Contents

0.1 KingKiosk Custom Widget SDK

Build custom widgets for KingKiosk using standard web technologies (HTML, CSS, JavaScript). Widgets run either in a local `customWebView` tile (InAppWebView bridge) or through the Remote Browser custom-widget bridge, and can communicate with the KingKiosk platform through the `window.KingKiosk` JavaScript API.

0.1.1 Table of Contents

- Quick Start
- Adding a Widget
- JavaScript Bridge API
- Receiving Commands

0.1.2 Quick Start

Create a simple widget in 3 steps:

0.1.2.1 1. Create your widget HTML

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>My Widget</title>
  <style>
    body {
      margin: 0;
      padding: 20px;
      background: #1a1a2e;
      color: white;
      font-family: -apple-system, BlinkMacSystemFont, sans-serif;
      display: flex;
      align-items: center;
      justify-content: center;
      min-height: 100vh;
    }
    .value {
      font-size: 72px;
      font-weight: bold;
    }
  </style>
</head>
<body>
  <div class="value" id="display">--</div>

  <script>
    // Wait for the KingKiosk bridge to be ready
    window.addEventListener('kingkiosk-ready', function() {
      console.log('KingKiosk bridge ready!');

      // Listen for commands from the platform
      window.KingKiosk.onCommand(function(command, payload) {
        if (command === 'set_value') {
          document.getElementById('display').textContent = payload.value;
        }
      });
    });
  </script>
</body>
</html>
```

0.1.2.2 2. Host your widget (or use inline HTML)

Option A: Host on a web server

```
https://your-server.com/widgets/my-widget/index.html
```

Option B: Use inline HTML (no hosting required) - Pass the HTML directly via MQTT command

0.1.2.3 3. Add the widget via MQTT

```
{
  "command": "create_remote_browser",
  "window_id": "my_widget_1",
  "name": "My Widget",
  "initial_url": "https://your-server.com/widgets/my-widget/",
  "auto_connect": true
}
```

0.1.3 Adding a Widget

There are two supported runtime paths today:

0.1.3.1 Option A: Remote Browser custom widget bridge (recommended, required on tvOS)

Create a remote browser window pointed at your widget URL:

```
{
  "command": "create_remote_browser",
  "window_id": "weather_widget_1",
  "name": "Weather Widget",
  "initial_url": "https://widgets.example.com/weather/",
  "auto_connect": true
}
```

Optional override (usually not needed): include `server_url` to force a specific Feature Server endpoint.

0.1.3.2 Option B: Reconfigure an existing local `customWebView` tile

Current dispatcher code does not expose a dedicated system command that creates a new local `customWebView` tile directly. If you already have one (for example from restored state), configure it via the window command topic:

Topic: `kingkiosk/{device_id}/window/{window_id}/command`

```
{
  "action": "configure",
  "url": "https://widgets.example.com/weather/",
  "title": "Weather",
  "storage": {
    "city": "San Francisco",
    "units": "fahrenheit"
  }
}
```

0.1.3.3 Local `customWebView` with inline HTML (simple widgets, no hosting)

```
{
  "action": "configure",
  "html": "<!DOCTYPE html><html><body><h1 id='count'>0</h1><script>window.addEventListener('kingkiosk-ready',()=>{window.KingKiosk.onCommand((cmd,p)=>{if(cmd==='increment')document.getElementById('count').textContent=parseInt(document.getElementById('count').textContent)+1;});});</script></body></html>"
}
```

0.1.3.4 Local `customWebView` with base64-encoded HTML (special characters, larger widgets)

```
{
  "action": "configure",
  "html_base64": "PCFET0NUWVBFIGh0bWw+PGh0bWw+PGJvZHk+PGgxPkhlbGxvIFdvcmxkPC9oMT48L2JvZHk+PC9odG1sPg=="
}
```

0.1.3.5 Window Configuration Options

Field	Path	Description
`window_id`	`create_remote_bro`	Required remote browser window ID.
`name`	`create_remote_bro`	Display name for remote browser window.

Field	Path	Description
`initial_url`	`create_remote_brow`	URL to load in remote browser session.
`server_url`	`create_remote_bro`	Optional Feature Server override (deprecated as required input).
`action`	window command topic	Use `"configure"` to reconfigure a local `customWebView` tile.
`url`	local `configure` action	URL to load (mutually exclusive with `html`/`html_base64`).
`html`	local `configure` action	Raw HTML content.
`html_base64`	local `configure` action	Base64-encoded HTML content.
`title`	local `configure` action	Optional title override.
`storage`	local `configure` action	Initial key-value storage map for the widget runtime.

0.1.3.6 Content Size Limits

When using inline HTML or base64-encoded content for local `customWebView`, be aware of MQTT message size limits:

Content Method	Recommended Max	Notes
URL	Unlimited	Widget hosted externally, only URL sent via MQTT
Inline HTML	500KB	JSON escaping adds overhead

Content Method	Recommended Max	Notes
Base64 HTML	375KB original	Becomes ~500KB after encoding (+33%)

MQTT Broker Limits: - MQTT protocol maximum: 256MB per message - Most production brokers: 256KB - 1MB default - AWS IoT Core: 128KB limit - Mosquitto default: 256MB (often configured lower)

Recommendations: - For simple widgets (< 50KB): Use inline `html` for convenience - For medium widgets (50KB - 375KB): Use `html_base64` to avoid JSON escaping issues - For complex widgets (> 375KB): Use `url` and host your widget externally

Base64 Encoding Example:

```
# Encode your widget HTML
cat my-widget.html | base64 > my-widget-base64.txt

# Check size (should be < 500KB after encoding)
wc -c my-widget-base64.txt
```

0.1.4 JavaScript Bridge API

The KingKiosk platform injects a `window.KingKiosk` object into your widget. Wait for the `kingkiosk-ready` event before using it.

```
window.addEventListener('kingkiosk-ready', function() {
  // Bridge is now available
  console.log('KingKiosk API ready');
});
```

0.1.4.1 API Reference

Method	Description
`KingKiosk.onCommand(callback)`	Register to receive commands
`KingKiosk.sendCommand(cmd, payload)`	Send command to platform

Method	Description
`KingKiosk.publishTelemetry(data)`	Publish telemetry data
`KingKiosk.storage.get(key)`	Get stored value (async)
`KingKiosk.storage.set(key, value)`	Store a value
`KingKiosk.storage.getAll()`	Get all stored values (async)
`KingKiosk.getWidgetInfo()`	Get widget metadata (async)

0.1.5 Receiving Commands

Register a callback to receive commands sent from the KingKiosk platform or MQTT.

```
window.KingKiosk.onCommand(function(command, payload) {
  console.log('Received:', command, payload);

  switch (command) {
    case 'set_value':
      updateDisplay(payload.value);
      break;

    case 'set_color':
      document.body.style.backgroundColor = payload.color;
      break;

    case 'refresh':
      fetchLatestData();
      break;

    default:
      console.log('Unknown command:', command);
  }
});
```

0.1.5.1 Sending Commands to Your Widget via MQTT

0.1.5.2 Local `customWebView` command path

Topic: `kingkiosk/{device_id}/window/{window_id}/command`

Use either explicit `widget_command`:

```
{
  "action": "widget_command",
  "command": "set_value",
  "payload": {
    "value": 42
  }
}
```

Or send any custom action directly (unknown actions are forwarded to the widget callback):

```
{
  "action": "set_temperature",
  "celsius": 22.5
}
```

Note: local `customWebView` commands are dispatched only after the widget registers a callback with `KingKiosk.onCommand(...)`.

0.1.5.3 Remote Browser custom widget bridge command path

Topic: `kingkiosk/{device_id}/element/{remote_browser_window_id}/cmd`

Use `command: "widget_command"` and one of the supported payload shapes:

Shape A:

```
{
  "command": "widget_command",
  "widget_command": "set_value",
  "payload": {
    "value": 42
  }
}
```

Shape B:

```
{
  "command": "widget_command",
  "payload": {
    "command": "set_value",
    "payload": {
      "value": 42
    }
  }
}
```

0.1.6 Sending Commands

Send commands from your widget to the KingKiosk platform. These are published to MQTT for external systems to consume.

```javascript
// Simple command
window.KingKiosk.sendCommand('button_pressed', { buttonId: 'start' });

// Command with data
window.KingKiosk.sendCommand('form_submitted', {
  name: 'John Doe',
  email: 'john@example.com',
  timestamp: Date.now()
});

// Emit an integration/event command for external consumers
window.KingKiosk.sendCommand('navigate', { url: '/settings' });
```

0.1.6.1 MQTT Output

Commands are published to:

```
kingkiosk/{device_id}/widget/{widget_id}/event
```

Payload format:

```json
{
  "type": "custom_command",
  "widget_id": "widget_abc123",
  "command": "button_pressed",
  "payload": { "buttonId": "start" },
  "timestamp": 1705432100000
}
```

0.1.7 Publishing Telemetry

Publish sensor data, metrics, or any telemetry from your widget.

```javascript
// Simple value
window.KingKiosk.publishTelemetry({ temperature: 72.5 });

// Multiple metrics
window.KingKiosk.publishTelemetry({
  cpu_usage: 45.2,
  memory_used: 8192,
  disk_free: 50000,
  uptime_seconds: 86400
});

// Periodic telemetry
setInterval(function() {
  window.KingKiosk.publishTelemetry({
    heartbeat: true,
    timestamp: Date.now()
  });
}, 30000);
```

0.1.7.1 MQTT Output

Telemetry is published to:

```
kingkiosk/{device_id}/widget/{widget_id}/telemetry
```

Payload format:

```json
{
  "widget_id": "widget_abc123",
  "data": {
    "temperature": 72.5
  },
  "timestamp": 1705432100000
}
```

0.1.8 Persistent Storage

Store and retrieve data using `KingKiosk.storage`.

- Local `customWebView`: storage is kept in the controller runtime while the tile exists.
- Remote Browser custom widget bridge: storage is persisted via app storage keys (`custom_widget_bridge_`
  `:*`) and restored across app restarts.

```javascript
// Store a value
window.KingKiosk.storage.set('theme', 'dark');
window.KingKiosk.storage.set('lastUpdate', Date.now());
window.KingKiosk.storage.set('settings', { volume: 80, muted: false });

// Retrieve a value (async)
const theme = await window.KingKiosk.storage.get('theme');
console.log('Current theme:', theme); // 'dark'

// Get all stored values
const allData = await window.KingKiosk.storage.getAll();
console.log('All storage:', allData);
// { theme: 'dark', lastUpdate: 1705432100000, settings: { volume: 80, muted: false } }
```

0.1.8.1 Initial Storage

Pre-populate storage when adding the widget:

```json
{
  "action": "configure",
  "url": "https://example.com/widget/",
  "storage": {
    "apiKey": "your-api-key",
    "refreshInterval": 60000,
    "theme": "dark"
  }
}
```

0.1.9 Widget Info

Get information about the widget and platform.

```
const info = await window.KingKiosk.getWidgetInfo();
console.log(info);
// {
//   widgetId: "widget_abc123",
//   platform: "macos"  // or "android", "ios", "windows", "linux", "web"
// }
```

Use this to adapt your widget for different platforms:

```
const info = await window.KingKiosk.getWidgetInfo();

if (info.platform === 'ios' || info.platform === 'android') {
  // Touch-optimized interface
  document.body.classList.add('touch-mode');
} else {
  // Desktop-like layout
  document.body.classList.add('desktop-mode');
}
```

0.1.10 MQTT Topics

0.1.10.1 Receiving Commands: Local `customWebView` (Platform -> Widget)

Topic: `kingkiosk/{device_id}/window/{window_id}/command`

```
{
  "action": "set_value",
  "value": 100
}
```

0.1.10.2 Receiving Commands: Remote Browser custom widget bridge (Platform -> Widget)

Topic: `kingkiosk/{device_id}/element/{remote_browser_window_id}/cmd`

```json
{
  "command": "widget_command",
  "widget_command": "set_value",
  "payload": { "value": 100 }
}
```

0.1.10.3 Widget Events (Widget -> Platform)

Topic: `kingkiosk/{device_id}/widget/{widget_id}/event`

```json
{
  "type": "custom_command",
  "widget_id": "my_widget",
  "command": "user_action",
  "payload": { "action": "clicked" },
  "timestamp": 1705432100000
}
```

0.1.10.4 Widget Telemetry (Widget -> Platform)

Topic: `kingkiosk/{device_id}/widget/{widget_id}/telemetry`

```json
{
  "widget_id": "my_widget",
  "data": { "sensor_value": 42 },
  "timestamp": 1705432100000
}
```

0.1.10.5 Widget State (remote browser bridge, retained)

Topic: `kingkiosk/{device_id}/widget/{widget_id}/state`

```json
{
  "widget_id": "my_widget",
  "type": "custom_webview",
  "has_command_handler": true,
  "storage": {},
  "timestamp": 1705432100000
}
```

0.1.11 Complete Example

A full-featured widget demonstrating all API features:

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Smart Thermostat Widget</title>
  <style>
    * { box-sizing: border-box; margin: 0; padding: 0; }

    body {
      font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, sans-serif;
      background: linear-gradient(135deg, #1a1a2e 0%, #16213e 100%);
      color: white;
      min-height: 100vh;
      display: flex;
      flex-direction: column;
      align-items: center;
      justify-content: center;
      padding: 20px;
    }

    .thermostat {
      text-align: center;
    }

    .temperature {
      font-size: 96px;
      font-weight: 200;
      line-height: 1;
    }

    .temperature .unit {
      font-size: 36px;
      vertical-align: top;
    }

    .label {
      font-size: 14px;
      text-transform: uppercase;
      letter-spacing: 2px;
      opacity: 0.7;
      margin-top: 8px;
    }

    .controls {
      display: flex;
      gap: 20px;
      margin-top: 40px;
    }

    .btn {
      width: 60px;
      height: 60px;
      border-radius: 50%;
      border: 2px solid rgba(255,255,255,0.3);
      background: rgba(255,255,255,0.1);
      color: white;
      font-size: 24px;
      cursor: pointer;
      transition: all 0.2s;
    }

    .btn:hover {
      background: rgba(255,255,255,0.2);
```

```
        transform: scale(1.1);
      }

      .btn:active {
        transform: scale(0.95);
      }

      .status {
        margin-top: 30px;
        font-size: 12px;
        opacity: 0.5;
      }

      .mode {
        margin-top: 20px;
        padding: 8px 16px;
        background: rgba(255,255,255,0.1);
        border-radius: 20px;
        font-size: 12px;
        text-transform: uppercase;
        letter-spacing: 1px;
      }

      .mode.heating { background: rgba(255,100,100,0.3); }
      .mode.cooling { background: rgba(100,100,255,0.3); }
      .mode.off { background: rgba(100,100,100,0.3); }
    </style>
  </head>
  <body>
    <div class="thermostat">
      <div class="temperature">
        <span id="temp">--</span><span class="unit">°F</span>
      </div>
      <div class="label">Target Temperature</div>

      <div class="controls">
        <button class="btn" id="btnDown">-</button>
        <button class="btn" id="btnUp">+</button>
      </div>

      <div class="mode" id="mode">OFF</div>

      <div class="status" id="status">Connecting...</div>
    </div>

    <script>
      // State
      let targetTemp = 72;
      let mode = 'off';
      let widgetId = 'unknown';

      // DOM elements
      const tempDisplay = document.getElementById('temp');
      const modeDisplay = document.getElementById('mode');
      const statusDisplay = document.getElementById('status');
      const btnUp = document.getElementById('btnUp');
      const btnDown = document.getElementById('btnDown');

      // Update display
      function updateDisplay() {
        tempDisplay.textContent = targetTemp;
        modeDisplay.textContent = mode.toUpperCase();
        modeDisplay.className = 'mode ' + mode;
      }

      // Send telemetry
      function sendTelemetry() {
        window.KingKiosk.publishTelemetry({
          target_temperature: targetTemp,
          mode: mode,
          timestamp: Date.now()
        });
      }

      // Initialize when bridge is ready
      window.addEventListener('kingkiosk-ready', async function() {
        statusDisplay.textContent = 'Connected';

        // Get widget info
        const info = await window.KingKiosk.getWidgetInfo();
        widgetId = info.widgetId;

        // Load saved state
        const savedTemp = await window.KingKiosk.storage.get('targetTemp');
        const savedMode = await window.KingKiosk.storage.get('mode');

        if (savedTemp) targetTemp = savedTemp;
        if (savedMode) mode = savedMode;

        updateDisplay();

        // Register command handler
        window.KingKiosk.onCommand(function(command, payload) {
          console.log('Command received:', command, payload);

          switch (command) {
            case 'set_temperature':
              targetTemp = payload.temperature;
              window.KingKiosk.storage.set('targetTemp', targetTemp);
              updateDisplay();
              sendTelemetry();
              break;

            case 'set_mode':
              mode = payload.mode;
              window.KingKiosk.storage.set('mode', mode);
              updateDisplay();
              sendTelemetry();
              break;

            case 'increment':
              targetTemp++;
              window.KingKiosk.storage.set('targetTemp', targetTemp);
              updateDisplay();
              sendTelemetry();
              break;

            case 'decrement':
              targetTemp--;
              window.KingKiosk.storage.set('targetTemp', targetTemp);
              updateDisplay();
              sendTelemetry();
              break;
          }
        });

        // Send initial telemetry
        sendTelemetry();

        // Periodic telemetry (every 30 seconds)
        setInterval(sendTelemetry, 30000);
      });

      // Button handlers
      btnUp.addEventListener('click', function() {
        targetTemp++;
        window.KingKiosk.storage.set('targetTemp', targetTemp);
        updateDisplay();
        sendTelemetry();
        window.KingKiosk.sendCommand('temperature_changed', {
          temperature: targetTemp,
          direction: 'up'
        });
      });

      btnDown.addEventListener('click', function() {
        targetTemp--;
        window.KingKiosk.storage.set('targetTemp', targetTemp);
        updateDisplay();
        sendTelemetry();
        window.KingKiosk.sendCommand('temperature_changed', {
          temperature: targetTemp,
          direction: 'down'
        });
      });
    </script>
  </body>
  </html>
```

0.1.12 Best Practices

0.1.12.1 1. Always Wait for the Bridge

```
// Good
window.addEventListener('kingkiosk-ready', function() {
  window.KingKiosk.onCommand(...);
});

// Bad - bridge may not be ready
window.KingKiosk.onCommand(...);
```

0.1.12.2 2. Handle Missing Bridge Gracefully

```
function initWidget() {
  if (window.KingKiosk) {
    // Running in KingKiosk
    setupBridgeHandlers();
  } else {
    // Running standalone (for testing)
    console.log('Running without KingKiosk bridge');
    setupMockData();
  }
}

window.addEventListener('kingkiosk-ready', initWidget);

// Fallback for standalone testing
setTimeout(function() {
  if (!window.KingKiosk) initWidget();
}, 1000);
```

0.1.12.3 3. Use Responsive Design

```
/* Support different window sizes */
body {
  min-height: 100vh;
  display: flex;
  align-items: center;
  justify-content: center;
}

/* Adapt to platform */
.tv-mode .text { font-size: 2em; }
.touch-mode .button { min-height: 44px; }
```

0.1.12.4 4. Minimize Network Requests

```
// Cache data locally
let cache = {};

async function getData(key) {
  if (!cache[key]) {
    cache[key] = await fetchFromApi(key);
  }
  return cache[key];
}
```

0.1.12.5 5. Use Throttled Telemetry

```
// Don't spam telemetry
let telemetryTimer = null;

function queueTelemetry(data) {
  if (telemetryTimer) clearTimeout(telemetryTimer);
  telemetryTimer = setTimeout(function() {
    window.KingKiosk.publishTelemetry(data);
  }, 1000);
}
```

0.1.12.6 6. Persist Important State

```
// Save state on every change
function updateTemperature(newTemp) {
  temperature = newTemp;
  window.KingKiosk.storage.set('temperature', temperature);
  updateDisplay();
}
```

0.1.13 Troubleshooting

0.1.13.1 Widget Shows "Not Configured"

- Ensure you provided `url`, `html`, or `html_base64`
- Check that URL is accessible (CORS may block some URLs)

0.1.13.2 Bridge Not Available

- Wait for `kingkiosk-ready` event
- Check browser console for errors
- Verify the widget is loaded in KingKiosk (not standalone browser)

0.1.13.3 Commands Not Received

- Local `customWebView`: verify topic `kingkiosk/{device_id}/window/{window_id}/command` and payload includes `"action"`
- Remote Browser bridge: verify topic `kingkiosk/{device_id}/element/{remote_browser_windo}/cmd` and payload uses `"command": "widget_command"`
- Ensure widget registered handler with `onCommand()`

0.1.13.4 Telemetry Not Publishing

- Check MQTT connection status
- Verify device name is set
- Look for errors in KingKiosk logs

0.1.13.5 Storage Not Persisting

- Local `customWebView`: storage is runtime-scoped to the active tile/controller
- Remote Browser bridge: storage persists via app storage (`custom_widget_bridge_storage_v1:*`)
- Verify you're calling `storage.set()` correctly
- Check that widget ID hasn't changed

0.1.13.6 Debug Tips

```
// Enable verbose logging
window.addEventListener('kingkiosk-ready', async function() {
  console.log('Bridge ready, widget ID:',
    (await window.KingKiosk.getWidgetInfo()).widgetId);

  window.KingKiosk.onCommand(function(cmd, payload) {
    console.log('[CMD]', cmd, JSON.stringify(payload));
  });
});

// Monitor all storage
setInterval(async function() {
  const all = await window.KingKiosk.storage.getAll();
  console.log('[STORAGE]', all);
}, 5000);
```

0.1.14 Platform Support

Platform	Local `customWebView`	Remote Browser custom widget bridge	Notes
macOS	Yes	Yes	Either path works.
iOS	Yes	Yes	Either path works.

Platform	Local `customWebView`	Remote Browser custom widget bridge	Notes
tvOS	No	Yes	tvOS uses Remote Browser path.
Android	Yes	Yes	Either path works.
Windows	Yes	Yes	Either path works.
Linux	Yes	Yes	Either path works.
Web	Yes	Depends on WebRTC/bridge support	Verify in your target browser/runtime.

0.1.14.1 tvOS Special Requirements

tvOS has no local `customWebView` tile runtime - it uses Remote Browser sessions.

Use a remote browser command (same command surface used on other platforms when desired):

```
{
  "command": "create_remote_browser",
  "window_id": "my_widget_tv",
  "name": "My Widget",
  "initial_url": "https://example.com/widget/",
  "auto_connect": true
}
```

`server_url` is optional; if omitted, the app uses configured Feature Server settings.

On tvOS (Remote Browser path), the custom widget bridge supports: - `KingKiosk.onCommand()` - `KingKiosk.sendCommand()` - `KingKiosk.publishTelemetry()` - `KingKiosk.storage.get/set/getAll()` - `KingKiosk.getWidgetInfo()`

For MQTT commands into widgets on tvOS, use the element command topic with `command: "widget_command"` (see sections above).

0.1.15 Need Help?

- Check the MQTT Widget Reference for more details
- Review the example widgets directory
- Open an issue on GitHub for bugs or feature requests