1 Acquiring an Access Token

1.1 Web Gadget Profile

This profile is suitable when the Client is a web application embedding a JavaScript Gadget on behalf of a User. This profile enables a Client to act on behalf of the User without acquiring a User's credentials.

This profile differs from the Web Application Profile in that it does not require the Client to make a call to the Authorization Server's Access Token URL and instead relies on a cross-origin browser communication channel to securely pass the access token to the Client.

1.1.1 Provisioning

Prior to initiating this profile, the Client MUST have obtained a Client Identifier from the Authorization Server. The Authorization Server MAY have also required the Client to register a Callback URL.

1.1.2 User Visits Client Web Application

The User visits a web page within the Client web application. The Client MUST include a reference to a WRAP Web Gadget Profile JavaScript implementation in order to retrieve access tokens on behalf of the User. Appendix B includes a reference implementation.

If the User's browser supports the HTML5 postMessage API, the implementation MUST register an event listener for the window "message" event. In down-level browsers, other initialization may occur.

1.1.3 User Initiates Access Request

The User initiates an access request by clicking a link or button within the browser that requests access to the User's information. When the User clicks the link or button, the Client SHOULD initiate an authorization request by invoking a JavaScript method that opens a browser popup window directing the User's browser to the Authorization Server's User Authorization URL. The method signature is:

```
WRAP.getAccessToken = function(
```

`authorizationUrl`
: REQUIRED. The Authorization Server's User Authorization URL. No query string parameters are required, but any that are included MUST be preserved.

`clientId`
: REQUIRED. The Client Identifier.

`callback`
: REQUIRED. The JavaScript callback function that is invoked when an access token is granted. The parameters passed to the callback are documented below.

`userState`
: OPTIONAL. The client MAY provide user state in the form of a JavaScript object that is preserved and provided to the JavaScript callback.

`scope`
: OPTIONAL. The Authorization Server MAY define authorization scope values for the Client to include.

)

1.1.4 Client Directs the User to the Authorization Server

The `WRAP.getAccessToken` method is responsible for mapping its input parameters to URL parameters that are added to the provided Authorization Server URL.

The method MUST validate that the `authorizationUrl` and `clientId` parameters are present and if not, an Error MUST be thrown. If their type is not "String" an Error MUST be thrown. The method MUST validate that the `callback` parameter is provided and is a function; if not, an Error MUST be thrown. If provided, the `scope` parameter must be of type "String"; if not, an Error MUST be thrown.

The provided User Authorization URL may include existing query string parameters; any existing parameters MUST be preserved. The method MUST generate a unique client state that can correlate the provided JavaScript callback and user state with an Access Token response. The method MUST generate a Callback URL using the Client's current window's window.location.href.

The method should take the validated input parameters and the generated parameters and add them to the Authorization Server's User Authorization URL as follows:

`wrap_client_id`
: REQUIRED. The Client Identifier.

`wrap_profile`
: REQUIRED. The value should be "WebGadgetProfile" to distinguish this profile from the Web Application Profile.

`wrap_callback`
: REQUIRED. The Callback URL. An absolute URL to which the Authorization Server will communicate the access token via a cross-origin communication channel. Authorization Servers MAY require that the wrap_callback URL match the previously registered value for the Client Identifier.

`wrap_client_state`
: OPTIONAL. An opaque value that the `WRAP.getAccessToken` implementation SHOULD use to correlate the access token request with the Client-provided JavaScript callback.

`wrap_scope`
: OPTIONAL. The Authorization Server MAY define authorization scope values for the Client to include.

The method SHOULD open a browser popup window that directs the User to the Authorization Server's User Authorization URL. The dimensions of the popup window SHOULD be 550px x 400px. The popup window MUST display the User Authorization URL in the window's title bar.

1.1.5 Authorization Server Confirms Delegation Request with User

Upon receiving an authorization request from the Client within a popup window of the User's browser, the Authorization Server authenticates the user, presents the User with the Protected Resource access that will be granted to the Client, and prompts the User to confirm the request.

If the User approves the request, the Authorization Server generates an access token and passes it in an HTTP response to the User's browser.

1.1.6 Authorization Server Communicates Access Token to Client

If the User approved the request, the Authorization Server MUST return the access token, callback, expiry and user state to the User's browser in the HTTP response.

1.1.6.1 Cross-Origin Communication (HTML5)

In browsers that support HTML5's postMessage API, JavaScript code executing in the User Authorization URL's domain MUST communicate the returned parameters as a JSON-encoded string to the opening window using the postMessage API. The first parameter to the postMessage API MUST be a JSON-encoded object containing the following fields:

`method`
: REQUIRED. The string "WRAP.getAccessToken"

`accessToken`
: REQUIRED. The Access Token.

`accessTokenExpiresIn`
: OPTIONAL. The lifetime of the Access Token in seconds. For example, 3600 represents one hour.

`clientState`
: REQUIRED. The client state that was provided by the `getAccessToken` call to correlate the access token request with the JavaScript callback.

The second parameter to the postMessage API MUST be the `wrap_callback` parameter provided to the User Authorization URL. The browser will enforce that the message is only delivered to the opening window if the domain names and schemes match.

Once the event handler for the Client window's "message" event is raised, the message MUST be deserialized into a JSON object. The handler MUST validate that the `method` field matches "WRAP.getAccessToken". If the `method` field does not exist or does not match, the message MUST be

ignored. Otherwise, the fields MUST be passed to the `WRAP.onAccessToken` method so that JavaScript callback can be invoked with the Access Token.

1.1.6.2 Cross-Origin Communication (Down-level Browser)

[Author's note: should we leave this undefined or provide specific details on how this can be accomplished using the HTML fragment hack, Flash LocalConnection or Silverlight? An alternative approach would leave this publisher-specific and the publisher would be responsible for invoking the WRAP.onAccessToken method as defined in 1.1.7.]

1.1.7 Access Token Dispatched to JavaScript Callback

Once the Access Token is communicated across the communication channel, it MUST be passed to the following JavaScript method in order to dispatch it to the Client application:

```
WRAP.onAccessToken = function(
```

`authorizationUrl`
: REQUIRED. The Authorization URL that granted the request.

`accessToken`
: REQUIRED. The Access Token.

`accessTokenExpiresIn`
: OPTIONAL. The lifetime of the Access Token in seconds. For example, 3600 represents one hour.

`clientState`
: REQUIRED. The client state that was provided by the `WRAP.getAccessToken` call to correlate the access token request with the JavaScript callback.

```
)
```

The `WRAP.onAccessToken` method SHOULD use the `clientState` in order to correlate this access token request with the initial `WRAP.getAccessToken` request. The provided `authorizationUrl` MUST match the `authorizationUrl` provided to the `WRAP.getAccessToken` request. If the `authorizationUrl` does not match, the request MUST be ignored. Otherwise, the callback that was provided to the WRAP.getAccessToken call MUST be invoked with the following parameters:

```
callback = function(
```

`accessToken`
: REQUIRED. The Access Token.

`accessTokenExpiresIn`

REQUIRED. The lifetime of the Access Token in seconds. For example, 3600 represents one hour. If the Authorization URL did not provide the lifetime, -1 should be provided.

`userState`

REQUIRED. The user state that was provided to the WRAP.getAccessToken method. If not provided, "null" should be used.

)

The Client may now use the Access Token to access the Protected Resource per §4.

2 Appendix A: WRAP.js Implementation

Below is a reference WRAP.js implementation that can be used in the Web Gadget Profile to retrieve access tokens on behalf of the Client. A Resource Provider that implements the Web Gadget Profile must support this interface for retrieving access tokens. See Appendix B for an example that uses this interface.

```
if (!window.WRAP) {
    window.WRAP = {};
}

(function() {

    WRAP.getAccessToken = function(authorizationUrl, clientId, callback, userState, scope) {
        userState = userState || null;
        scope = scope || "";

        if (typeof (authorizationUrl) !== "string" || authorizationUrl.length === 0) {
            throw new Error("The authorizationUrl parameter is invalid");
        }

        if (typeof (clientId) !== "string" || clientId.length === 0) {
            throw new Error("The clientId parameter is invalid");
        }

        if (typeof (callback) !== "function") {
            throw new Error("The callback parameter is invalid");
        }

        if (typeof (scope) !== "string") {
            throw new Error("The scope parameter is invalid");
        }

        var clientState = (nextClientState++).toString();

       var options =
"width=550,height=400,status=no,resizable=yes,toolbar=no,menubar=no,scrollbars=yes";

        authorizationUrl +=
            (authorizationUrl.indexOf("?") < 0 ? "?" : "&") +
            "wrap_client_id=" + encodeURIComponent(clientId) + "&" +
            "wrap_callback=" + encodeURIComponent(window.location.href) + "&" +
            "wrap_client_state=" + encodeURIComponent(clientState) + "&" +
            "wrap_scope=" + encodeURIComponent(scope);

        callbacks[clientState] = { "callback": callback, "userState": userState,
"authorizationUrl": authorizationUrl };

        window.open(authorizationUrl, "WRAP", options);
    };

    WRAP.onAccessToken = function(authorizationUrl, accessToken, accessTokenExpiresIn,
clientState) {
        accessTokenExpiresIn = accessTokenExpiresIn || -1;

        if (typeof (origin) !== "string" || origin.length == 0) {
            throw new Error("The origin parameter is invalid");
        }

        if (typeof (accessToken) !== "string" || accessToken.length == 0) {
            throw new Error("The accessToken parameter is invalid");
        }

        if (typeof (accessTokenExpiresIn) !== "number") {
            throw new Error("The accessTokenExpiresIn parameter is invalid");
        }

        var state = callbacks[clientState];

        if (state) {

            if (state.authorizationUrl !== authorizationUrl) {
                // Origin does not match. Ignore.
                return;
            }
```

```
            delete callbacks[clientState];

            callback(accessToken, accessTokenExpiresIn, state.userState);
        }
    };

    function onMessage(e) {
        var m = JSON.parse(e.data);
        if (m &&
            m.method &&
            m.method === "WRAP.getAccessToken") {

            WRAP.onAccessToken(e.origin, m.accessToken, m.accessTokenExpiresIn, m.clientState);
        }
    }

    var nextClientState = 0;

    var callbacks = {};

    window.addEventListener("message", onMessage);
} ());
```

3 Appendix B: Web Gadget Example

This example demonstrates sample code provided by a Resource Provider to enable a Client to retrieve an Access Token in the Web Gadget Profile. In this example, the Resource Provider uses the HTML5 postMessage API to handle secure cross-origin communication, but a Resource Provider can use other technologies as well.

3.1 Resource Provider Code

The following code is used by the Resource Provider to send the access token to the originating Client web page.

```
// Resource Provider code (URL: http://auth.example.com/user_authorization)

var callbackUrl = "http://music.example.com/";

var message = JSON.stringify({
    "method": "WRAP.getAccessToken",
    "accessToken": "<%= accessToken %>",
    "accessTokenExpiresIn": 28800,
    "clientState": "<%= clientState %>"
});

if (window.opener) {
    window.opener.postMessage(message, callbackUrl);
}

window.close();
```

3.2 Client Code

The following code is provided by the Resource Provider to the Client and is loaded in the Client's web page to handle receipt of the access token across cross-origin browser communication channel.

```
// Client code (URL: http://music.example.com/)

function getAccessToken() {
    WRAP.getAccessToken(
        "http://auth.example.com/user_authorization",
        "status.example.com",
        onAccessToken);
}
```

```
function onAccessToken(accessToken, accessTokenExpiresIn) {
}
```