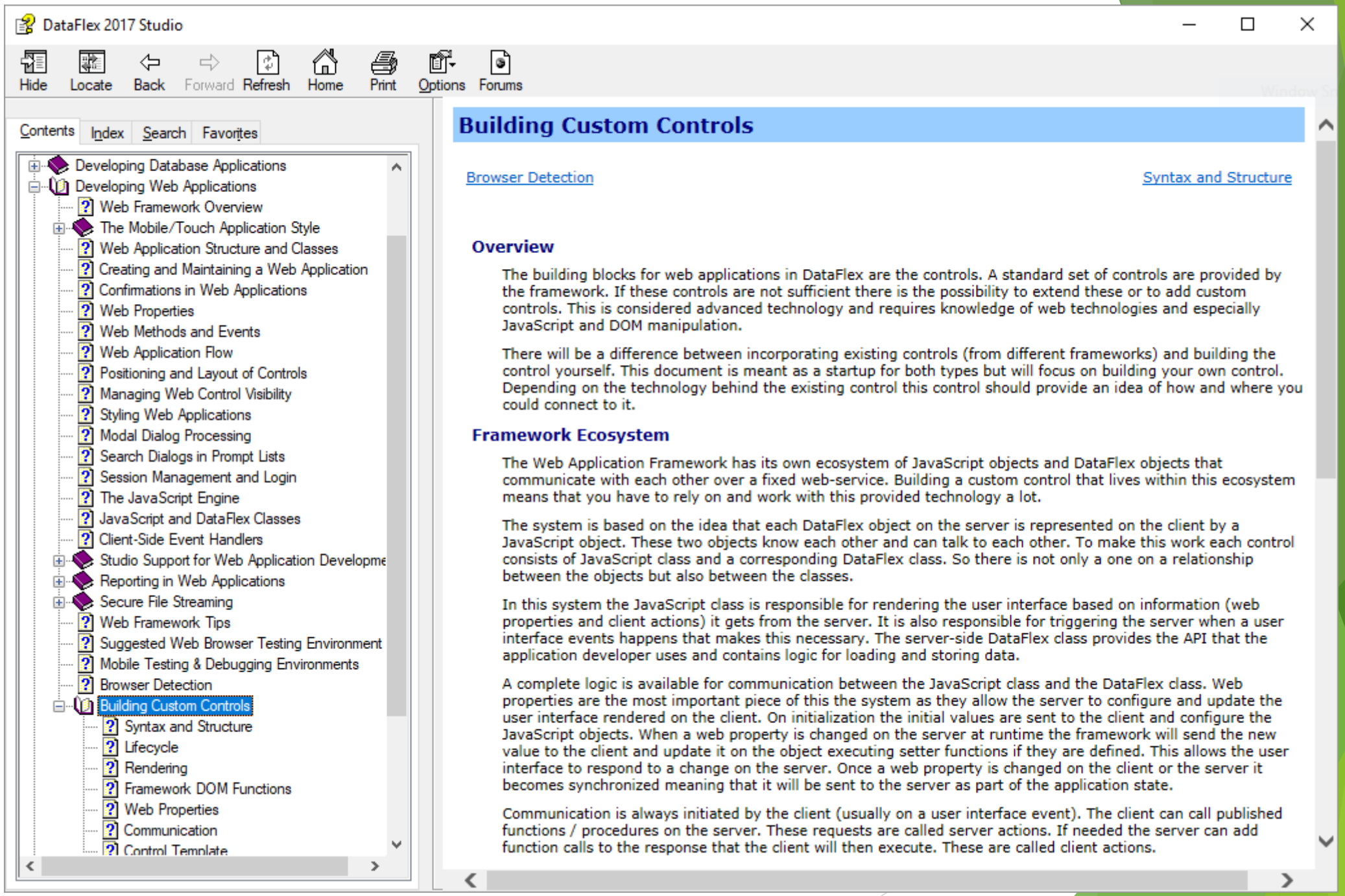
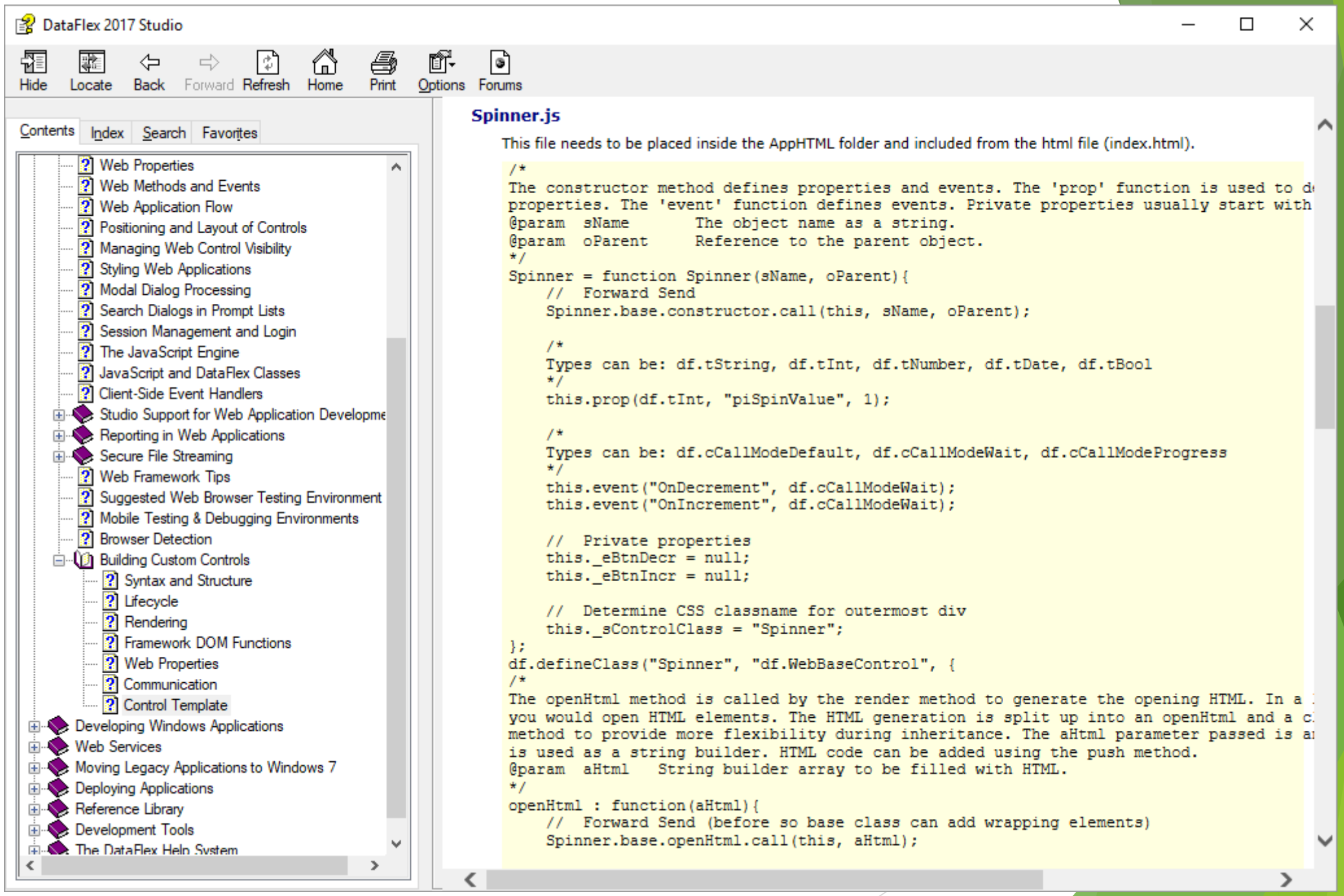
Custom Components

Using ECMAScript

Jakob Kruse | Sture ApS


DataFlex 2017 Studio
Hide
Locate
Back
Forward
Refresh
Home
Print
Options
Forums
Contents
Index
Search
Favorites
Developing Database Applications
Developing Web Applications
Web Framework Overview
The Mobile/Touch Application Style
Web Application Structure and Classes
Creating and Maintaining a Web Application
Confirmations in Web Applications
Web Properties
Web Methods and Events
Web Application Flow
Positioning and Layout of Controls
Managing Web Control Visibility
Styling Web Applications
Modal Dialog Processing
Search Dialogs in Prompt Lists
Session Management and Login
The JavaScript Engine
JavaScript and DataFlex Classes
Client-Side Event Handlers
Studio Support for Web Application Developme
Reporting in Web Applications
Secure File Streaming
Web Framework Tips
Suggested Web Browser Testing Environment
Mobile Testing & Debugging Environments
Browser Detection
Building Custom Controls
Syntax and Structure
Lifecycle
Rendering
Framework DOM Functions
Web Properties
Communication
Control Template
Building Custom Controls
Browser Detection
Syntax and Structure
Overview
The building blocks for web applications in DataFlex are the controls. A standard set of controls are provided by the framework. If these controls are not sufficient there is the possibility to extend these or to add custom controls. This is considered advanced technology and requires knowledge of web technologies and especially JavaScript and DOM manipulation.
There will be a difference between incorporating existing controls (from different frameworks) and building the control yourself. This document is meant as a startup for both types but will focus on building your own control. Depending on the technology behind the existing control this control should provide an idea of how and where you could connect to it.
Framework Ecosystem
The Web Application Framework has its own ecosystem of JavaScript objects and DataFlex objects that communicate with each other over a fixed web-service. Building a custom control that lives within this ecosystem means that you have to rely on and work with this provided technology a lot.
The system is based on the idea that each DataFlex object on the server is represented on the client by a JavaScript object. These two objects know each other and can talk to each other. To make this work each control consists of JavaScript class and a corresponding DataFlex class. So there is not only a one on a relationship between the objects but also between the classes.
In this system the JavaScript class is responsible for rendering the user interface based on information (web properties and client actions) it gets from the server. It is also responsible for triggering the server when a user interface events happens that makes this necessary. The server-side DataFlex class provides the API that the application developer uses and contains logic for loading and storing data.
A complete logic is available for communication between the JavaScript class and the DataFlex class. Web properties are the most important piece of this the system as they allow the server to configure and update the user interface rendered on the client. On initialization the initial values are sent to the client and configure the JavaScript objects. When a web property is changed on the server at runtime the framework will send the new value to the client and update it on the object executing setter functions if they are defined. This allows the user interface to respond to a change on the server. Once a web property is changed on the client or the server it becomes synchronized meaning that it will be sent to the server as part of the application state.
Communication is always initiated by the client (usually on a user interface event). The client can call published functions / procedures on the server. These requests are called server actions. If needed the server can add function calls to the response that the client will then execute. These are called client actions.

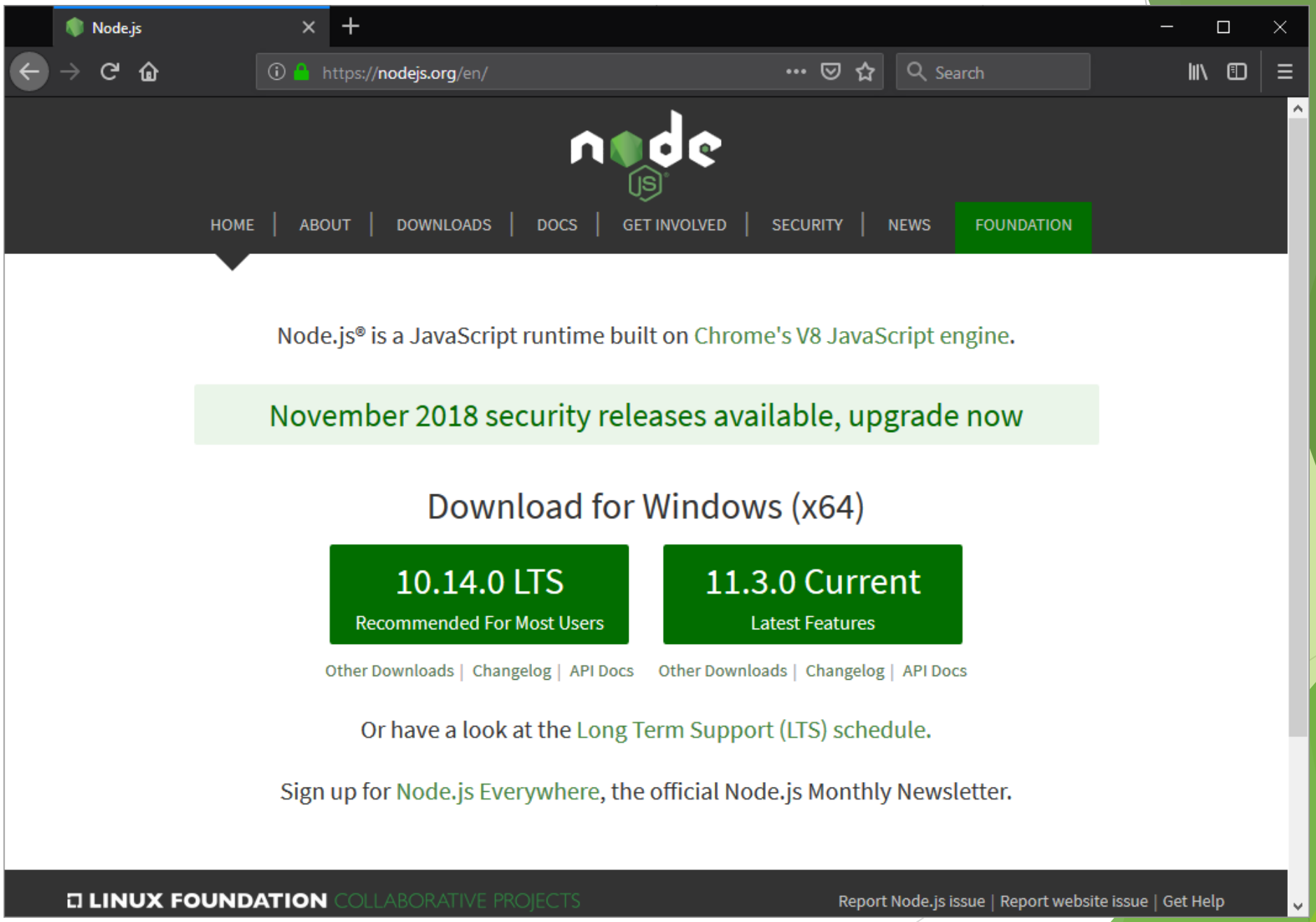

DataFlex 2017 Studio
Hide
Locate
Back
Forward
Refresh
Home
Print
Options
Forums
Contents
Index
Search
Favorites
Web Properties
Web Methods and Events
Web Application Flow
Positioning and Layout of Controls
Managing Web Control Visibility
Styling Web Applications
Modal Dialog Processing
Search Dialogs in Prompt Lists
Session Management and Login
The JavaScript Engine
JavaScript and DataFlex Classes
Client-Side Event Handlers
Studio Support for Web Application Developme
Reporting in Web Applications
Secure File Streaming
Web Framework Tips
Suggested Web Browser Testing Environment
Mobile Testing & Debugging Environments
Browser Detection
Building Custom Controls
Syntax and Structure
Lifecycle
Rendering
Framework DOM Functions
Web Properties
Communication
Control Template
Developing Windows Applications
Web Services
Moving Legacy Applications to Windows 7
Deploying Applications
Reference Library
Development Tools
The DataFlex Help System
Spinner.js
This file needs to be placed inside the AppHTML folder and included from the html file (index.html).
/*
The constructor method defines properties and events. The 'prop' function is used to d
properties. The 'event' function defines events. Private properties usually start with
@param sName The object name as a string.
@param oParent Reference to the parent object.
*/
Spinner = function Spinner(sName, oParent){
// Forward Send
Spinner.base.constructor.call(this, sName, oParent);
/*
Types can be: df.tString, df.tInt, df.tNumber, df.tDate, df.tBool
*/
this.prop(df.tInt, "piSpinValue", 1);
/*
Types can be: df.cCallModeDefault, df.cCallModeWait, df.cCallModeProgress
*/
this.event("OnDecrement", df.cCallModeWait);
this.event("OnIncrement", df.cCallModeWait);
// Private properties
this._eBtnDecr = null;
this._eBtnIncr = null;
// Determine CSS classname for outermost div
this._sControlClass = "Spinner";
};
df.defineClass("Spinner", "df.WebBaseControl", {
/*
The openHtml method is called by the render method to generate the opening HTML. In a
you would open HTML elements. The HTML generation is split up into an openHtml and a c
method to provide more flexibility during inheritance. The aHtml parameter passed is a
is used as a string builder. HTML code can be added using the push method.
@param aHtml String builder array to be filled with HTML.
*/
openHtml : function(aHtml){
// Forward Send (before so base class can add wrapping elements)
Spinner.base.openHtml.call(this, aHtml);

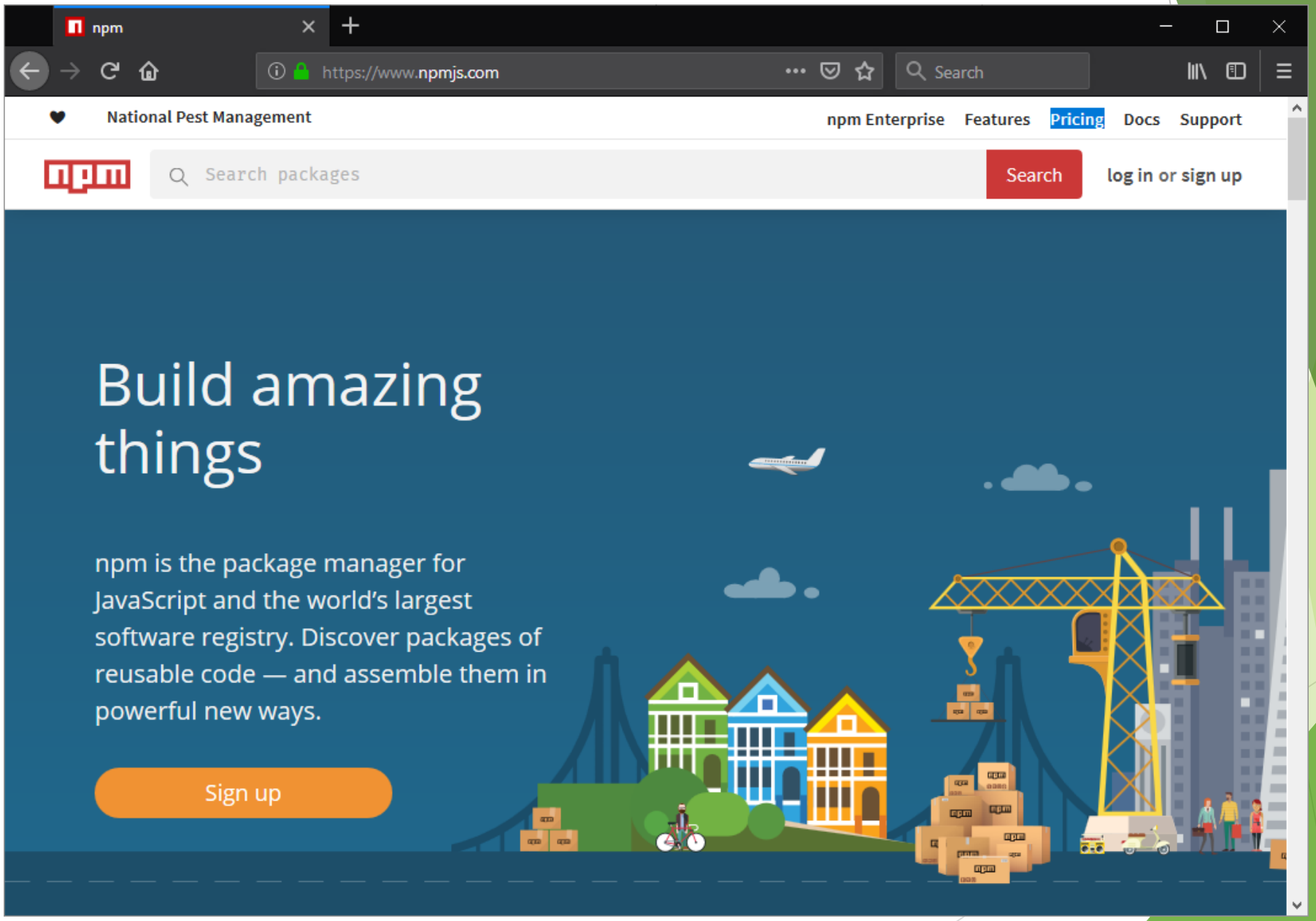
JavaScript

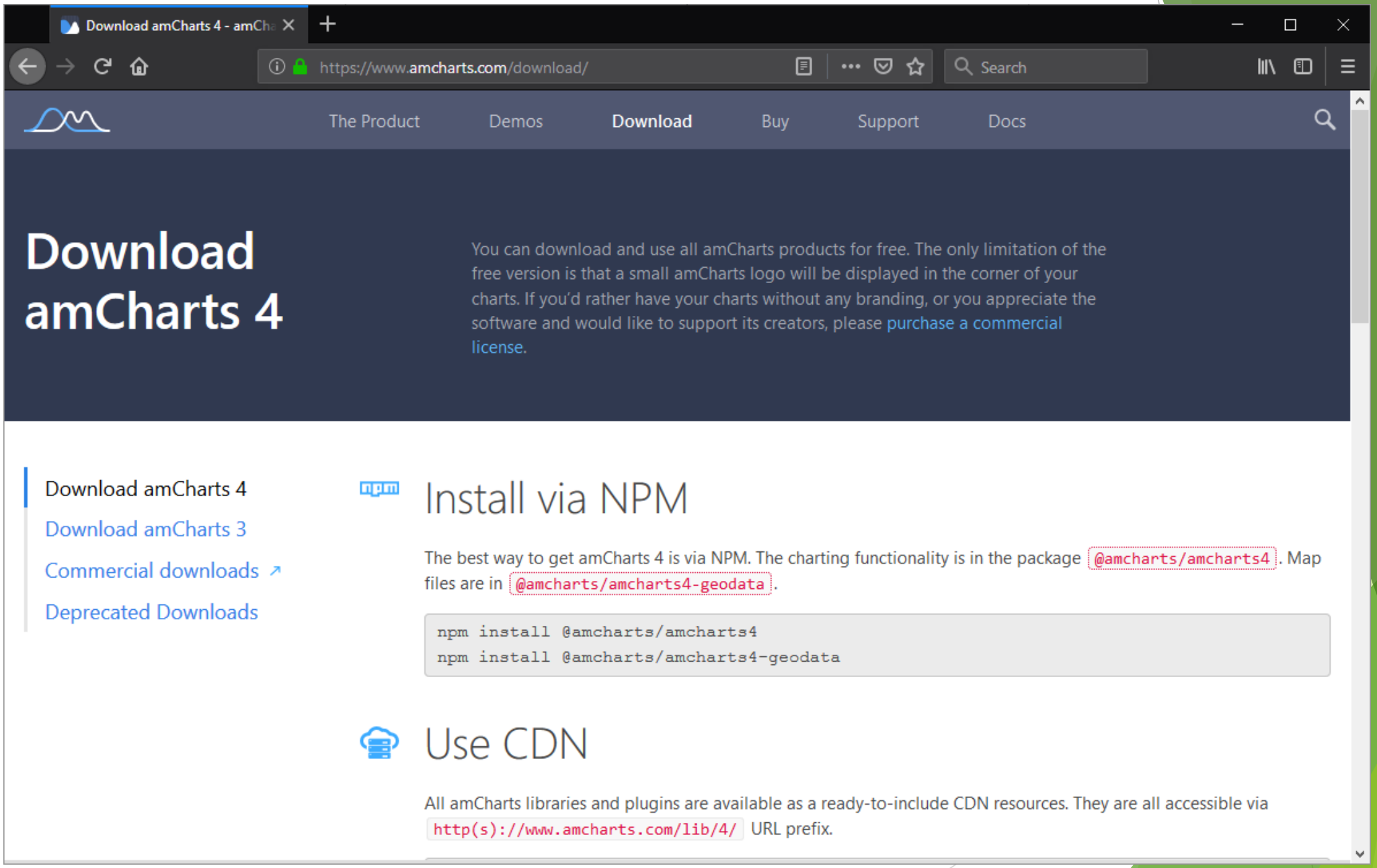
```
MyComponent = function MyComponent(sName, oParent) {
    MyComponent.base.constructor.call(this, sName, oParent);

    this.prop(df.tString, 'psName', 'Scanduc');
};
df.defineClass('MyComponent', 'df.WebBaseControl', {
    openHtml: function (aHtml) {
        MyComponent.base.openHtml.call(this, aHtml);
        aHtml.push('<div>');
    },

    closeHtml: function (aHtml) {
        aHtml.push('</div>');
        MyComponent.base.closeHtml.call(this, aHtml);
    },

    afterRender: function () {
        this._eControl = df.dom.query(this._eElem, 'div');
        MyComponent.base.afterRender.call(this);
        this.set_psName(this.psName);
    },

    set_psName: function (sVal) {
        if (this._eControl) {
            df.dom.setText(this._eControl, 'Hello ' + sVal);
        }
    }
});
```

ECMAScript

```
class MyComponent extends df.WebBaseControl {
    constructor(sName, oParent) {
        super(sName, oParent);

        this.prop(df.tString, 'psName', 'Scanduc');
    }

    openHtml(aHtml) {
        super.openHtml(aHtml);
        aHtml.push('<div>');
    }

    closeHtml(aHtml) {
        aHtml.push('</div>');
        super.closeHtml(aHtml);
    }

    afterRender() {
        this._eControl = df.dom.query(this._eElem, 'div');
        super.afterRender();
        this.set_psName(this.psName);
    }

    set_psName(sVal) {
        if (this._eControl) {
            df.dom.setText(this._eControl, 'Hello ' + sVal);
        }
    }
}
```


Node.js
https://nodejs.org/en/
Search
HOME
ABOUT
DOWNLOADS
DOCS
GET INVOLVED
SECURITY
NEWS
FOUNDATION
Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.
November 2018 security releases available, upgrade now
Download for Windows (x64)
10.14.0 LTS
Recommended For Most Users
11.3.0 Current
Latest Features
Other Downloads | Changelog | API Docs
Other Downloads | Changelog | API Docs
Or have a look at the Long Term Support (LTS) schedule.
Sign up for Node.js Everywhere, the official Node.js Monthly Newsletter.
LINUX FOUNDATION COLLABORATIVE PROJECTS
Report Node.js issue | Report website issue | Get Help


npm
https://www.npmjs.com
Search
National Pest Management
npm Enterprise
Features
Pricing
Docs
Support
Search packages
Search
log in or sign up
Build amazing things
npm is the package manager for JavaScript and the world's largest software registry. Discover packages of reusable code — and assemble them in powerful new ways.
Sign up


Download amCharts 4 - amCha
https://www.amcharts.com/download/
Search
The Product
Demos
Download
Buy
Support
Docs
Download amCharts 4
You can download and use all amCharts products for free. The only limitation of the free version is that a small amCharts logo will be displayed in the corner of your charts. If you'd rather have your charts without any branding, or you appreciate the software and would like to support its creators, please purchase a commercial license.
Download amCharts 4
Download amCharts 3
Commercial downloads
Deprecated Downloads
npm
Install via NPM
The best way to get amCharts 4 is via NPM. The charting functionality is in the package @amcharts/amcharts4. Map files are in @amcharts/amcharts4-geodata.
npm install @amcharts/amcharts4
npm install @amcharts/amcharts4-geodata
Use CDN
All amCharts libraries and plugins are available as a ready-to-include CDN resources. They are all accessible via http(s)://www.amcharts.com/lib/4/ URL prefix.

```
import * as core from "@amcharts/amcharts4/core";
import * as charts from "@amcharts/amcharts4/charts";
import animated from "@amcharts/amcharts4/themes/animated";

core.useTheme(animated);

class AmChartsComponent extends df.WebBaseControl {
    constructor(sName, oParent) {
        super(sName, oParent);
        this._eChart = null;
    }

    openHtml(aHtml) {
        super.openHtml(aHtml);
        aHtml.push('<div class="AmChartsComponent">');
    }

    closeHtml(aHtml) {
        aHtml.push('</div>');
        super.closeHtml(aHtml);
    }

    afterRender() {
        super.afterRender();
        this._eControl = df.dom.query(this._eControlWrp, 'div');
        this.sizeHeight(-1);
    }

    setChartConfig(sType, sVal) {
        this._eChart?.dispose();
        if (sVal) {
            const config = JSON.parse(sVal);
            this._eChart = core.createFromConfig(config, this._eControl, sType);
        } else {
            this._eChart = null;
            this._eControl.innerHTML = '';
        }
    }
}

global.AmChartsComponent = AmChartsComponent;
```

```
Use cWebBaseControl.pkg

Class cAmChart is a cWebBaseControl

    Procedure Construct_Object
        Forward Send Construct_Object

        Set psJSClass to "AmChartsComponent"

        { WebProperty=Client }
        Property Integer piHeight 0
    End_Procedure

    Procedure End_Construct_Object
        Forward Send End_Construct_Object
    End_Procedure

    Procedure SetChartConfig String sType String sJSON
        String[] params
        Move sType to params[0]
        Move sJSON to params[1]
        Send ClientAction "setChartConfig" params
    End_Procedure

End_Class
```

Example: amCharts component

https://github.com/stureaps/AmChartsComponent

Thank you!