTTP 1.1 using a RESTful architecture. Using standard and timeless protocols and architectures such as HTTP 1.1 and REST means that access to and from a CAStor cluster is intuitive, scalable and future-proof. Additionally we offer a Software Development Kit (SDK) that provides fully documented client library support for integrating the Caringo object storage software suite into any application or device. The SDK implements CAStor's HTTP interface and includes Caringo's best practices for CAStor client implementations.

The CAStor SDK includes:

- Consistent, documented API for Java, Python, C++, and C#
- Content Router Publisher API
- Validation of basic functions for implemented clients running in validation mode
- Implementations covering deficiencies in standard off-the-shelf HTTP libraries

Widely Available Libraries

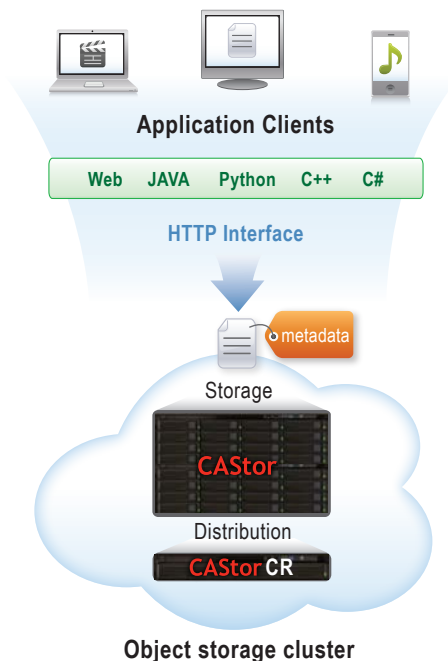
Access to and from CAStor leverages the standard HTTP 1.1 requests such as POST, GET, HEAD, and DELETE. Stable, well-used libraries and commented code examples are available. Your developers will be able to integrate with CAStor from any application or device in minutes.

Robust SDK for all Major Languages

The CAStor Software Development Kit (SDK) provides libraries and sample code for accessing the Caringo object storage platform from any application or device. The SDK describes a consistent set of features using a common API in each of several popular programming languages and includes a reference implementation for Java, Python, C++ and C#. All example clients are synchronous and thread-safe. High performance applications can call any of the clients from multiple threads and/or multiple processes without interference or deadlocks.

SCSP Proxy

The SDK also includes an integrated proxy that enables balanced communication between clients and CAStor clusters on private and public networks. The proxy enables flexible cluster topologies with the ability to write to local and remote clusters in a single transaction. For clusters on a private network, the proxy manages content transfer between external clients and the cluster. The proxy also includes a locator feature that generates a list of CAStor nodes available in a cluster for optimal read/write performance.



About Caringo

To learn more about Caringo and our products visit our website or e-mail info@caringo.com.

Caringo, Inc.

6801 North Capital of Texas Highway
Building 2, Suite 200
Austin, Texas 78731
512-782-4490
www.caringo.com

OS Support

The Caringo CAStor Software Development Kit includes install packages for the following platforms:

- Microsoft Server 2003 R2 SP2
- Microsoft Server 2008 SP2
- Microsoft Server 2008 R2
- Red Hat Enterprise Linux 5.5 and 6.0
- SUSE Linux Enterprise Server 11.2

Beyond the install packages listed above, the SDK source code is available to the developer for additional platforms of choice.

Commands and Objects Supported:

The primary SDK execution class provides procedural methods for execution of the various HTTP commands.

The API supports the following basic commands:

- Write, Read, Info, Delete for all objects
- Update, Copy, Append for mutable objects

Additional specialized methods include:

- Remote synchronous writes to both the local and a remote cluster using the SCSP proxy
- Aggregate Info of a defined list of objects in either a local or remote cluster
- Info of an object in any defined remote cluster for complex topologies

The basic methods supported by the API can be extended with the addition of both standard HTTP request headers and CAStor-specific headers and/or query arguments.

CAStor supports the following general types of objects:

- Immutable system named objects, which can be deleted but not changed. If you delete a system named object, its Universally Unique ID (UUID) is not reused.
- Mutable system named objects, which have contents that can be replaced but UUIDs that never change.
- Mutable user named objects, which allow easy integration into existing namespaces and applications. A user named object's name can be reused after deletion.