

Using the Object Store

The Object Store allows you to store data of any (i.e. large) size by implementing a chunking mechanism, allowing you to for example store and retrieve files (i.e. the object) of any size by associating them with a path and a file name (i.e. the key). You obtain a `ObjectStoreManager` object from your JetStream [context](#).

Go

```
// ObjectStoreManager is used to manage object stores. It provides methods
// for CRUD operations on object stores.
type ObjectStoreManager interface {
    // ObjectStore will look up and bind to an existing object store
    // instance.
    //
    // If the object store with given name does not exist, ErrBucketNotFound
    // will be returned.
    ObjectStore(ctx context.Context, bucket string) (ObjectStore, error)

    // CreateObjectStore will create a new object store with the given
    // configuration.
    //
    // If the object store with given name already exists, ErrBucketExists
    // will be returned.
    CreateObjectStore(ctx context.Context, cfg ObjectStoreConfig) (ObjectStore, error)

    // UpdateObjectStore will update an existing object store with the given
    // configuration.
    //
    // If the object store with given name does not exist, ErrBucketNotFound
    // will be returned.
    UpdateObjectStore(ctx context.Context, cfg ObjectStoreConfig) (ObjectStore, error)

    // CreateOrUpdateObjectStore will create a new object store with the given
    // configuration if it does not exist, or update an existing object store
    // with the given configuration.
    CreateOrUpdateObjectStore(ctx context.Context, cfg ObjectStoreConfig) (ObjectStore, error)
```

```

    // DeleteObjectStore will delete the provided object store.
    //
    // If the object store with given name does not exist, ErrBucketNotFound
    // will be returned.
    DeleteObjectStore(ctx context.Context, bucket string) error

    // ObjectStoreNames is used to retrieve a list of bucket names.
    // It returns an ObjectStoreNamesLister exposing a channel to receive
    // the names of the object stores.
    //
    // The lister will always close the channel when done (either all names
    // have been read or an error occurred) and therefore can be used in a
    // for-range loop.
    ObjectStoreNames(ctx context.Context) ObjectStoreNamesLister

    // ObjectStores is used to retrieve a list of bucket statuses.
    // It returns an ObjectStoresLister exposing a channel to receive
    // the statuses of the object stores.
    //
    // The lister will always close the channel when done (either all statuses
    // have been read or an error occurred) and therefore can be used in a
    // for-range loop.
    ObjectStores(ctx context.Context) ObjectStoresLister
}

// ObjectStore contains methods to operate on an object store.
// Using the ObjectStore interface, it is possible to:
//
// - Perform CRUD operations on objects (Get, Put, Delete).
//   Get and put expose convenience methods to work with
//   byte slices, strings and files, in addition to streaming [io.Reader]
// - Get information about an object without retrieving it.
// - Update the metadata of an object.
// - Add links to other objects or object stores.
// - Watch for updates to a store
// - List information about objects in a store
// - Retrieve status and configuration of an object store.
type ObjectStore interface {
    // Put will place the contents from the reader into a new object. If the

```

```

// object already exists, it will be overwritten. The object name is
// required and is taken from the ObjectMeta.Name field.
//
// The reader will be read until EOF. ObjectInfo will be returned, containing
// the object's metadata, digest and instance information.
Put(ctx context.Context, obj ObjectMeta, reader io.Reader) (*ObjectInfo, error)

// PutBytes is convenience function to put a byte slice into this object
// store under the given name.
//
// ObjectInfo will be returned, containing the object's metadata, digest
// and instance information.
PutBytes(ctx context.Context, name string, data []byte) (*ObjectInfo, error)

// PutString is convenience function to put a string into this object
// store under the given name.
//
// ObjectInfo will be returned, containing the object's metadata, digest
// and instance information.
PutString(ctx context.Context, name string, data string) (*ObjectInfo, error)

// PutFile is convenience function to put a file contents into this
// object store. The name of the object will be the path of the file.
//
// ObjectInfo will be returned, containing the object's metadata, digest
// and instance information.
PutFile(ctx context.Context, file string) (*ObjectInfo, error)

// Get will pull the named object from the object store. If the object
// does not exist, ErrObjectNotFound will be returned.
//
// The returned ObjectResult will contain the object's metadata and a
// reader to read the object's contents. The reader will be closed when
// all data has been read or an error occurs.
//
// A GetObjectShowDeleted option can be supplied to return an object
// even if it was marked as deleted.
Get(ctx context.Context, name string, opts ...GetObjectOpt) (ObjectResult, error)

// GetBytes is a convenience function to pull an object from this object

```

```

    // store and return it as a byte slice.
    //
    // If the object does not exist, ErrObjectNotFound will be returned.
    //
    // A GetObjectShowDeleted option can be supplied to return an object
    // even if it was marked as deleted.
    GetBytes(ctx context.Context, name string, opts ...GetObjectOpt) ([]byte, error)

    // GetString is a convenience function to pull an object from this
    // object store and return it as a string.
    //
    // If the object does not exist, ErrObjectNotFound will be returned.
    //
    // A GetObjectShowDeleted option can be supplied to return an object
    // even if it was marked as deleted.
    GetString(ctx context.Context, name string, opts ...GetObjectOpt) (string, error)

    // GetFile is a convenience function to pull an object from this object
    // store and place it in a file. If the file already exists, it will be
    // overwritten, otherwise it will be created.
    //
    // If the object does not exist, ErrObjectNotFound will be returned.
    // A GetObjectShowDeleted option can be supplied to return an object
    // even if it was marked as deleted.
    GetFile(ctx context.Context, name, file string, opts ...GetObjectOpt) error

    // GetInfo will retrieve the current information for the object, containing
    // the object's metadata and instance information.
    //
    // If the object does not exist, ErrObjectNotFound will be returned.
    //
    // A GetObjectInfoShowDeleted option can be supplied to return an object
    // even if it was marked as deleted.
    GetInfo(ctx context.Context, name string, opts ...GetObjectInfoOpt) (*ObjectInfo, error)

    // UpdateMeta will update the metadata for the object.
    //
    // If the object does not exist, ErrUpdateMetaDeleted will be returned.
    // If the new name is different from the old name, and an object with the
    // new name already exists, ErrObjectAlreadyExists will be returned.

```

```
func UpdateMeta(ctx context.Context, name string, meta ObjectMeta) error
```

```
// Delete will delete the named object from the object store. If the object
// does not exist, ErrObjectNotFound will be returned. If the object is
// already deleted, no error will be returned.
```

```
//
```

```
// All chunks for the object will be purged, and the object will be marked
// as deleted.
```

```
func Delete(ctx context.Context, name string) error
```

```
// AddLink will add a link to another object. A link is a reference to
// another object. The provided name is the name of the link object.
// The provided ObjectInfo is the info of the object being linked to.
```

```
//
```

```
// If an object with given name already exists, ErrObjectAlreadyExists
// will be returned.
```

```
// If object being linked to is deleted, ErrNoLinkToDeleted will be
// returned.
```

```
// If the provided object is a link, ErrNoLinkToLink will be returned.
```

```
// If the provided object is nil or the name is empty, ErrObjectRequired
// will be returned.
```

```
func AddLink(ctx context.Context, name string, obj *ObjectInfo) (*ObjectInfo, error)
```

```
// AddBucketLink will add a link to another object store. A link is a
// reference to another object store. The provided name is the name of
// the link object.
```

```
// The provided ObjectStore is the object store being linked to.
```

```
//
```

```
// If an object with given name already exists, ErrObjectAlreadyExists
// will be returned.
```

```
// If the provided object store is nil ErrBucketRequired will be returned.
```

```
func AddBucketLink(ctx context.Context, name string, bucket ObjectStore) (*ObjectInfo, error)
```

```
// Seal will seal the object store, no further modifications will be allowed.
```

```
func Seal(ctx context.Context) error
```

```
// Watch for any updates to objects in the store. By default, the watcher will send the latest
// info for each object and all future updates. Watch will send a nil
// entry when it has received all initial values. There are a few ways
// to configure the watcher:
```

```

//
// - IncludeHistory will have the watcher send all historical information
// for each object.
// - IgnoreDeletes will have the watcher not pass any objects with
// delete markers.
// - UpdatesOnly will have the watcher only pass updates on objects
// (without latest info when started).
Watch(ctx context.Context, opts ...WatchOpt) (ObjectWatcher, error)

// List will list information about objects in the store.
//
// If the object store is empty, ErrNoObjectsFound will be returned.
List(ctx context.Context, opts ...ListObjectsOpt) ([]*ObjectInfo, error)

// Status retrieves the status and configuration of the bucket.
Status(ctx context.Context) (ObjectStoreStatus, error)
}

```

See more at jetstream/object.go

Java

```

/**
 * Object Store Management context for creation and access to key value buckets.
 */
public interface ObjectStore {

    /**
     * Get the name of the object store's bucket.
     * @return the name
     */
    String getBucketName();

    /**
     * Place the contents of the input stream into a new object.
     */
    ObjectInfo put(ObjectMeta meta, InputStream inputStream) throws IOException,
        JetStreamApiException, NoSuchAlgorithmException;

    /**

```

```

    * Place the contents of the input stream into a new object.
    */
    ObjectInfo put(String objectName, InputStream inputStream) throws IOException,
JetStreamApiException, NoSuchAlgorithmException;

/**
    * Place the bytes into a new object.
    */
    ObjectInfo put(String objectName, byte[] input) throws IOException, JetStreamApiException,
NoSuchAlgorithmException;

/**
    * Place the contents of the file into a new object using the file name as the object
name.
    */
    ObjectInfo put(File file) throws IOException, JetStreamApiException,
NoSuchAlgorithmException;

/**
    * Get an object by name from the store, reading it into the output stream, if the object
exists.
    */
    ObjectInfo get(String objectName, OutputStream outputStream) throws IOException,
JetStreamApiException, InterruptedException, NoSuchAlgorithmException;

/**
    * Get the info for an object if the object exists / is not deleted.
    */
    ObjectInfo getInfo(String objectName) throws IOException, JetStreamApiException;

/**
    * Get the info for an object if the object exists, optionally including deleted.
    */
    ObjectInfo getInfo(String objectName, boolean includingDeleted) throws IOException,
JetStreamApiException;

/**
    * Update the metadata of name, description or headers. All other changes are ignored.
    */
    ObjectInfo updateMeta(String objectName, ObjectMeta meta) throws IOException,

```

```

JetStreamApiException;

/**
 * Delete the object by name. A No-op if the object is already deleted.
 */
ObjectInfo delete(String objectName) throws IOException, JetStreamApiException;

/**
 * Add a link to another object. A link cannot be for another link.
 */
ObjectInfo addLink(String objectName, ObjectInfo toInfo) throws IOException,
JetStreamApiException;

/**
 * Add a link to another object store (bucket).
 */
ObjectInfo addBucketLink(String objectName, ObjectStore toStore) throws IOException,
JetStreamApiException;

/**
 * Close (seal) the bucket to changes. The store (bucket) will be read only.
 */
ObjectStoreStatus seal() throws IOException, JetStreamApiException;

/**
 * Get a list of all object [infos] in the store.
 */
List<ObjectInfo> getList() throws IOException, JetStreamApiException,
InterruptedException;

/**
 * Create a watch on the store (bucket).
 */
NatsObjectStoreWatchSubscription watch(ObjectStoreWatcher watcher,
ObjectStoreWatchOption... watchOptions) throws IOException, JetStreamApiException,
InterruptedException;

/**
 * Get the ObjectStoreStatus object.
 */

```

```
ObjectStoreStatus getStatus() throws IOException, JetStreamApiException;
```

Python

```
async def object_store(self, bucket: str) -> ObjectStore:

async def create_object_store(
    self,
    bucket: str = None,
    config: Optional[api.ObjectStoreConfig] = None,
    **params,
) -> ObjectStore:
    """
    create_object_store takes an api.ObjectStoreConfig and creates a OBJ in JetStream.
    """

async def delete_object_store(self, bucket: str) -> bool:
    """
    delete_object_store will delete the underlying stream for the named object.
    """
```

C#

```
// dotnet add package NATS.Net

/// <summary>
/// NATS Object Store context.
/// </summary>
public interface INatsObjContext
{
    /// <summary>
    /// Provides access to the JetStream context associated with the Object Store operations.
    /// </summary>
    INatsJSContext JetStreamContext { get; }

    /// <summary>
    /// Create a new object store.
    /// </summary>
    /// <param name="bucket">Bucket name.</param>
    /// <param name="cancellationToken">A <see cref="CancellationToken"/> used to cancel the
    API call.</param>
```

```

    /// <returns>Object store object.</returns>
    ValueTask<INatsObjStore> CreateObjectStoreAsync(string bucket, CancellationToken
cancellationToken = default);

    /// <summary>
    /// Create a new object store.
    /// </summary>
    /// <param name="config">Object store configuration.</param>
    /// <param name="cancellationToken">A <see cref="CancellationToken"/> used to cancel the
API call.</param>
    /// <returns>Object store object.</returns>
    ValueTask<INatsObjStore> CreateObjectStoreAsync(NatsObjConfig config, CancellationToken
cancellationToken = default);

    /// <summary>
    /// Get an existing object store.
    /// </summary>
    /// <param name="bucket">Bucket name</param>
    /// <param name="cancellationToken">A <see cref="CancellationToken"/> used to cancel the
API call.</param>
    /// <returns>The Object Store object</returns>
    ValueTask<INatsObjStore> GetObjectStoreAsync(string bucket, CancellationToken
cancellationToken = default);

    /// <summary>
    /// Delete an object store.
    /// </summary>
    /// <param name="bucket">Name of the bucket.</param>
    /// <param name="cancellationToken">A <see cref="CancellationToken"/> used to cancel the
API call.</param>
    /// <returns>Whether delete was successful or not.</returns>
    ValueTask<bool> DeleteObjectStore(string bucket, CancellationToken cancellationToken);
}

    /// <summary>
    /// NATS Object Store.
    /// </summary>
public interface INatsObjStore
{
    /// <summary>
    /// Provides access to the JetStream context associated with the Object Store operations.

```

```

/// </summary>
INatsJSContext JetStreamContext { get; }

/// <summary>
/// Object store bucket name.
/// </summary>
string Bucket { get; }

/// <summary>
/// Get object by key.
/// </summary>
/// <param name="key">Object key.</param>
/// <param name="cancellationToken">A <see cref="CancellationToken"/> used to cancel the
API call.</param>
/// <returns>Object value as a byte array.</returns>
ValueTask<byte[]> GetBytesAsync(string key, Cancellation_token cancellationToken =
default);

/// <summary>
/// Get object by key.
/// </summary>
/// <param name="key">Object key.</param>
/// <param name="stream">Stream to write the object value to.</param>
/// <param name="leaveOpen"><c>true</c> to not close the underlying stream when async
method returns; otherwise, <c>false</c></param>
/// <param name="cancellationToken">A <see cref="CancellationToken"/> used to cancel the
API call.</param>
/// <returns>Object metadata.</returns>
/// <exception cref="NatsObjException">Metadata didn't match the value retrieved e.g. the
SHA digest.</exception>
ValueTask<ObjectMetadata> GetAsync(string key, Stream stream, bool leaveOpen = false,
Cancellation_token cancellationToken = default);

/// <summary>
/// Put an object by key.
/// </summary>
/// <param name="key">Object key.</param>
/// <param name="value">Object value as a byte array.</param>
/// <param name="cancellationToken">A <see cref="CancellationToken"/> used to cancel the
API call.</param>
/// <returns>Object metadata.</returns>

```

```

    ValueTask<ObjectMetadata> PutAsync(string key, byte[] value, CancellationToken
cancellationToken = default);

    /// <summary>
    /// Put an object by key.
    /// </summary>
    /// <param name="key">Object key.</param>
    /// <param name="stream">Stream to read the value from.</param>
    /// <param name="leaveOpen"><c>true</c> to not close the underlying stream when async
method returns; otherwise, <c>false</c></param>
    /// <param name="cancellationToken">A <see cref="CancellationToken"/> used to cancel the
API call.</param>
    /// <returns>Object metadata.</returns>
    /// <exception cref="NatsObjException">There was an error calculating SHA
digest.</exception>
    /// <exception cref="NatsJSApiException">Server responded with an error.</exception>
    ValueTask<ObjectMetadata> PutAsync(string key, Stream stream, bool leaveOpen = false,
CancellationToken cancellationToken = default);

    /// <summary>
    /// Put an object by key.
    /// </summary>
    /// <param name="meta">Object metadata.</param>
    /// <param name="stream">Stream to read the value from.</param>
    /// <param name="leaveOpen"><c>true</c> to not close the underlying stream when async
method returns; otherwise, <c>false</c></param>
    /// <param name="cancellationToken">A <see cref="CancellationToken"/> used to cancel the
API call.</param>
    /// <returns>Object metadata.</returns>
    /// <exception cref="NatsObjException">There was an error calculating SHA
digest.</exception>
    /// <exception cref="NatsJSApiException">Server responded with an error.</exception>
    ValueTask<ObjectMetadata> PutAsync(ObjectMetadata meta, Stream stream, bool leaveOpen =
false, CancellationToken cancellationToken = default);

    /// <summary>
    /// Update object metadata
    /// </summary>
    /// <param name="key">Object key</param>
    /// <param name="meta">Object metadata</param>
    /// <param name="cancellationToken">A <see cref="CancellationToken"/> used to cancel the

```

```

API call.</param>
    /// <returns>Object metadata</returns>
    /// <exception cref="NatsObjException">There is already an object with the same
name</exception>
    ValueTask<ObjectMetadata> UpdateMetaAsync(string key, ObjectMetadata meta,
CancellationTokentoken cancellationToken = default);

    /// <summary>
    /// Add a link to another object
    /// </summary>
    /// <param name="link">Link name</param>
    /// <param name="target">Target object's name</param>
    /// <param name="cancellationToken">A <see cref="CancellationTokentoken"/> used to cancel the
API call.</param>
    /// <returns>Metadata of the new link object</returns>
    ValueTask<ObjectMetadata> AddLinkAsync(string link, string target, CancellationTokentoken
cancellationToken = default);

    /// <summary>
    /// Add a link to another object
    /// </summary>
    /// <param name="link">Link name</param>
    /// <param name="target">Target object's metadata</param>
    /// <param name="cancellationToken">A <see cref="CancellationTokentoken"/> used to cancel the
API call.</param>
    /// <returns>Metadata of the new link object</returns>
    ValueTask<ObjectMetadata> AddLinkAsync(string link, ObjectMetadata target,
CancellationTokentoken cancellationToken = default);

    /// <summary>
    /// Add a link to another object store
    /// </summary>
    /// <param name="link">Object's name to be linked</param>
    /// <param name="target">Target object store</param>
    /// <param name="cancellationToken">A <see cref="CancellationTokentoken"/> used to cancel the
API call.</param>
    /// <returns>Metadata of the new link object</returns>
    /// <exception cref="NatsObjException">Object with the same name already
exists</exception>
    ValueTask<ObjectMetadata> AddBucketLinkAsync(string link, INatsObjStore target,
CancellationTokentoken cancellationToken = default);

```

```

    /// <summary>
    /// Seal the object store. No further modifications will be allowed.
    /// </summary>
    /// <param name="cancellationToken">A <see cref="CancellationTok
API call.</param>
    /// <exception cref="NatsObjException">Update operation failed</exception>
    ValueTask SealAsync(CancellationTok
    cancellationToken = default);

    /// <summary>
    /// Get object metadata by key.
    /// </summary>
    /// <param name="key">Object key.</param>
    /// <param name="showDeleted">Also retrieve deleted objects.</param>
    /// <param name="cancellationToken">A <see cref="CancellationTok
API call.</param>
    /// <returns>Object metadata.</returns>
    /// <exception cref="NatsObjException">Object was not found.</exception>
    ValueTask<ObjectMetadata> GetInfoAsync(string key, bool showDeleted = false,
    CancellationTok
    cancellationToken = default);

    /// <summary>
    /// List all the objects in this store.
    /// </summary>
    /// <param name="opts">List options</param>
    /// <param name="cancellationToken">A <see cref="CancellationTok
API call.</param>
    /// <returns>An async enumerable object metadata to be used in an <c>await
foreach</c></returns>
    IAsyncEnumerable<ObjectMetadata> ListAsync(NatsObjListOpts? opts = default,
    CancellationTok
    cancellationToken = default);

    /// <summary>
    /// Retrieves run-time status about the backing store of the bucket.
    /// </summary>
    /// <param name="cancellationToken">A <see cref="CancellationTok
API call.</param>
    /// <returns>Object store status</returns>
    ValueTask<NatsObjStatus> GetStatusAsync(CancellationTok
    cancellationToken = default);

    /// <summary>

```

```

    /// Watch for changes in the underlying store and receive meta information updates.
    /// </summary>
    /// <param name="opts">Watch options</param>
    /// <param name="cancellationToken">A <see cref="Cancellation.Token"/> used to cancel the
API call.</param>
    /// <returns>An async enumerable object metadata to be used in an <c>await
foreach</c></returns>
    IEnumerable<ObjectMetadata> WatchAsync(NatsObjWatchOpts? opts = default,
Cancellation.Token cancellationToken = default);

    /// <summary>
    /// Delete an object by key.
    /// </summary>
    /// <param name="key">Object key.</param>
    /// <param name="cancellationToken">A <see cref="Cancellation.Token"/> used to cancel the
API call.</param>
    /// <exception cref="NatsObjException">Object metadata was invalid or chunks can't be
purged.</exception>
    ValueTask DeleteAsync(string key, Cancellation.Token cancellationToken = default);
}

```

📄 #1

📄 cocobook-sync 📄 2026-10-28 11:13:08 UTC

📄 cocobook-sync 📄 2026-10-28 11:13:08 UTC