Cloud Device and Job States

(Semantic State support for Google Cloud Print)

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 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 device types it supports.

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

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

  // Supported device states.
  enum StateType {

    // Device is ready to accept jobs. Self-testing, low power and all other
    // states in which the 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;
  }

  // Version of the CDS in the form "X.Y" where changes to Y are backwards
  // compatible, and changes to X are not (required).
  optional string version = 1;

  // 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 = 2;

  // Defined for devices with printing capabilities.
  optional PrinterStateSection printer = 3;
```

```
  // Defined for devices with scanning capabilities.
  optional ScannerStateSection scanner = 4;
}
```

2.2 Printer State Section

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

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

  // Current printer state (required).
  optional CloudDeviceState.StateType state = 1;

  // State of the input trays.
  optional InputTrayState input_tray_state = 2;

  // State of the output bins.
  optional OutputBinState output_bin_state = 3;

  // State of the markers.
  optional MarkerState marker_state = 4;

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

  // State of the printer media paths.
  optional MediaPathState media_path_state = 6;

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

2.3 Scanner State Section

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

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

  // Current scanner state (required).
  optional CloudDeviceState.StateType 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;
  NOT_CONFIGURED = 1;
  ONLINE = 2;
  OFFLINE = 3;
}
```

Device state details which cannot be expressed in the defined semantic messages should be delivered as VendorState messages. It is encouraged to omit periods from the end of all vendor messages if possible.

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

  message Item {

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

    // Severity of the state (required).
    optional StateType state = 1;
    // User-readable state description (required).
    optional string description = 2;
  }

  repeated Item item = 1;
}
```

```
// An example of vendor state.
{
  "item": [
    {
      "state": "ERROR",
      "description": "Interface board or control board failure"
    },
    {
      "state": "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 (refers to CDD model) (required).
    optional string vendor_id = 1;

    // Current tray state (required).
    optional StateType state = 2;
    // Loaded media level, percent. Ranges from 0 (empty) to 100 (fully loaded).
    optional int32 level_percent = 3;
    // Vendor-specific message, ignored when state == OK.
    optional string vendor_message = 101;
  }

  repeated Item item = 1;
}
```

```
// An example of input tray state.
{
  "item": [
```

```
    {
      "vendor_id": "...",
      "state": "OK",
      "level_percent": 20
    },
    {
      "vendor_id": "...",
      "state": "EMPTY",
      "level_percent": 0
    },
    {
      "vendor_id": "...",
      "state": "FAILURE",
      "vendor_message": "Input tray not detected"
    }
  ]
}
```

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 any more output. 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 (refers to CDD model) (required).
    optional string vendor_id = 1;

    // Current bin state (required).
    optional StateType state = 2;
    // Used space, percent. Ranges from 0 (empty) to 100 (full).
    optional int32 level_percent = 3;
    // Vendor-specific message, ignored when state == OK.
    optional string vendor_message = 101;
  }

  repeated Item item = 1;
```

```
}
```

```
// An example of output bin state.
{
  "item": [
    {
      "vendor_id": "...",
      "state": "OK",
      "level_percent": 70
    },
    {
      "vendor_id": "...",
      "state": "FULL",
      "level_percent": 100
    },
    {
      "vendor_id": "...",
      "state": "FAILURE",
      "vendor_message": "Service required"
    }
  ]
}
```

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 (refers to CDD model) (required).
    optional string vendor_id = 1;

    // Current marker state (required).
    optional StateType state = 2;
    // Marker supply amount, percent. Ranges from 0 to 100.
```

```
    optional int32 level_percent = 3;
    // Estimated number of pages for which the marker supply amount will last.
    optional int32 level_pages = 4;
    // Vendor-specific message, ignored when state == OK.
    optional string vendor_message = 101;
  }

  repeated Item item = 1;
}
```

```
// An example of marker state.
{
  "item": [
    {
      "vendor_id": "...",
      "state": "OK",
      "level_percent": 50,
      "level_pages": 300
    },
    {
      "vendor_id": "...",
      "state": "EXHAUSTED",
      "level_percent": 0,
      "level_pages": 0
    },
    {
      "vendor_id": "...",
      "state": "FAILURE",
      "vendor_message": "Toner sensor not working properly or cartridge defective"
    }
  ]
}
```

4.4 Cover state

The state of doors, covers and other types of user-accessible compartments is reported and stored in the 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 (refers to CDD model) (required).
    optional string vendor_id = 1;

    // Current cover state (required).
    optional StateType state = 2;
    // Vendor-specific message, ignored when state == OK.
    optional string vendor_message = 101;
  }

  repeated Item item = 1;
}
```

```
// An example of cover state.
{
  "item": [
    {
      "vendor_id": "...",
      "state": "OK"
    },
    {
      "vendor_id": "...",
      "state": "OPEN",
      "vendor_message": "Duplex unit cover is open"
    },
    {
      "vendor_id": "...",
      "state": "FAILURE",
      "vendor_message": "Front cover sensor failure"
    }
  ]
}
```

4.5 Media path state

```
// 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 present, but not functioning properly. Treated as error.
      FAILURE = 2;
    }
```

```
    // ID of the media path (refers to CDD model) (required).
    optional string vendor_id = 1;

    // Current state (required).
    optional StateType state = 2;
    // Vendor-specific message, ignored when state == OK.
    optional string vendor_message = 101;
  }

  repeated Item item = 1;
}
```

```
// An example of media path state.
{
  "item": [
    {
      "vendor_id": "...",
      "state": "OK"
    },
    {
      "vendor_id": "...",
      "state": "MEDIA_JAM",
      "vendor_message": "Paper jam in duplex unit"
    },
    {
      "vendor_id": "...",
      "state": "FAILURE",
      "vendor_message": "Sensor failure"
    }
  ]
}
```

4.6 Examples

Here's an example of the typical inkjet printer state, ready to print, no issues registered, as reported by the device. Notice that the "cloud_connection_state" field is omitted; it will be set by the server in the CDS returned by the printer lookup and search APIs (see section 6.1.3).

```
// Top-level CloudDeviceState message.
{
  "version": "1.0",

  "printer": {
    "state": "IDLE",

    "input_tray_state": {
      "item": [
        {
          "vendor_id": "tray1",
```

```
        "state": "OK",
        "level_percent": 20
      },
      {
        "vendor_id": "manual",
        "state": "OK"
      }
    ]
  },

  "output_bin_state": {
    "item": [
      {
        "vendor_id": "bin1",
        "state": "OK",
        "level_percent": 0
      }
    ]
  },

  "marker_state": {
    "item": [
      {
        "vendor_id": "ink",
        "state": "OK",
        "level_percent": 70
      }
    ]
  },

  "cover_state": {
    "item": [
      {
        "vendor_id": "front",
        "state": "OK"
      },
      {
        "vendor_id": "back",
        "state": "OK"
      }
    ]
  },

  "media_path_state": {
    "item": [
      {
        "vendor_id": "media_path",
        "state": "OK"
      }
    ]
  }
}
```

```
}
```

If a device has a large number of units commonly in the OK state with no additional information such as “level_percent”, then the CDS messages can be quite large but contain very little significant information. To minimize the size of the message, the server assumes that device units whose state is not mentioned in the CDS are in the OK state. Here’s the state of the same printer as above, all units are functional, except for the empty ink cartridge. Jobs are pending, hence the STOPPED device state.

```
// Top-level CloudDeviceState message.
{
  "version": "1.0",
  "printer": {
    "state": "STOPPED",
    "marker_state": {
      "item": [
        {
          "vendor_id": "ink",
          "state": "EXHAUSTED",
          "level_percent": 0
        }
      ]
    }
  }
}
```

5. Semantic Job State

The device-type agnostic portion of the state of a job on a cloud device is represented by a JobState message. The Type enumeration defines the basic set of states defined for the generic job life cycle. These state types are not specific to *printer* jobs, but may apply to jobs on any device type.

5.1 Job State

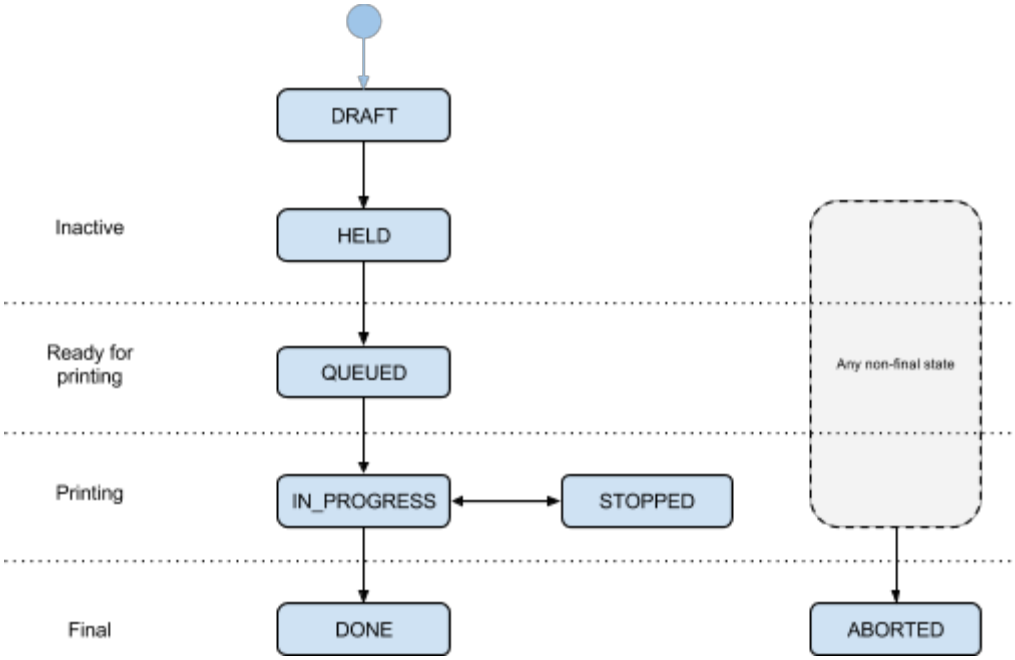
```
// Contains the device-agnostic state of a job on a cloud device.
message JobState {

  // Supported job state types.
  enum Type {

    // 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 error or user intervention.
    STOPPED = 4;

    // Processed successfully.
    DONE = 5;

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

  message UserActionCause {

    enum ActionCode {
      // User has cancelled the job.
      CANCELLED = 0;
      // User has paused the job.
      PAUSED = 1;
      // User has performed some other action.
      OTHER = 100;
    }

    // Code for the user action which caused the current job state (required).
    optional ActionCode action_code = 1;
  }

  message DeviceStateCause {

    enum ErrorCode {
      // Error due to input tray problem.
      INPUT_TRAY = 0;
      // Error due to marker problem.
      MARKER = 1;
      // Error due a problem in the media path.
      MEDIA_PATH = 2;
      // Error due to media size problem.
      MEDIA_SIZE = 3;
      // Error due to media type problem.
      MEDIA_TYPE = 4;
      // Error due to some other device state.
      OTHER = 100;
    }

    // Error code for the device state which caused the current job state
    // (required).
    optional ErrorCode error_code = 1;
  }
```

```
message DeviceActionCause {

  enum ErrorCode {
    // Error while downloading job.
    DOWNLOAD_FAILURE = 0;
    // Error due to invalid ticket.
    INVALID_TICKET = 1;
    // A generic printing error occurred.
    PRINT_FAILURE = 2;
    // Error due to some other device action.
    OTHER = 100;
  }

  // Error code for the device action which caused the current job state
  // (required).
  optional ErrorCode error_code = 1;
}

message ServiceActionCause {

  enum ErrorCode {
    // Contains error codes set internally by the Cloud Print server.
  }

  // Error code for the service action which caused the current job state
  // (required).
  optional ErrorCode error_code = 1;
}

// Current job state type (required).
optional Type type = 1;

// Exactly one of the following four fields must be set if and only if the
// state type is ABORTED or STOPPED.
// For example:
// - {"type": "ABORTED", "user_action_cause": {"action_code": "CANCELLED"}}
//   interpreted as the job was cancelled by the user.
// - {"type": "STOPPED", "device_state_cause": {"error_code": "MEDIA_PATH"}}
//   interpreted as the job was stopped due to a temporary problem with the
//   media path, such as paper jam (the specific cause will be discerned from
//   the device state by the server).
// - {"type": "ABORTED",
//    "device_action_cause": {"error_code": "DOWNLOAD_FAILURE"}}
//   interpreted as the job was aborted due to a download failure.

// If present, job state was changed due to user action.
optional UserActionCause user_action_cause = 2;

// If present, job state was changed due to device state change.
optional DeviceStateCause device_state_cause = 3;

// If present, job state was changed due to device action.
```

```
  optional DeviceActionCause device_action_cause = 4;

  // If present, job state was changed due to service (Cloud Print) action.
  // Should only be set by the Cloud Print server.
  optional ServiceActionCause service_action_cause = 5;
}
```

Job state type transitions expected and supported by Cloud Print:



A single device can have multiple functions (e.g. printing and scanning), but a job on such a device is performed by only one function. Therefore, there is a top-level job state format for each supported device function.

5.2 Print Job State

Print job state (PJS) is stored on the server and can be fetched by all clients with access to the job.

```
// Represents the current state of a print job on a cloud device.
message PrintJobState {

  // Version of the PJS in the form "X.Y" where changes to Y are backwards
  // compatible, and changes to X are not (required).
  optional string version = 1;

  // Current state of the job (required).
  optional JobState state = 2;

  // Number of successfully printed pages. Printer should use this value to
  // restart interrupted/suspended print jobs from the next page.
  // Printer can only increase the number of pages printed.
  optional int32 pages_printed = 3;
```

```
  // Number of attempts to deliver the print job (should only be set by the
  // Cloud Print server).
  optional int32 delivery_attempts = 4;
}
```

5.3 Print Job State Diff

Cloud devices may request job state changes for any jobs that are not in a final state (DONE or ABORTED) by sending a PrintJobStateDiff message.

```
// Diff that can be applied to a PrintJobState message. Any omitted field will
// not be changed.
message PrintJobStateDiff {

  // New job state.
  optional JobState state = 1;

  // New number of pages printed.
  optional int32 pages_printed = 2;
}
```

5.4 Examples

Here's an example of the state of a print job in progress, 4 pages printed so far.

```
// Top-level PrintJobState message.
{
  "version": "1.0",
  "state": {
    "type": "IN_PROGRESS"
  },
  "pages_printed": 4
}
```

Print job printed successfully (number of pages is optional to report).

```
// Top-level PrintJobState message.
{
  "version": "1.0",
  "state": {
    "type": "DONE"
  }
}
```

Print job was cancelled by the user.

```
// Top-level PrintJobState message.
{
  "version": "1.0",
```

```
  "state": {
    "type": "ABORTED",
    "user_action_cause": {"action_code": "CANCELLED"}
  },
  "pages_printed": 7
}
```

Print job stopped due to a temporary problem with an input tray, typically an empty input tray (the actual issue is reported in the printer's CDS, see CloudDeviceState message).

```
// Top-level PrintJobState message.
{
  "version": "1.0",
  "state": {
    "type": "STOPPED",
    "device_state_cause": {"error_code": "INPUT_TRAY"}
  },
  "pages_printed": 7
}
```

Print job state diff for a job that has newly entered the QUEUED state.

```
// PrintJobStateDiff message.
{
  "state": {
    "type": "QUEUED"
  }
}
```

6. Reporting and Retrieving Semantic Device and Job States

While the CDS and PJS data structures are defined in Protobuf format, they are communicated to Google Cloud Print 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 PJS are given in this JSON format.

6.1 Semantic Device State

There are a few GCP APIs where the CDS format can be used:

- Registering devices (/register)
- Printer update (/update)
- Printer lookup (/printer) and search (/search)

6.1.1 Printer Registration

The parameter “semantic_state” of the /register API can be used to supply the registration-time CDS of a printer as a JSON string. CDD must be provided too via the “capabilities” parameter as a file or a string in JSON format, and “use_cdd=true” must be included.

6.1.2 Printer Update

The parameters “semantic_state” and “semantic_state_diff” of the /update API can be used to update the CDS of an existing printer.

- If “semantic_state” is used, then the provided CDS message overwrites the CDS stored for the printer (or sets it for the first time).
- If “semantic_state_diff” is used, then the provided CDS message is interpreted as a “diff” on the CDS stored for the printer. Fields omitted in the diff are not changed in the original CDS. Nested message fields within PrinterStateSection and ScannerStateSection that are provided with empty values are removed. Fields of primitive or enum type and nested message fields that are provided with nonempty values are added or changed. The “version” field can always be omitted when using the “semantic_state_diff” parameter. When it is omitted and the printer did not previously have its CDS set, the Cloud Print server will assume the provided diff is using the latest version and it will set the version field accordingly.

When updating a printer’s CDS, if the printer’s CDD has not already been provided, it needs to be provided via the “capabilities” parameter and “use_cdd=true” must be included.

6.1.3 Printer Lookup and Search

The /printer (lookup) and /search APIs have a parameter “extra_fields” which is a comma-separated list of optional fields to include in the JSON response. The currently supported optional fields are “connectionStatus”, “semanticState”, “uiState”, and “queuedJobsCount”, so to include “semanticState” in the response, include the parameter “extra_fields=semanticState” in the request.

6.2 Semantic Job State

6.2.1 Print Job Update/Control

The parameter “semantic_state_diff” of the /control API can be used to provide a PrintJobStateDiff to update the state of an existing print job (or set it for the first time).

6.2.2 Print Job Lookup and Search

To include “semanticState” in the JSON response returned by the /job (lookup) and /jobs (search) APIs, include the parameter “extra_fields=semanticState” in the request.

7. Changelog

This section lists the **incompatible** changes in this document since June 3, 2013.

1. Added required “version” field to device and job state messages.
2. Changed the name of the field “state_type” in InputTrayState.Item, OutputBinState.Item, etc. to “state”.
3. Changed the type of all numeric fields to int32: the “level_percent” field of InputTrayState.Item, OutputBinState.Item and MarkerState.Item, the “level_pages” field of MarkerState.Item, and the “pages_printed” field of PrintJobState.
4. Reorganized job state to have a top-level format for each device function. The top-level state format for print jobs is now PrintJobState rather than CloudJobState.
5. Reorganized job state causes so that the JobState message has four message-type fields “user_action_cause”, “device_state_cause”, “device_action_cause” and “service_action_cause” (where it is required that exactly one is set) so that additional information can be included about the cause.
6. Changed example device state vendor messages to not end with a period. We now prefer the “description” of VendorState items and the optional “vendor_message” of printer part state items to not end with a period if possible.
7. Improved the semantics for causes in JobState messages (section 5.1).
   (a) Exactly one of the four cause fields must be set if the state type is ABORTED or STOPPED, otherwise none of the cause fields is allowed to be set.
   (b) User action causes have a required action code, and device state and device action causes have a required error code. The code can be specified as OTHER for user action, device state, and device action causes if none of the other codes accurately describe the cause.