This is document is a draft. If you have any suggestions on important capabilities to support c anything else, please let us know. Specifically, we would like feedback on:

- Important capabilities that have been left out
- Complexity or simplicity of the design
- Difficulty in implementation

GCP Semantic States

1 Introduction

Cloud Device semantic State (CDS) and Cloud Job semantic State (CJS) are based upon and augment the CDD format, providing the means to collect, store and retrieve device and job states and state transitions.

2 Semantic Device State

CDD strives to be device type agnostic, therefore, the basic set of defined states apply not only

to printers, but to any cloud ready devices.

2.1 Device State Type

```
// Supported device states.
enum DeviceStateType {

  // Device is ready to accept jobs. Self-testing, low power and all other
  // states in which device can start processing newly submitted jobs without user
  // intervention should be mapped into this state.
  IDLE = 0;

  // Processing jobs (e.g. printing).
  PROCESSING = 1;

  // Device cannot process jobs. User should fix the problem to resume the processing
  // (e.g. printer is out of paper).
  STOPPED = 2;
}
```

Initialization and shutdown states are intentionally left out as a device should not connect to Cloud Print until it is ready to accept jobs.

2.2 Cloud Device State

The entire state of the device is represented by the CloudDeviceState message. Devices of different types utilize different sections of this message. Multifunctional devices can use several or all of the defined sections. More sections are expected to be added in the future, as Cloud Print expands the set of supported device types.

```
// Represents the entire cloud-connected device state.
message CloudDeviceState {

  // Whether device is connected to the server. It is not intended to be reported
  // by the device, it's set by the server.
  optional CloudConnectionStateType cloud_connection_state = 1;

  // Defined for the device with printing capabilities.
  optional PrinterStateSection printer = 2;

  // Defined for the device with scanning capabilities.

}
optional ScannerStateSection scanner = 3;
```

2.3 Printer

Devices with printing capabilities use this section to reflect the state of the printing unit.

```
// Represents the printer state.
message PrinterStateSection {

  // Current printer state.
  optional DeviceStateType state = 1;

  // State of the input trays.
  optional InputTrayState input_tray_state = 3;

  // State of the output bins.
  optional OutputBinState output_bin_state = 4;

  // State of the printer doors/covers/etc.
  optional CoverState cover_state = 5;

  // State of the markers.
  optional MarkerState marker_state = 6;

  // State of the printer media paths.
  optional MediaPathState media_path_state = 7;

  // Vendor-specific printer state.
  optional VendorState vendor_state = 101;
}
```

2.4 Scanner

Devices with scanning capabilities use this section to reflect the state of the scanning unit.

```
// Represents the scanner state.
message ScannerStateSection {

  // Current scanner state.
  optional DeviceState state = 1;

  // Vendor-specific scanner state.
  optional VendorState vendor_state = 101;
}
```

3 Semantic Cloud Device State Details

State details are set to describe the device state (defined in 2.1 Device State Type) in order to further specify the generic *singular device state*. Whenever possible, device and job types (e.g. printer and print job) should utilize specific state messages tailored for the purpose of these types. State messages are designed to be extended later as we discover more generally applicable and useful state details.

Connectivity state is not intended to be reported by the device itself, it's set by the server.

```
// Device cloud connectivity state.
enum CloudConnectionStateType {
  UNKNOWN = 0;
  ONLINE = 1;
  OFFLINE = 2;
}
```

Device state details which cannot be expressed in the defined semantic messages, should be delivered as VendorState messages.

```
// Vendor specific state.
message VendorState {

  message Item {

    enum StateType {
      ERROR = 0;
      WARNING = 1;
      INFO = 2;
    }

    // Severity of the state.
    optional StateType state_type = 1;
    // User readable state description.
    optional string description = 2;
  }

  repeated Item item = 1;
}
```

```
// An example of vendor state.
```

```
{
  “item”: [
    {
      “state_type”: 0 /*ERROR*/,
      “description”: “System error 404.”
    },
    {
      “state_type”: 1 /*WARNING*/,
      “description”: “The waste toner box is almost full.”
    }
  ]
}
```

4 Semantic Printer State Details

Printer specific state details are listed in this section.

4.1 Input tray state

```
// State of the device’s input trays.
message InputTrayState {

  message Item {

    enum StateType {
      // Tray is functional.
      OK = 0;
      // Tray is out of media. Treated as error.
      EMPTY = 1;
      // Tray is open. Treated as error.
      OPEN = 2;
      // Tray is installed, but turned off or disconnected. Treated as error.
      OFF = 3;
      // Tray is present, but not functioning properly. Treated as error.
      FAILURE = 4;
    }

    // ID of the tray.
    optional InputTray.ID id = 1;

    // Current tray state.
    optional StateType state_type = 2;
    // Loaded media level, percent.
    // Ranges from 0.0 (empty) to 1.0 (fully loaded).
    optional double level_percent = 3;
    // Vendor specific message.
    optional string vendor_message = 101;
  }

  repeated Item item = 1;
}
```

```
// An example of input tray state.
```

```
{
  “item”: [
    {
      “id”: …,
      “state_type”: 0 /*OK*/,
      “level_percent”: 0.2
    },
    {
      “id”: …,
      “state_type”: 1 /*EMPTY*/,
      “level_percent”: 0.0
    },
    {
      “id”: …,
      “state_type”: 4 /*FAILURE*/,
      “vendor_message”: “SC123”
    }
  ]
}
```

4.2 Output bin state

```
// State of the device’s output bins.
message OutputBinState {

  message Item {

    enum StateType {
      // Bin is functional.
      OK = 0;
      // Bin is full and cannot receive output anymore. Treated as error.
      FULL = 1;
      // Bin is open. Treated as error.
      OPEN = 2;
      // Bin is installed, but turned off or disconnected. Treated as error.
      OFF = 3;
      // Bin is present, but not functioning properly. Treated as error.
      FAILURE = 4;
    }
```

```
    // ID of the bin.
    optional OutputBin.ID id = 1;

    // Current bin state.
    optional StateType state_type = 2;
    // Used space, percent. Ranges from 0.0 (empty) to 1.0 (full).
    optional double level_percent = 3;
    // Vendor specific message.
    optional string vendor_message = 101;
  }

  repeated Item item = 1;
}
```

```
// An example of output bin state.
{
  “item”: [
    {
      “id”: …,
      “state_type”: 0 /*OK*/,
      “level_percent”: 0.7
    },
    {
      “id”: …,
      “state_type”: 1 /*FULL*/,
      “level_percent”: 1.0
    },
    {
      “id”: …,
      “state_type”: 4 /*FAILURE*/,
      “vendor_message”: “SC123”
    }
  ]
}
```

4.3 Marker state

The state of toner, ink, staples and other consumables (besides media) is represented in the MarkerState message.

```
// State of the device markers (toner/ink/staples/etc).
```

```
message MarkerState {

  message Item {

    enum StateType {
      // Marker is functional.
      OK = 0;
      // Marker resource is exhausted. Treated as error.
      EXHAUSTED = 1;
      // Marker is removed. Treated as error.
      REMOVED = 2;
      // Marker is present, but not functioning properly. Treated as error.
      FAILURE = 3;
    }

    // ID of the marker.
    optional Marker.ID id = 1;

    // Current marker state.
    optional StateType state_type = 2;
    // Marker supply amount, percent. Ranges from 0.0 to 1.0.
    optional double level_percent = 3;
    // The estimated number of pages available marker supply amount would last.
    optional double level_pages = 4;
    // Vendor specific message.
    optional string vendor_message = 101;
  }

  repeated Item item = 1;
}
```

```
// An example of marker state.
{
  “item”: [
    {
      “id”: …,
      “state_type”: 0 /*OK*/,
      “level_percent”: 0.5,
      “level_pages”: 300
    },
    {
```

```
      “id”: …,
      “state_type”: 1 /*EXHAUSTED*/,
      “level_percent”: 0.0,
      “level_pages”: 0
    },
    {
      “id”: …,
      “state_type”: 3 /*FAILURE*/,
      “vendor_message”: “SC123”
    }
  ]
}
```

4.4 Media path

```
// State of the device media paths.
message MediaPathState {

  message Item {

    enum StateType {
      // Path is functioning.
      OK = 0;
      // Media is jammed. Treated as error.
      MEDIA_JAM = 1;
      // Path is is present, but not functioning properly. Treated as error.
      FAILURE = 2;
    }

    // ID of the media path.
    optional MediaPath.ID id = 1;

    // Current state.
    optional StateType state_type = 2;
    // Vendor specific message.
    optional string vendor_message = 101;
  }

  repeated Item item = 1;
}
```

```
// An example of media path state.
{
  “item”: [
    {
      “id”: …,
      “state_type”: 0 /*OK*/
    },
    {
      “id”: …,
      “state_type”: 2 /*FAILURE*/,
      “vendor_message”: “SC123”
    }
  ]
}
```

4.5 Cover state

The state of doors, covers and other types of user accessible compartments is reported and stored in CoverState message.

```
// State of the device covers (door/cover/etc).
message CoverState {

  message Item {

    enum StateType {
      // Default cover state (closed, does not need any attention).
      OK = 0;
      // Cover is open. Treated as error.
      OPEN = 1;
      // Cover is not functioning properly. Treated as error.
      FAILURE = 2;
    }

    // ID of the cover.
    optional Cover.ID id = 1;

    // Current cover state.
    optional StateType state_type = 2;
    // Vendor specific message.
```

```
    optional string vendor_message = 101;
  }

  repeated Item item = 1;
}
```

```
// An example of cover state.
{
  “item”: [
    {
      “id”: …,
      “state_type”: 0 /*OK*/
    },
    {
      “id”: …,
      “state_type”: 2 /*FAILURE*/,
      “vendor_message”: “SC123”
    }
  ]
}
```

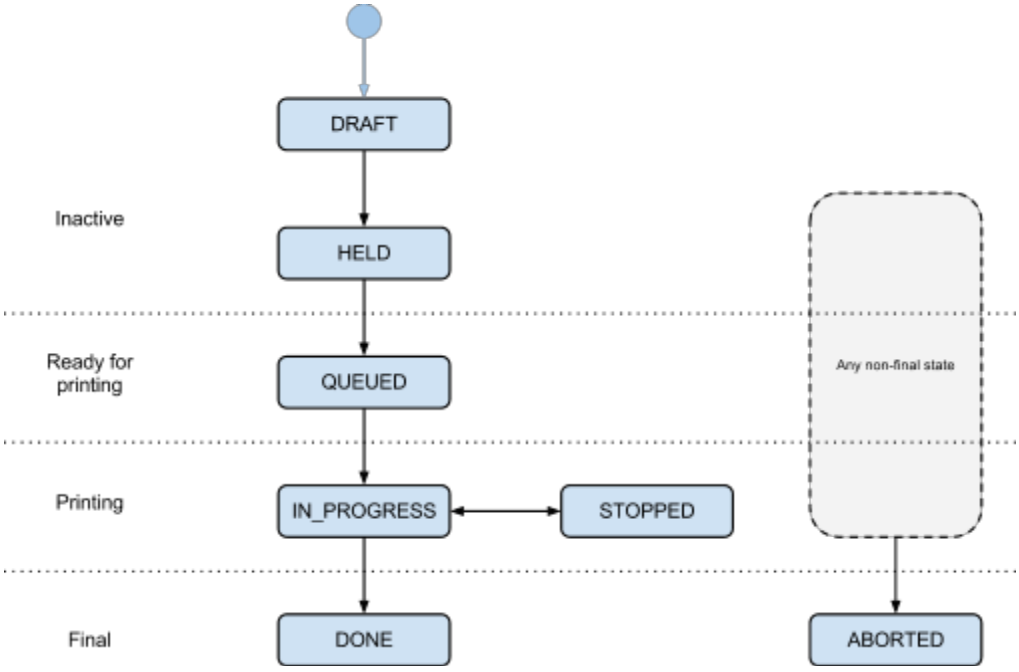
5 Semantic Job State

The basic set of states defined for generic job life cycle. These states are not specific to *printer* jobs, but may apply to jobs of any device type.

5.1 Job state

```
// Supported job states.
enum JobStateType {

  // Job is being created and is not ready for processing yet.
  DRAFT = 0;

  // Submitted and ready, but should not be processed yet.
  HELD = 1;

  // Ready for processing.
  QUEUED = 2;

  // Currently being processed.
  IN_PROGRESS = 3;

  // Was in progress, but stopped due to the error or user intervention.
  STOPPED = 4;

  // Processed successfully.
  DONE = 5;

  // Aborted due to the error or by the user action (canceled).
  ABORTED = 6;
}
```

Job state transitions expected and supported by Cloud Print:



5.2 Print Job State

Print job state is stored on the server and can be fetched by all clients who have access to the print job. Printers may request print job state changes for any jobs that are not in final state.

```
// Part of the device state holding the current state of the print job.
message PrintJobState {

  message Cause {

    enum Type {
      // Print job state was changed due to user action.
      USER_ACTION = 0;

      // Print job state was changed due to printer state change.
      PRINTER_STATE = 1;

      // Print job state was changed due to printer action.
      PRINTER_ACTION = 2;

      // Print job state was changed due to service (Cloud Print) action.
      SERVICE_ACTION = 3;
    }
```

```
    // Type of the cause.
    optional Type type = 1;
  }

  // Current state of the print job.
  optional JobStateType state = 1;

  // Whether current job state was caused by user or printer action, for example:
  // - {“state”: 6 /*ABORTED*/, “cause”: {“type”: 0 /*USER_ACTION*/}} interpreted
  //   as the job was canceled by the user;
  // - {“state”: 4 /*STOPPED*/, “cause”: {“type”: 1 /*PRINTER_STATE*/}} interpreted
  //   as the job was stopped due to temporary printer issues, such as paper jam
  //   (the actual cause can be discerned from the printer state).
  // - {“state”: 6 /*ABORTED*/, “cause”: {“type”: 2 /*PRINTER_ACTION*/}} interpreted
  //   as the job was aborted due to fatal printer problems, such as out of memory.
  optional Cause cause = 2;

  // Number of successfully printed pages. Printer should use this value to restart
  // interrupted/suspended print job from the next page.
  // Printer can only increase the number of pages printed.
  optional int64 pages_printed = 3;
}
```

```
// An example of print job in QUEUED state.
{
  “state”: 2 /*QUEUED*/
}

// An example of print job in IN_PROGRESS state.
{
  “state”: 3 /*IN_PROGRESS*/,
  “pages_printed”: 7
}

// An example of the canceled print job state.
{
  “state”: 6 /*ABORTED*/,
  “cause”: {
    “type”: 0 /*USER_ACTION*/
  }
```

```
  “pages_printed”: 7
}
```

6 Reporting and Retrieving Semantic Device and Job States

While CDS and CJS data structures are defined in Protobuf format, they are communicated to GCP via a corresponding JSON format. This format is a JSON serialized version of the protobuf messages. Notice that the various examples used in describing CDS and CJS are given in this JSON format.

The API to report and retrieve device and job state has not yet been determined.