1010DATA

# Embedded QuickApps

(212) 405.1010 | info@1010data.com | Follow: @1010data | www.1010data.com

# Contents

# Embedded QuickApps

QuickApps can be embedded inside web applications built outside of 1010data, and they can interact with elements in the application by sending and receiving values that are meaningful to both.

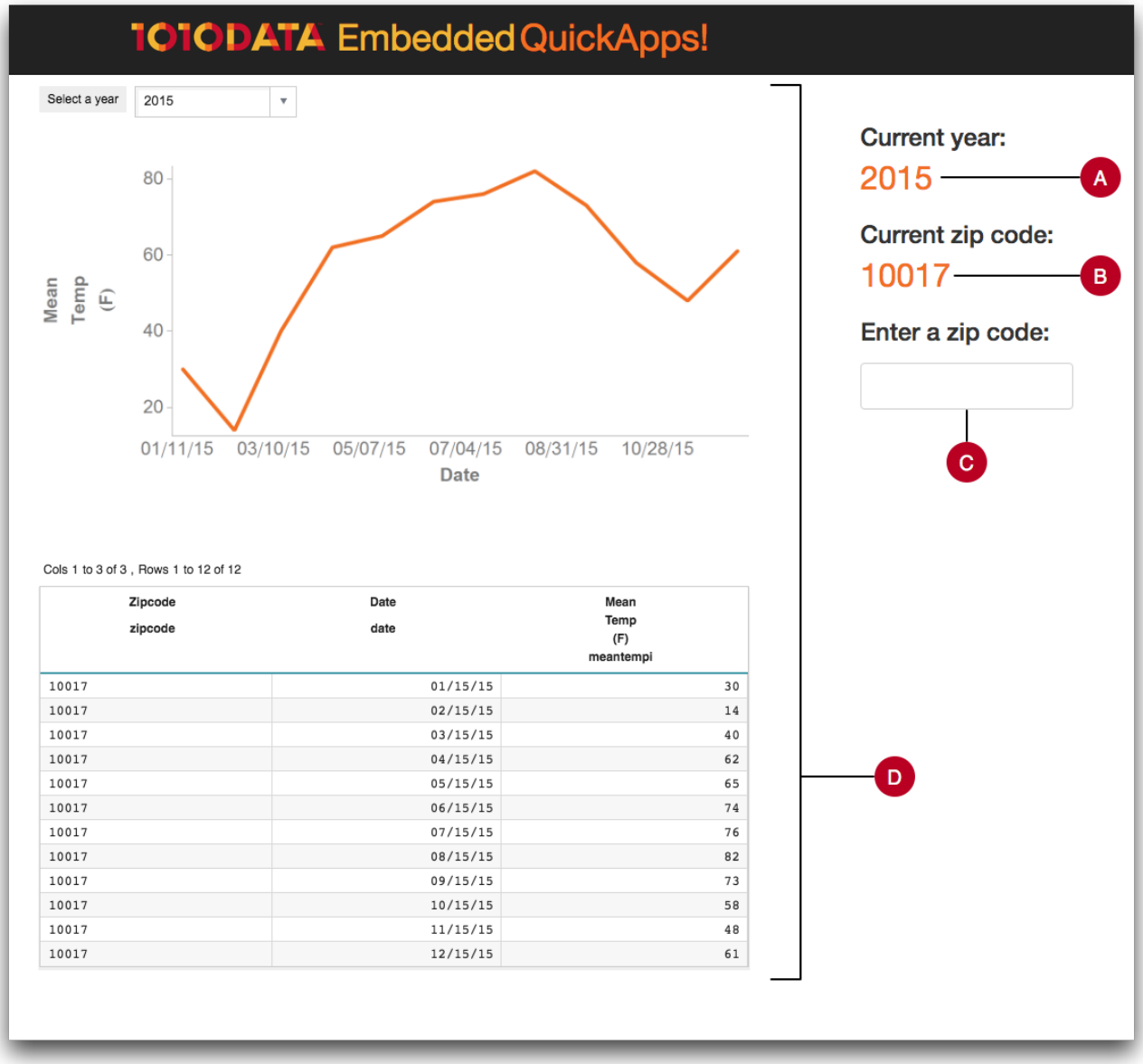
Embedding a QuickApp in a website that is external to 1010data is as simple as creating an `<iframe>` and pointing it at the QuickApp's URL. For reference, a QuickApp can be accessed directly using the following URL format:

`https://www2.1010data.com/cgi-bin/`*`[VERSION]`*`/quickapp?path=`*`[PATH_TO_QUICKAPP]`*

Pointing a web browser to a valid QuickApp URL will prompt the user for 1010data credentials and then display the QuickApp in *standalone* mode. This means that the user can only interact with the UI elements of the QuickApp, not with any of the other components of the 1010data web interface, such as the **Folder and Tables** browser. Standalone QuickApps are useful if the objective is to avoid the need for the end user to log into 1010data and manually open the QuickApp. This feature also permits embedding a QuickApp into another webpage via an `<iframe>` HTML element.

While the ability to embed a standalone QuickApp is useful in and of itself, it becomes more so when the QuickApp can interact with an enclosing web application. This can be accomplished with some basic event handling and message construction on the JavaScript side, and two special widgets, `transmitter` and `receiver`, on the QuickApp side.

This tutorial demonstrates the critical components of such interactions using a simple web application. The following is an annotated screenshot of the sample application:




**A. Current year**

Displays the value of the current year selected.

This value is used to filter the data displayed in the chart and grid within the embedded QuickApp.

In this example, the input for the year is provided via a drop-down menu in the embedded QuickApp, not the enclosing web application.

**B. Current zip code**

Displays the value of the current zip code entered.

When first launched, the enclosing web application will show an initial value for the zip code.

The value can be changed via the **Enter a zip code** field in the web application.

**C. Enter a zip code**

This field accepts a zip code value.

If a valid zip code is entered, the data displayed in the QuickApp will reflect the new value.

The value is also shown under **Current zip code** in the web application.

**D. Embedded QuickApp**

The 1010data QuickApp that is embedded in the external web application via an `<iframe>`. The QuickApp displays a drop-down menu to select the desired year as well as a chart and grid that show the results of the current selections.

The process of embedding a QuickApp in a web application is fast and fun! With a little bit of JavaScript and some help from `transmitter` and `receiver`, QuickApps can be even more customized and more flexible.

## Sample QuickApp

This basic QuickApp provides an example of Macro Language code that sends and receives messages to and from an enclosing JavaScript application.

When communicating with an external JavaScript application, understanding how the `transmitter` and `receiver` widgets facilitate that communication is important.

In this example, the QuickApp uses the `receiver` widget to receive a message sent from a web application via the JavaScript `postMessage` method. The message specifies values that should be assigned to particular dynamic variables. See *`receiver`* in the *1010data Reference Manual* for more information on this widget.

The `transmitter` widget lets the QuickApp send a message (via a JavaScript `postMessage` call) that will be received by the encapsulating JavaScript application. The `transmitter` widget sends a message whenever a change is made to any of the dynamic variables it references. The JavaScript application will then update its variables based on the values it receives from the `transmitter` widget. See *`transmitter`* in the *1010data Reference Manual* for more information on this widget.

This example QuickApp is very basic and uses techniques that should be familiar to QuickApp developers at almost any level. Here is a basic description of what the QuickApp does:

- Provides a drop-down menu to select a year to filter the results for the data to be plotted
- Plots a line chart consisting of the mean temperature for the 15th of every month over the period of a given year for a particular zip code
- Displays the plotted data in tabular form
- Shows a progress bar while the QuickApp is running any queries

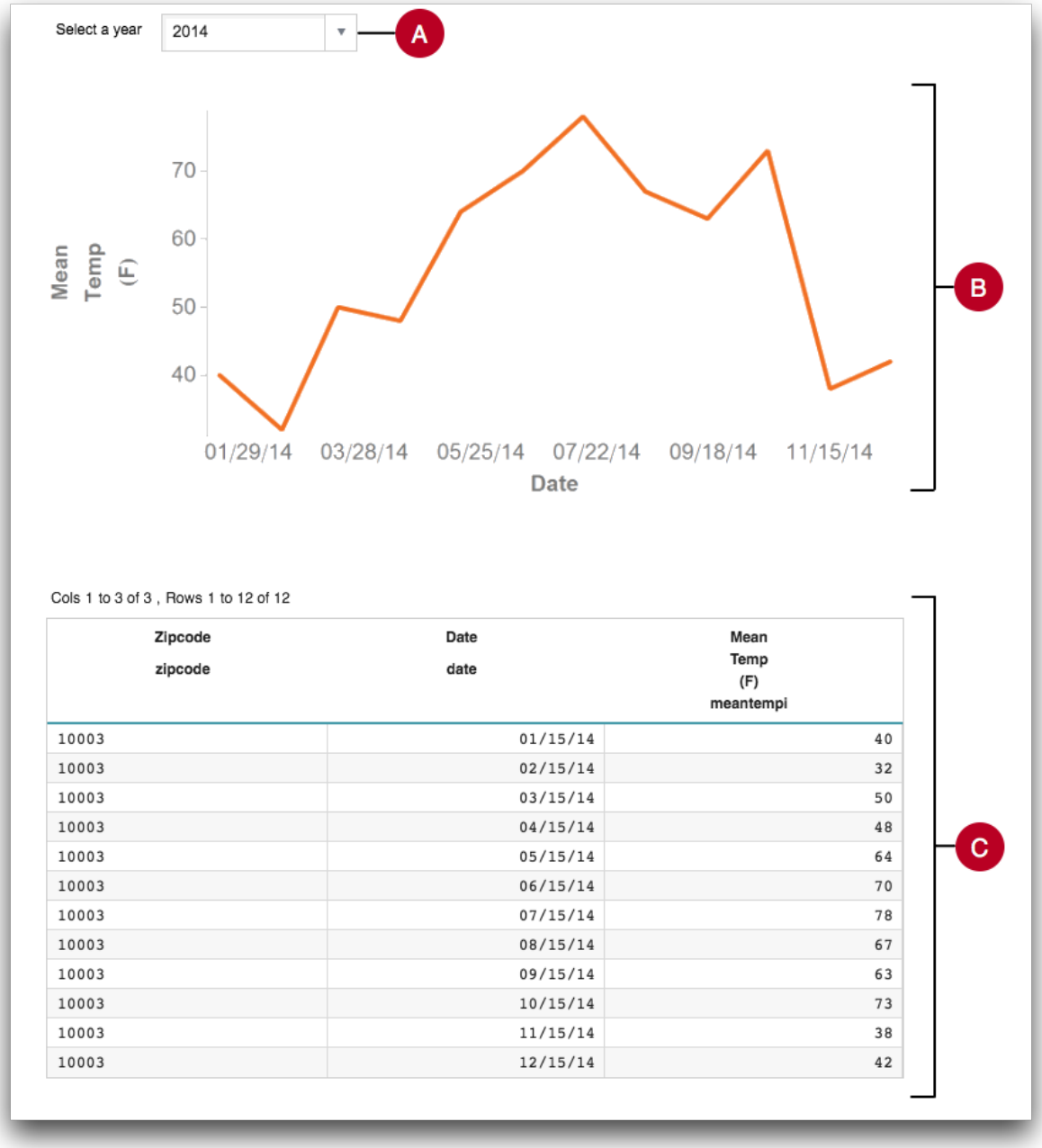
The complete QuickApp code is shown below:

```
<defblock name="sel_year">
    <table>2014;2015
    </table>
</defblock>
<defblock name="to_plot" zipcode="10017" year="2015">
  <base table="pub.demo.weather.wunderground.observed_daily"/>
  <sel value="year(date)={@year}"/>
  <sel value="zipcode={@zipcode}"/>
  <sel value="day(date)=15"/>
  <colord cols="zipcode,date,meantempi"/>
  <sort col="date" dir="up"/>
</defblock>
<dynamic year="2015" zipcode="10017">
  <widget class_="receiver"/>
  <widget class_="transmitter" message_="{@year},{@zipcode}" onrender_="1"/>
  <widget class_="progressbar" display_="top"/>
  <layout arrange_="v">
    <widget class_="dropdown" label_="Select a year"
     value_="@year" insert_="sel_year"/>
```

```
    <widget class_="graphics" width_="700"
     base_="pub.demo.weather.wunderground.observed_daily">
        <call block="to_plot" year="{@year}" zipcode="{@zipcode}"/>
        <graphspec width="600" height="400">
            <chart type="line" samples="100000">
                <style seriescolors="#F26F21" linemarkershow="0"/>
                <data x="date" y="meantempi" yrot="65"/>
            </chart>
        </graphspec>
    </widget>
    <widget class_="grid" type_="scroll"
     base_="pub.demo.weather.wunderground.observed_daily">
        <call block="to_plot" year="{@year}" zipcode="{@zipcode}"/>
    </widget>
  </layout>
</dynamic>
```

Notice that the QuickApp filters by zip code, but doesn't provide any UI for the user to make a selection on that metric. The input mechanism will be provided by the web application.

The following is an annotated screenshot of the QuickApp:




### A. Select a year

A drop-down menu that provides the end user with two options: **2014** or **2015**. The values are provided by the `<defblock>` named `sel_year`.

### B. Mean temperature over time chart

A line chart displaying the mean temperature (in Fahrenheit) for the 15th of each month of the selected year for a particular zip code. The data that will be plotted is specified via the `<defblock>` named `to_plot`.

### C. Mean temperature over time grid

A grid widget that displays the data that is being graphed in the line chart. Data for this grid is also specified by the `<defblock>` named `to_plot`.

The QuickApp itself is defined within the `<dynamic>` element, which for this example contains the following widgets:

- `receiver` - receives messages from the JavaScript application
- `transmitter` - sends messages to the JavaScript application
- `progressbar` - shows a progress bar while the QuickApp is running any queries
- `dropdown` - allows the user to select a year to filter results
- `graphics` - displays the resultant data in the form of a line chart
- `grid` - displays the resultant data in tabular form

A `<layout>` element is used to vertically arrange the three visible widgets.

> **Note:** Neither the `transmitter` nor the `receiver` widget has any visible manifestation when the QuickApp is run.

The `receiver` widget is created with the following Macro Language code:

```
<widget class_="receiver"/>
```

When it receives a message sent from a web application via the JavaScript `postMessage` method, it sets the dynamic variables specified in the message to the supplied values.

The `transmitter` widget is created with the following Macro Language code:

```
<widget class_="transmitter" message_="{@year},{@zipcode}" onrender_="1"/>
```

The `message_` attribute specifies the format of the message the `transmitter` widget will send to the JavaScript application. In this example, the message will consist of the values of the dynamic variables `year` and `zipcode`, separated by a comma. When either of these dynamic variables change, a message is sent to the JavaScript application. When it sends this message to the DOM of the enclosing web application, the application will store the value of the message as a comma-separated list and process it accordingly. Because `onrender_="1"`, a message will be sent to the containing web application when the QuickApp is initially rendered.

In addition, it is highly advisable to include a `progressbar` widget in your QuickApp if it is going to be embedded within an external web application. This gives the user a visual indicator that the QuickApp has received the inputs and is running the applicable queries. The `progressbar` widget in this example is created with the following line of code:

```
<widget class_="progressbar" display_="top"/>
```

Since `display_="top"`, the progress bar will appear over the top of the QuickApp. See *`progressbar`* in the *1010data Reference Manual* for more information.

See *Sample web application* on page 8 for an example of JavaScript code that can be used to connect this QuickApp with an external web application.

## Sample web application

HTML and JavaScript are used to build a small web application, embed a QuickApp into it, and interconnect both for shared data and user interactions.

This example web application is relatively simple and is designed specifically to demonstrate how to create interactions between an embedded QuickApp and an enclosing web application. To begin, here is a quick list of features of the sample application:

- Shows the value of the year selected in the embedded QuickApp
- Shows the value of the zip code entered in the web application
- Provides an input field for entering a new zip code value
- Provides additional branding/styling

Sample HTML and JavaScript code for the web application is shown below:

```
<!DOCTYPE html>
<html>
  <head>
    <title>1010data Embedded QuickApps!</title>
    <meta name="viewport" content="width=device-width,
          initial-scale=1.0" charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge;chrome=1" />
    <link href="1010data-DC-BS.css" rel="stylesheet" media="screen"/>
    <style>
      .rn {text-decoration: underline; }
    </style>
    <link href="styles.css" rel="stylesheet" media="screen"/>
    <script>
      window.onload = function() {
        var iframe = document.querySelector("iframe");
        var win = iframe.contentWindow;
        var zip = document.querySelector('#input_zip');
        win.postMessage({msg:  "init",
                         uid:  "[1010data_USERNAME]",
                         pswd: "[1010data_PASSWORD]"}, iframe.src);
        zip.addEventListener("change", function(e) {
          win.postMessage({set: {"zipcode": e.target.value}}, iframe.src);
          var current_zip = document.querySelector("#current_zip");
          current_zip.innerHTML = e.target.value;
        });
        window.addEventListener("message", function(e) {
          var data = e.data.split(',')
          var current_year = document.querySelector("#current_year");
          current_year.innerHTML = data[0];
          var current_zip = document.querySelector("#current_zip");
          current_zip.innerHTML = data[1];
        });
      }
    </script>
  </head>
  <body>
    <div class="navbar navbar-inverse navbar-fixed-top" role="">
      <div class="container">
        <div class="navbar-title">
            <p class="1010logo">
            <img src="1010datalogo.png"
                 style="margin-top: .7em;display: inline;float: left;"/>
            <p style="color: white;display: inline-block;
                      padding-left: .2em;font-size: 24pt;">
            <p style="display: inline;padding-top: 10px;">
            <p style="display: inline;color: #F1B434;
                      font-size: 36px;margin-top: 5px;">Embedded</p>
            <p style="display: inline;color: #F26F21;
                      font-size: 36px;margin-top: 5px;"> QuickApps!</p>
        </p>
        </div>
      </div>
    </div>
    <div class="container_exchange">
      <div class="inner">
        <div class="stuff">
          <h3>Current year:</h3>
          <div id="current_year">2015</div>
          <h3>Current zip code:</h3>
          <div id="current_zip">10017</div>
          <h3>Enter a zip code:</h3>
          <input type="field" name="zip" id="input_zip">
        </div>
```

```
      </div>
      <div>
        <iframe class="embedded"
         src="https://www2.1010data.com/cgi-bin/beta-latest/quickapp?
path=pub.doc.samples.embedded_qa.sample_qa"
         style="width: 800px; height: 1000px" frameborder="0"/>
      </div>
    </div><!--end container_exchange-->
    <script src="jQuery-2.0.3.js"></script>
    <script src="bootstrap.min.js"></script>
  </body>
</html>
```

> **Note:** This brief overview will not address the HTML used in the construction of the web application. Instead, it will focus on the JavaScript necessary to create interactions between the QuickApp and the enclosing application.

Below is the JavaScript function that is called when the web application finishes loading:

```
window.onload = function() {
  var iframe = document.querySelector("iframe");
  var win = iframe.contentWindow;
  win.postMessage({msg:  "init",
                   uid:  "[1010data_USERNAME]",
                   pswd: "[1010data_PASSWORD]"}, iframe.src);
  var zip = document.querySelector('#input_zip');
  zip.addEventListener("change", function(e) {
    win.postMessage({set: {"zipcode": e.target.value}}, iframe.src);
    var current_zip = document.querySelector("#current_zip");
    current_zip.innerHTML = e.target.value;
  });
  window.addEventListener("message", function(e) {
    var data = e.data.split(',')
    var year = document.querySelector("#current_year");
    year.innerHTML = data[0];
    var zip = document.querySelector("#current_zip");
    zip.innerHTML = data[1];
  });
}
```

The JavaScript function first creates a variable, `iframe`, to reference the `<iframe>` element that contains the QuickApp in the web application.

```
var iframe = document.querySelector("iframe");
```

It then creates a variable, `win`, to access the window object inside the iframe. This is how the QuickApp's DOM is accessed.

```
var win = iframe.contentWindow;
```

An initialization message is sent to the QuickApp using the `postMessage` method. This message will be interpreted by the `receiver` widget in the QuickApp.

```
win.postMessage({msg:  "init",
                 uid:  "[1010data_USERNAME]",
                 pswd: "[1010data_PASSWORD]"}, iframe.src);
```

> **Note:** The message in this example contains placeholders for the 1010data credentials; however, you should not enter your 1010data username and password directly in the JavaScript code. You should provide a secure means within your web application for obtaining these values from the user and specifying them to `postMessage`. Alternatively, you could omit this `postMessage` call altogether, in which case the user will be prompted to enter their credentials via the standard 1010data login page when the web application loads.

When valid 1010data credentials are specified, a 1010data session will be authenticated.

A variable, `zip`, is created to access the input field where the user can enter a zip code. This input field is the HTML element with an ID of `input_zip`.

```
var zip = document.querySelector('#input_zip');
```

The `zip.addEventListener` method registers a function that is called when the `input_zip` element's value is changed by the user. When the value is changed, the function associated with the event listener sends a message to the QuickApp via the `postMessage` method telling it to set the dynamic variable `zipcode` in the QuickApp to the new value from the input field. The input field's value is accessed using the `e` parameter to the event listener function. The event listener function then sets the value of the HTML element with an ID of `current_zip` to the value of the `input_zip` element, so that the new zip code will be displayed in the web application.

```
zip.addEventListener("change", function(e) {
  win.postMessage({set: {"zipcode": e.target.value}}, iframe.src);
  var current_zip = document.querySelector("#current_zip");
  current_zip.innerHTML = e.target.value;
});
```

An `window.addEventListener` method registers a function that is called when the window receives data through a websocket. When information is sent from the QuickApp via the `transmitter` widget, the function associated with the event listener splits the comma-separated message that has been received from the QuickApp into a JavaScript array. It then updates the year and zip code that are being displayed by the HTML elements with the IDs `current_year` and `current_zip`, respectively, using the values in the array.

```
window.addEventListener("message", function(e) {
  var data = e.data.split(',')
  var year = document.querySelector("#current_year");
  year.innerHTML = data[0];
  var zip = document.querySelector("#current_zip");
  zip.innerHTML = data[1];
});
```

When the `window.onload` function runs, the JavaScript application will be set up to send messages to and receive messages from the embedded QuickApp.

See *Sample QuickApp* on page 5 for an example of a QuickApp that uses the `transmitter` and `receiver` widgets to communicate to this external web application.