



DURAFLEX™

Software Databook and Design Guide

Rev #: 1.00

SW Version: R3.1.1

Date: 09 June 2020

Memjet Confidential

© 2020 Memjet Technology Ltd. This document and information contained herein are the confidential and proprietary property of Memjet Technology Ltd.

© Copyright 2020 Memjet Technology Limited.

VersaPass and DuraLink are registered trademarks of Memjet Technology Limited.

This document is provided to the recipient subject to and in accordance with the terms of

- Non-Disclosure agreement;
- Provision of Technical Documentation and Samples Standard Terms and Conditions; and/or
- Component OEM Agreement

signed by recipient and Memjet.

To the extent permitted by law, this document is PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, INCLUDING BUT NOT LIMITED TO, IMPLIED WARRANTIES OF ACCURACY, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF ANY THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

Memjet will not be liable for any misstatements or omissions, including but not limited to misstatements or omissions in relation to instructions for the operation, use or maintenance of any equipment, samples, components or software. Recipient acknowledges and agrees that whilst care has been taken in compiling this information, it may contain estimates and draft information, and may not be current, accurate or complete. The information contained herein is subject to change without notice.

In the event of any conflict between the terms of this document and any executed agreement between recipient and Memjet, the terms in such executed agreement shall control and prevail. For the sake of clarity, only warranties expressly made in executed agreements will be binding on recipient and Memjet. Nothing herein should be construed as constituting an additional warranty.

Revision History

Doc. Version	SW Release	Date	Details
V1.00	R3.1.1	09-Jun-20	Initial release



Contents

1	Introduction.....	5
1.1	Aim and Audience	5
1.2	Prerequisites and Scope.....	5
1.3	Software Release Version.....	6
1.4	Typographic Conventions	6
1.5	Related Documentation.....	6
1.6	Further Information	6
2	Software Architecture.....	7
2.1	Memjet Software	8
2.1.1	PES State Machine.....	9
2.2	OEM-Provided Printer Control Software	10
3	PES Interfaces	11
3.1	PES Command Interface	11
3.2	PES Event Interface.....	11
3.3	PES Commands.....	12
3.4	Sequence Diagrams	16
4	PES Client Software Development.....	18
4.1	Common Client Operations	18
4.1.1	Preparing to Print.....	18
4.1.2	Printing.....	19
4.1.3	Clearing Faults.....	20
5	Getting Started with Development.....	21
5.1	Copyright and EULA.....	21
5.2	Software Requirements.....	21
5.3	DuraFlex Print Engine Network Configuration.....	22
5.3.1	Detect the Print Engine Hostname	22
5.4	OEM Software Development Distributable (OEM ISO Image).....	23
5.4.1	Kareela Development Package	23
5.5	Cookbook PES Application	25
5.5.1	Run the Cookbook PES Application from a Windows Client PC	25
5.5.2	Cookbook PES Application Usage	26
5.6	PES API Source Code Generation	29

Figures

Figure 1	Example Software Configuration – 1-Wide.....	7
Figure 2	Example Software Configuration – N-Wide	8
Figure 3	PES State Machine.....	9
Figure 4	Simple PDF Print Sequence.....	16
Figure 5	Simple RIP Print Sequence	17
Figure 6	Example Output in Windows PowerShell	22
Figure 7	Kareela Development Package.....	23
Figure 8	Directories in the Kareela Installer ZIP File	23



Figure 9 – Example Output of the Help Command	26
Figure 10 – Example Output to Invoke PES Commands	27
Figure 11 – Example Output of print_job Command	28

Tables

Table 1 – PES Commands.....	12
Table 2 – Code Example for Preparing to Print.....	18
Table 3 – Python Code Example for Printing	19
Table 4 – Code Example for Clearing Faults	20
Table 5 – Software Tools.....	21



1 Introduction

The DuraFlex Software Databook and Design Guide forms part of the documentation suite for the Memjet DuraFlex Modules. Other Databook and Design Guides provide technical detail about specific modules for targeted engineering audiences.

1.1 Aim and Audience

The aim of this document is to provide DuraFlex Print Engine Supervisor (PES) programming information to OEM software developers to assist them with the development of their own printer control software.

After reading this document the reader is expected to be able to begin development on the OEM components that control the DuraFlex Print Engine Modules via the PES Software Application Program Interface (API) to perform printing operations.

This document is also expected to be used as a reference document for OEM software developers during software development. This document is targeted at OEM software and firmware engineers who are responsible for integrating the DuraFlex printing system with the OEM print engine.

1.2 Prerequisites and Scope

This document assumes general knowledge about inkjet printing and printing systems. This document provides guidance on development of the OEM software required to control a DuraFlex print engine. It covers:

- DuraFlex software architecture information, Print Engine Software interface, and code examples
- Instructions for setting up a Software Development Environment (SDE)
- The Cookbook PES Application: An example OEM Printer Control Software developed by Memjet for use as a reference design to aid OEM development
- Troubleshooting information

This document does not cover:

- Hardware drivers and other low-level software. The PES Software interfaces with an array of smaller software components that control the many aspects of the Print Engine. Similarly, to the hardware specification, detailed knowledge of these components is not necessary to use DuraFlex by design.
- Information pertaining to the development of OEM-specific RIP solutions. Reference information can be found within the [jobSubmissionLib.h](#) supplied with the PES software installation package.
- Hardware specifications that can be found in the *DuraFlex Mechanical and Fluidic Databook and Design Guide*.
- Mechanical, electrical, design, installation, operation, troubleshooting, or servicing of a DuraFlex-based printing system.

This document does not describe other DuraFlex software interfaces such as the host driver, image stitching and alignment, color profile generation, and the Job Submission Library (JSL):

- The host driver is used in 1-wide systems to generate PDF files and deliver to the DuraFlex print engine for printing. It can be branded and customized by the OEM.
- Image stitching and alignment apply to N-wide systems to ensure the array of DuraFlex print engines produce the expected print quality.
- Color profiles are created to provide the desired visual output on various media.



- The JSL is required when using an external RIP and is used by the RIP for generating “GBOR format” files for consumption by the DuraFlex print engine. An external RIP is required in N-wide systems and is optional in 1-wide systems.

1.3 Software Release Version

The information in this document applies to the software release reflected in the [Revision History](#) table.

1.4 Typographic Conventions

Throughout this document, the following typographic conventions are used:

Code Character	<code>Courier</code> font is used to identify HTTP GET and POST commands with associated arguments, as well as references to source code, job states, registry settings, directory/file names, XCI commands, and XML settings.
Bold	Text that appears on-screen in the user interface is shown in bold font. This includes UI buttons, engine states, warning codes, and fault codes.
Yellow Highlighting	Yellow highlighting indicates sections that are new or updates in this version of the document, compared to the previous version.

1.5 Related Documentation

Other documents, besides this guide, provide further details for specific readers:

- *Product Foundation and Design Fundamentals Databook* – For OEM managers and non-technical personnel charged with evaluating the DuraFlex components for use within their products. This document describes the DuraFlex concept and Memjet-supplied DuraFlex components and gives an overview of the operational considerations. It introduces the components an OEM is required to design and manufacture to ensure the DuraFlex Modules function as designed in a DuraFlex-based print engine.
- *Mechanical and Fluidics Databook and Design Guide* – For mechanical design engineers and developers, providing details of the Memjet hardware modules and components (including printhead and maintenance system).
- *Electrical and Electronics Databook and Design Guide* – For electrical design engineers and developers, providing details of the Memjet power requirements, electronic assemblies, and connections.
- *Installation and Testing Guide* – For OEM personnel who are installing and commissioning a new printing system.
- *Demo GUI User Guide* – For OEM personnel who are controlling the printing tasks using the DuraFlex Demo GUI.
- *Operations and Troubleshooting Guide* – For OEM engineers to carry out operational tasks, identify symptoms and resolve issues.
- *Technical Bulletins* – For various audiences to announce product or process update or to provide specifics on single-subject technical topics.
- *PDFs of CAD and Schematics* – For various audiences to provide detailed dimensions related to specific areas.

1.6 Further Information

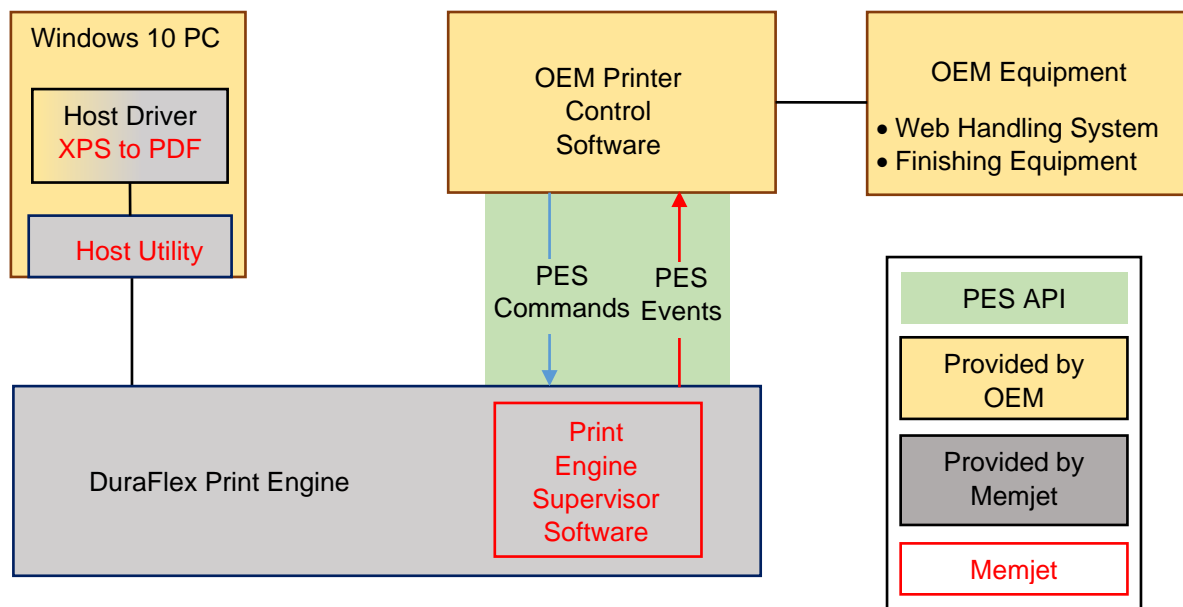
For Technical Documentation access or questions, please contact your Memjet technical support team.



2 Software Architecture

To support the modular design of the DuraFlex printing system, DuraFlex software includes a series of clearly defined functional interfaces. An overview of a 1-wide DuraFlex-based printer is shown below.

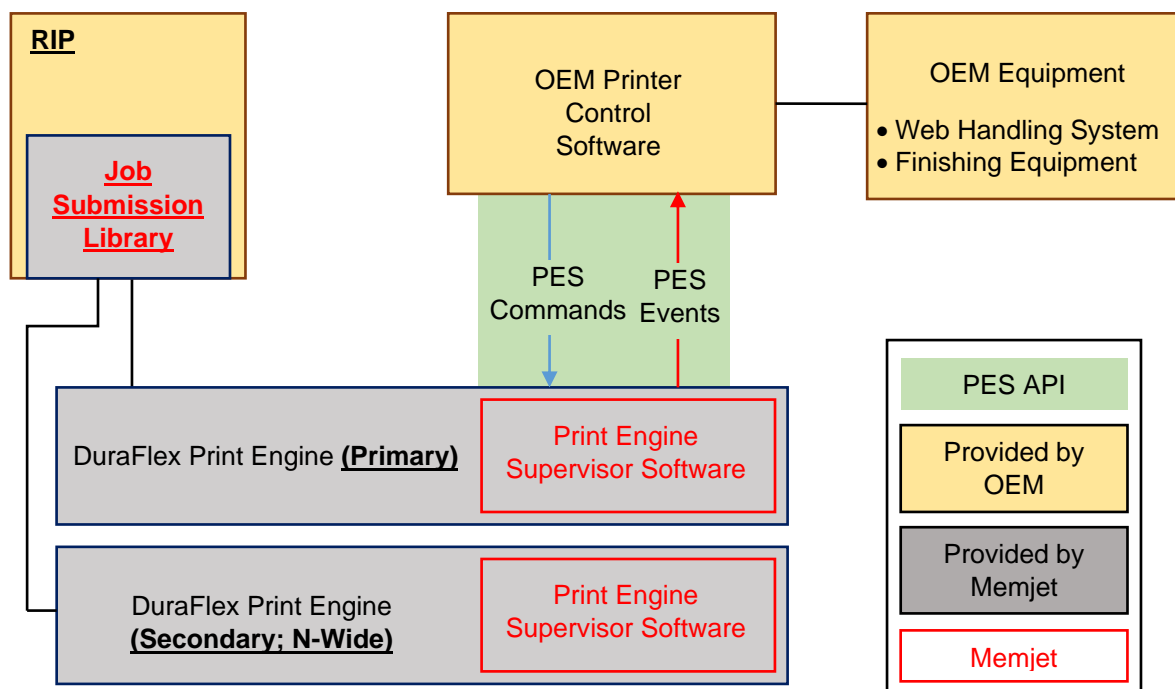
Figure 1 – Example Software Configuration – 1-Wide



In an N-wide printer system, where the value of N is between 1 and 4, one of the Print Modules will be configured as the “primary” engine, shown in [Figure 2](#), and will provide the Print Engine Supervisor (PES) interface for the OEM.

Note: PES commands are the same as the 1-wide implementation and will use an index for engine-specific needs (such as replacing the wiper in a specific engine as opposed to starting a print job across all engines).



Figure 2 – Example Software Configuration – N-Wide

Note: The differences between two graphics above are shown in **bold text with underline** in [Figure 2](#).

Memjet provides the PES interface as a set of Apache Thrift Interface Description Language (IDL) files along with API documentation and source code for C++ and Python, plus a simple test application written in Python to interface with the PES. OEMs will need to develop their own Printer Control Software to communicate through the PES interface using a programming language of their choice.

To control the printing system, the PES software serves as the “back-end”, while the OEM Printer Control Software works as the “front-end” user interface. The back-end and front-end communicate with each other through the PES interface, which is the API (Application Programming Interface). Each action from the user in the front-end (i.e. button click or data entry) triggers a request, and the back-end will send over an immediate response to present on the front-end. While numerous interactions occur in between, the PES API enables the OEM Printer Control Software and the PES software to work seamlessly together to fulfill the user's printing requirements.

2.1 Memjet Software

The PES software is a top-level software application that provides a single point of control and reporting to the various software and hardware components in a DuraFlex-based Print Engine. The PES software provides Apache Thrift interfaces (PES Command/Events). The PES Software also expects the OEM Printer Control Software to have provided its own Thrift interfaces in accordance with the requirements given in this document to connect with the PES software supervisor.

The functions in [Figure 3](#) are the PesCommand Thrift Service calls that trigger state transitions. For details of the use of these functions, refer to the Thrift IDL documentation.

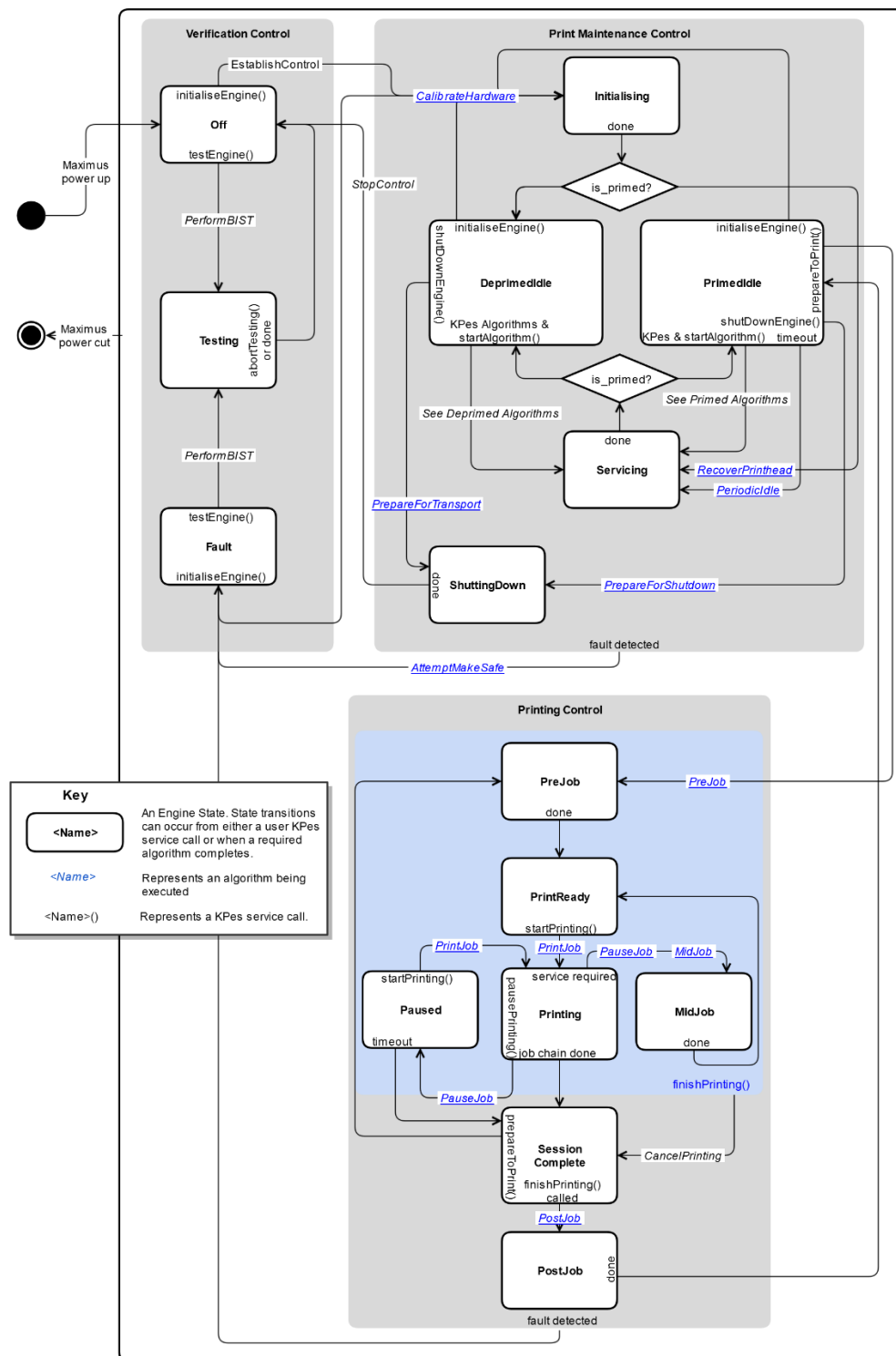
To perform an operation with the PES Software, the print engine must first be put into the correct state. Then the desired operation can be triggered.



2.1.1 PES State Machine

Operation of the PES software follows the state machine shown in [Figure 3](#).

Figure 3 – PES State Machine



09-Jun-20

Memjet Confidential

© 2020 Memjet Technology Ltd. This document and information contained herein are the confidential and proprietary property of Memjet Technology Ltd.



DURAFLEX™

2.2 OEM-Provided Printer Control Software

The OEM-provided Printer Control Software provides the User Interface for the DuraFlex-based printer as defined/required by the OEM. At a minimum, it must be capable of controlling the DuraFlex-based print engine and interfacing with the media handling system.

Note: [Section 4 PES Client Software Development](#) provides an example of a simple Printer Control Software to assist OEMs. The OEM may choose to add control of associated equipment into their implementation to control OEM equipment such as web handling systems or finishing equipment.



3 PES Interfaces

There are two primary interfaces provided to the OEM; a Command Interface (PES Commands) and an Event Notification Interface (PES Events) as shown in [Figure 1](#) and [Figure 2](#). Both are served by the PES software and accessed by OEM clients. The Command Interface provides a set of high-level commands to control the movements of a DuraFlex print engine, while the Event Interface provides reports whenever an event (progress or error) occurs within the print engine. These two interfaces are working individually to provide synchronous command/response, and asynchronous event notification.

3.1 PES Command Interface

This is a simple blocking, synchronous, Command-and-Response interface. All commands will be responded to within a few milliseconds. If the response is only an acknowledgement, a notification of operation completion will be provided via the Event interface. Multiple clients may connect with the Command Interface to request status information. To prevent conflicts between clients only one client should issue commands that may cause a change in print engine state. The PES does not monitor the source of commands, so it is up to the OEM to enforce this restriction.

The commands are documented in the `PesCommand.thrift` Thrift IDL file, which can be grouped into the following categories:

- System Configuration - e.g. get product info, status (read only) and get/set settings (read/write).
- Print Process control - e.g. initialise, prepare, start, pause and finish printing.
- Maintenance control - e.g. prime, deprime and service.

For a walkthrough of some common use cases of the Command Interface, see [Section 3.3 PES Commands](#).

3.2 PES Event Interface

The Event Interface is also blocking however, in this case, a client's call to the interface will be expected to block until an event occurs. If there are new events already available a call will return immediately. If a client is blocked waiting for the next event when the OEM wishes to shut it down, a call to the `abortEventSession` command will force the event server to release the block. The session to be aborted is identified by a `sessionId` which may be obtained ahead of time via a call to `getSessionId` on this Event interface. See [Section 3.3 PES Commands](#).

The PES software maintains an event sequence number so that the client always knows which events it has received. At each call to `getNewEvents`, the client provides the last sequence number that it knew about, and the PES will either return all events that have occurred since that one, or block until a later-numbered event occurs. At restart, the PES always resets the sequence number to 1. Hence, every time a client connects to the Event Interface, they should initialise their view of the printer by calling `getStatus` on the Command Interface. This will provide both the current status and the latest sequence number. From that point, clients can use the Event Interface to maintain their view of the print engine status.

The event cache is a fixed size store that will wrap and overwrite the oldest message when it is full, so if the client leaves it too long before calling the Event interface again, events may be lost.

If this occurs the client should restart their view by calling `getStatus` as described above. While polling `getStatus` is possible, Memjet recommends that the Event Interface be used instead. The use of polling in a complicated system can cause unpredictable behavior.



3.3 PES Commands

The main commands used to control the DuraFlex Print Module are in the Thrift IDL file `KPesCommands.thrift` and are listed here in simpler form to aid developers in gaining familiarity with the system. These commands are used in the Cookbook PES Application in the `PesCommander` class in `pesClientExample.py`.

Table 1 – PES Commands

Command	Description
<code>getApiBuild()</code> // see <code>pesClientExample.py</code> <code>__init__()</code>	Returns the Build ID of the API supported by the command server to check compatibility between client and server
<code>isApiCompatible()</code> // see <code>pesClientExample.py</code> <code>__init__()</code>	Runs the compatibility check between client and server
<code>getProductInfo()</code>	Not yet implemented
<code>getSettings()</code>	<ul style="list-style-type: none"> Not yet implemented The <code>getSettings()</code> and <code>storeSettings()</code> commands will be used to read and set items as noted in <code>KPesSettings.thrift</code> The settings for KWS, TOF sensor, Encoder are described in the Operations and Troubleshooting guide
<code>storeSettings()</code>	Not yet implemented
<code>getStatus()</code> // see <code>pesClientExample.py</code> <code>do_status()</code>	<ul style="list-style-type: none"> Returns a snapshot of engine status Can be called at any time
<code>abortEventSession()</code> // see <code>pesClientExample.py</code> <code>do_exit()</code>	<ul style="list-style-type: none"> Unlocks an event client before shutting down the client Causes calls to <code>getNewEvents()</code> to return <code>RESULT_ABORTED</code> Can be called at any time
<code>clearJobQueue()</code> // see <code>pesClientExample.py</code> <code>do_clear_job_queue()</code>	<ul style="list-style-type: none"> Removes all jobs from the job queue Call in states other than <code>EngineState.PRINTING</code> Otherwise throws exception and returns <code>RESULT_BUSY</code>
<code>setJobAllowNext()</code>	<ul style="list-style-type: none"> Not yet implemented Will be used for job chaining. Until implemented, concatenating PDFs into a single file would be equivalent in concept
<code>getJobCustomInfo()</code>	<ul style="list-style-type: none"> Not yet implemented OEMs can add custom information to print jobs and this is the command to get that data from the print engine Until implemented, any custom information that would be used will have to be managed outside of DuraFlex Returns <code>RESULT_BAD_PARAM</code> if job specified is not found in the queue



Command	Description
<pre>startMovingPrintheads() // see pesClientExample.py do_move_printheads()</pre>	<ul style="list-style-type: none"> • This is a lower-level command to move the printhead to a specific position (such as cap, wipe, or print) • This is not used in the normal course of printing but is used after a call to <code>replaceWipers()</code> • Must only be called during <code>EngineState.PRIMED_IDLE</code> or <code>EngineState.DEPRIMED_IDLE</code> • Otherwise throws exception and returns <code>RESULT_BUSY</code>
<pre>replaceWipers() // see pesClientExample.py do_replace_wipers()</pre>	<ul style="list-style-type: none"> • This moves the wiper to the far side of the engine for removal • Must be followed by a <code>startMovingPrintheads()</code> command to move the printhead and wiper back to cap position • Must only be called during <code>EngineState.PRIMED_IDLE</code> or <code>EngineState.DEPRIMED_IDLE</code> <ul style="list-style-type: none"> • Otherwise throws exception and returns <code>RESULT_BUSY</code>
<pre>initialiseEngine() // see pesClientExample.py do_clear_fault()</pre>	<ul style="list-style-type: none"> • Initialises the print engine. Transitions to the <code>EngineState.PRIMED_IDLE</code> or <code>EngineState.DEPRIMED_IDLE</code> state via the <code>EngineState.INITIALISING</code> state. • Must only call during <code>EngineState.OFF</code> <ul style="list-style-type: none"> • Otherwise throws exception and returns <code>RESULT_BUSY</code>
<pre>prepareToPrint() // see pesClientExample.py do_print_job()</pre>	<ul style="list-style-type: none"> • Activates a new print session and transitions to <code>EngineState.PRINT_READY</code> via <code>EngineState.PRE_JOB</code> • Must only be called during <code>EngineState.PRIMED_IDLE</code> • Otherwise throws exception and returns <code>RESULT_BUSY</code> • Normal return values: <ul style="list-style-type: none"> • Maximum media speed (unit: mm/s) based on the job y-resolution • <code>EngineState.RESULT_NOT_READY</code> if there was no valid, fully assembled job at the front of the job queue
<pre>startPrinting() // see pesClientExample.py do_print_job()</pre>	<ul style="list-style-type: none"> • This commands the engine to start printing pages of a loaded job • Can be called during either <code>EngineState.PRINT_READY</code> or <code>EngineState.PAUSED</code> • Should be called when the prints speed is stable, and the finishing equipment is ready • Transitions from <code>EngineState.PRINT_READY</code> to <code>EngineState.PRINTING</code> • This transitions to <code>EngineState.PRINTING</code> immediately • Can be called during <code>EngineState.PRE_JOB</code> or <code>EngineState.MID_JOB</code> and will transition to <code>EngineState.PRINTING</code> after servicing completes • Transitions to <code>EngineState.SESSION_COMPLETE</code> when job finished printing • Returns <code>RESULT_NOT_READY</code> if called when not in the following <code>EngineStates</code>: <ul style="list-style-type: none"> • <code>PRE_JOB</code>, <code>PRINT_READY</code>, <code>PAUSED</code>, <code>MID_JOB</code>



Command	Description
<code>pausePrinting()</code>	<ul style="list-style-type: none"> • Not yet implemented • No similar workaround • Pauses a print job. If a target page is given and the page has not yet printed, the pause will occur at that target page. Otherwise, the first available page will pause. • The pause will occur on a page boundary • The engine will transition to <code>EngineState.PAUSED</code> when the pause takes effect • Once paused, <code>startPrinting()</code> will resume and must be called before the pause timeout (used to protect the health of the printhead) <ul style="list-style-type: none"> • If a timeout occurs, state will change to <code>EngineState.SESSION_COMPLETE</code> • Job can then be continued by completing servicing then calling <code>prepareToPrint()</code> or can be ended by calling <code>finishPrinting()</code> • Can be called during <code>EngineState.PRINTING</code> or <code>EngineState.SESSION_COMPLETE</code>. If called in <code>EngineState.SESSION_COMPLETE</code>, it will be a null operation but not an error • If called from other states: <ul style="list-style-type: none"> • Throws exception and returns <code>RESULT_NOT_READY</code> • Returns error if target page not in job: <ul style="list-style-type: none"> • Throws exception and returns <code>RESULT_BAD_PARAM</code>
<pre>finishPrinting() // see pesClientExample.py do_print_job()</pre>	<ul style="list-style-type: none"> • Finish a print session and start post-job servicing • Transitions to <code>EngineState.PRIMED_IDLE</code> via <code>EngineState.POST_JOB</code> • If called during <code>EngineState.PRINTING</code> the current job will be cancelled • Can be called from any of these <code>EngineStates</code>: <ul style="list-style-type: none"> • <code>PRE_JOB</code>, <code>PRINT_READY</code>, <code>PAUSED</code>, <code>MID-JOB</code>, <code>PRINTING</code>, or <code>SESSION_COMPLETE</code> • If called from other states: <ul style="list-style-type: none"> • Throws exception and returns <code>RESULT_NOT_READY</code>
<pre>startPriming() // see pesClientExample.py do_start_priming()</pre>	<ul style="list-style-type: none"> • Starts priming the print engine • Must only be called from <code>EngineState.DEPRIMED_IDLE</code> • Transitions to <code>EngineState.PRIMED_IDLE</code> • If called from other states: <ul style="list-style-type: none"> • Throws exception and returns <code>RESULT_BUSY</code>
<pre>startDepriming() // see pesClientExample.py do_start_depriming()</pre>	<ul style="list-style-type: none"> • Starts depriming the print engine • Must only be called from <code>EngineState.PRIMED_IDLE</code> • Transitions to <code>EngineState.DEPRIMED_IDLE</code> • If called from other states: <ul style="list-style-type: none"> • Throws exception and returns <code>RESULT_BUSY</code>



Command	Description
<code>startServicing()</code> // see <code>pesClientExample.py</code> <code>do_start_service()</code>	<ul style="list-style-type: none">Starts specified service routines (such as <code>ServiceType.LIGHT</code>, <code>ServiceType.HEAVY</code>)Must only be called from <code>EngineState.PRIMED_IDLE</code> or <code>EngineState.DEPRIMED_IDLE</code>Transitions to <code>EngineState.SERVICING</code>If called from other states:<ul style="list-style-type: none">Throws exception and returns <code>RESULT_BUSY</code>
<code>startAlgorithm()</code>	For development use only
<code>shutDownEngine()</code> // see <code>pesClientExample.py</code> <code>do_clear_fault()</code>	<ul style="list-style-type: none">Used to transition to <code>EngineState.OFF</code> via <code>EngineState.SHUTTING_DOWN</code>Must only be called from <code>EngineState.PRIMED_IDLE</code> or <code>EngineState.DEPRIMED_IDLE</code>If called from other states:<ul style="list-style-type: none">Throws exception and returns <code>RESULT_BUSY</code>
<code>generateJobId()</code>	Only used in multi-engine setups

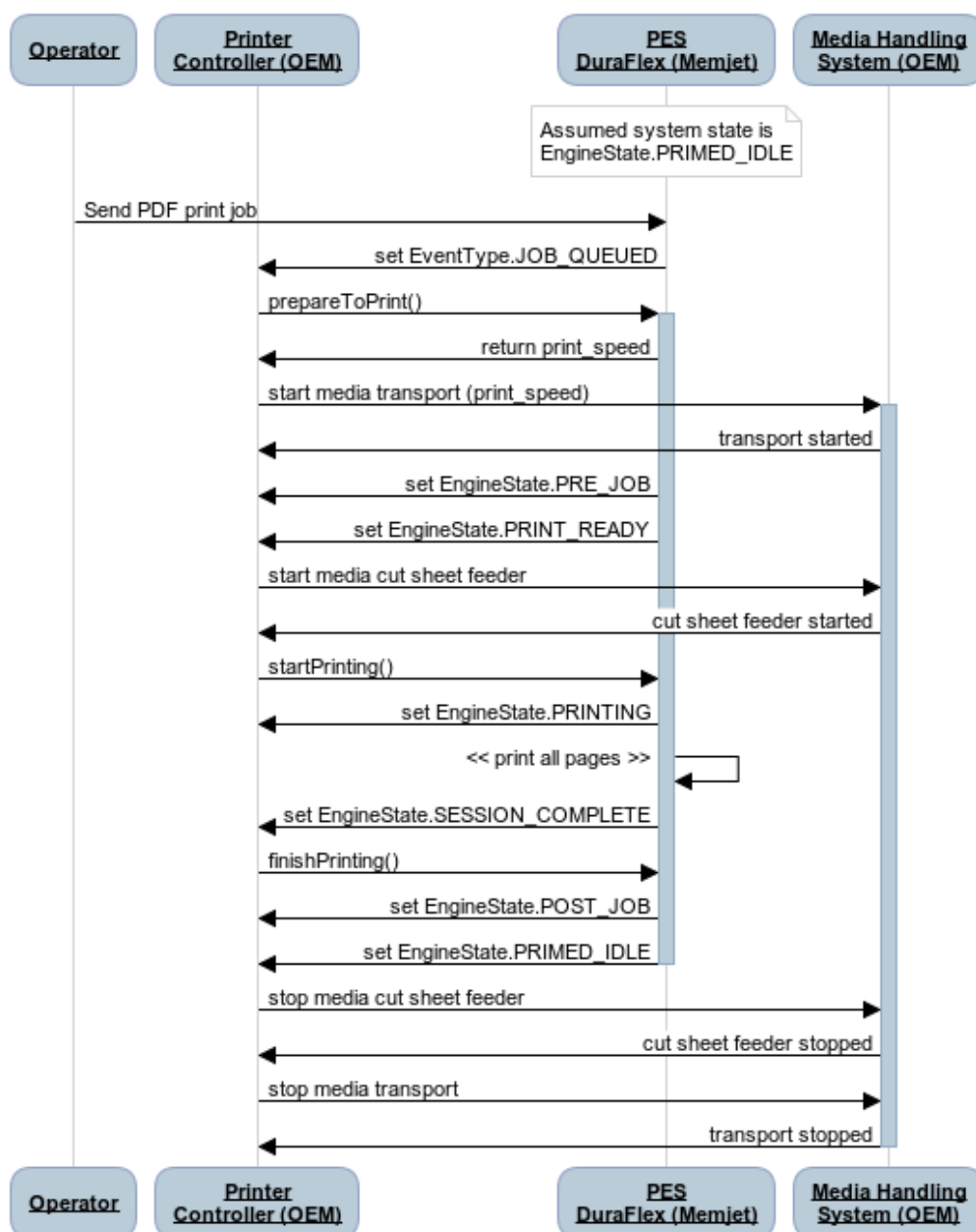


3.4 Sequence Diagrams

From the PES software perspective, the PDF and RIP print sequences share a lot of similarities. For example, when the user submits the print job (either PDF in the Embedded RIP mode or GBOR in the External RIP mode), the print job is queued, then the `prepareToPrint()` command enables the system to return print speed and start the media. When the `startPrinting()` command is running, the print job gets started. When the print engine is in the `SESSION_COMPLETE` state, the `finishPrinting()` command sets the print engine state as `PRIMED_IDLE` and stops the media.

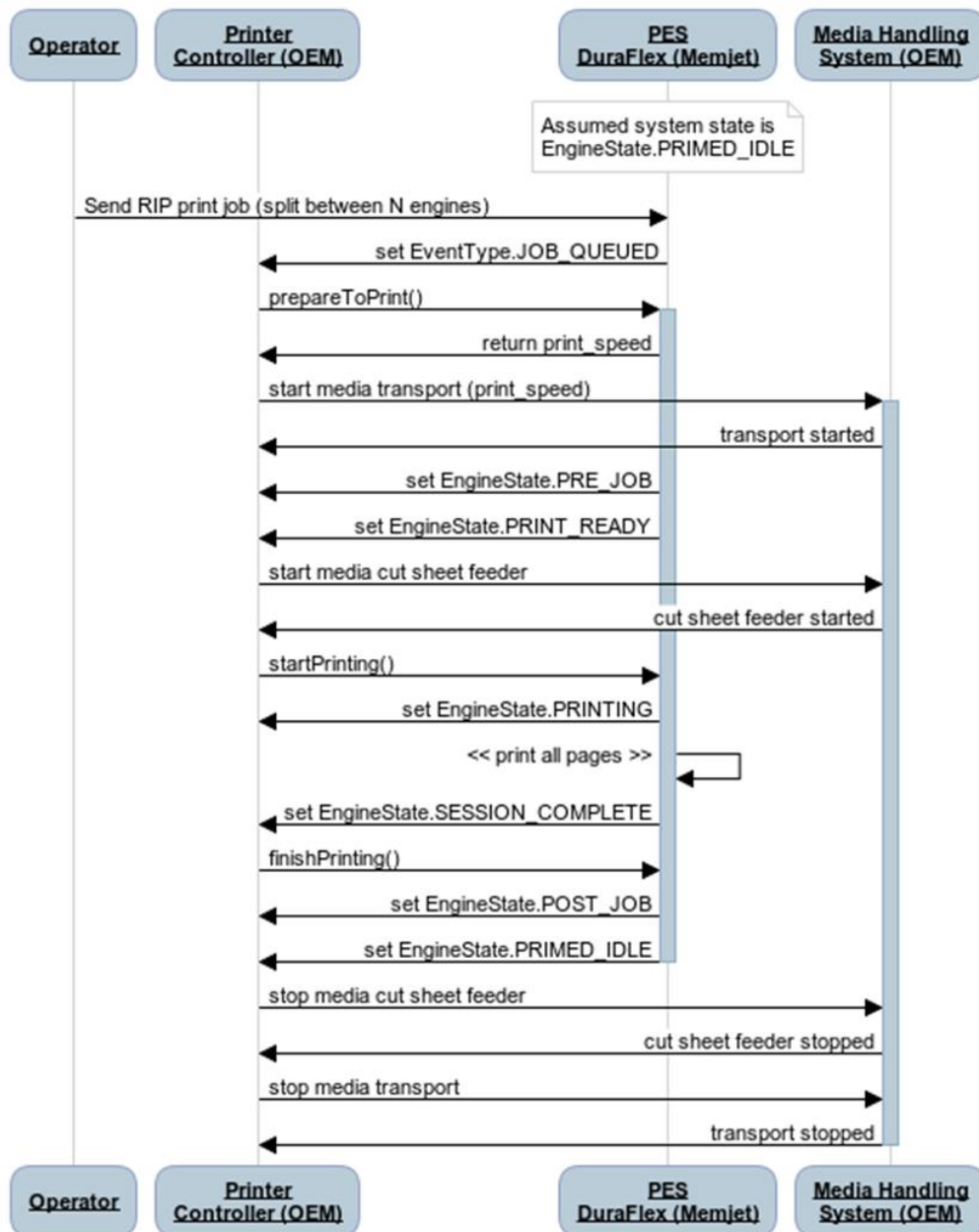
The diagram below shows the basic interactions and sequence involved in printing a job.

Figure 4 – Simple PDF Print Sequence



The diagram below shows the basic interactions and sequence involved in printing a RIP-based job.

Figure 5 – Simple RIP Print Sequence



4 PES Client Software Development

4.1 Common Client Operations

Refer to [Figure 4](#) for context of a print job sequence.

To assist with software development, the following Python examples demonstrate common printing scenarios. All examples assume the PES Software is running on a Virtual Machine (VM) emulator.

4.1.1 Preparing to Print

The following table shows the Python code for printing preparation:

Table 2 – Code Example for Preparing to Print

Python
<pre> print "Please Fire a Print Job." print "Waiting for Job..." # Set up event to wait for queued job def is_job_queue(event): return (event.type == evtConstants.EventType.JOB_QUEUED and event.details.job.state == comConstants.JobState.IDLE) # Test if a job is already queued, if not wait for a job. if (status.jobQueue and status.jobQueue[0].state == comConstants.JobState.IDLE) \ or self._wait_for_event_state(is_job_queue, status): print "Job Detected." else: return # A job is ready, prepare to print and get print speed print "Preparing to Print..." max_media_speed = self.cmd_client.prepareToPrint() # Can then use print speed to set up OEM media transport </pre>

Note: To allow for the necessary printing preparation, the PES software requires a job to be added to the print queue before the `prepareToPrint()` call is made.

The `prepareToPrint()` call allows the caller to get the maximum possible print speed for the job.



4.1.2 Printing

When waiting for a session to complete or a change in the status, monitor the events for an announcement. This is the same as changing the `EngineState` parameter, as it scales much better into large systems than making polling calls to the server.

Table 3 – Python Code Example for Printing

Python
<pre># Assumes prepareToPrint() has been called already # Wait for the PRINT_READY engine state after prepareToPrint() def is_print_ready(event): return (event.type == evtConstants.EventType.ENGINE_STATUS and event.details.engineStatus.state == stsType.EngineState.PRINT_READY) if not self._wait_for_event_state(is_print_ready, status): return print "Engine is Ready to Print." # Start the print! # Be sure to set up any OEM equipment prior (such as media feed) self.cmd_client.startPrinting() # Wait for the print job to complete printing. def is_session_complete(event): return (event.type == evtConstants.EventType.ENGINE_STATUS and event.details.engineStatus.state == stsType.EngineState.SESSION_COMPLETE) if not self._wait_for_event_state(is_session_complete, status): return print "Print Session Complete." # Resolve the print session and go back to IDLE. print "Finish Printing..." self.cmd_client.finishPrinting()</pre>

Note: The Python programming language assumes that the job prints successfully and may hang if the job is not successful. If you use the C++ programming language, it handles all error cases.



4.1.3 Clearing Faults

When problems are detected, for instance out-of-ink, missing hardware, etc., the engine will go into the FAULT state.

In this example, the `causefault()` call emulates some problems that cause the system to go into the FAULT state and then removes the cause of the faults so a later recovery can succeed. Normally the source of a fault will be something external to the software, meaning user action is required to rectify the fault. Once the external fault condition is removed, shut down and reinitialise to clear the software FAULT state. If the cause of the fault is still present, the engine will be in the IDLE state after reinitialisation. However, if the cause of the fault it still present, the engine will then go into the FAULT state again.

The following table shows the Python code for fault clearing:

Table 4 – Code Example for Clearing Faults

Python
<pre> status = self.cmd_client.getStatus() if status.engineStatus.state != stsType.EngineState.FAULT: print "FAULT state not detected." return # Shutdown the engine self.cmd_client.shutdownEngine(cmdType.ShutDownEngineParams()) # Set up event to wait for the OFF state. def is_engine_off(event): return (event.type == evtConstants.EventType.ENGINE_STATUS and event.details.engineStatus.state == stsType.EngineState.OFF) # Now wait for OFF state if not self._wait_for_event_state(is_engine_off, status): return print "Engine is OFF" # Initialise the engine from the OFF state self.cmd_client.initialiseEngine() # Set up event to wait for IDLE state def is_engine_idle(event): return (event.type == evtConstants.EventType.ENGINE_STATUS and (event.details.engineStatus.state == stsType.EngineState.PRIMED_IDLE or event.details.engineStatus.state == stsType.EngineState.DEPRIMED_IDLE)) # Now wait for idle if self._wait_for_event_state(is_engine_idle, status): print "Engine is IDLE" </pre>



5 Getting Started with Development

This section provides a high-level overview of the main steps involved in preparing for PES client software development and provides examples of software development environment setup. It adds context and recommended practice information to partner the technical information provided in the various Readme.txt files included with the PES software release package.

5.1 Copyright and EULA

All Memjet software is protected by copyright and its use is subject to an End-User License Agreement (EULA) located with the PES software installation. All Thrift IDLs and example code provided as part of the Printer Engine Supervisor software package are protected by Memjet's copyright and cannot be used without express permission.

5.2 Software Requirements

The Client PC should have Windows 10 operating system up and running.

The OEM also needs to install the following software tools on the Windows Client PC:

Table 5 – Software Tools

S/N	Software Name	Usage	URL for Downloading
1	Apache Thrift Library 0.9.3	Cross-language RPC framework	http://archive.apache.org/dist/thrift/0.9.3/thrift-0.9.3.exe
2	Bonjour for Windows	Resolves mDNS hostnames	https://support.apple.com/kb/DL999?locale=en_US
3	Python 2.7	Enables the OEM to run Python-based applications	https://www.python.org/downloads/release/python-2716/



5.3 DuraFlex Print Engine Network Configuration

The Print Engine (Datapath PCA) relies on an external DHCP server to obtain an IP address in the 1 Gigabit Ethernet (GbE) network. OEM needs to provide the network and DHCP server.

The Print Engine implements the following DNS protocols:

- **Multi-Cast DNS (mDNS):** This allows the hostname resolution of a DuraFlex Print Engine without the presence of DNS server.

The following software tools can resolve mDNS hostnames:

- Bonjour for Windows
- Avahi
- **DNS Service Discovery (DNS-SD):** This allows other hosts on the same Local Area Network (LAN) to discover the DuraFlex Print Engine and its services. This can be useful to identify DuraFlex Print Engine without knowing the hostname or IP address.

A mDNS or DNS-SD browser such as “avahi-browse” on Linux or “dns-sd” on Windows can discover the services by resolving the following service names:

- _kareela-command._tcp
- _kareela-event._tcp

5.3.1 Detect the Print Engine Hostname

Using an mDNS or DNS-SD browser, a device connected to the Print Engine network can detect the hostname or IP address and services that are broadcast from the Print Engine.

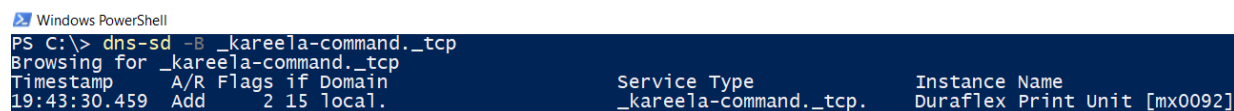
5.3.1.1 In the Windows Environment

1. Connect the Windows Client PC to the Print Engine’s network.
2. Install Bonjour for Windows. This provides the “dns-sd” browser utility and enables the mDNS hostname resolution.
3. In the Windows PowerShell, enter the following command:

```
dns-sd -B _kareela-command._tcp
```

The following screenshot shows an example of the response, where [mx0092] under the Instance Name is the hostname for this particular print unit.

Figure 6 – Example Output in Windows PowerShell



```

Windows PowerShell
PS C:\> dns-sd -B _kareela-command._tcp
Browsing for _kareela-command._tcp
Timestamp A/R Flags if Domain Service Type Instance Name
19:43:30.459 Add 2 15 local. _kareela-command._tcp. Duraflex Print Unit [mx0092]
  
```



5.4 OEM Software Development Distributable (OEM ISO Image)

DuraFlex software packages are distributed in the OEM ISO release, e.g. [Duraflex-OEM-R3.1.1.iso](#).

The ISO release provides a method of bundling and delivering DuraFlex development software packages that the OEMs will need to begin their development.

On the Windows Client PC, right-click [Duraflex-OEM-R3.1.1.iso](#) and select **Mount** from the pop-up menu. This allows the OEM to view the files on the ISO image as if it is a disk drive.

Any relevant files and packages for development are under the `\OEM` directory. You need to copy this directory to the local file system.

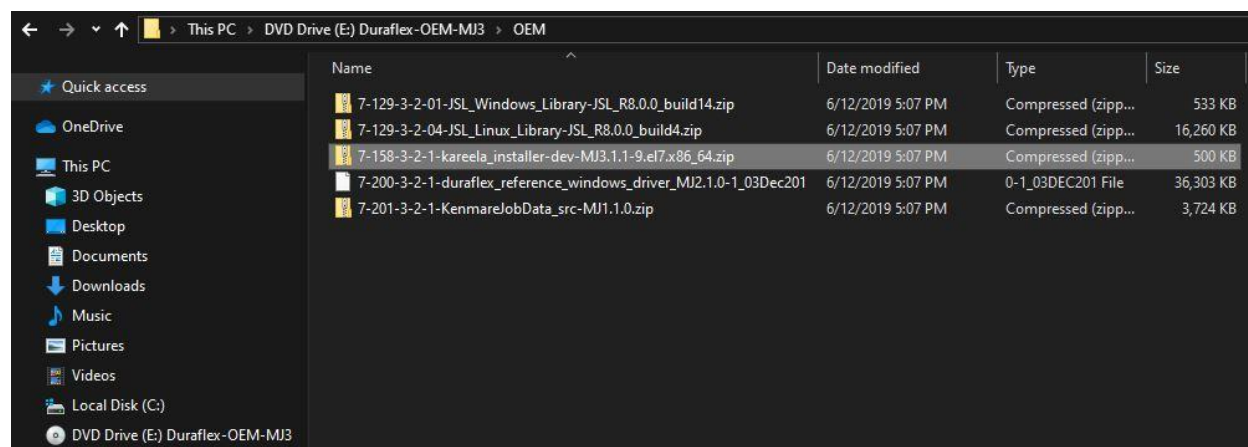
Kareela development package is required for PES API development:

[7-158-3-2-1-kareela_installer-dev-MJ3.1.1-9.el7.x86_64.zip](#)

Note: Within the software code, Kareela is the Memjet project name for PES software.

The following screenshot shows the Kareela development package in the `\OEM` directory:

Figure 7 – Kareela Development Package

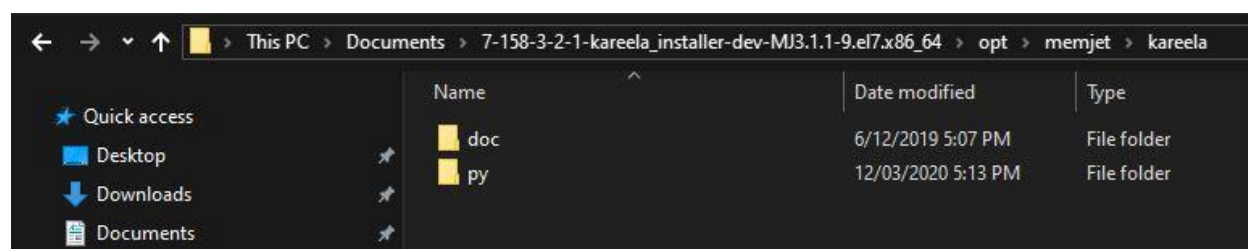


5.4.1 Kareela Development Package

Extract the Kareela development ZIP package, which includes the following contents:

- The `\opt\memjet\kareela\doc` directory contains the PES Software Interface documents.
- The `\opt\memjet\kareela\py` directory contains the Cookbook PES Application.

Figure 8 – Directories in the Kareela Installer ZIP File



5.4.1.1 PES Software Interface Documents in the Kareela Development Package

The `\opt\memjet\kareela\doc` directory contains the following subfolders and files:

```
\---Idl
|      KPesCommand.thrift
|      KPesCommon.thrift
|      KPesEvent.thrift
|      KPesProduct.thrift
|      KPesSettings.thrift
|      KPesStatus.thrift
|
\---ThriftLib
    \---Common
        MjCore.thrift
        Result.thrift
```

Apache Thrift IDL (Interface Definition Language) collection represents the PES Software Interface.

Refer to [Section 5.6 PES API Source Code Generation](#) to compile the IDL to a specific development language. The OEM must compile every `.thrift` file listed below to enable the PES Software Interface.

5.4.1.2 Cookbook PES Application in the Kareela Development Package

The `\opt\memjet\kareela\py` directory contains the following subfolders and files, including the pre-built Cookbook PES Application:

```
|      pesClientExample.py
|      ...
|
\---Memjet
|      ...
|
\---thrift
```

- The `pesClientExample.py` file means the "Cookbook PES Application". For usage details, refer to [Section 5.5.2 Cookbook PES Application Usage](#).
- The `\Memjet` directory contains the compiled Thrift IDL files of the PES software interface to python. This is used to invoke the commands on the PES interface in python and will be hooked to the Thrift network stack in `pesClientExample.py`.
- The `\thrift` directory contains the Thrift Library 0.9.3 for Python. This is responsible for implementing Thrift's Remote Procedure Call (RPC) network stack to enable communication between systems. The OEM needs to install Thrift Library 0.9.3 on the system or virtual environment.



5.5 Cookbook PES Application

An example printer controller ([pesClientExample.py](#)), which is the Cookbook PES Application, provides a solution to operate the Print Engine and serves as a cookbook with source code that provides code examples to invoke and navigate the PES Thrift Interface.

The Cookbook PES Application provides the following features:

- Connect to the Thrift service
- Initialise the Print Engine
- Initialise a service routine
- Wait for Print Engine state transitions or events
- Perform a printing job, and so on

5.5.1 Run the Cookbook PES Application from a Windows Client PC

1. Ensure that the Windows Client PC is able to detect the Print Engine on the network. For example, use the command `ping mxNNNN.local` to check.

The OEM needs to install Bonjour for Windows to resolve the mDNS hostname. For more information on how to set up the system network, see [Section 5.3 DuraFlex Print Engine Network Configuration](#).

2. Install Python 2.7 (See [Table 5](#) to find the URL for downloading).
3. Copy and extract the Kareela Development Package (See [Section 5.4 OEM Software Development Distributable \(OEM ISO Image\)](#)).
4. Open a Windows PowerShell terminal. Navigate to the following location in the extracted Kareela Development Package directory:

```
\opt\memjet\kareela\py
```

5. Run the following command:

```
python pesClientExample.py mxNNNN.local
```

Note: Replace “mxNNNN.local” with the hostname or IP address of the targeted Print Engine on the network.

6. While the commands are executed from [pesClientExample.py](#), all events raised from the Print Engine are logged into [pes_client.log](#) (See [KPesEvent.thrift:EventInfo](#)).

[pes_client.log](#) is located in the current directory where [pesClientExample.py](#) has been called from.

7. Open a new Windows PowerShell terminal. Navigate to the directory which contains the extracted Kareela Development Package.

Run the following command:

```
Get-Content pes_client.log -Wait
```

This command will monitor the log file and provides an output with any event updates from the Print Engine.



5.5.2 Cookbook PES Application Usage

When successfully connected to Print Engine, `pesClientExample.py` displays the PES API version and the current engine status.

A built-in command line interpreter invokes the primary PES commands to operate the Print Engine.

- If the OEM enters `help`, the response will display all available commands.
- If the OEM enters `help [command name]`, the response will display the usage details, parameters required, and the IDL documentation of the specified command.

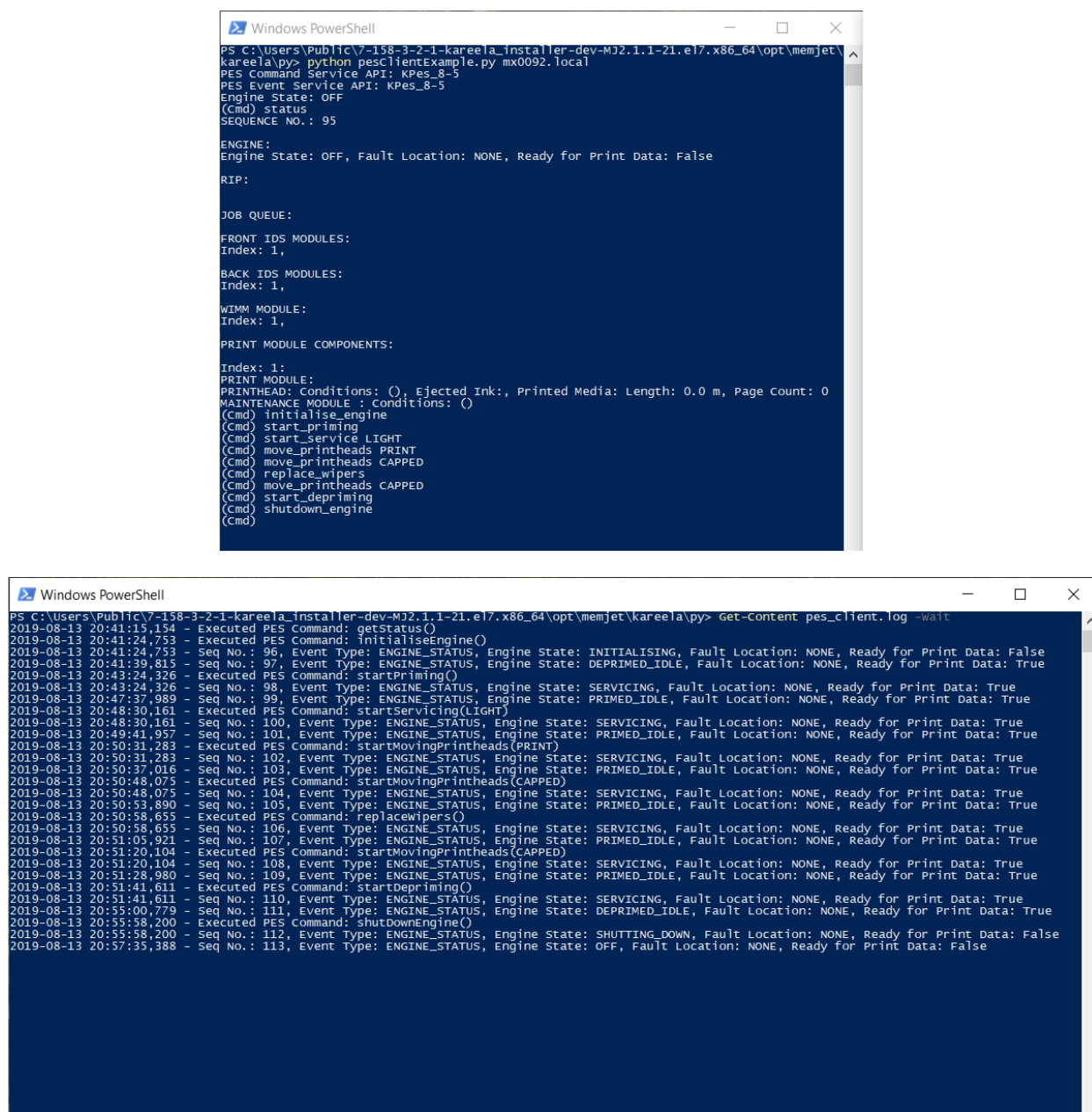
Figure 9 – Example Output of the Help Command

```

Windows PowerShell
PS C:\Users\Public\7-158-3-2-1-kareela_installer-dev-Mj2.1.1-21.e17.x86_64\opt\memjet\kareela\py> python pesClientExample.py mx0092.local
PES Command Service API: KPes_8-5
PES Event Service API: KPes_8-5
Engine State: DEPRIMED_IDLE
(cmd) help
Documented commands (type help <topic>):
=====
clear_fault      help            print_job       start_priming
clear_job_queue  initialise_engine replace_wipers  start_printing
exit             move_printheads shutdown_engine start_service
finish_printing  prepare_to_print start_depriming status
(cmd) help status
Syntax: status
Thrift IDL Doc:
    @return a snapshot of the full engine status
(cmd) help initialise_engine
Syntax: initialise_engine
Thrift IDL Doc:
    Initialise the engine.
    Used to transition to IDLE via INITIALISING.
    @note Must only be called during EngineState.OFF.
    @throw MjException The operation was not performed. Possible values
    of errorCode are:
    - RESULT_BUSY Function was called while not in EngineState.OFF.
(cmd)

```



Figure 10 – Example Output to Invoke PES Commands


```

Windows PowerShell
PS C:\Users\Public\7-158-3-2-1-kareela_installer-dev-MJ2.1.1-21.e17.x86_64\opt\memjet\kareela\py> python pesClientExample.py mx0092.local
PES Command Service API: KPes_8-5
PES Event Service API: KPes_8-5
Engine State: OFF
(cmd) status
SEQUENCE NO.: 95

ENGINE:
Engine State: OFF, Fault Location: NONE, Ready for Print Data: False

RIP:

JOB QUEUE:

FRONT IDS MODULES:
Index: 1,

BACK IDS MODULES:
Index: 1,

WIMM MODULE:
Index: 1,

PRINT MODULE COMPONENTS:
Index: 1:
PRINT MODULE:
PRINTHEAD: Conditions: (), Ejected Ink:, Printed Media: Length: 0.0 m, Page Count: 0
MAINTENANCE MODULE : Conditions: ()
(cmd) initialise_engine
(cmd) start_priming
(cmd) start_service LIGHT
(cmd) move_printheads PRINT
(cmd) move_printheads CAPPED
(cmd) replace_wipers
(cmd) move_printheads CAPPED
(cmd) start_depriming
(cmd) shutdown_engine
(cmd)

Windows PowerShell
PS C:\Users\Public\7-158-3-2-1-kareela_installer-dev-MJ2.1.1-21.e17.x86_64\opt\memjet\kareela\py> Get-Content pes_client.log -wait
2019-08-13 20:41:15,154 - Executed PES Command: getStatus()
2019-08-13 20:41:24,753 - Executed PES Command: initialiseEngine()
2019-08-13 20:41:24,753 - Seq No.: 96, Event Type: ENGINE_STATUS, Engine State: INITIALISING, Fault Location: NONE, Ready for Print Data: False
2019-08-13 20:41:39,815 - Seq No.: 97, Event Type: ENGINE_STATUS, Engine State: DEPRIMED_IDLE, Fault Location: NONE, Ready for Print Data: True
2019-08-13 20:43:24,326 - Executed PES Command: startPriming()
2019-08-13 20:43:24,326 - Seq No.: 98, Event Type: ENGINE_STATUS, Engine State: SERVICING, Fault Location: NONE, Ready for Print Data: True
2019-08-13 20:47:37,989 - Seq No.: 99, Event Type: ENGINE_STATUS, Engine State: PRIMED_IDLE, Fault Location: NONE, Ready for Print Data: True
2019-08-13 20:48:30,161 - Executed PES Command: startServicing(LIGHT)
2019-08-13 20:48:30,161 - Seq No.: 100, Event Type: ENGINE_STATUS, Engine State: SERVICING, Fault Location: NONE, Ready for Print Data: True
2019-08-13 20:49:41,957 - Seq No.: 101, Event Type: ENGINE_STATUS, Engine State: PRIMED_IDLE, Fault Location: NONE, Ready for Print Data: True
2019-08-13 20:50:31,283 - Executed PES Command: startMovingPrintheads(PRINT)
2019-08-13 20:50:31,283 - Seq No.: 102, Event Type: ENGINE_STATUS, Engine State: SERVICING, Fault Location: NONE, Ready for Print Data: True
2019-08-13 20:50:37,016 - Seq No.: 103, Event Type: ENGINE_STATUS, Engine State: PRIMED_IDLE, Fault Location: NONE, Ready for Print Data: True
2019-08-13 20:50:48,075 - Executed PES Command: startMovingPrintheads(CAPPED)
2019-08-13 20:50:48,075 - Seq No.: 104, Event Type: ENGINE_STATUS, Engine State: SERVICING, Fault Location: NONE, Ready for Print Data: True
2019-08-13 20:50:53,890 - Seq No.: 105, Event Type: ENGINE_STATUS, Engine State: PRIMED_IDLE, Fault Location: NONE, Ready for Print Data: True
2019-08-13 20:50:58,655 - Executed PES Command: replaceWipers()
2019-08-13 20:50:58,655 - Seq No.: 106, Event Type: ENGINE_STATUS, Engine State: SERVICING, Fault Location: NONE, Ready for Print Data: True
2019-08-13 20:51:05,921 - Seq No.: 107, Event Type: ENGINE_STATUS, Engine State: PRIMED_IDLE, Fault Location: NONE, Ready for Print Data: True
2019-08-13 20:51:20,104 - Executed PES Command: startMovingPrintheads(CAPPED)
2019-08-13 20:51:20,104 - Seq No.: 108, Event Type: ENGINE_STATUS, Engine State: SERVICING, Fault Location: NONE, Ready for Print Data: True
2019-08-13 20:51:28,980 - Seq No.: 109, Event Type: ENGINE_STATUS, Engine State: PRIMED_IDLE, Fault Location: NONE, Ready for Print Data: True
2019-08-13 20:51:41,611 - Executed PES Command: startDepriming()
2019-08-13 20:51:41,611 - Seq No.: 110, Event Type: ENGINE_STATUS, Engine State: SERVICING, Fault Location: NONE, Ready for Print Data: True
2019-08-13 20:55:00,779 - Seq No.: 111, Event Type: ENGINE_STATUS, Engine State: DEPRIMED_IDLE, Fault Location: NONE, Ready for Print Data: True
2019-08-13 20:55:58,200 - Executed PES Command: shutdownEngine()
2019-08-13 20:55:58,200 - Seq No.: 112, Event Type: ENGINE_STATUS, Engine State: SHUTTING_DOWN, Fault Location: NONE, Ready for Print Data: False
2019-08-13 20:57:35,388 - Seq No.: 113, Event Type: ENGINE_STATUS, Engine State: OFF, Fault Location: NONE, Ready for Print Data: False

```

All PES commands will return an exception if the invoked command is not acceptable for the current engine state. For example, `prepareToPrint()` will return an exception if the engine is in a deprimed state (`DEPRIMED_IDLE`).

Cookbook PES Application provides minimal abstraction over the PES command interface. Cookbook PES Application will return immediate responses for any invoked PES commands and print any Memjet or Thrift defined exceptions if these occur. See `def thrift_err_handler()` in `pesClientExample.py`.

Events (see `KPesEvent.thrift:EventInfo`) that are generated from the invoked command from the Print Engine will be logged to `pes_client.log`.



The OEM Printer Control Software is expected to handle any exceptions appropriately, as well as maintain and update its state based on events received from the PES Event Interface. It is the OEM's responsibility to develop the Printer Control Software.

The "EventLogger()" class in `pesClientExample.py` is a contained example on using the PES event interface. It has logging functionality to save any new events received from the PES to `pes_client.log`.

Not all commands available in the Cookbook PES Application are direct calls on the PES Command Interface.

The following commands are examples on how to sequence multiple PES command calls and wait for necessary events that indicate engine state transitions:

- `print_job`:
 1. Run the command `prepareToPrint()`
 2. Wait for the response to show `EngineState.PRINT_READY`
 3. Run the command `startPrinting()`
 4. Wait for the response to show `EngineState.SESSION_COMPLETE`
 5. Run the command `finishPrinting()`
- `clear_fault`:
 1. Run the command `shutdownEngine()`
 2. Wait for the response to show `EngineState.OFF`
 3. Run the command `initialiseEngine()`
 4. Wait for the response to show `EngineState.DEPRIMED_IDLE` or `EngineState.PRIMED_IDLE`

Figure 11 – Example Output of print_job Command

```

Windows PowerShell
PS C:\Users\Public\Documents> python pesClientExample.py mx0097.local
PES Command Service API: KPes_8-5
PES Event Service API: KPes_8-5
Engine State: PRIMED_IDLE
(Cmd) print_job
Please Submit a Print Job.
waiting for Job...
Job detected.
Are you confident the job has been fully buffered? Press Enter to continue...
Preparing to Print...
Please Start Media Movement. Max Media Speed: 492.125 mm/s
Engine is Ready to Print.
Press Enter to Start Printing...
Print Session Complete.
Finish Printing...
Please Stop Media Movement.
(Cmd)

Windows PowerShell
PS C:\Users\Public\Documents> Get-Content pes_client.log -wait
2019-08-14 18:32:22.582 - Executed PES Command: getStatus()
2019-08-14 18:32:35.568 - Seq No.: 20, Event Type: JOB_QUEUED, Job ID: 00000013, Job State: IDLE, Allow Next: False, Total Page Count: 4
2019-08-14 18:32:52.726 - Executed PES Command: prepareToPrint()
2019-08-14 18:32:52.729 - Seq No.: 21, Event Type: ENGINE_STATUS, Engine State: PRE_JOB, Fault Location: NONE, Ready for Print Data: True
2019-08-14 18:32:53.868 - Seq No.: 22, Event Type: ENGINE_STATUS, Engine State: PRINT_READY, Fault Location: NONE, Ready for Print Data: True
2019-08-14 18:33:03.256 - Executed PES Command: startPrinting()
2019-08-14 18:33:09.088 - Seq No.: 23, Event Type: JOB_STATUS, Job ID: 00000013, Job State: PRINTING, Allow Next: False, Total Page Count: 4
2019-08-14 18:33:10.112 - Seq No.: 24, Event Type: ENGINE_STATUS, Engine State: PRINTING, Fault Location: NONE, Ready for Print Data: True
2019-08-14 18:33:14.092 - Seq No.: 25, Event Type: JOB_STATUS, Job ID: 00000013, Job State: PRINTING, Allow Next: False, Total Page Count: 4, Completed Page Count: 1
2019-08-14 18:33:17.592 - Seq No.: 26, Event Type: JOB_STATUS, Job ID: 00000013, Job State: PRINTING, Allow Next: False, Total Page Count: 4, Completed Page Count: 2
2019-08-14 18:33:21.598 - Seq No.: 27, Event Type: JOB_STATUS, Job ID: 00000013, Job State: PRINTING, Allow Next: False, Total Page Count: 4, Completed Page Count: 3
2019-08-14 18:33:25.604 - Seq No.: 28, Event Type: JOB_COMPLETION, Job ID: 00000013, Result: SUCCESS, Ink used: CYAN: 0.07043630907 ml MAGENTA: 0.08026131956 ml YELLOW: 0.08026131956 ml
m. Page Count: 4, Summary: JobCondition: COMPLETE
2019-08-14 18:33:25.611 - Seq No.: 29, Event Type: ENGINE_STATUS, Engine State: SESSION_COMPLETE, Fault Location: NONE, Ready for Print Data: True
2019-08-14 18:33:25.615 - Executed PES Command: finishPrinting()
2019-08-14 18:33:25.617 - Seq No.: 30, Event Type: ENGINE_STATUS, Engine State: SERVICING, Fault Location: NONE, Ready for Print Data: True
2019-08-14 18:33:34.164 - Seq No.: 31, Event Type: ENGINE_STATUS, Engine State: PRIMED_IDLE, Fault Location: NONE, Ready for Print Data: True
  
```



5.6 PES API Source Code Generation

Apache Thrift is a cross-language RPC framework. DuraFlex Print Engine Supervisor (PES) uses the Thrift framework to enable OEMs to operate PES API in a variety of programming languages and environments.

For more details about Apache Thrift, refer to the official site: <https://thrift.apache.org/docs/>

It is required that OEM developers become familiar with the Thrift framework and the PES API Thrift IDL.

This section provides instructions to generate source code for a target programming language from the PES API Thrift IDL files provided in the Kareela Development Package.

The IDL files define the PES interface. The generated source code needs to be integrated with the OEM Print Controller Software to operate the PES interface.

1. Set up the Thrift Compiler in the Windows operating system by downloading Apache Thrift from its official site (See [Table 5](#)).
2. Extract the Kareela Development Package to obtain access to the PES API IDL files:

```
7-158-3-2-1-kareela_installer-dev-MJ3.1.1-9.e17.x86_64.zip
```

3. Navigate to the following directory:

```
\opt\memjet\kareela\doc
```

4. Run the following commands for each IDL file:

- a. Run `thrift-0.9.3.exe -I .\ --gen csharp Idl\KPesCommand.thrift`
- b. Run `thrift-0.9.3.exe -I .\ --gen csharp Idl\KPesCommon.thrift`
- c. Run `thrift-0.9.3.exe -I .\ --gen csharp Idl\KPesEvent.thrift`
- d. Run `thrift-0.9.3.exe -I .\ --gen csharp Idl\KPesProduct.thrift`
- e. Run `thrift-0.9.3.exe -I .\ --gen csharp Idl\KPesSettings.thrift`
- f. Run `thrift-0.9.3.exe -I .\ --gen csharp Idl\KPesStatus.thrift`
- g. Run `thrift-0.9.3.exe -I .\ --gen csharp ThriftLib\Common\MjCore.thrift`
- h. Run `thrift-0.9.3.exe -I .\ --gen csharp ThriftLib\Common\Result.thrift`

The above example generates C# source code. The generated source files will be located under the directory: `\opt\memjet\kareela\doc\gen-csharp`

The `--gen` option specifies the language to target. The above commands also assume that the `thrift-0.9.3.exe` file is accessible from the system path.

5. The generated source code will be dependent on the language-specific Thrift Library in order to compile and build and also enable the network stack. Refer to the `thrift-0.9.3\lib` directory within Thrift.

