Job Submission Library Guide

Rev #: 1.01

Date: 30 April 2021

Revision History

Doc. Version	JSL SW Release	Date	Details
V1.01	R8.0.0	30-Apr-21	• Added 1.5 Glossary • Added 1.6 Additional Documentation or Access • Minor editorial updates
V1.00	R8.0.0	19-Mar-21	Initial Release

Contents

1 Introduction 4

1.1 Aim and Audience 4
1.2 Prerequisites and Scope 4
1.3 Typographic Conventions 4
1.4 Related Documentation 4
1.5 Glossary 5
1.6 Additional Documentation or Access 5

2 Overview 6

2.1 DuraFlex Architecture 6
2.2 RIP Farm 6
2.3 Job Submission Library 7
2.4 Development Tools 8

3 Job Submission Library 9

3.1 Installation 9
3.2 Printing Strip Specification 9
3.2.1 printableOffset and printableWidth 11
3.2.2 EngineStage and stripIndex 11
3.2.3 Color Specification 12
3.3 Operation 12
3.3.1 Logging 12
3.3.2 Initialization 12
3.4 Job Flow 13
3.4.1 Open Job 13
3.4.2 Add Page 14
3.4.3 Close Job 15
3.4.4 Abort Job and Thread Safety 15

4 Configuration File Setup 16

5 JSL Development Tools 17

5.1 Library Test App 17
5.2 Library Stress Test App 17

Figures

Figure 1 – DuraFlex Software Architecture and RIP 7
Figure 2 – Physical and Virtual Printhead Layout Viewed from Above 10
Figure 3 – RIP Strip Definitions and Parameters 10
Figure 4 – JSL Job Flow Sequence Diagram 13
Figure 5 – Example JslConfig.xml 16

Tables

Table 1 – PES API Data Fields 9
Table 2 – JSL Index Variants 12

1 Introduction

This document is part of the OEM technical documentation suite for Memjet DuraFlex® module-based printing systems.

1.1 Aim and Audience

The aim of this document is to assist software engineers with using the DuraFlex Job Submission Library (JSL) to implement a third-party RIP of their own choice for processing of printer data.

This document is intended for software engineers who are integrating the DuraFlex print engine with a third-party RIP.

1.2 Prerequisites and Scope

The reader is expected to be familiar with software engineering, at least one of the development environments (Windows 10 or CentOS), commercial printing practices, and implementation.

This document assumes prior knowledge of:

- the C programming language
- digital printing process flow
- software development practices and terminology

The reader should also be familiar with the DuraFlex concept and Memjet-supplied DuraFlex components, especially the Print Engine Supervisor (PES) Software API.

Note:	Full details of the JSL API are available in the `jobSubmissionLib.h` file included in the JSL release package. This document serves as an introduction to the detailed information available in that file.

1.3 Typographic Conventions

Throughout this document, the following typographic conventions are used:

`Code Character`	`Courier` 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**. 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.

1.4 Related Documentation

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

- *System Overview* – 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 Fluidic Databook and Design Guide* – For mechanical design engineers and developers, providing details of the Memjet hardware modules and components (including printhead and maintenance system) and specifications of the ink delivery system fluidics.
- *Electrical Databook and Design Guide* – For electrical design engineers and developers, providing details of the Memjet power requirements, electronic assemblies, and connections.
- *Software Databook and Design Guide* – For software and firmware engineers who need to understand the software interfaces, commands, scripts, and reference software applications.
- *Demo GUI User Guide* – For OEM personnel using the DuraFlex Demo GUI reference application.
- *Installation and Commissioning Guide* – For OEM personnel who are installing and commissioning a new printing system.
- *Operations Guide* – For OEM engineers and operators to perform operational tasks.
- *Troubleshooting Guide* – For OEM engineers and technicians to identify symptoms and resolve issues.
- *Service and Repair Guide* – For OEM engineers and technicians to perform DuraFlex inspection and maintenance tasks and component and consumable replacement.
- *Job Submission Library Guide* – For OEM software engineers to incorporate the Job Submission Library (JSL) into their chosen Raster Image Processor (RIP).
- *Technical Bulletins* – For various audiences to announce product or process update or to provide specifics on single-subject technical topics.
- *CAD and Schematics* – For various audiences to provide detailed dimensions related to specific areas.

Note:	All technical documentation is available on your Memjet Partner Site.

1.5 Glossary

For terms, acronyms, and abbreviations used in this guide and some product-specific terms, see the *DuraFlex Glossary*.

Note:	This document is hyperlinked to the glossary. For offline reading, download the DuraFlex Glossary file from your Memjet Partner Site.

1.6 Additional Documentation or Access

For additional product-related technical documents, go to your Memjet Partner Site.

If you need Partner Site access, enter a case in Service Desk (*https://OEMsupport.memjet.com*), send an email to Memjet Customer Support (*customer.support@memjet.com*), or contact your Technical Account Manager.

2 Overview

The JSL is a software library that allows OEMs to integrate a RIP with the DuraFlex print engine.

2.1 DuraFlex Architecture

The DuraFlex product includes an embedded RIP, that is, a RIP that is part of the print engine. In this mode of operation, a source document (such as a PDF file) is sent directly to the print engine, which processes it to produce the print data that is used to drive the printheads.

Note:	The terms embedded RIP and internal RIP (internal to the PES) are synonymous and can be used interchangeably.

OEMs can choose from RIP vendors who have already integrated their RIPs with the Memjet DuraFlex system. An OEM may choose to use one of these solutions if they:

- do not have in-house RIP technology or expertise
- do not have an agreement, or prior arrangement with another RIP vendor

If the embedded RIP is not suitable for an OEM's printer design, then an external RIP can be used instead. In this mode of operation, known as *External RIP mode,* because it bypasses the embedded RIP, the source document is processed by a RIP that is run external to the print engine, and the resulting print data is sent to the print engine via a network connection. The JSL is a software library that allows OEM partners to integrate such an external RIP with the DuraFlex print engine. For more information on differences between RIP options, see the *DuraFlex System Overview*.

2.2 RIP Farm

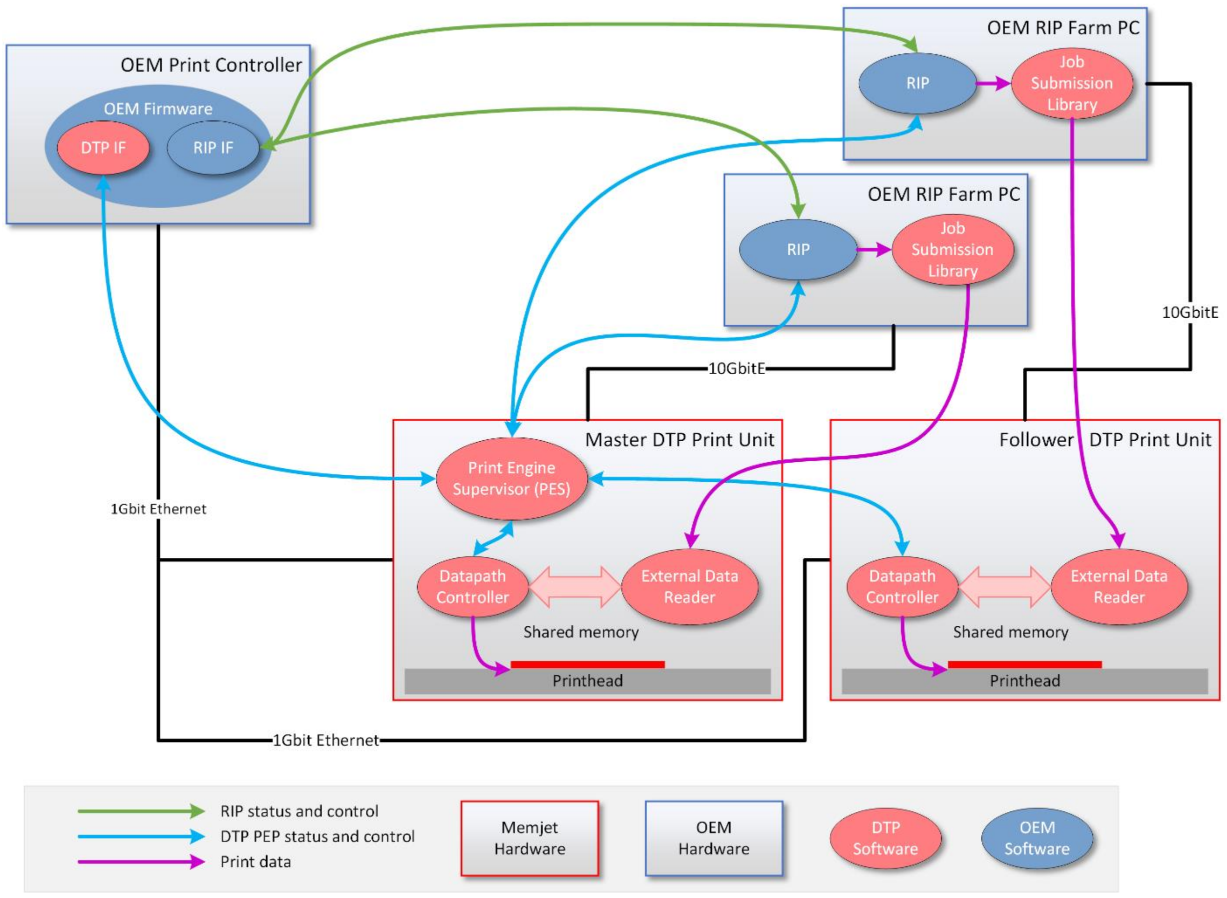
The RIP farm is the computing infrastructure used by the OEM to convert documents into the rasterized print data required to drive the printer. The RIP farm is typically implemented as one or more computers (PCs) physically connected to the DuraFlex ethernet switch infrastructure. For example, *Figure 1* shows a printer that is two printheads wide. Each print module it receives print data from a separate RIP running on its own PC in the RIP farm.

It is possible that the RIP farm may consist of only a single PC, for example where the printer has a single Print Module. Alternately, where the source document complexity or printer width does not warrant a separate RIP Farm PC, the RIP (and JSL) may run, for example, on the OEM Print Controller PC.

The RIP farm produces rasterized print data in the Memjet file format (ABT/Borealis files) required by the print modules. Memjet does not provide an external RIP. Instead, OEMs source an external RIP for use with the printer based on their specific application.

Note:	If an OEM wants to use RIP farm infrastructure sourced from a RIP vendor that has not yet integrated their system with DuraFlex, then the JSL will need to be integrated with their RIP technology and installed on their equipment. The JSL is designed to be universal, and Memjet can offer some assistance with integration if required. Contact your TAM for more information.

Figure 1 – DuraFlex Software Architecture and RIP



2.3 Job Submission Library

The Job Submission Library provides a simple interface between the print data output of the RIP farm and the DuraFlex print modules. The JSL accepts job and page metadata and image data from the RIP, translates it into Memjet's format, and either sends it to any waiting print modules or writes it to files for later use. It is provided as a library to be linked into the RIP application.

`JobSubmissionLib` gives the RIP a simplified interface that takes care of the details associated with:

- formatting of print job metadata to Memjet's file format specification
- sending the job information and the print data over the network to each print unit

The header file which defines the `JobSubmissionLib` API is in the `jobSubmissionLib.h` file. For complete documentation on the API refer to that file.

Note: The JSL is released as a Windows Library and a CentOS7 static library. The JSL provides a C interface. Using C avoids issues with exporting mangled names.

The flow of data from the OEM printer controller software to the print engine and from the RIP farm to the print modules is illustrated in *Figure 1* above.

Note:	Software engineers who are integrating the JSL should also read the *DuraFlex Software Databook and Design Guide* for further information on the PES API.

2.4 Development Tools

Two development tools are available from Memjet to assist with the use and testing of the Job Submission Library:

- *Library Test App*
- *Library Stress Test App*.

These are available for both Windows and Linux. These are explained in detail in Section *5 JSL Development Tools*.

3 Job Submission Library

The Job Submission Library uses a single 'C' language interface which provides library initialization and process logging, as well as job control, i.e., `jslibOpenJob()`, `jslibAddPage()`, `jslibCloseJob()` and `jslibAbortJob()`. In addition, using the library requires access to a configuration file for static system configuration, an example of which is provided as part of the JSL software release.

The interface is one-way only. It accepts data from the RIP for transmission to the printheads, but it cannot be queried. It will issue error codes and log output if it is unable to perform a requested task.

The Job Submission Library will not throw exceptions on client calls. It is the responsibility of the RIP to obtain the system status from the print engine supervisor via the PES API and modify its calls accordingly. Note that the RIP can use the PES events to monitor and report completion status of print jobs.

Because system control functionality is outside the capabilities of the Job Submission Library, it is not discussed further here. The *DuraFlex Software Databook and Design Guide* provides instructions on how to use the PES Command and Event APIs.

3.1 Installation

The RIP vendor may decide where to install JSL and as such OEMs should consult with their RIP vendor if it is necessary to install or update the JSL.

3.2 Printing Strip Specification

The RIP must structure the data provided to the JSL to match the configuration of the print engine as shown in *Table 1*. This information is derived from the PES API and is detailed in the *DuraFlex Software Databook and Design Guide*. The RIP may perform many functions before it talks to JSL, for instance, color management and halftoning. The RIP is also responsible for taking wide jobs and deciding which vertical strip gets sent to which print unit, and it must manage one JSL instance per print unit.

Table 1 lists the information required by the RIP to generate the correct images for the print engine. The PES API Thrift files should be consulted for detailed definitions of these data fields. All the fields are available from the PES API by calling either `getProductInfo()` or `getSettings()`, refer to *Figure 4* for the job flow sequence.

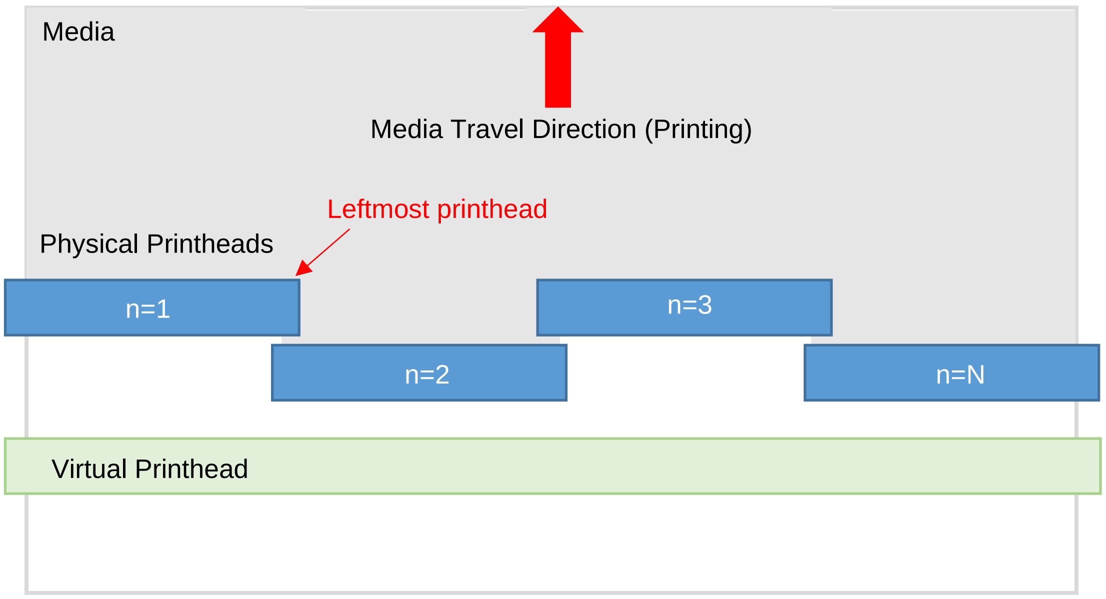
Table 1 – PES API Data Fields

Required Information	PES API Call	Map	PES API Data Field
Colors	`getProductInfo()`	`printUnit`	`PrintUnitProductInfo.inkColors` `PrintUnitProductInfo.engineStage`
Strip Width and Offset	`getSettings()`	`pm`	`PmSettings.printableOffset` `PmSettings.printableWidth`

All distance-related fields in the PES API (including `printableOffset` and `printableWidth`) are expressed in micrometres (μm), while the JSL uses printed dots. The units must be converted using the job's horizontal resolution 1,600 dpi or 15.875 μm/dot.

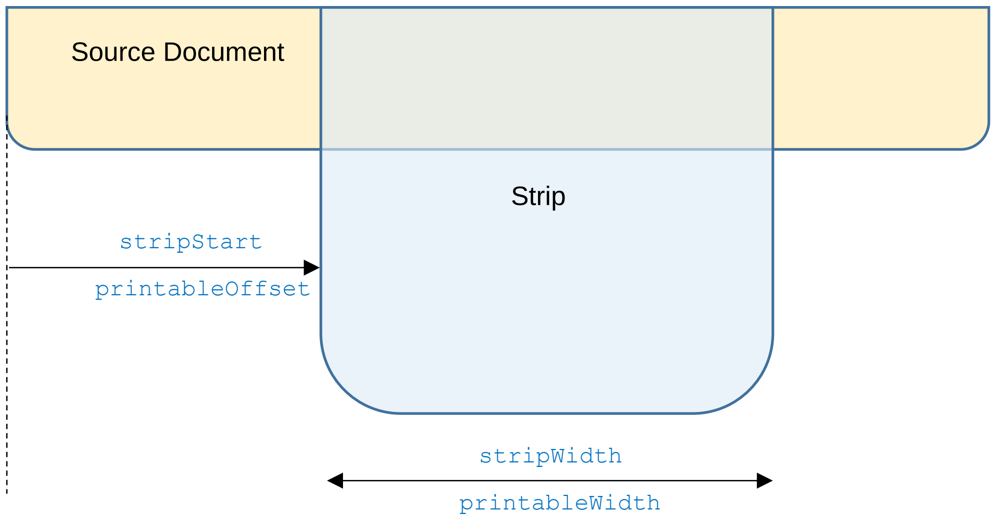
The PES software can be thought of as providing a 'virtual printhead' that spans the entire width of all print modules in each configuration as shown in *Figure 2*. For a system that is N printheads wide, then the printheads are labelled from 1 to N from left to right. The leftmost printhead is identified when viewed from above the printed media and when facing in the direction in which the media moves relative to the printheads.

Figure 2 – Physical and Virtual Printhead Layout Viewed from Above



The `printableOffset` positions the image strip for each printhead across this virtual printhead as shown in *Figure 3*. Typically, for the leftmost print module, the `printableOffset` will be 0, but the PCS configuration of the system may also set the `printableOffset` greater than 0, so the RIP should not depend on a `printableOffset` of 0.

Figure 3 – RIP Strip Definitions and Parameters



3.2.1 printableOffset and printableWidth

The information provided here describes configuring the JSL’s `stripStart` and `stripWidth` to produce print data for aligned printheads. The RIP must process the printed image into “strips” suitable for printing by each printhead along a print bar. As shown in *Figure 2*, the strip is defined in the PES API by `printableOffset` and `printableWidth`, which correspond directly to JSL’s `stripStart` and `stripWidth`. The RIP must convert these PES API values from micrometres to dots for JSL. These JSL fields should be thought of as metadata, describing the data that is being supplied, not configuration parameters used to configure the job. Consult `jobSubmisisonLib.h` for additional information on `stripStart` and `stripWidth`.

Even if a strip is narrower than a printhead, the RIP must supply full-width data to meet the printableWidth configuration, including any padding needed. The RIP may need to add zero padding to the image data for the following reasons:

- to meet the `printableWidth` requirement.
  Even if the image data are narrower than the `printableWidth`, the data supplied in the strip must be zero padded to reach the `printableWidth`.
- to meet the 128-bit multiple requirement
  If the `printableWidth` is not a multiple of 128, additional zero padding is required to extend the strip beyond the `printableWidth` to make the strip a multiple of 128 bits wide. However, even when zero padding is added, the `stripWidth` parameter is set to the PES API’s `printableWidth`.
- to center justify or right-justify image data.
  In this case, the image may be padded on the left.

During the RIP process, the RIP need only consider ideally aligned printheads. During printing the PES software takes care of any horizontal alignment adjustments necessary to properly position the printable strip, modifying the margins as necessary to maintain alignment from one printhead to the next.

3.2.2 EngineStage and stripIndex

An engine stage is a grouping of printheads, typically used in a duplex printer to distinguish between printheads printing on the front of the media from those printing on the back. For a simplex printer there will only be one engine stage, containing all printheads. The engine stage is represented in JSL by the `engineStage` field, which is a number starting from 1.

Within each engine stage, the strips are numbered consecutively starting at 1. This is the `stripIndex`. Typically, the strips, taken in ascending `stripIndex` order, will have increasing `stripStart` values.

To get the `engineStage` for each print unit, use the `getProductInfo()` request to obtain the `printUnit` map. The `PrintUnitProductInfo.engineStage` field is found within that map.

The `stripIndex` for each print unit can be generated from the `printUnit` map, by allocating an index for each entry in print unit order, resetting the index to 1 when the `engineStage` value changes.

For example, for a 2-wide duplex the RIP would set the engine stage and strip index values as shown in *Table 2*:

Table 2 – JSL Index Variants

Print Unit index	`PrintUnitProductInfo.engineStage`	JSL `engineStage`	JSL `stripIndex`
1	1	1	1
2	1	1	2
3	2	2	1
4	2	2	2

3.2.3 Color Specification

The colors used in the printhead must be supplied to JSL via the fields `numColors` and `colorArray`. Note that the color order supplied in `colorArray` must match the order in which the colors are connected to the printhead. Color connection information is available from the PES API via the `getProductInfo()` request. See *Table 1* for more information on which API calls to use.

3.3 Operation

3.3.1 Logging

The Job Submission Library logs human-readable information on errors and general progress. Log messages can be filtered on the level of significance (priority) so the client code can examine as much or as little of the provided output as required. Log messages are passed to the client via a callback mechanism. A call is generally made in the context of the caller's thread and will block, so the client's callback function should extract what information it needs and return as soon as is possible. Where processing throughput is of maximum importance, it is best to not set the filter to the `JSLIB_LOGGING_LEVEL_DEBUG` level as this will log many messages during 'correct' operation.

Logging may be configured at any time, even before initialization. It is recommended that logging is performed prior to initialization so that errors in that step can be understood. If logging is desired, then the minimum configuration is to set the callback function via `jslibSetLoggingCallback()`. To disable logging, call the function with a null parameter. `jslibSetLoggingCallback()` may be called as often as desired, and each call will overwrite the previous one. Each time the function is called with a non-zero value, and if the logging level is set to `JSLIB_LOGGING_LEVEL_INFO` or lower, then an immediate callback will be made with a string indicating the library version before `jslibSetLoggingCallback()` returns.

By default, the priority level is set to `JSLIB_LOGGING_LEVEL_INFO`, but it may be changed via calls to `jslibSetLoggingLevel()`. For more detail, refer to documentation in `jobSubmissionLib.h`.

3.3.2 Initialization

After logging has been configured, the library must also be configured via a call to `jslibInit()`. Input parameters are the mode, a fixed ID number for this printer model, and the XML configuration file that provides printer specific information required by the library. Both the mode number and the XML file are provided by Mem

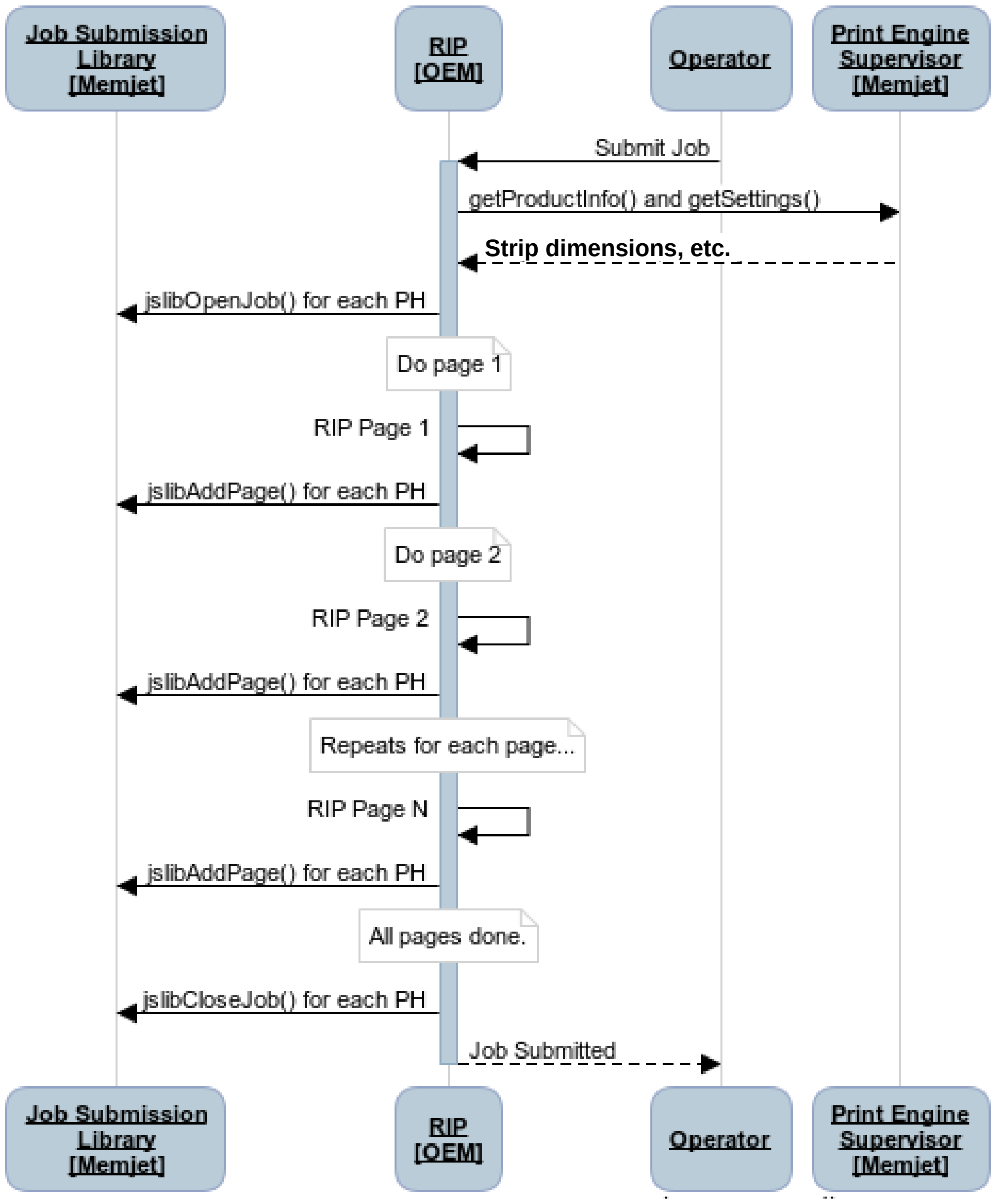
If a RIP uses multiple processes on the same node, then those processes can all call the DLL independently. However, any one handle returned from `jslibOpenJob()` for a given print module can only be used by one process on one node, see Section *3.4.1 Open Job*.

3.4 Job Flow

3.4.1 Open Job

Once initialized, print jobs may be opened, processed, and closed. Each job may be spread across multiple printheads, and the job must be opened separately for each print unit with a call to `jslibOpenJob()`. This is shown and described in *Figure 4* below, which also shows optional requests sent to the PES API to obtain printer attributes.

Figure 4 – JSL Job Flow Sequence Diagram


Job Submission Library [Memjet]
RIP [OEM]
Operator
Print Engine Supervisor [Memjet]
Submit Job
getProductInfo() and getSettings()
Strip dimensions, etc.
jslibOpenJob() for each PH
Do page 1
RIP Page 1
jslibAddPage() for each PH
Do page 2
RIP Page 2
jslibAddPage() for each PH
Repeats for each page...
RIP Page N
jslibAddPage() for each PH
All pages done.
jslibCloseJob() for each PH
Job Submitted
Job Submission Library [Memjet]
RIP [OEM]
Operator
Print Engine Supervisor [Memjet]

For example, in a 2-wide color printer, there will be 2 calls to `jslibOpenJob()`, one for each print module. Each call will return a handle for that job on that print module which must be used in all subsequent calls to `jslibAddPage()`, `jslibCloseJob()` and `jslibAbortJob()`. For every call to `jslibOpenJob()` on a printhead, there must be a corresponding call to `jslibCloseJob()` prior to calling `jslibShutdown()`. The handle returned by `jslibOpenJob()` may only be used by the process that called it. It cannot be shared between processes or other nodes in a RIP farm.

Some of the parameters that are passed to `jslibOpenJob()` will typically be the same for each printhead, some of it must be the same, while some module-specific information will differ. These differences are noted in the documentation for the `JslibJobDescription` struct in `jobSubmissionLib.h`. with the statements "...will typically supply identical..." or "...must supply identical...". Where neither of these clauses appear, then the data is printhead-specific. For example, as the `jobId` field in the `JslibJobDescription` struct is documented with "All files/streams for the same job must supply an identical `jobId`", then it is the responsibility of the client code to ensure that this is the case.

The PES API command `KPesCommand`.`generateJobID()` may be used to generate a `jobId`. It is recommended that the RIP uses this command to get a `jobId`, instead of generating its own ID, to avoid issues with jobs printed via the PES.

3.4.2 Add Page

The print data software (Gymea) requires that 5 pages be preloaded before the job starts, and at least 3 pages be continuously loaded during printing. Buffering multiple pages of data allows the system to handle preparation for the next page while the current page is printing.

The RIP will call the `jslibAddPage()` function to add a page. The `pageHeight` and `numCopies` get passed as parameters to this function.

Once a job is opened on all modules, pages can be added using `jslibAddPage()`. This blocking call will not return until the data has been queued for transfer to the print module. There is no requirement regarding the order in which different print modules are called; printing will not commence until all the modules have received the preload number of pages (5).

The RIP decides when to submit print data to the JSL, which will send data via network socket as fast as it can. The print modules have a large data buffer, and it is preferable to keep it as full as possible to avoid data under-run due to network throughput variations. Thus, it is best to continue to call `jslibAddpage()` adding as many pages as possible until the call blocks. If this is being done in a single threaded system, then the calls for each print module should be done in strict rotation so that no module is without data when another has blocked the thread.

The number of pages to preload is the number of times the RIP must call `jslibAddPage()` when `numCopies` is set to 1 before printing can be started. There are some exceptions to this case:

1. If the total number of pages in the job is less than the number of pages to preload, then all the print job pages must be loaded before printing can start.

2. If `numCopies` is set to a value other than 1 (0 is not allowed), then each call to `jslibAddPage()` will send `numCopies` pages to the print unit.

For example, if "number of pages to preload" is 5 and `numCopies` is set to 5 then printing can commence after the first call to `jslibAddPage()` returns successfully. Note, in this (streaming) mode the RIP must ensure that it can add the remaining pages fast enough to avoid data under-runs.

There are two ways to print a page multiple times; either the page can be added multiple times by calling `jslibAddpage()`, or the `numCopies` member of the `JslibPageDescription` struct may be set to the required quantity. Both methods are functionally identical.

The last page to be added to a job must have the `isLastPage` member of the `JslibPageDescription` struct set to `True`. Failure to do this will prevent `jslibJobClose()` from running successfully.

If the function is in a blocked state when the job needs to be aborted for some reason, e.g., due to a stall in the printer, then it can be freed by calling `jslibAbortJob()` on another thread, see Section *3.4.4*.

3.4.3 Close Job

When all pages of a job have been added, `jslibCloseJob()` must be successfully called for all print modules before any new job can be opened. When `jslibCloseJob()` returns, the handle that was originally supplied by `jslibOpenJob()` is no longer valid and must not be used. New handles will be supplied with the new calls to `jslibOpenJob()`.

3.4.4 Abort Job and Thread Safety

The Job Submission Library is thread-safe and calls to job/page functions for different print modules may be called sequentially from a single thread or in parallel from multiple threads if required. This is a matter of design choice for the client software. However, given that calls to `jslibAddPage()` may block, it is imperative that there is a separate thread from which to call `jslibAbortJob()` if required.

If `jslibAbortJob()` is called for one or more print modules, then `jslibCloseJob()` must still be called for those modules before either a new job can be opened, or the library can be shut down.

An abort call on one module does not directly affect any other modules; any module which is in the process of adding a page will complete its data transmission before returning. However, as a job cannot print until all modules have received their data, once one print module has had a job aborted, the job should be aborted for all print modules.

Note:	Discarding a job is a printer control activity which is outside the scope of the Job Submission Library.

4 Configuration File Setup

The Job Submission Library sends print data to printheads via network connections, and so needs the network addresses of the print modules.

For DuraFlex this network information is supplied via the JSL configuration file passed as a parameter to the `jslibInit()` call.

Because this information will vary between installations, the JSL release package contains a configuration file (`JslConfig.xml`) with a commented-out example of the required information (*Figure 5*).

Figure 5 – Example JslConfig.xml

Example JslConfig,xml

```
<!—Remove Commenting
<printheadModules>
  <engineStage number="1">
    <strip number="1">
      <printheadModule color="JSLIB_CYAN JSLIB_MAGENTA JSLIB_YELLOW JSLIB_BLACK">
        <socket type = "primary">
          <hostAddress>__hostname or address__</hostAddress>
          <port>__IP port number__</port>
        </socket>
      </printheadModule>
    </strip>
    <strip number="2">
      <printheadModule color="JSLIB_CYAN JSLIB_MAGENTA JSLIB_YELLOW JSLIB_BLACK">
        <socket type = "primary">
          <hostAddress>__hostname or address__</hostAddress>
          <port>__IP port number__</port>
        </socket>
      </printheadModule>
    </strip>
  </engineStage>
</printheadModules>
-->
```

Update the file with the attributes appropriate for the printer:

- an `engineStage` entry for each engine stage, numbered from 1
- within `engineStage`, a `strip` entry for each printhead, numbered from 1
- fill in the appropriate `hostAddress` (node name or IP address).
- for `port` use 9100 unless instructed otherwise by your Memjet technical support team.
- ensure that the `color` attribute for the `printheadModule` lists the colors in the order they are connected in the printer.
- delete the green highlighted text at the top and bottom of the code to remove the comments

Note: Although this section describes the procedure for manually editing this file, it is possible that in practice the file will instead be used as a template which will be updated by some automated process as part of the RIP or OEM printer software installation process.

5 JSL Development Tools

Two development tools are available to assist with the use and testing of the Job Submission Library: the Library Test App and the Library Stress Test App. While helpful during development, neither of these tools are intended for a production environment.

5.1 Library Test App

The Library Test App is a command-line application which exercises the Job Submission Library in much the same way as a RIP, though in a simplified fashion. While primarily designed as a test tool for the library itself, it may be useful to RIP developers to compare its output with similarly configured output from their RIP.

Each 'run' of the application invokes one JSL job, where each supplied TIFF file is added as a page in that job. Because it is single threaded, it cannot not exercise the abort capability of the JSL and is potentially less efficient than a multi-threaded version when it is blocked by communications throughput.

The application's command line parameters specify the configuration of the printer for which it is generating Memjet's data and the names of one or more TIFF files containing the images to be printed. For more information on the command line parameters, run the command line help:

- `LibraryTestApp /h` for Windows or
- `./libraryTestApp -h` for Linux.

5.2 Library Stress Test App

The Library Stress Test App is similar to the Library Test App, except:

- It is multi-threaded, using one or two threads per print module.
- It accepts only one TIFF image (and thus one page).
- After conversion into Memjet's format, it sends that page as many times as requested in a job, and that job can be sent as many times as requested, all done as fast as it can.

This application was primarily designed as a stress test for the Job Submission Library but may be useful to RIP developers as a throughput benchmark.

- For more information on the command line parameters, run the command line help: `LibraryStressTestApp /h` for Windows or `./libraryStressTestApp -h` for Linux.