



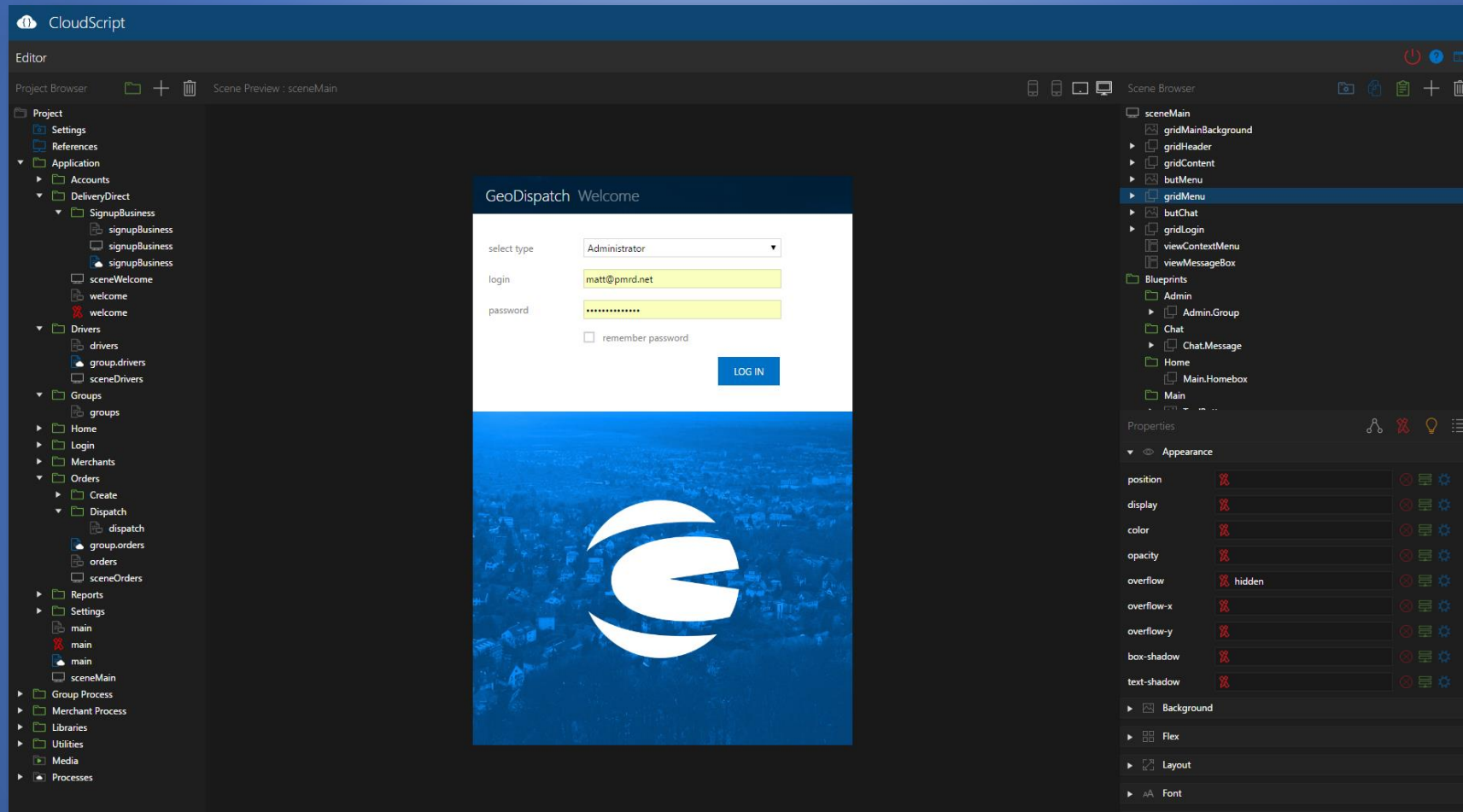
CloudScript

What is CloudScript?

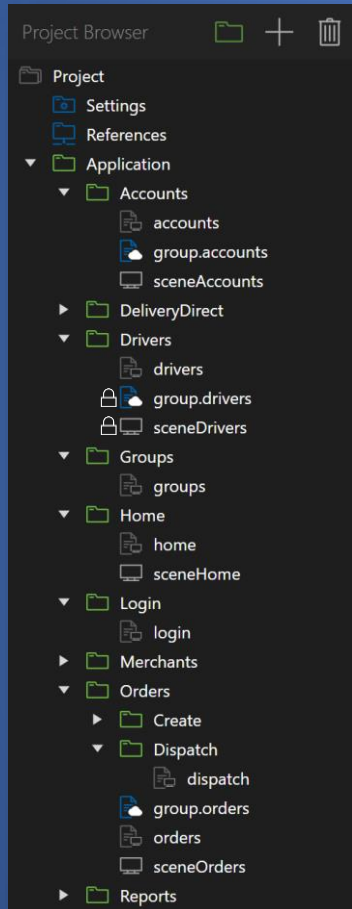
CloudScript is a web-based IDE for web application development. CloudScript is easy for beginners and powerful for advanced users. We like to think it's the middle-ground between Wix and Visual Studio.

- Multi-platform: HTML5 and JavaScript (if it has a modern browser it works!)
- Built with .NET Framework and pure JavaScript. No third-party libraries.
- Amazing performance. Extremely lightweight.
- Integrated hosting: Simply point your DNS to our provided IP and it's ready to use!
- Easy to use interface editor with visual scripting, bindings, controls, device-size variables and a lot more!
- Powerful code editor with IntelliSense and hundreds of built-in functions.
- Create simple or complex cloud-hosted APIs written in JavaScript.
- Extremely easy to use multithreaded server and inter-thread communication.
- Client-side libraries for API communication and interface interaction.
- Team features: Versioning, file locking and more!
- Easy to publish your finished product.

Editor

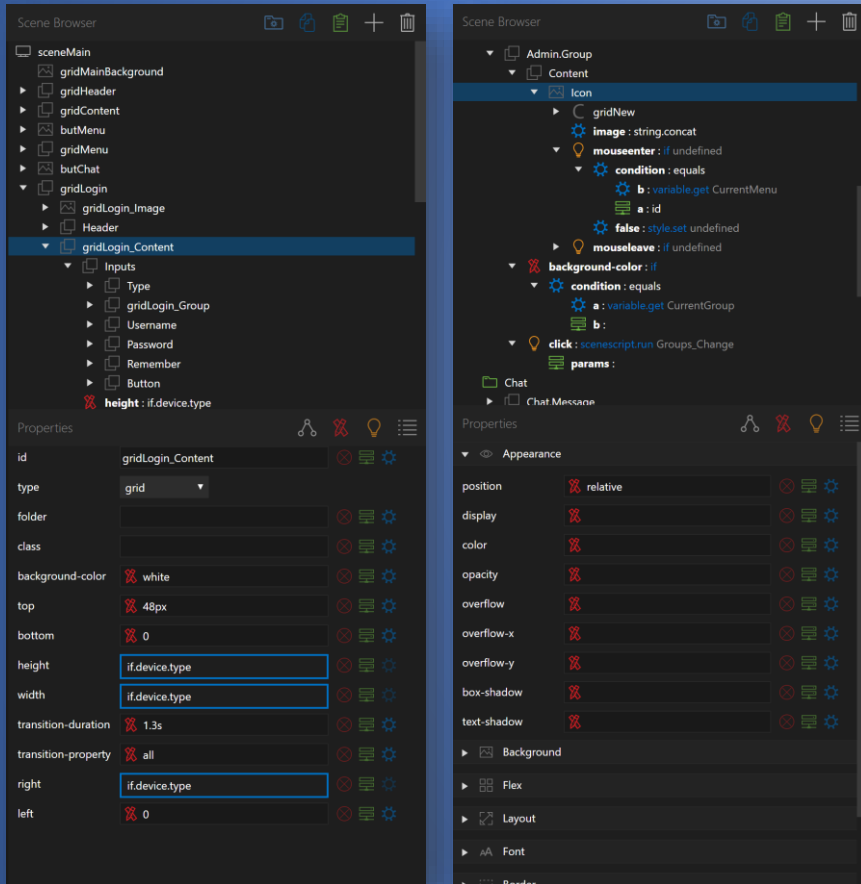


Editor - Project Browser



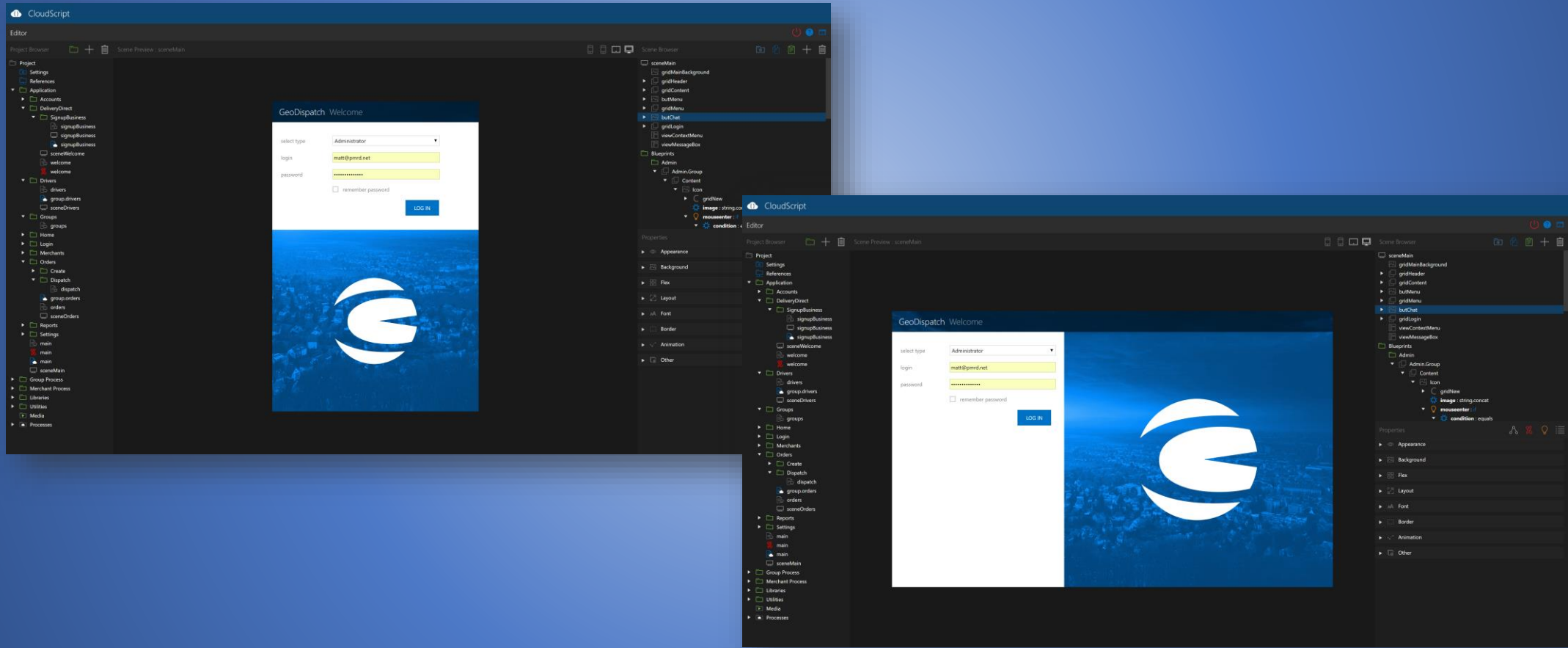
- Create and manage folders and files: Scenes, cloudscripts, scenescrpts, controls, html and css.
- Instantly updates when working with a team
- See if team members are working on files
- Media manager: Images, sounds, etc.
- Process manager
- Project settings

Editor - Scene Browser

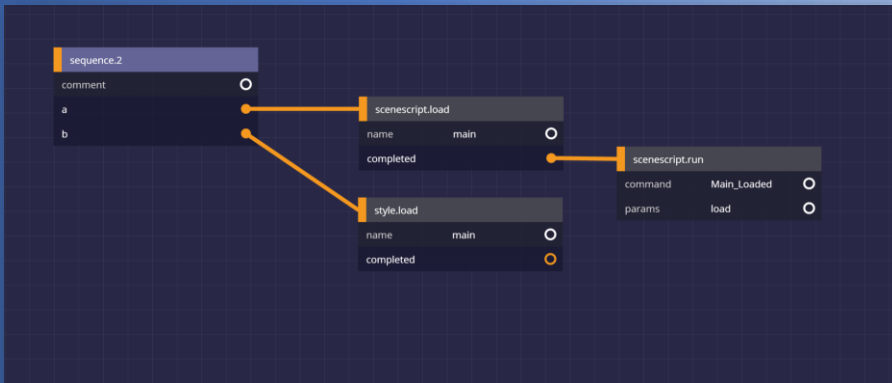
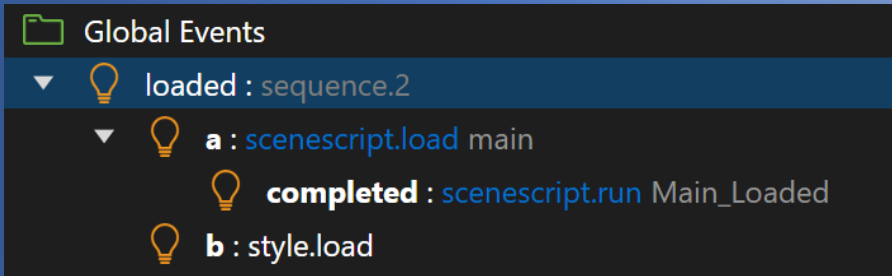


- Create and manage scene elements
- Dozens of scene elements: grid, label, textbox, dropdown, checkbox, optionbox, listview, sceneview, blueprintview, htmlview, googlemap and a lot more!
- Element property editor
- Element style editor
- Element event editor
- Visual Scripting

Editor - Device Preview

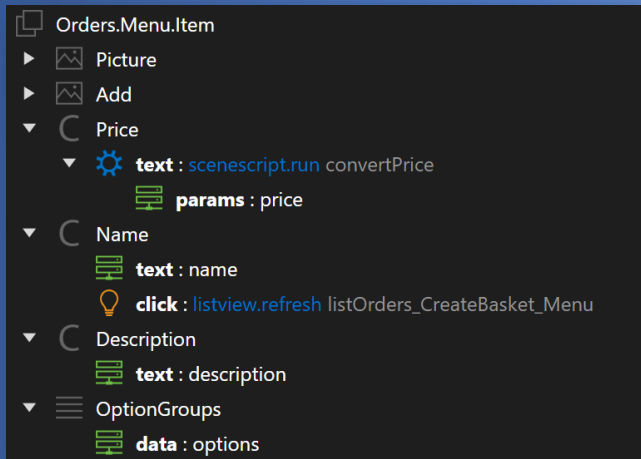


Editor - Visual Scripting

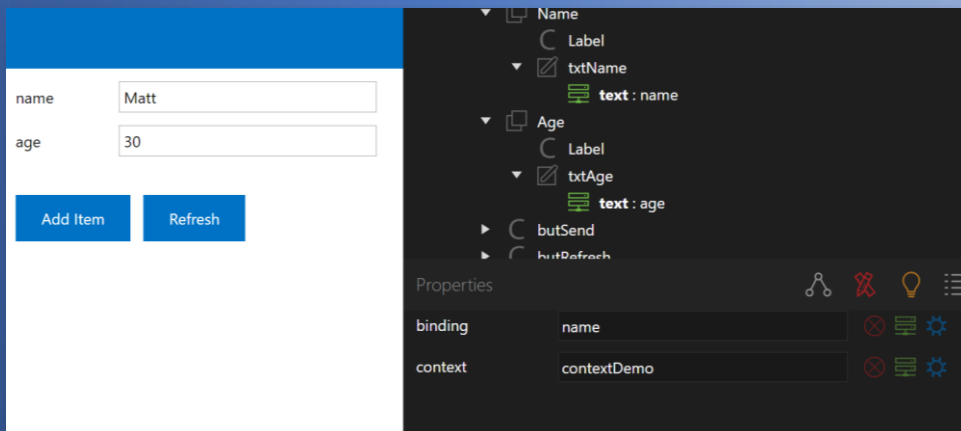


- Create simple or complex functions without a single line of code
- Can be applied to any property including styles
- 100+ built-in functions
- Create function macros
- Interact with client-side code
- Execute server-side API calls
- Schema view
- Device-size variables
- Language variables

Editor – Controls, Bindings and ContextBindings

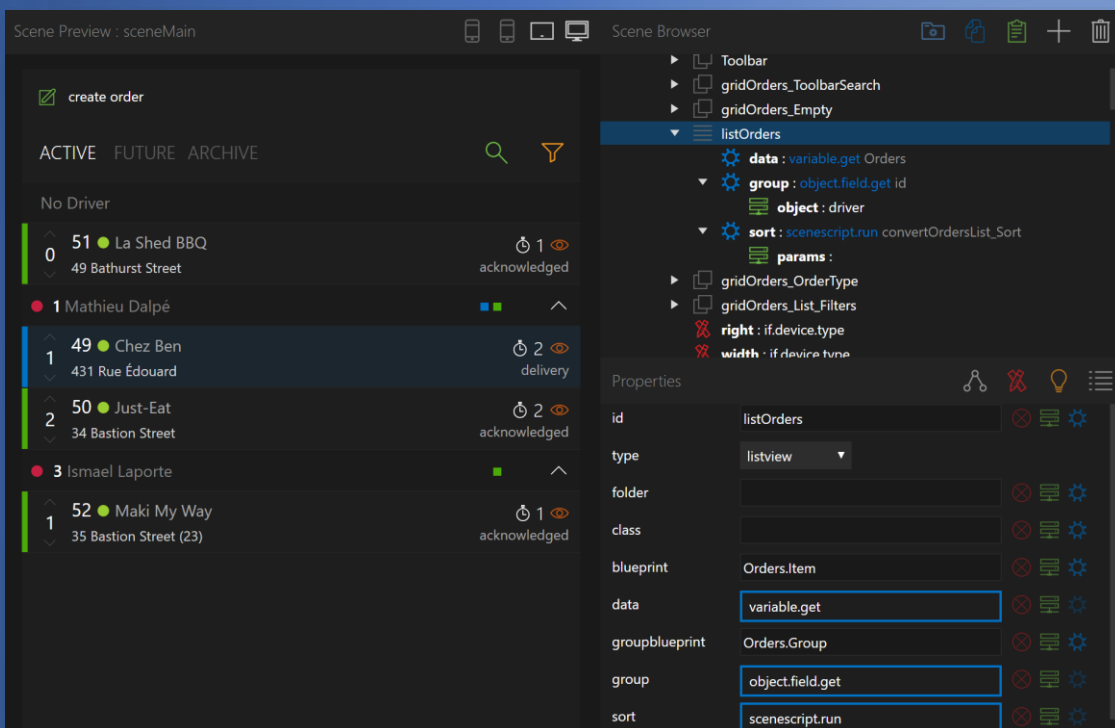


- Create controls for ListViews or other elements
- Use bindings to display data in controls
- Pass binding data to code
- Use ContextBindings to quickly apply or read data across multiple textboxes, labels, listviews or other elements.
- ContextBinding data is easy to set and get from code or visual scripting



```
var values = { name : "Matt", age : 30 }  
cs.context("contextDemo").data(values);  
var getvalues = cs.context("contextDemo").data();
```


Editor - ListView



- Display arrays of data easily
- Render huge lists with great performance thanks to virtualization
- Use controls for items and group headers
- Simple and complex grouping property
- Simple and complex sorting property
- ListView data is easy to set and get from code or visual scripting

```
var list = [{ name : "Matt", age : 30 }, { name : "Mike", age : 32 }];  
cs("listDemo").data(list);  
var getlist = cs("listDemo").data();
```

Editor – API Documenter

Easily enter and test API functions for your team or external developers who want to use your API

GeoDispatch API

- main**
 - login
- group**
 - merchants:list
 - orders:list
- merchant**
 - menu:categories
 - menu:categories_create
 - menu:categories_save
 - menu:categories_delete
 - menu:categories_setdefault
 - menu:items
 - menu:items_single
 - menu:items_create
 - menu:items_save
 - menu:items_duplicate
 - menu:items_delete
 - settings:zones
 - settings:zones_create
 - settings:zones_save
 - settings:zones_delete

login

Login for admins, merchants, drivers and accounts

<https://portal.geodispatch.net/api/main/login>

Parameters

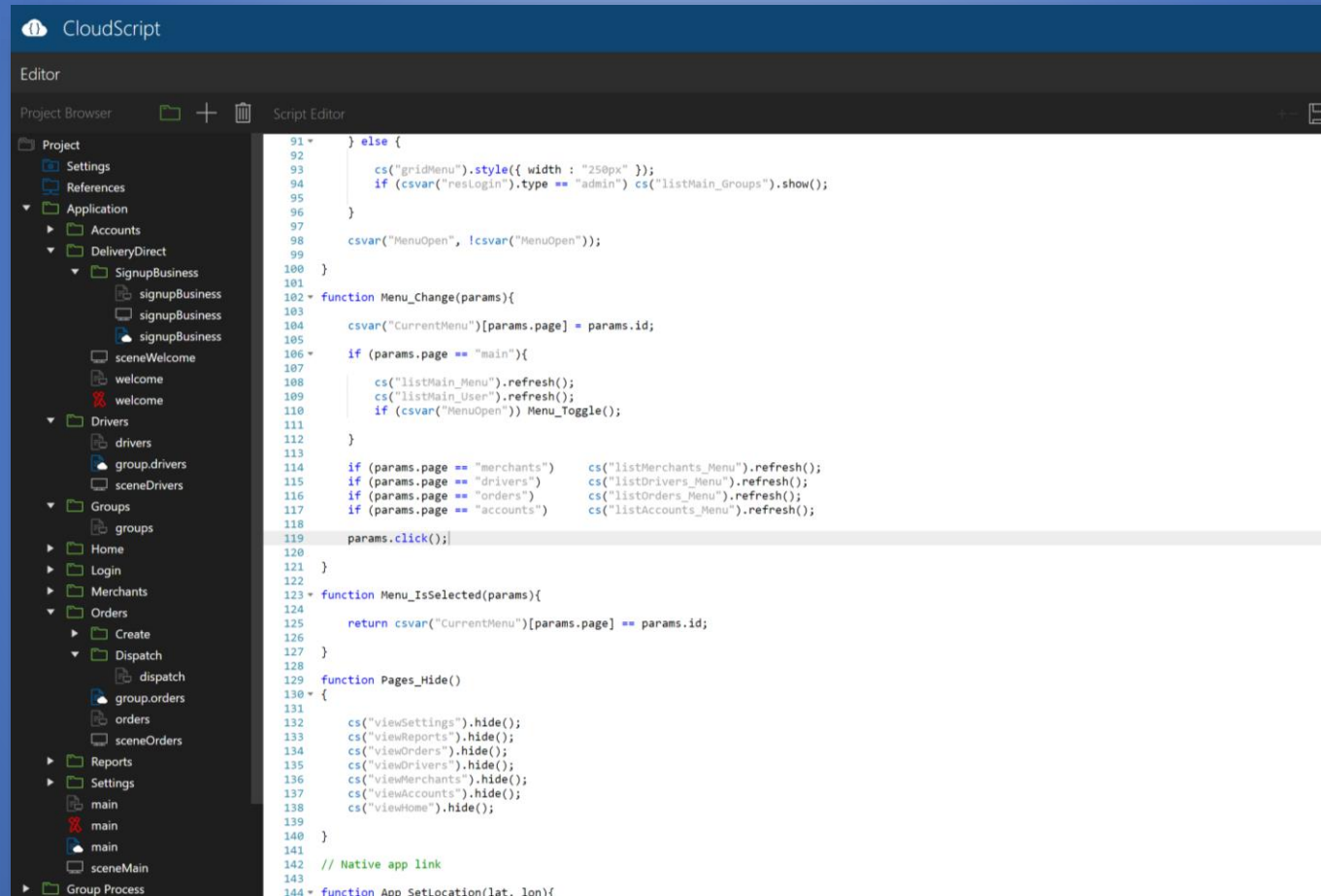
Default Session Check

type	string	<input type="text" value="admin"/>	Type of user (admin, merchant, driver, account)
username	string	<input type="text" value="matt@pmrd.net"/>	User name
password	string	<input type="password" value="*****"/>	Password
group	string	<input type="text"/>	Group (not required for admin)

[Send](#)

```
{
  "success": true,
  "type": "admin",
  "sessionid": "cc7c62fe-59a6-4fd2-be2b-9ef85377d2d9",
  "name": "Mathieu DaIpé",
  "groups": [
    {
      "name": "MenuLivraison.ca",
      "value": "menulivraison.ca"
    },
    {
      "name": "OrderIn.ca",
      "value": "orderin.ca"
    },
    {
      "name": "DeliveryDirect.ca",
      "value": "deliverydirect.ca"
    }
  ]
}
```

Code Editor



The image shows the CloudScript Code Editor interface. On the left is a Project Browser with a tree view of a project structure. The main area is a Script Editor displaying JavaScript code. The code includes a function `Menu_Change` that updates the current menu and refreshes content based on the page type. It also includes a `Pages_Hide` function to hide various views.

```
91 *   } else {
92
93     cs("gridMenu").style({ width : "250px" });
94     if (csvgar("resLogin").type == "admin") cs("listMain_Groups").show();
95   }
96 }
97
98 csvar("MenuOpen", !csvgar("MenuOpen"));
99
100 }
101
102 * function Menu_Change(params){
103
104   csvar("CurrentMenu")[params.page] = params.id;
105
106 *   if (params.page == "main"){
107
108     cs("listMain_Menu").refresh();
109     cs("listMain_User").refresh();
110     if (csvgar("MenuOpen")) Menu_Toggle();
111   }
112
113
114   if (params.page == "merchants")   cs("listMerchants_Menu").refresh();
115   if (params.page == "drivers")     cs("listDrivers_Menu").refresh();
116   if (params.page == "orders")     cs("listOrders_Menu").refresh();
117   if (params.page == "accounts")   cs("listAccounts_Menu").refresh();
118
119   params.click();
120
121 }
122
123 * function Menu_IsSelected(params){
124
125   return csvar("CurrentMenu")[params.page] == params.id;
126 }
127
128
129 function Pages_Hide()
130 * {
131
132   cs("viewSettings").hide();
133   cs("viewReports").hide();
134   cs("viewOrders").hide();
135   cs("viewDrivers").hide();
136   cs("viewMerchants").hide();
137   cs("viewAccounts").hide();
138   cs("viewHome").hide();
139 }
140
141
142 // Native app link
143
144 * function App_SetLocation(lat, lon){
```

Code Editor – IntelliSense

CloudScript functions are documented, context-sensitive and appear automatically

```
61   csvar("CurrentMenu").orders = "info";
62
63   csvar("CurrentAccount", {});
64   csvar("CurrentTeams", {});
65   csvar("CurrentDriver", {});
66   csvar("CurrentOrder", {});
67   cs.
68   cs.
69   cs.
70   cs.
71   cs.
72   cs.
73   cs.
74   cm.
75   arguments
76   cs.
77
78 }
```

execute	scenescript	<i>fn(string process, string command, ? params, fn(?) callback)</i>
context	scenescript	Executes a comand on a local CloudScript process and returns the data asynchronously.
loadscript	scenescript	
executeremote	scenescript	
document	keyword	
window	keyword	
prototype	keyword	
arguments	keyword	

```
66   csvar("CurrentOrder", {});
67   csvar("Current
68   cs("txtLogin_
69   cs("txtLogin_
70   cs("txtLogin_
71   cs("txtLogin_
72   cs("cmbLogin_
73   cs("cmbLogin_
74   cmbLogin_Type
75   data
76   cs("lblText").
77
78 }
```

sort	scenescript	<i>fn(string text) -> string</i>
scene	scenescript	Get or set the text for the specified object
refresh	scenescript	
show	scenescript	
focus	scenescript	
style	scenescript	
text	scenescript	
data	scenescript	

Code Editor – Debugging

The console provides real-time and detailed information when compilation errors or exceptions occur

```
Console

5:58:16 AM  cloudscript      System restarted
5:58:15 AM  cloudscript      CloudScript saved : signupBusiness
5:57:57 AM  signupprocess    SyntaxError: Unexpected identifier at signupprocess:4:11 -> functions Init(params) {
5:57:57 AM  cloudscript      System restarted
5:57:55 AM  cloudscript      CloudScript saved : signupBusiness
```

The code editor alerts you when your syntax is invalid

```
⚠ 4 ▾ function Init(params {
5
6     CloudScript.LoadScript("classes");
7     CloudScript.LoadScript("utilities");
8
9     CloudScript.LoadScript("moment");
10    CloudScript.LoadScript("moment-timezone");
11
12    API_Key = param.api_key;
13
```

Let's get technical a bit...

Server – Built-in Functions

- FileSystem
- WebClient
- WebSocket
- Email
- Encode
- And more!

```
// FileSystem
FileSystem.CreateFolder("demo");
let file = FileSystem.Load("demo/filename");
FileSystem.Save("demo/filename", file);

// WebClient
let clientDemo = new WebClient();

let values = { name : "Matt", email : "matt@pmrd.net", password: "" };

clientDemo.UploadValues("http://someserver.com/register.php", values, function(result){
    CloudScript.Log(result)
})

// WebSocket Server
let socketDemo = new WebSocket("https://demo.com:443/socketdemo", Open, Receive, Close);

function Open(id, ip) { }
function Receive(id, data) { }
function Close(id) { }

socketDemo.Send(id, "Hello World!");
```

Server - API Communication

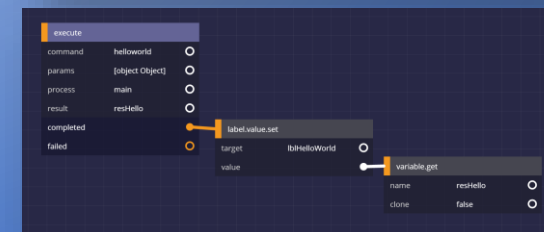
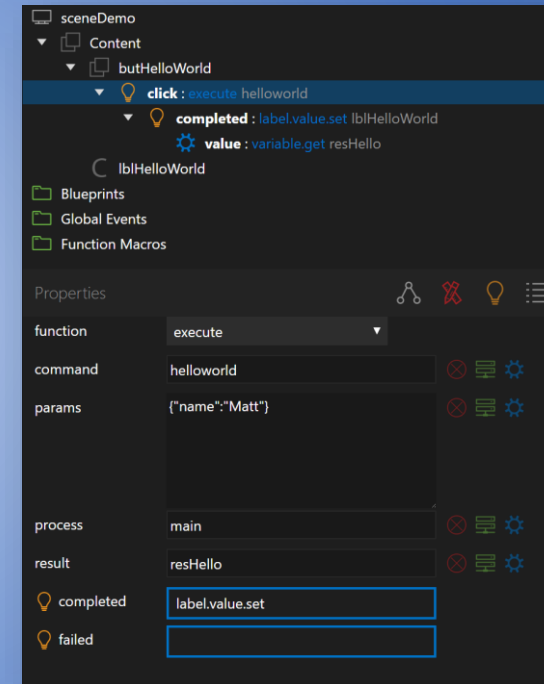
Server-side code

```
function Execute(command, params, domain, ip) {  
  
    // Hello World Command  
    if (command == "helloworld"){  
  
        // Return "Hello Name!"  
        return "Hello " + params.name + "!";  
  
    }  
}
```

Client-side code

```
function clickHelloWorld(){  
  
    // Call server function  
    cs.execute("main", "helloworld", { name : "Matt" }, function(result){  
  
        // Apply result to label  
        cs("lblHelloWorld").text(result);  
  
    })  
}
```

Visual scripting



Server - Threaded Processes

By default, CloudScript creates a server-side process called "main" with the server-side script "main". You can however create more processes with other server-side scripts. These processes run on separate threads and will not lock-up other processes when running large tasks. The minimum server-side code required for a process script is an **Init** and **Execute** function shown below.

```
function Init(params){  
  
}  
  
function Execute(command, params, domain, ip){  
  
}
```

A new process can be created from server-side code using the built-in **CreateProcess** function and accessed from server-side code using the **Execute**, **ExecuteRemote**, **ExecuteAsync** or **ExecuteRemoteAsync** functions or on the client-side using the **execute** or **executeremote** function or with visual scripting.

```
CloudScript.CreateProcess("demo", "demoscript", optionalparams);
```

Server-side code

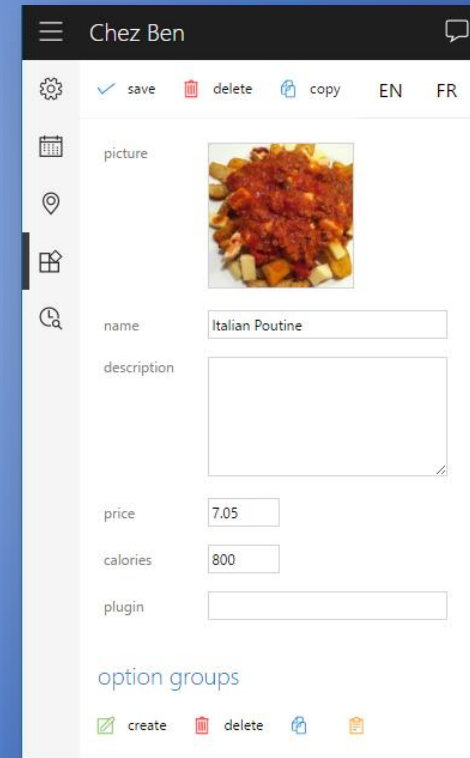
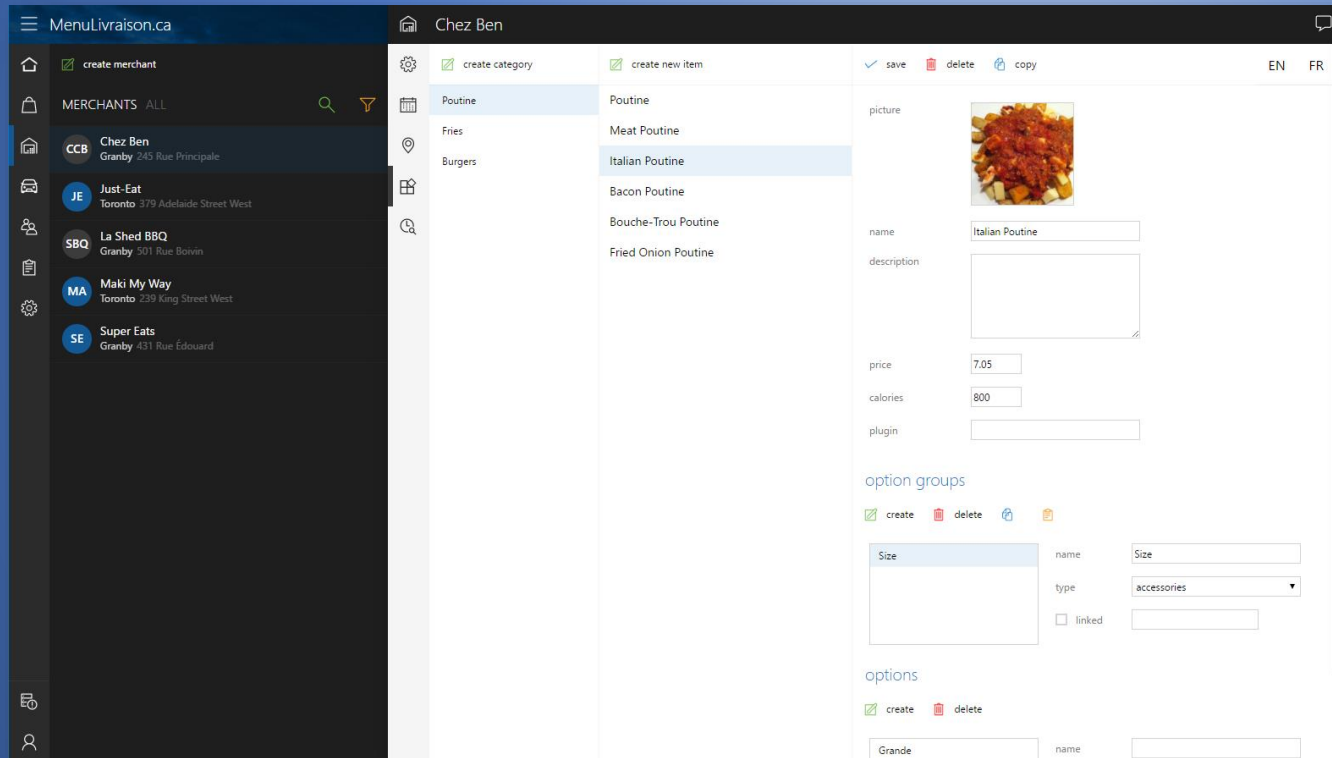
```
// Synchronous local  
let result = CloudScript.Execute("demo", "helloworld", { name : "Matt" });  
CloudScript.Log(result);  
  
// Asynchronous local  
CloudScript.ExecuteAsync("demo", "helloworld", { name : "Matt" }, function(result) {  
    CloudScript.Log(result);  
})  
  
// Synchronous remote  
let result = CloudScript.ExecuteRemote("someserver.com", "demo", "helloworld", { name : "Matt" });  
CloudScript.Log(result);  
  
// Asynchronous remote  
CloudScript.ExecuteRemoteAsync("someserver.com", "demo", "helloworld", { name : "Matt" }, function(result) {  
    CloudScript.Log(result);  
})
```

Client-side code

```
// Local  
cs.execute("demo", "helloworld", { name : "Matt" }, function(result){  
    console.log(result);  
})  
  
//Remote  
cs.executeremote("someserver.com", "demo", "helloworld", { name : "Matt" }, function(result){  
    console.log(result);  
})
```

CloudScript – Today

CloudScript apps are currently being used internally by our delivery companies with GeoDispatch. CloudScript also powers the order bridges for our delivery partners such as Just-Eat, Cara, Menu.ca, Boston Pizza, Super Eats and many more.



CloudScript – Tomorrow

Users will be able to sign up and create instantly ready-to-use projects from CloudScript.net or from the Microsoft Azure Portal. They will be able to select how many processor cores and memory they need for their project and pay a monthly fee accordingly.

