PAPPL Scanning Support

GSoC ‘23 and ‘24
IDEA ??
IDEA: SANE BECOMING INSANE

- Example: Let’s consider 3 applications and 4 devices
- TWAIN: Needs 12 different programs to work.
- SANE: Reduces the number of programs to 7 but still a lot.
- Driverless Scanning: Reduced to 3.
- PAPPL: Framework for driverless printing through CUPS. Scanning support project being implemented.
The Architecture

Communicates with the Scanner Application through IP which in our case is eSCL built on HTTP

Basically a frontend (Scan image + sane-airscan backend) which does a DNS-SD browse for eSCL servers.

Browse Avahi and find the registered scanning service.

Looks for eSCL servers in the network

Uses a standard network protocol "eSCL" to find scanners and scan on them

If eSCL is not supported then we need Scanner Application Framework which emulates an eSCL scanner and communicates with the actual scanner

Scanner Application Framework

Responsible for communicating with the physical scanner through callback functions via a proprietary protocol

Since driverless scanning and printing work in the same manner it makes sense to have a printer application that supports both Printing and Scanning. This is what we want to do with PAPPL

To emulate an eSCL server for a non-eSCL supported device

Basically a server daemon which is running and registers the service through DNS-SD
Work / Code
Phase 1
Creating ESCL Endpoints

- Go through the MOPRIA Scan Specifications to know the endpoints.
- HTTPS/eSCL endpoints needed for
  1) /{root}/ScannerCapabilities
  2) /{root}/ScannerStatus
  3) /{root}/ScanBufferInfo
  4) /{root}/ScanJobs and many more
ESCL endpoints

Implemented under pappl/client.c under HTTP_STATE_GET and HTTP_STATE_POST

```c
case HTTP_STATE_POST:
    if (strcmp(client->uri, "/eSCL/ScanJobs") == 0)
    {
        // Process the ScanJobs request
        size_t content_length = (size_t)httpGetLength2(client->http);
        char *xml_content = (char *)malloc(content_length + 1);

if (strcmp(client->uri, "/eSCL/ScanJobs/"))
    // Check if the requested URI is "/eSCL/ScannerCapabilities"
    if (strcmp(client->uri, "/eSCL/ScannerCapabilities") == 0)
    {
        addOrUpdateIP(&iplist, client->host_field);

    case HTTP_STATE_GET:
        const char *dummyFilePath = "NULL";
        addOrUpdateIP(&iplist, client->host_field);

        // Check if the requested URI is "/eSCL/ScanJobs/JobUri/NextDocument"
        if (IsMatchingRequest(client->uri) == 1)
        {
            dummyFilePath = "/DummyDriver/pappl-160.png";

            // Send an external file...
            int fd;  // Resource file descriptor
            char buffer[BUFFSIZE];  // Copy buffer
            ssize_t bytes;  // Bytes read/written
```
Creating XML Parser

● Since eSCL communications is essentially through XML we would have to create a XML Parser that can sort of interpret various details and call the required functions.

● The XML parser is based on regex matching done using patterns as demonstrated in MOPRIA Scan Specifications.

● Divided the main function into many branched out functions for micro features.
// Functions...

char *readXmlContent(const char *filePath);
void initScanSettingsXml(ScanSettingsXml &settings, const char *s);
char *getString(ScanSettingsXml &settings, const char *name, const char *pattern);
double getNumber(ScanSettingsXml &settings, const char *name, const char *pattern);
bool ClientAlreadyAirScan(pappl_client_t *client);

ScanSettingsXml &ScanSettingsFromXml(const char *xmlString, pappl_client_t *client) {
    ScanSettingsXml scanSettings;
    initScanSettingsXml(&scanSettings, xmlString);

    char *versionPattern = "<cmsg:Version>(.*)</cmsg:Version>";
    char *version = getString(&scanSettings, "Version", versionPattern);

    char *intentPattern = "<cmsg:Intent>(.*)(.*)</cmsg:Intent>";
    char *intent = getString(&scanSettings, "Intent", intentPattern);

    char *heightPattern = "<cmsg:Height>(.*)</cmsg:Height>";
    char *height = getNumber(&scanSettings, "Height", heightPattern);

    char *contentRegionUnitsPattern = "<cmsg:ContentRegionUnits>(.*)</cmsg:ContentRegionUnits>";
    char *contentRegionUnits = getString(&scanSettings, "ContentRegionUnits", contentRegionUnitsPattern);

    char *widthPattern = "<cmsg:Width>(.*)</cmsg:Width>";
    double width = getNumber(&scanSettings, "Width", widthPattern);

    char *xOffsetPattern = "<cmsg:Offset>(.*)</cmsg:Offset>";
    double xOffset = getNumber(&scanSettings, "XOffset", xOffsetPattern);

    char *yOffsetPattern = "<cmsg:Offset>(.*)</cmsg:Offset>";
    double yOffset = getNumber(&scanSettings, "YOffset", yOffsetPattern);

    char *inputSourcePattern = "<cmsg:InputSource>(.*)</cmsg:InputSource>";
    char *inputSource = getString(&scanSettings, "InputSource", inputSourcePattern);

    char *colorModePattern = "<cmsg:ColorMode>(.*)</cmsg:ColorMode>";
    char *colorMode = getString(&scanSettings, "ColorMode", colorModePattern);
}
Add a dummy driver (initially xml files, later turned to text files) that emulates a driver fetching data from a scanner.

The following scan driver emulation files had to be added:

1) ScannerStatus
2) ScannerCapabilites
3) ScannerBufferInfo
DUMMY DRIVER

Example of Initial Buffer Info file

```xml
<?xml version="1.0" encoding="UTF-8"?>
<scan:ScanBufferInfo
  xsi:schemaLocation="http://schemas.hp.com/imaging/escl/2011/05/03 esCL.xsd"
  xmlns:scan="http://schemas.hp.com/imaging/escl/2011/05/03"
  xmlns:pwg="http://www.pwg.org/schemas/2010/12/sm"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <scan:ScanSettings>
    <pwg:Version>2.6</pwg:Version>
    <scan:Intent>Photo</scan:Intent>
    <pwg:ScanRegions>
      <pwg:ScanRegion>
        <pwg:Height>1200</pwg:Height>
        <pwg:ContentTypeUnits>escl:ThreeHundredthsOfInches</pwg:ContentTypeUnits>
        <pwg:Width>1800</pwg:Width>
        <pwg:XOffset>0</pwg:XOffset>
        <pwg:YOffset>0</pwg:YOffset>
      </pwg:ScanRegion>
    </pwg:ScanRegions>
    <scan:DocumentFormatExt>image/jpeg</scan:DocumentFormatExt>
    <pwg:ContentType>Photo</pwg:ContentType>
    <pwg:InputSource>Platen</pwg:InputSource>
    <scan:XResolution>300</scan:XResolution>
    <scan:YResolution>300</scan:YResolution>
    <scan:ColorMode>Grayscale8</scan:ColorMode>
</scan:ScanBufferInfo>
```
Creating SANE Interface for Pappl Retrofit

- During the later half of GSoC had to understand the working of SANE and how it works.
- Started with understanding the documentation and capabilities of SANE, and finally modelled a PR.
- Independent application completing the SANE driver behaviour already developed and tested.
- Work on combining the SANE Driver from PAPPL Retrofit with PAPPL for creating a Scanner Application still ongoing.
```c
void initializeSane()
{
    SANE_Int versionCode = 0;
    sane_init(&versionCode, authenticationCallback);
    printf("Version: %d\n", versionCode);
}

SANE_Status getScanningDevices(const SANE_Device ***deviceList)
{
    printf("Getting all Scanning Devices\n");
    SANE_Status status = sane_get_devices(deviceList, SANE_FALSE);
    if (status)
    {
        printf("Could not retrieve devices: %s\n", sane_status(status));
    }
    return status;
}

SANE_Status openScanningDevice(SANE_Device *device, SANE_Handle *handle)
{
    SANE_Status status = sane_open(device->name, handle);
    if (status)
    {
        printf("Scanning device could not be opened %s: %s\n", device->name, sane_status(status));
    }
    return status;
}

void cancelScan(SANE_Handle handle)
{
    sane_cancel(handle);
}
```
Work / Code
Phase 2
PAPPL API Bridging for Scanner Applications

- Create printer-like class object structures for scanners.
- Add functionalities and data structures for scan job creation and linking with normal job objects.
- Modify the DNS-SD advertisement to support both printing and scanning.
- Merging into Upstream 😁