

A simple TODO list using HTML5 WebDatabases



- [Introduction](#)
 - [Pre-requisites](#)
 - [Asynchronous and Transactional](#)
 - [Step 1. Opening the database](#)
 - [Step 2. Creating a table](#)
 - [Step 3. Adding data to a table](#)
 - [Step 4. Selecting data from a table](#)
 - [Step 4a. Rendering data from a table](#)
 - [Step 5. Deleting data from a table](#)
 - [Step 6. Hooking it all up](#)
-

{% Aside 'warning' %} On November 18, 2010, the [W3C announced](#) that Web SQL database is a deprecated specification. This is a recommendation for web developers to no longer use the technology as effectively the spec will receive no new updates and browser vendors aren't encouraged to support this technology.

Instead, developers are encouraged to use [IndexedDB](#), the replacement technology. {% endAside %}

Introduction

[Web Databases](#) are new in HTML5. Web Databases are hosted and persisted inside a user's browser. By allowing developers to create applications with rich query abilities it is envisioned that a new breed of web applications will emerge that have the ability to work online and off-line.

The example code in this article demonstrates how to create a very simple todo list manager. It is a very high level tour of some of the features available in HTML5.

Pre-requisites

This sample uses a namespace to encapsulate the database logic.

```
var html5rocks = {};  
html5rocks.webdb = {};
```

Asynchronous and Transactional

In the majority of cases where you are using Web Database support you will be using the [Asynchronous API](#). The Asynchronous API is a non-blocking system and as such will not get data through return values, but rather will get data delivered to a defined callback function.

The Web Database support through HTML is transactional. It is no possible to execute SQL statements outside of a transaction. There are two types of transactions: read/write transactions (`transaction()`) and read only transactions (`readTransaction()`). Please note, read/write will lock the entire database.

Step 1. Opening the database

The database needs to be opened before it can be accessed.
You need to define the name, version, description and the size of the database.

```
html5rocks.webdb.db = null;  
  
html5rocks.webdb.open = function() {  
  var dbSize = 5 * 1024 * 1024; // 5MB  
  html5rocks.webdb.db = openDatabase("Todo", "1", "Todo manager", dbSize);  
}  
  
html5rocks.webdb.onError = function(tx, e) {  
  alert("There has been an error: " + e.message);  
}  
  
html5rocks.webdb.onSuccess = function(tx, r) {  
  // re-render the data.
```

```
// loadTodoItems is defined in Step 4a
html5rocks.webdb.getAllTodoItems(loadTodoItems);
}
```

Step 2. Creating a table

You can only create a table by executing a CREATE TABLE SQL statement inside a transaction.

We have defined a function that will create a table in the body onload event. If the table doesn't already exist, a table will be created.

The table is called todo and has 3 columns.

- ID - a incrementing sequential ID column
- todo - a text column that is the body of the item
- added_on - the time that the todo item was created

```
html5rocks.webdb.createTable = function() {
  var db = html5rocks.webdb.db;
  db.transaction(function(tx) {
    tx.executeSql("CREATE TABLE IF NOT EXISTS " +
                  "todo(ID INTEGER PRIMARY KEY ASC, todo TEXT, added_on DATETIME)", []);
  });
}
```

Step 3. Adding data to a table

We are building a todo list manager so it is pretty important that we are able to add todo items in to the database.

A transaction is created, inside the transaction an INSERT into the todo table is performed.

executeSql takes several parameters, the SQL to execute and the parameters values to bind the query.

```
html5rocks.webdb.addTodo = function(todoText) {
  var db = html5rocks.webdb.db;
  db.transaction(function(tx){
```

```
var addedOn = new Date();
tx.executeSql("INSERT INTO todo(todo, added_on) VALUES (?,?)",
    [todoText, addedOn],
    html5rocks.webdb.onSuccess,
    html5rocks.webdb.onError);
});
}
```

Step 4. Selecting data from a table

Now that the data is in the database, you need a function that gets the data back out. In Chrome, Webdatabase's use standard SQLite SELECT queries.

```
html5rocks.webdb.getAllTodoItems = function(renderFunc) {
var db = html5rocks.webdb.db;
db.transaction(function(tx) {
tx.executeSql("SELECT * FROM todo", [], renderFunc,
    html5rocks.webdb.onError);
});
}
```

Note that all of these commands used in this sample are asynchronous and as such the data is not returned from the transaction or the executeSql call. The results are passed through to the success callback.

Step 4a. Rendering data from a table

Once the data has been fetched from the table, the loadTodoItems method will be called.

The onSuccess callback takes two parameters. The first being the transaction of the query and the second being the result set. It is fairly simple to iterate across the data:

```
function loadTodoItems(tx, rs) {
var rowOutput = "";
var todoItems = document.getElementById("todoItems");
for (var i=0; i < rs.rows.length; i++) {
```

```

rowOutput += renderTodo(rs.rows.item(i));
}

todoItems.innerHTML = rowOutput;
}

function renderTodo(row) {
return "<li>" + row.todo +
    " [<a href='javascript:void(0);' onclick=\"html5rocks.webdb.deleteTodo(" +
    row.ID + ");\">Delete</a>]</li>";
}

```

The effect of this method call is that the todo list is rendered into a DOM Element called "todoItems".

Step 5. Deleting data from a table

```

html5rocks.webdb.deleteTodo = function(id) {
var db = html5rocks.webdb.db;
db.transaction(function(tx){
tx.executeSql("DELETE FROM todo WHERE ID=?", [id],
    html5rocks.webdb.onSuccess,
    html5rocks.webdb.onError);
});
}

```

Step 6. Hooking it all up

When the page loads, open the database and create the table (if needed) and render any todo items that might already be in the database.

```

....
function init() {
html5rocks.webdb.open();
html5rocks.webdb.createTable();
html5rocks.webdb.getAllTodoItems(loadTodoItems);
}
</script>

```

```
<body onload="init();">
```

A function that takes the data out of the DOM is needed so, call the `html5rocks.webdb.addTodo` method

```
function addTodo() {  
  var todo = document.getElementById("todo");  
  html5rocks.webdb.addTodo(todo.value);  
  todo.value = "";  
}
```

📄 #2

📄 📄 📄 2026-10-31 22:28:18 UTC
📄 📄 📄 2026-10-31 22:28:19 UTC