IndexedDB: definitions

> This chapter can be read as is, it but it is primarily given as a reference. We recommend you skim read it, then do the next section ("using IndexedDB"), then come back to this page if you need any clarification.

These definitions come from the W3C specification. Read this page to familiarize yourself with the terms.

DATABASE

- Each origin (you may consider as "each application") has an associated set of databases. A *database* comprises one or moreobject stores which hold the data stored in the database.
- Every database has a *name* that identifies it within a specific origin. The name can be any string value, including the empty string, and stays constant for the lifetime of the database.
- Each database also has a current version. When a database is first created, its version is 0, if not specified otherwise. Each database can only have one version at any given time. A database can't exist in multiple versions at once.
- The act of opening a database creates *a connection*. There may be multiple connections to a given database at any given time.

OBJECT STORE

- An object store is the mechanism by which data are stored in the database.
- Every object store has *a name*. The name is unique within the database to which it belongs.
- The object store persistently holds records (JavaScript objects), which are key-value

pairs. One of these keys is a kind of "primary key" in the SQL database sense. This "key" is a property that every object in the datastore *must* contain. Values in the object store are structured, but this structure may vary (i.e., if we store persons in a database, and use the email as "the key all objects must define", some may have first name and last name, others may have an address or no address at all, etc.)

- Records within an object store are sorted according to the keys in ascending order.
- Every object store also optionally has *a key generator* and an optional *key path*. If the object store has a key path, it is said to use in-line keys. Otherwise it is said to use *out-of-line* keys.
- The object store can derive the key from one of three sources:

    1. A key generator. A key generator generates a monotonically increasing number every time a key is needed. This is somewhat similar to auto-incremented primary keys in a SQL database.
    2. Keys can be derived via a key path.
    3. Keys can also be explicitly specified when a value is stored in the object store.

Further details will be given in the next chapter "Using IndexedDB".

VERSION

- When a database is first created, its version is the integer 0. Each database has one version at a time; a database can't exist in multiple versions at once.
- The only way to change the version is by opening it with a higher version than the current one. This will start a `versionchange`transaction and fire an `upgradeneeded` event. The only place where the schema of the database can be updated is inside the handler of that event.

This definition describes the most recent specification, which is only implemented in up-to-date browsers. Old browsers implemented the now deprecated and removed `IDBDatabase.setVersion()` method.

TRANSACTION

From the specification: "*A transaction is used to interact with the data in a database. Whenever data is read or written to the database, this is done by using a transaction.*

All transactions are created through a connection, which is the transaction's connection. The transaction has a mode (`read`,`readwrite` *or* `versionchange`*) that determines which types of interactions can be performed upon that transaction. The mode is set when the transaction is created and remains fixed for the life of the transaction. The transaction also has a scope that determines the object stores with which the transaction may interact*."

A transaction in IndexedDB is similar to a transaction in a SQL database. It defines: "*An atomic and durable set of data-access and data-modification operations*". Either all operations succeed or fail.

A database connection can have several active transactions associated with it at a time, but these write transactions cannot have overlapping scopes (*they cannot work on the same data at the same time*). The scope of a transaction, which is defined at creation time, determines which concurrent transactions can read or write the same data (multiple reads can occur, while writes will be sequential, only one at a time), and remains constant for the lifetime of the transaction.

So, for example, if a database connection already has a writing transaction with a scope that just covers the `flyingMonkey` object store, you can start a second transaction with a scope of the`unicornCentaur` and `unicornPegasus` object stores. As for reading transactions, you can have several of them, and they may even overlap. A "`versionchange`" transaction never runs concurrently with other transactions (reminder: we usually use such transactions when we create the object store or when we modify the schema).

Generally speaking, the above requirements mean that "`readwrite`" transactions which have overlapping scopes always run in the order they were created, and never run in parallel. A "`versionchange`" transaction is automatically created when a database version number is provided that is greater than the current database version. This transaction will be active inside the `onupgradeneeded`event handler, allowing the creation of new object stores andindexes.

REQUEST

The operation by which reading and writing on a database is done. Every request represents one read or one write operation. Requests are always run within a transaction. The example below adds a customer into the object store named "customers".

```
// Use the transaction to add data...
var objectStore =transaction.objectStore("customers");
for (var i in customerData) {
    var request =objectStore.add(customerData[i]);
    request.onsuccess = function(event) {
        // event.target.result == customerData[i].ssn
 };
}
```

INDEX

It is sometimes useful to retrieve records in an object store through other means than their key.

An *index* allows the user to look up records in an object store using the properties of the values in the object stores records. Indexes are a common concept in databases. Indexes can speed up object retrieval and allow multi-criteria searches. For example, if you store persons in your object store, and add an index on the "email" property of each person, then looking for some person by email will be much faster.

An index is a specialized persistent key-value storage and has a*referenced* object store. For example, you have your "persons" object store that is the referenced data store, and this reference store can have an index storage associated with it that contains indexes that map email values to key values in the reference store.

The index has *a list of records* which holds the data stored in the index. The records in an index are automatically populated whenever records in the referenced object store are inserted, updated or deleted. There can be several indexes referencing the same object store, in which changes to the object store cause all such indexes to get updated.

An index contains *a unique flag*. When this flag is set to `true`, the index enforces that no two records in the index have the same key. If a user attempts to insert or modify a record in the index's referenced object store, such that evaluating the index's key path on the record's new value yields a result which already exists in the index, then the attempted modification to the object store fails.

KEY AND VALUES

KEY

A data value by which stored values are organized and retrieved in the object store. The object store can derive the key from one of three sources: a key generator, a key path, and an explicitly specified value.

The key must be of a data type that has a number that is greater than the one before. Each record in an object store must have a key that is unique within the same store, so you cannot have multiple records with the same key in a given object store.

A key can be one of the following types: string, date, `float`, and**array**. For arrays, the key can range from an empty value to infinity. And you can include an array within an array.

Alternatively, you can also look up records in an object store usingthe index.

KEY GENERATOR

A mechanism for producing new keys in an ordered sequence. If an object store does not have a key generator, then the application must provide keys for records being stored. Similar to auto-generated primary keys in SQL databases.

IN-LINE KEY

A key that is stored as part of the stored value. Example: the email of a person or a student number in an object representing a student in a student store. It is found using *a key path*. An in-line key can be generated using a generator. After the key has been generated, it can then be stored in the value using the key path, or it can also be used as a key.

OUT-OF-LINE KEY

A key that is stored separately from the value being stored, for instance, an auto-incremented id that is not part of the object. Example: you store persons `{name:Buffa, firstName:Michel}` and `{name:Gandon, firstName: Fabien}`, each will have a key (think of it as a primary key, an id...) that can be auto-generated or specified, but that is not part of the object stored.

KEY PATH

Defines where the browser should extract the key from a value in the object store or index. **A valid key path can include one of the following: an empty string, a JavaScript identifier, or multiple JavaScript identifiers separated by periods. It cannot include spaces.**

VALUE

Each record has a value, which could include anything that can be expressed in JavaScript, including: boolean, number, string, date,object, array, regexp, undefined, and null.

When an object or an array is stored, the properties and values in that object or array can also be anything that is a valid value.

Blobs and files can be stored, (supported by all major browsers, IE > 9). The example in the next chapter stores images using blobs.

RANGE AND SCOPE

SCOPE

The set of object stores and indexes to which a transaction applies. The scopes of read-only transactions can overlap and execute at the same time. On the other hand, the scopes of writing transactions cannot overlap. You can still start several transactions with the same scope at the same time, but they just queue up and execute one after another.

CURSOR

A mechanism for iterating over multiple records with a *key range*. The cursor has a source that indicates which index or object store it is iterating. It has a position within the range, and moves in a direction that is increasing or decreasing in the order of record keys. For the reference documentation on cursors, see IDBCursor.

KEY RANGE

A continuous interval over some data type used for keys. Records can be retrieved from object stores and indexes using keys or a range of keys. You can limit or filter the range using lower and upper bounds. For example, you can iterate over all values of a key between x and y.

For the reference documentation on key range, see IDBKeyRange.