HAPI Chrome OS App Tutorial

Description

The Chrome OS App provides a sample Chrome OS File System Provider API App using HAPI. The app allows Active Directory (AD) authentication and access to both on-premise AD and HAPI agent files and folders as a virtual file system in Chrome OS' file manager. The entire Chrome File System Provider API has been implemented to include mount, un-mount, download, upload, edit, create, rename, cut, copy and paste for files and folders.

Tutorial

Introduction

This sample Chrome OS app is written mainly in javascript and demonstrates how to use HAPI to:

- Login with AD credentials
- Navigate files and folder published in AD or HAPI agents
- Get a list of all the users HAPI agents
- Download a file to the device
- Upload a local device file or folder to AD or on agents
- Truncate a file stored in AD or on agents
- Delete a file or folder stored in AD or on agents
- Create a file or folder stored in AD or on agents
- Edit a file stored in AD or on agents
- Rename a file or folder stored in AD or on agents
- Copy a file or folder stored locally, in AD or on agents to a file or folder stored locally, in AD or on agents

This app implements all of the Chrome OS File System Provider API (up through Chrome 44) and a subset of the HAPI functionality. By default, the app makes calls to the HAPI Gatekeeper in our sandbox environment, which is supported by a HAPI Gateway and sandbox Active Directory instance. FullArmor can supply a temporary sandbox Active Directory account, complete with sample data, upon request.

The HAPI interface is defined on our API Documentation pages at https://sandbox.fullarmorhapi.com/route/swagger and https://sandbox.fullarmorhapi.com/swagger. These are Gateway and Gatekeeper calls respectively. All HAPI calls are directed to the Gatekeeper. Urls with /route/hapi will be routed to the proper Gateway. Urls with /route/agent/<id> will be routed to the proper HAPI Agent. Urls with /api access the interface presented by the Gatekeeper.

App Operations

Login

HAPI Calls: Login (/route/hapi/Login) Post

HAPI login takes Active Directory credentials and creates a token required for all other HAPI calls. In the sandbox environment, the token will expire with 15 minutes of idle time. This sample app, will show a login dialog when it is either started, or on selects configure from a mounted HAPI file system.

Here is code from the sample app performing a login.

```
this.login = function (username, password, success, error) {
    _username = username;
    _password = password;

    var xhr = new XMLHttpRequest();
    xhr.open("POST", _gatekeeper + '/route/hapi/login', true);
    xhr.setRequestHeader("Content-Type", "application/json;charset=UTF-8");
    xhr.setRequestHeader('HAPIToken', _token);
    xhr.onload = function(e) {
        var data = JSON.parse(this.responseText);
        if (data.Success) {
            _token = data.Token;
            _usersid = data.UserSID;
            _userdisplayname = data.UserDisplayName;
            success();
        }
        else {
            this.hapiToken = false;
            error(data.Error);
        }
    };
    xhr.send(JSON.stringify({UserName: username, Password: password}));
};
```

Note: For all code snippets, the _gatekeeper variable is set to the default for the sandbox ("https://sandbox.fullarmorhapi.com").

View Files and Folders

HAPI Calls: Share(s) (/route/hapi/Share(s)/GetFilesAndFolders) Post and Agent Share(s) (/route/agent/<agentId>/Share(s)/GetFilesAndFolders) Post

HAPI includes many file providers (Box, Dropbox, Office365, SharePoint, Active Directory, My Computer, etc.). The GetFilesAndFolders method for all of them are the same. The call allows for filtering, flat multi-level enumeration, and other functionality. For this sample app, we just call to get the current folder's files and folders.

We start with calling the Shares provider to get the root shares published by Active Directory. Then as the user selects to enter a share, we call the Share provider to get the file and folders for that specific folder.

Here is code from the sample app performing file and folder enumeration.

```
this.getfilesandfolders = function (controller, folderEnumRequest, success, error) {
        var xhr = new XMLHttpRequest();
        if (controller.startsWith("agent")) {
            xhr.open("POST", _gatekeeper + '/route/' + controller +
'/GetFilesAndFolders', true);
        }
        else {
            xhr.open("POST", _gatekeeper + '/route/hapi/' + controller +
'/GetFilesAndFolders', true);
        }
        xhr.setRequestHeader("Content-Type", "application/json;charset=UTF-8");
        xhr.setRequestHeader('HAPIToken', _token);
        xhr.onload = (function (e) {
            if (this.status > 299) {
                if (controller.startsWith("agent")) {
                  success({Items: []});
                }
                else {
                  error(this.status);
                }
            }
            else
                success(JSON.parse(this.responseText));
        });
        xhr.send(JSON.stringify(folderEnumRequest));
    };
```

Download a File

HAPI Calls: Share (/route/hapi/Share/Download) Get and Agent Share (/route/agent/agentId/Share/Download) Get

The HAPI providers all have a Download method. It takes a string containing the file identifier and returns a stream filled with the file content. The stream has a ContentDispositionHeader of type attachment with a value that is the filename.

Here is code from the sample app performing file download.

```
this.downloadfile = function(controller, path, start, end, success, error) {
        var xhr = new XMLHttpRequest();
        if (controller.startsWith("agent")) {
           xhr.open("GET", _gatekeeper + '/route/' + controller +
'/Download?fileIdentifier=' + path, true);
        }
        else {
           xhr.open("GET", _gatekeeper + '/route/hapi/' + controller +
'/Download?fileIdentifier=' + path, true);
        }
        xhr.responseType = 'arraybuffer';
        xhr.setRequestHeader('HAPIToken', _token);
        xhr.setRequestHeader('Range', 'bytes=' + start + '-' + end);
        xhr.onload = (function (e) {
           if (this.status > 299) {
              error(this.status);
           }
           else
              success(this.response);
        });
        xhr.send();
     };
```

Upload a File

HAPI Calls: Share (/route/hapi/Share/UploadFile) Post and Agent Share (/route/agent/<agentId>/Share/UploadFile) Post

The HAPI providers all have an UploadFile method. It takes a Mime multipart request with the file identifier and chunking parameters. The HAPI Upload method can upload the file all in one call or in chunks. The sample code uploads the file (not in chunks, as the Chrome OS File System Provider API send us the file/folder in write chunks on its own).

Here is code from the sample app performing file upload.

```
this.uploadfile = function(controller, path, name, offset, arraybuffer, success, error) {
        var xhr = new XMLHttpRequest();
        var slice = new Blob([arraybuffer]);
        if (controller.startsWith("agent")) {
           xhr.open("POST", _gatekeeper + '/route/' + controller + '/UploadFile', true);
        }
        else {
           xhr.open("POST", _gatekeeper + '/route/hapi/' + controller + '/UploadFile',
true);
        }

        var headers = {
           "HAPIToken": _token,
           "Accept": "application/json",
           "Cache-Control": "no-cache",
           "X-Requested-With": "XMLHttpRequest"
        };

        for (var headerName in headers) {
           var headerValue = headers[headerName];
           xhr.setRequestHeader(headerName, headerValue);
        }

        xhr.onload = (function (e) {
           if (this.status > 299) {
              error(this.status);
           }
           else
              success(this.response);
        });

        var form = new FormData();
        form.append("fileidentifier", path);
        form.append("chunk", -1);
        form.append("chunkStart", offset);
        form.append("file", slice, name);

        xhr.send(form);

    };
```