XML Pipeline Server

User's Guide

Contents

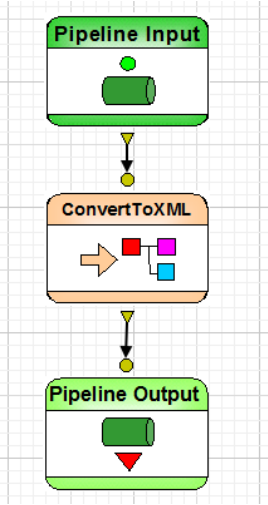
XML Pipeline Server User's Guide

XML Pipeline Server is a lightweight deployment environment for executing Stylus Studio XML pipelines with a large variety of built-in features.

XML Pipeline

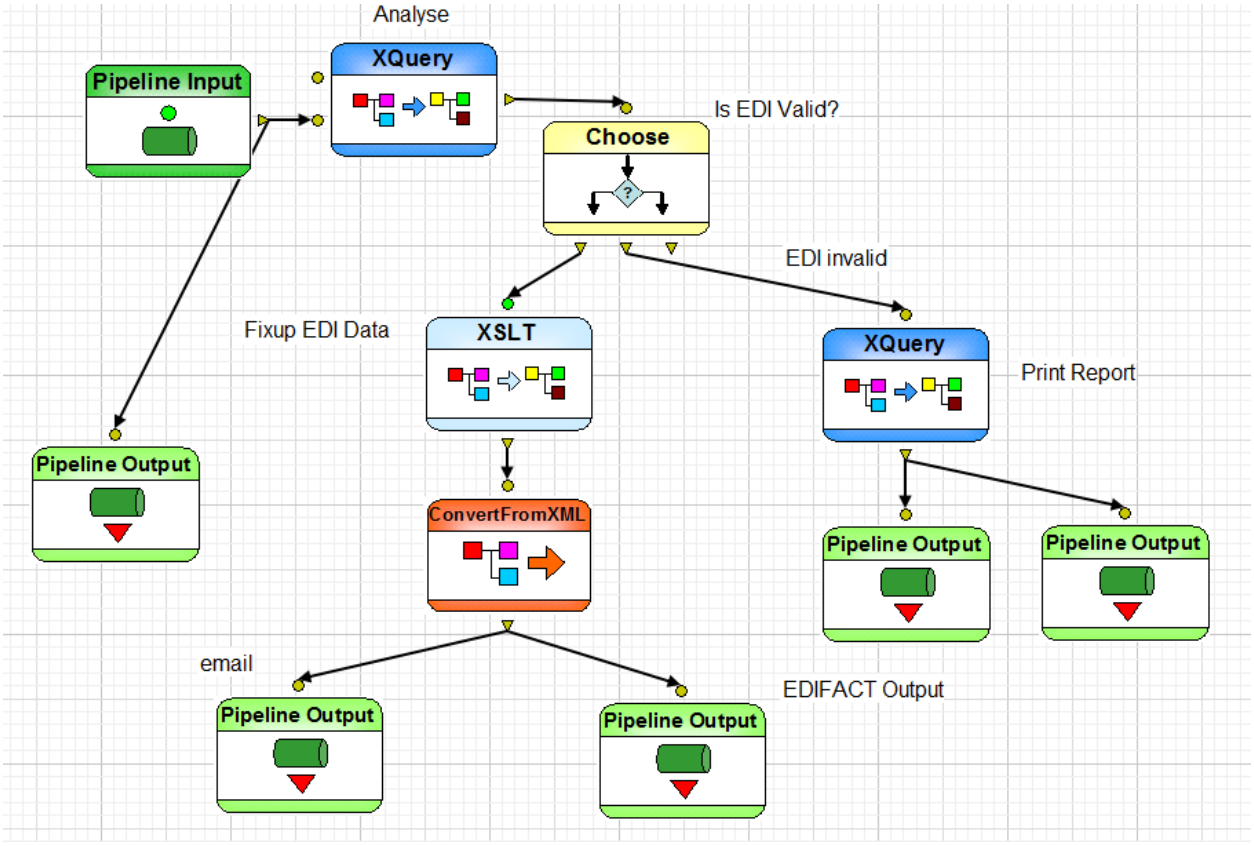
A pipeline is a chain of operations which starts with an input port and terminates with one or more output ports or a Stop operation.

The following simple pipeline converts EDI transactions to XML:



A pipeline can perform far more complex tasks which involve several operations.

The following example shows a pipeline performing advanced EDI analysis and validation, data augmentation and sends the resulting EDI to an email box.



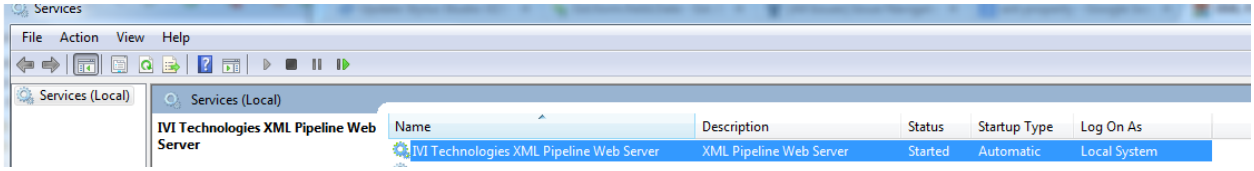
Installation Guide

Windows 64 bits

XML Pipeline Server runs on Windows 10, Windows 8 (64 bit only), Windows 7, Windows Vista, and Windows Server 2008 or later. It bundles all necessary third party applications including the Oracle Java Virtual Machine 1.8 which is installed as a private copy and does not interfere with other virtual machines installed in the system.

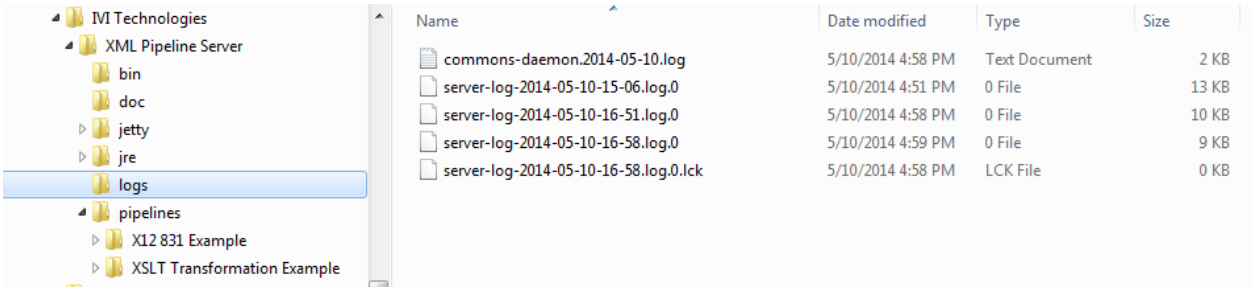
The installation package is a single executable named **XMLPipelineServer_setup.exe**

The installation procedure copies all necessary files into the installation folder and then registers the XML Pipeline built-in Web Server as Windows registry. This operation requires elevated privileges.



The service registration is automatically executed during the installation through a batch file located in **bin\RegisterWebServerService.bat**

If the default port is used by another program the server will not start and you will not be able to access the Web UI. If the server is not reachable, check the logs in the log folder for possible errors.



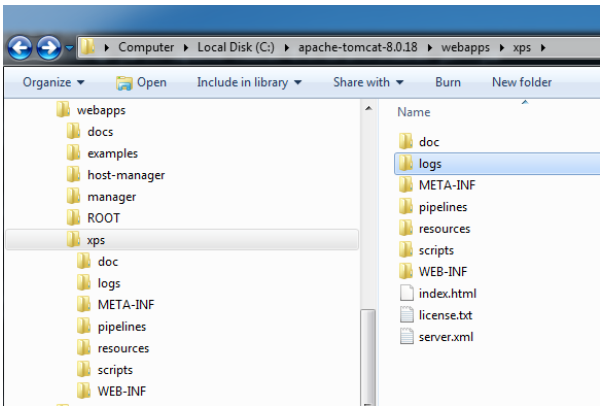
If the default port is used, you can change the value in tomcat\conf\server.xml. See Apache Tomcat documentation for more information.

War Deployment

XML Pipeline Server is available in two distributions: Window Installation package and WAR file.

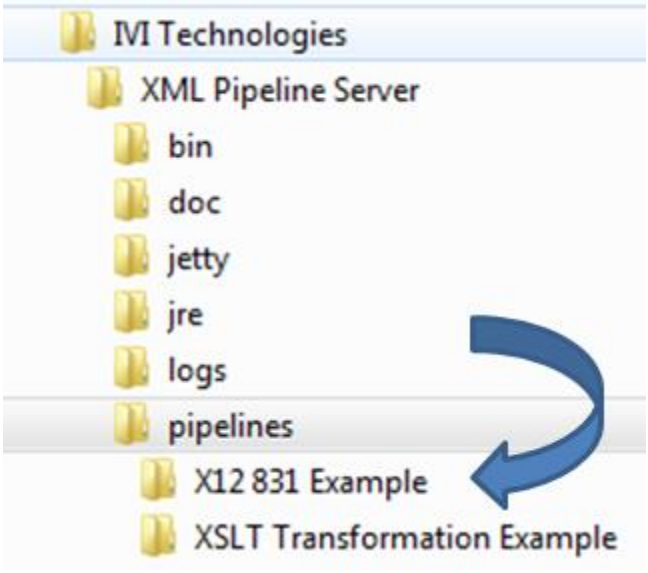
When deploy using the WAR distribution the product root is the servlet root. In the following example the war was deployed on Tomcat 8, notice that the license file and the "pipelines" folder are located in the servlet folder.

XML Pipeline Server requires a Servlet Container 3.1 and Java 1.8 or greater.



Deploy an XML Pipeline

Deploying a pipeline is just a matter of copying the pipeline file and all referenced artifacts (XQuery queries, XSLT transformations, XSD schemas, etc.) to a sub folder of your choice, under the pipelines folder



The XML Pipeline Server monitors the pipelines folder. As soon as you copy a pipeline the server loads the definition and binds inputs and outputs.

If you overwrite an existing pipeline while an instance is running, the server stops the instance and unbinds its ports before loading the new definition.

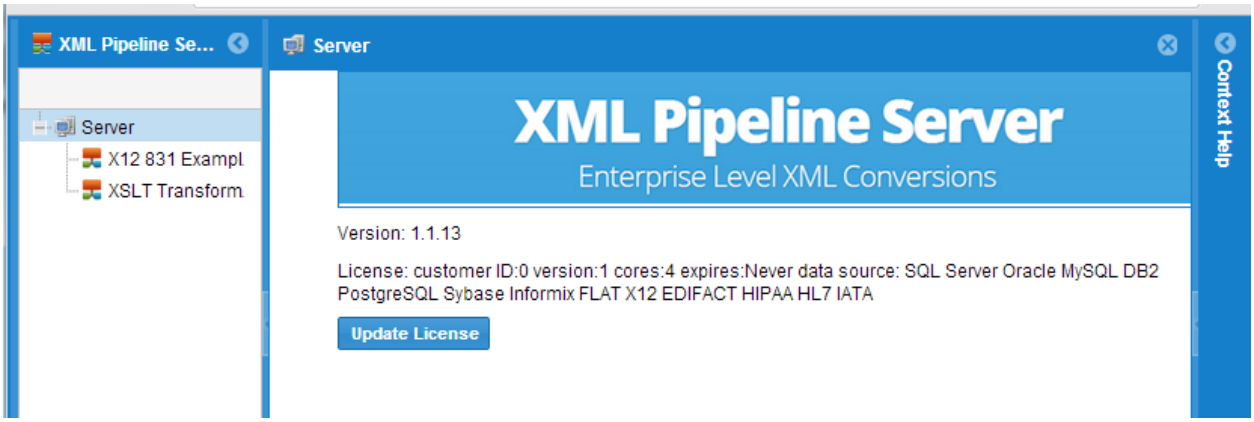
All relative URLs are resolved with the pipeline URL as the base or the including resource as the base. For example, a query which includes a module or an XSLT which includes/imports a library will use the main module as the base.

Web Dashboard

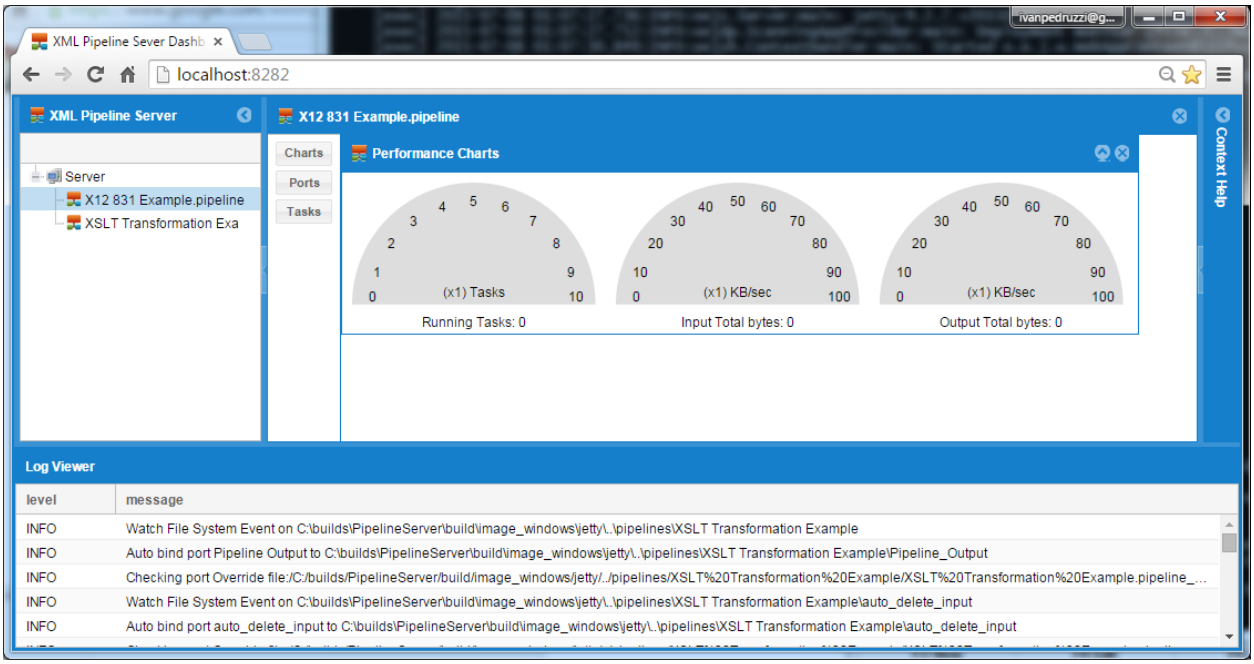
XML Pipeline Server features a Web UI which runs on its built-in web server. By default this runs on port 81.

The Web UI is divided in three areas: a tree control on the left inside which shows the server node and its active pipelines, the details panel on the right which provide details of what you have selected in the tree control, and the Log Viewer at the bottom which shows server notification in real-time.

Clicking on the server node shows the license details.



When clicking on a pipeline node, the right inside panel shows the number of operations currently running and the data in input and output.



Operations

XML Pipeline Server supports the following operations:

- XQuery
- XSLT
- XML Schema Validation
- XSL-FO to PDF

- Convert To and From XML
- Choose
- Stop and Warning

End-Points

XML Pipeline Server supports the following end points:

- File System Folders
- FTP, FTPS and SFTP
- SMTP , IMAP and POP
- HTTP
- Timer
- WebSocket
- MQTT
- LDAP
- Microsoft Graph
- DropBox
- Google Drive
- Microsoft One Drive
- SMS

-

Data Source, Data Target

XML Pipeline Server supports the following sources and targets:

- Relational database
  - Microsoft SQL Server 2000 - 2016
  - Oracle 8g to 12c R2
  - IBM DB2
  - MySQL 5 and 6 commercial
  - MySQL 5 and 6 Community Edition, plug & play external JDBC drive
  - Sybase 11
  - Informix
- EDI
  - X12
  - EDIFACT, EANCOM, TRADACOMS, EDIGAS, NCPDP
  - HIPAA
  - HL7
  - IATA
- Flat File
  - CSV
  - JSON
  - Tab separated
  - Fixed Width [Input only]
  - Windows INI, Java Properties
  - SYLK (Microsoft Excel legacy)
  - Dbase III, IV and V
  - RTF
- Binary
  - Base 64
  - Base 2-36

Binding Pipeline Port to End-Point

XML Pipeline Server provides multiple ways to bind a pipeline port. By default, an input port is bound to a sub folder with the port's name.

XML Pipeline Server registers for file system events and triggers the pipeline execution as soon as a file is copied in the monitored folders. The server performs several checks to determine if the file is ready to be processed: It monitors the file size which must not change for one second. It then tries to get an exclusive lock to make sure no other processes own the file. Finally, it starts reading the data.

To bind a port to a different end-point, you need to create a binding file which must have the same pipeline file name. For example if your pipeline name is **X12 831 Example.pipeline** the binding file needs to be named **X12 831 Example.pipeline.binding** and has to be located in the same folder as the pipeline file.

Email End-point

Here is an example of a binding file which binds an input port to a Gmail IMAP end-point and an output port to a Gmail SMTP end-point. Notice that the attribute name's value ("Email Input", "Email Output") on the port element, highlighted below has to match the pipeline port's name.

Property "pullEventIntervalInMilliseconds" determines how frequently the IMAP client checks for new messages on the email server, the default value is 1 second.

Multiple filters can be used to restring the number of message pulled from the IMAP server

subject_filter	The comparison is case-insensitive. The pattern is a simple string that must appear as a substring in the Subject.
from_filter	The comparison is case-insensitive. The pattern is a simple string that must appear as a substring in the From address
to_filter	The comparison is case-insensitive. The pattern is a simple string that must appear as a substring in the To address
cc_filter	The comparison is case-insensitive. The pattern is a simple string that must appear as a substring in the CC address
bcc_filter	The comparison is case-insensitive. The pattern is a simple string that must appear as a substring in the CC address
older_than_in_seconds_filter	Find messages that are older than a given interval (in seconds). Relies on the server implementing the WITHIN search extension (RFC 5032). **None of Email servers tested support this IMAP extension**
younger_than_in_seconds_filter	Find messages that are younger than a given interval (in seconds). Relies on the server implementing the WITHIN search extension (RFC 5032).**None of Email server tested support this IMAP extension**
move_emails_to_folder	Instruct the email client to move the message to another folder instead of using the fingerprint file to track which emails have been processed
set_done_before_loading_email_body	Set to true to instruct the email client to mark the message "done" before loading the message body. In case of error while processing the body, the email will not be reprocessed.
starting_n_days_ago	Set the number of days to go back reading emails, for example reads just the last 7 days. Default is -1 which means no restriction. Mutual Exclusive with start_year
starting_year	Allow to set the starting date from when to read the emails Default is -1 which means no restriction.
starting_month	Allow to set the starting date from when to read the emails
starting_day	Allow to set the starting date from when to read the emails
starting_hour	Allow to set the starting date from when to read the emails
starting_minute	Allow to set the starting date from when to read the emails
startingFrom	Starting date expressed as YYYY-MM-DD
endingTo	Ending date expressed as YYYY-MM-DD
last_uid_filter	Fetch messages that have UID greater than the value provided. **When this filter is set all other filters are evaluated client-side**

```
<binding xmlns="http://www.ivitechnologies.com/pipeline/server">
 <port_manager>
  <port
        name="Email Input"
        protocol="email"
        mail.imap.host="imap.gmail.com"
        mail.imap.port="993"
        mail.imap.user="<your email>@gmail.com"
        mail.imap.password="<your password>"
        mail.imap.ssl.enable="true"
        folder="inbox"
        starting_year="2014"
        starting_month="1"
        starting_day="1"
        subject_filter="XML Pipeline Server - Test 1"
        from_filter="support@xmlpipelineserver.com"

        message_parts="1"
        pullEventIntervalInMilliseconds="1000"/>

  <port
        name="Email Output"
        protocol="email"
        mail.smtp.host="smtp.gmail.com"
        mail.smtp.port="587"
        mail.smtp.user="<your email>@gmail.com"
        mail.smtp.password="<your password>"
        mail.smtp.auth="true"
        mail.smtp.starttls.enable="true"
        mail.smtp.from="<your email>@gmail.com"
        recipients="<your email>@gmail.com "
        subject="XML Pipeline Server - Test 1"
        body="This message was sent by XML Pipeline Server"
        send_data_in_message_body="false"
        message_mime_type="html"
        message_character_encoding="utf-8"
        send_email_n_retries="5"
        send_email_wait_before_retry_in_milliseconds="10000"
/>
 </port_manager>
</binding>
```

How New Emails are detected

The IMAP client can use multiple ways to detect new emails:

File Fingerprint

By default, the IMAP client creates a file named as the unique id returned by the IMAP server in the local email cache, a folder located in

```
<xps folder>\MailCache\<imap server host name>\<email username>
```

The IMAP client skips emails for which a file fingerprint exists in the email cache.

Using the file fingerprint is mutual exclusive with using the email flag.

Moving Email on a different IMAP folder

Using the following binding property in the pipeline binding file. Notice that this feature can be used in combination with other two.

```
move_emails_to_folder="Trash"
```

Using one of the standard IMAP email flags:

Flags.Flag.ANSWERED
Flags.Flag.DELETED
Flags.Flag.DRAFT
Flags.Flag.FLAGGED
Flags.Flag.RECENT
Flags.Flag.SEEN

Using the following binding property in the pipeline binding file

```
use_custom_flag="Flags.Flag.FLAGGED"
custom_flag_value="false"
```

Property custom_flag_value is used to retrieve emails which have the flag set to the specific value. In the example above the IMAP client retrieves all emails have flag “FLAGGED” set to false. After the email is processed the flag is set to true, so they are not process again.

Using the email flag is mutual exclusive with using the file fingerprint.

File System End-point

You can augment the File System port using the following definition.

If the port name starts with **auto_delete_** the file which is triggering the pipeline execution will be deleted after the pipeline execution is completed. When the property "auto_delete" is true the file which is triggering the pipeline execution will be deleted after the pipeline execution is completed.

The property "filter" allows catching events of file names which match the given regular expression. In this example only files with the .txt extension will trigger the execution.

Property "folderFullPath" allows overwriting the location on the file system the port will be using. The value can be a local path name like c:\temp\InputPortTest or UNC path like \\server\share\folder

Property "treatTheseFileExtAsZip" allows to specify a list of file extensions separated by comma, for example ".zip, .jar" for file which will be treated as zip file. Each file stored in the archive (which can be optionally filtered with the filter property) will trigger a pipeline execution.

Property “delayAfterDetectingNewFileInMilliseconds” allows to alter the delay after which a file is process. When files are created/copied at very low speed and the process that creates the file does not lock the file, XPS may attempt to start processing before the file has been completely copied. Increasing the delay may mitigate the error condition.

A very common pattern is making a copy of the input file to an archive folder or to a stage area from which the file is deleted when the pipeline is completed. The following properties allow to specify a folder where to copy/move the input file.

copy_before_processing_folder
copy_after_processing_folder
move_after_processing_folder

```
<binding xmlns="http://www.ivitechnologies.com/pipeline/server">
 <port_manager>
  <port
   name="Pipeline Input"
   protocol="file"
   filter="(.*).txt"
   folderFullPath="c:\temp\InputPortTest"
   treatTheseFileExtAsZip=".zip"
   auto_delete="true"
   delayAfterDetectingNewFileInMilliseconds="2000"

   copy_before_processing_folder="stage_before"
   copy_after_processing_folder="stage_after"
   move_after_processing_folder="stage_after"
  />
 </port_manager>
</binding>
```

The File System output port allows to hard code a file name which will be used when the data is saved on disk. It is also possible to append the data rather than overwrite the file on each execution

```
<binding xmlns="http://www.ivitechnologies.com/pipeline/server">
 <port_manager>
  <port
   name="Pipeline Output"
   protocol="file"
   outputFileName="books.xml"
   appendDataToOutputFile="true"
  />
 </port_manager>
</binding>
```

Starting with build 230, it is possible to configure File System input ports to watch a folder, scanning file names on a timer instead of using the OS file event notifications. This is useful because under some conditions like monitoring file on a remote network share does not behaves reliably.

Setting binding property “useTimer” to true activates an alternative directory watch monitor. Binding property “pullEventIntervalInMilliseconds” allows to configure the interval for how often the directory is scanned.

```
<port
name="Input"
folderFullPath="\\localhost\XPSUNCTest"
protocol="file"
auto_delete="true"
useTimer="true"
pullEventIntervalInMilliseconds=”1000”
filter=".*\.xml$"
/>
```

Timer End-point

The timer end-point is useful when a resource needs to be pull on an interval, for example querying a database looking for new rows. The attribute "active" is convenient for pausing the timer.

A port bound to a timer executes the pipeline in sequence. It waits until the pipeline is terminated before attempting to start another which may result in an interval greater than the one defined in the binding.

To start the timer to a specific date set attribute “start_timer_first_Date” to MM-DD.

To start the timer to specific time set the attribute “start_timer_first_time” to HH:MM:SS, in the example below the timer start at 11 PM.

If the pipeline bound to this port starts after the start time, the timer will start the day after. For example, say that the start time is 22:00:00 and the pipeline starts 1 hour and 10 minutes later at 23:10:00, the timer will start the next day at 22:00:00.

To exclude some of days of the week you can use the following properties: exclude_SUNDAY, exclude_MONDAY, exclude_TUESDAY, exclude_WEDNESDAY, exclude_THURSDAY, exclude_FRIDAY, exclude_SATURDAY. In the example below the timer does not execute in the weekend;

To define a time window in which the timer executes for a specific day of the week you can use combination of following properties, in the example below the timer on Friday will run only 4 hours in the morning.

start_time_SUNDAY	end_time_SUNDAY
start_time_MONDAY	end_time_MONDAY
start_time_TUESDAY	end_time_TUESDAY
start_time_WEDNESDAY	end_time_WEDNESDAY
start_time_THURSDAY	end_time_THURSDAY
start_time_FRIDAY	end_time_FRIDAY
start_time_SATURDAY	end_time_SATURDAY

To exclude specific dates, you can set property “exclude_days_MM_DD_separated_by_comma”. In the following examples On Christmas and first day of year the timer does not execute.

```
<binding xmlns="http://www.ivitechnologies.com/pipeline/server">
	<port_manager>
		<port
			name="input"
			protocol="timer"
			active="true"
			interval="5000"
			time-unit="MILLISECONDS"
			start_timer_first_time="23:00:00"

			exclude_SUNDAY="true"
			exclude_SATURDAY="true"

			start_time_FRIDAY="08:00:00"
			end_time_FRIDAY="12:00:00"

			exclude_days_MM_DD_separated_by_comma="12-25,01-01"
			/>
	</port_manager>
</binding>
```

There are situations in which you may want to execute the timer only on specific business days.

For example, says you want to run a report every 5th business day of the month.

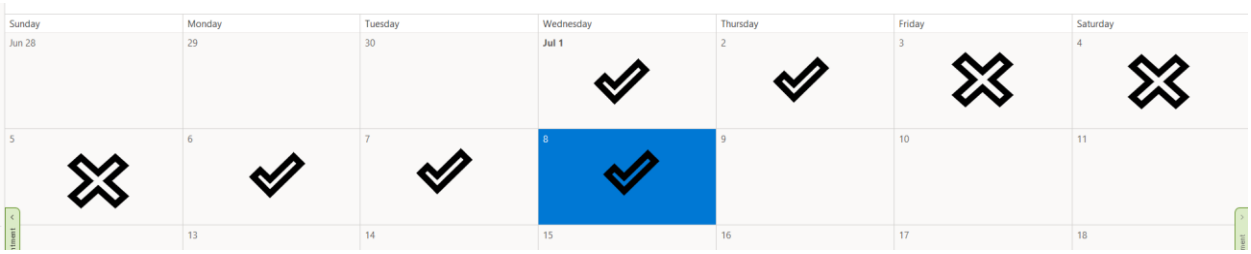
```
<binding xmlns="http://www.ivitechnologies.com/pipeline/server">
 <port_manager>
  <port
   name="input"
   protocol="timer"
   active="true"
   interval="24"
   time-unit="HOURS"
```

```
   start_timer_first_time="23:00:00"
holidays_separated_by_comma="NewYearDay,MemorialDay,IndependenceDay,LaborDay,Thanksgiving,BlackFr
iday,Christmas"
   executes_only_on_these_business_days_separated_by_comma="5"
   exclude_SATURDAY="true"
   exclude_SUNDAY="true"
   />
 </port_manager>
</binding>
```

To explain how business days are calculate let' take the 5th business day of July 2020 which was July 8.

The following calendar explains how the business day is calculated.

Independence Day (4 of July) which is a federal holiday in United States was on Saturday therefore it was celebrated the day before, July 3.



Property "holidays_separated_by_comma" allows to specify which holidays to include when counting the business day index. Supported values are: NewYearDay, MartinLutherKingDay, PresidentsDay, MemorialDay, IndependenceDay, LaborDay, ColumbusDay, VeteransDay, Thanksgiving, BlackFriday and Christmas.

FTP, FTPS and SFTP End-point

In the following example, we bind an FTP end-point to an input port and a SFTP end-point to an output port.

Property "max_connection_tries" determines how many times the connections is retried before throwing a severe error, the default is 1.

Property "wait_before_retry_in_millisecond" determines how long to wait between retries, the default value is 1 second.

Property "pullEventIntervalInMilliseconds" determines how frequently the FTP/SFTP client checks for new files on the server, the default value is 1 second.

Property “start_time_window_HH24_MM” prevents checking for new files if the current time is less than the value indicated.
Property “end_time_window_HH24_MM” prevents checking for new files if the current time is greater than the value indicated.

Property “skips_days_of_the_week” prevents checking for new files on specific weekdays.

Property “skip_dates” prevents checking for new files on specific dates express as month and day

```
<port
        name="FTP Input"
        protocol="ftp"
        host="127.0.0.1"
        port="21"
        path="/"
        user="ftpuser"
        password="ftpuser"
        filter="(.*).xml"
        pullEventIntervalInMilliseconds="10000"

        start_time_window_HH24_MM="14:00"
        end_time_window_HH24_MM="18:00"
        skips_days_of_the_week="SATURDAY,SUNDAY"
        skip_dates="12-25,01-01"/>
```

Property "ignore_modified_time_when_detecting_new_files" prevents to detect changes in the file last modified date. When property is set to true and a file on the server is overwritten the change is ignored. Default value is false.

Property "true_false_passive_mode" allows to enable PASSIVE mode. In passive mode FTP the client initiates both connections to the server, solving the problem of firewalls filtering the incoming data port connection to the client from the server. By default, the passive mode is on.

```
<binding xmlns="http://www.ivitechnologies.com/pipeline/server">
 <port_manager>

        <port
                name="FTP Input"
                protocol="ftp"
                host="127.0.0.1"
                port="21"
                path="/"
```

```
                user="ftpuser"
                password="ftpuser"
                filter="(.*).xml"
                true_false_passive_mode="true"
                milliseconds_connection_timeout="10000"
                pullEventIntervalInMilliseconds="1000"
                ignore_modified_time_when_detecting_new_files="false"/>

        <port
                name="Pipeline Output"
                protocol="sftp"
                host="127.0.0.1"
                port="22"
                path="/output"
                user="sftp"
                password="sftp"
                milliseconds_connection_timeout="10000"/>

 </port_manager>
</binding>
```

To enable FTPS set property "ftps_protocol" = SSL

```
<binding xmlns="http://www.ivitechnologies.com/pipeline/server">
<port_manager>
  <port name="in"
        protocol="ftp" ftps_protocol="SSL"
        host="127.0.0.1" port="23" path="/" user="ftps" password="ftps"/>
  <port name="out"
        protocol="ftp" ftps_protocol="SSL"
        host="127.0.0.1" port="23" path="/upload" user="ftps" password="ftps"/>
</port_manager>
</binding>
```

Property ftps_certificate_policy_all_valid_none can be set to "all" [default], "valid" or "none". It tells the FTPS client which SSL certificate to accept.

Property ftps_protocol_implict_true_false can be set to "true" or "false" [default]. When implicit is set to true the FTPS client assume that the entire transmission is encrypted. When the value is set to false only a portion of the transmission is encrypted, the encryption is negotiated with the server.

The following configuration properties instruct the output port to wait until the target file on the remote FTP server is deleted.

Property Name	Values	Default
wait_until_target_file_exists	true, false	True
wait_until_target_file_exists_sleeping_interval_in_milliseconds	number	10000
wait_until_target_file_exists_timeout_in_milliseconds	number	3600000

SFTP Authentication using Private/Public Key

The following configuration properties instruct the **SFTP** input/output port to use a private/public key for authentication

Property Name	Values

identity_private_key	<full path private key file>
identity_public_key	<full path public key file>
identity_passphrase	<Private key passphrase>

Here an example of binding

```
<binding xmlns="http://www.ivitechnologies.com/pipeline/server">
<port_manager>
  <port
   name="in"
   host='localhost' port='22' path='/'
   user='sftp_identity_test'
   identity_private_key='c:\users\sftp_identity_test\priv.ppk'
  />
 </port_manager>
</binding>
```

SFTP Low Level Logging

Set port property enable_sftp_logging to true to enable low level logging.

```
<binding xmlns="http://www.ivitechnologies.com/pipeline/server">
 <port_manager>
        <port
                name="Pipeline Output"
                protocol="sftp"
                host="127.0.0.1"
                port="22"
                path="/output"
                user="sftp"
                password="sftp"
                enable_sftp_logging="true"
                milliseconds_connection_timeout="10000"/>

 </port_manager>
</binding>
```

SFTP Legacy Algorithms

Starting from version 154, XPS bundles a new version of JSCH SFTP client which by default disables legacy algorithms ssh-rsa and ssh-dss.

By default, XPS enables ssh-rsa and ssh-dss, but expose a property to allow the user to disable them

```
enable_legacy_algorithms = 'false'
```

In addition, XPS allows to override the following session properties to specify which algorithms to use

```
pubkey_accepted_key_types = ''
server_host_key = ''
```

MQTT End-point

MQTT endpoints allow to publish and subscribe messages into a queue hosted by a MQTT broker like ActiveMQ Artemis

Getting started (mqtt.org)
MQTT · ActiveMQ Artemis Documentation (apache.org)

An input port bound to a MQTT queue triggers a pipeline execution when a new message is received from the broker.

An output port bound to a MQTT queue sends data to a broker 's queue.

MQTT protocol is designed to transmit small messages, up to few MB, it is not advisable to send large amount of data.

Here an example of port binding for MQTT. **You must use unique client_id for each port.**

```
<binding xmlns="http://www.ivitechnologies.com/pipeline/server">
    <port_manager>
        <port
            name="mqtt_input"
            protocol="mqtt"
            broker_url="tcp://localhost:1883"
            user="ivan"
            password=""
            destination="/queue_1"
            client_id="mqtt pipeline receiver"/>

        <port
            name="mqtt_output"
            protocol="mqtt"
            broker_url="tcp://localhost:1883"
            user="ivan"
            password=""
            destination="/queue_1"
            client_id="mqtt pipeline sender"
            message_retained="false"/>
    </port_manager>
</binding>
```

Websocket End-point

The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code.

XPS supports Websocket 1.1, it allows to bind input and output port to Websocket addresses.

By default, websocket is disabled, the following property has to be added to server.xml

```
<websocket_server enable="true"/>
```

The bound input port listens for inbound messages and triggers the pipeline execution when a new message is received. The message payload is used as input to the pipeline.

The bound output port sends any data produced by the pipeline execution to the configured websocket address.

```
<binding xmlns="http://www.ivitechnologies.com/pipeline/server">
    <port_manager>
        <port
            name="input_websocket"
            protocol="websocket"
            active="true"
            url="ws://localhost:9999/websocket/websocket.pipeline/receiver"
            />
        <port
            name="output_websocket"
            protocol="websocket"
            active="true"
            url="ws://localhost:9999/websocket/echo1/sender"
            />
    </port_manager>
</binding>
```

When the server-side connection is hosted by the XPS's Tomcat instance, XPS assigns a special meaning to the websocket URL:

```
ws://localhost:9999/websocket/<operation>/<role>
```

All connections to the same operation receive messages sent to this end point except for the connection with role = sender. For example you may have two input ports with the following end points

```
            url="ws://localhost:9999/websocket/websocket.pipeline/receiver1"
            url="ws://localhost:9999/websocket/websocket.pipeline/receiver2"
```

One output port

```
          url="ws://localhost:9999/websocket/websocket.pipeline/sender"
```

When the above outport sends a message, both input ports receive It, because they are bound to the same operation.

LDAP End-point

XPS supports LDAP server as data source and allows to bind a port.

Here is an example of binding:

```
<binding xmlns="http://www.ivitechnologies.com/pipeline/server">
    <port_manager>
        <port
           protocol="ldap"
           name="myldap"
           pullEventIntervalInMilliseconds="60000"
           url="ldap://localhost:389"
           dn="ou=users,dc=example,dc=com"
           Context.SECURITY_AUTHENTICATION="simple"
           Context.SECURITY_PRINCIPAL="uid=admin,ou=system"
```

```
            Context.SECURITY_CREDENTIALS="mypassword"
            property_list_to_omit_separated_by_comma="objectGUID,objectSid"
            query="(objectClass=*)"
            property_list_returning_attributes_separated_by_comma="*"
            returning_object_true_false="false"
            read_timeout_in_milliseconds="1000"
            connection_timeout_in_milliseconds="5000"
            sortBy="cn"
            paginationTrueFalse="true"
            pageSize="500"
            />
    </port_manager>
</binding>
```

Starting with build 229, the LDAP client supports connection retry logic with the following properties

```
            connect_n_retries="2"
            connect_before_retry_in_milliseconds="60000"
```

The resulting XML is structured as follows, the element under element “object” depends on the LDAP implementation and the type of object. Every object should always have a sub element “objectclass”

```
<result>
   <object>
      <dc name="dc" type="text">example</dc>
      <objectclass name="objectclass" type="text">domain</objectclass>
   </object>
</result>
```

Microsoft Graph End-point

XPS supports Microsoft Graph as data source and allows to bind a port.

Here is an example of binding:

```
<binding xmlns="http://www.ivitechnologies.com/pipeline/server">
<port_manager>
    <port
     name="msgraph_input"
     protocol="msgraph"
     pullEventIntervalInMilliseconds="60000"
     client_id='…'
     client_secret='…'
     tenant_id='…'
     timeout_in_seconds='10'
     select='id,givenName,surname,jobTitle,mail,mobilePhone'
     filter='surname eq &apos;Pedruzzi&apos;'
     expand='manager($select=mail)'
     orderby='surname'
     top='50'
     />
 </port_manager>
</binding>
```

Google Drive End-point

Google Drive end-points can be used as source or target for accessing and generating files.

To grant access to the Google Drive end-points you need to create a project in the Google Development Console, enable the Google Drive API and create a service account which can be exported as Json file.

You also need to need to grant access to the folder In your drive to the email specified in property client_email in the service account.

The following binding file example shows how to bind input and output ports:

```
<binding xmlns="http://www.ivitechnologies.com/pipeline/server">
    <port_manager>
        <port
              name="gdrive_input"
              protocol="gdrive"
              path="/Google API Test Folder/Folder1"
              service_account="service_account.json"/>
      <port
              name="gdrive_output"
              protocol="gdrive"
              path="/Google API Test Folder/output"
              outputFileName="AUP_2024.xml"
              service_account="service_account.json"/>
          </port_manager>
      </binding>
```

The service_account property must point to a file generated by the Google development console, here an example:

```
{
  "type": "service_account",
  "project_id": "…",
  "private_key_id": "…",
  "private_key": "…",
  "client_email": "…",
  "client_id": "…",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "…",
  "universe_domain": "googleapis.com"
}
```

The Google Drive end point uses a local cache to keep track of which files have been process the same way as the FTP end-point does, a folder named GDriveCache is created either in the product folder or in the xps_data folder.

The output port support these additional following properties:

- outputFileName to hard code the file output name
- appenDataToOutputFile to keep appending data to an existing file

DropBox End-point

DropBox end-points can be used as source or target for accessing and generating files.

To grant access to the DropBox end-points you need to create DropBox “Application” in the DropBox Console which has two properties the app key and the app secret.

You need grant access for your DropBox “Application” to a folder in your DropBox.

You will need to navigate to the following URL to authorize the DropBox “Application” and retrieve the authorization code:
https://www.dropbox.com/oauth2/authorize?client_id=<APP_KEY>&token_access_type=offline&response_type=code

Once you have obtained the authorization code you need to retrieve a permanent refresh_token

Open a command prompt and navigate to <XPS folder>\tools\dropbox and execute the following command

```
dropbox_refresh_token.bat "https://api.dropboxapi.com/oauth2/token" "<AUTHENTICATION CODE>" "<APP KEY>" "<APP SECRET>"
```

The command line will generate the permanent refresh token, that you will need to set in the binding file.

The following binding file example shows how to bind input and output ports:

```
<binding xmlns="http://www.ivitechnologies.com/pipeline/server">
    <port_manager>
       <port

        name="input"
        protocol="dropbox"
        app_key="<APP KEY>"
        app_secret="<APP SECRET>"
        refresh_token="<REFRESH TOKEN>"/>

        <port
        name="output"
        protocol="dropbox"
        app_key="<APP KEY>"
        app_secret="<APP SECRET>"
        refresh_token="<REFRESH TOKEN>"/>
        </port_manager>
        </binding>
```

The DropBox end point uses a local cache to keep track of which files have been process the same way as the FTP end-point does, a folder named DropBoxCache is created either in the product folder or in the xps_data folder.

Microsoft One Drive End-point

Microsoft One Drive end-points can be used as source or target for accessing and generating files.

To grant access to the Microsoft One Drive end-points you need to create an “Application” in your Microsoft Entra ID in the Microsoft Azure Console. The application will have a client id, in addition you will need to create a client secret.

You need grant access for your “Application” the following API permissions

API / Permissions name	Type	Description
Microsoft Graph (7)		
Files.Read	Delegated	Read user files
Files.Read.All	Delegated	Read all files that user can access
Files.ReadWrite	Delegated	Have full access to user files
Files.ReadWrite.All	Delegated	Have full access to all files user can access
offline_access	Delegated	Maintain access to data you have given it access to
Sites.ReadWrite.All	Application	Read and write items in all site collections
User.Read	Delegated	Sign in and read user profile

You need to set a redirect URI under Authentication -> Add Platform

You need to navigate to the following URL to authorize the “Application” and retrieve the authorization code, scope will need to list all above API permissions separated by space.

```
https://login.microsoftonline.com/{tenant id}/oauth2/authorize
?client_id={client_id}&scope={scope}&response_type=code&redirect_uri={redirect_uri}
```

Once you have obtained the authorization code you need to retrieve a permanent refresh_token, open a command prompt and navigate to <XPS folder>\tools\dropbox and execute the following command

```
SET tenant_id=
SET authorization_code=
SET client_id=
SET client_secret=
SET redirect_uri=
SET authorization_URL=https://login.microsoftonline.com/%tenant_id%/oauth2/token
dropbox_refresh_token.bat "%authorization_URL%" "%authorization_code%" "%client_id%"
"%client_secret%" "%redirect_uri%"
```

The above command obtains a permanent refresh_token.

The following binding file example shows how to bind input and output ports:

```
<binding xmlns="http://www.ivitechnologies.com/pipeline/server">
 <port_manager>

       <port

        name="input"
        protocol="onedrive"
        client_id="<client id>"
        client_secret="<client secret>"
        refresh_token="<REFRESH TOKEN>"/>

        <port
        name="output"
        protocol=" onedrive"
        client_id="<client id>"
        client_secret="<client secret>"
        app_secret="<APP SECRET>"
        refresh_token="<REFRESH TOKEN>"/>

 </port_manager>

</binding>
```

The One Drive end point uses a local cache to keep track of which files have been process the same way as the FTP end-point does, a folder named OneDriveCache is created either in the product folder or in the xps_data folder.

SMS End-point

SMS end-points allow triggering a pipeline execution using a SMS message. The SMS message is passed as input to the pipeline using the following format

```
<message>
    <sid>SMec3e3584cb0d37f9b085baa7aab6feca</sid>
    <from></from>
    <to>+19787375185</to>
    <body>2024-11-15T01:29:26.414Z</body>
    <sent>2024-11-15T01:29:28Z</sent>
</message>
```

Output ports bound to the SMS end-point send text payload as SMS messages.

The SMS end-point is implemented using Twilio SMS Service infostructure.

IVI Technologies offers its own Twilio account and can allocate local phone numbers for you as part of the SMS end-point licensing.

It's important to understand that the "from_phone" must point to a phone number allocated on the Twilio SMS Service infostructure which allows queuing SMS messages, such queue can be pulled by the SMS end-point.

Here is an example for how to bind input and output ports

```
<binding xmlns="http://www.ivitechnologies.com/pipeline/server">
 <port_manager>
  <port
    name="input"
    protocol="sms"
    account_sid="…"
    account_token="…"
    from_phone="…"
    since_time_unit="HOURS"
    since ="4"
  />

  <port
    name="output"
    protocol="sms"
    to_phone="+19787375185"
    account_sid="…"
    account_token="…"
    from_phone="…"/>
 </port_manager>
</binding>
```

Th SMS end point uses a local cache to keep track of which SMS have been process the same way as the FTP end-point does, a folder named SMSCache is created either in the product folder or in the xps_data folder.

HTTP End-point

HTTP end points allow triggering a pipeline execution using an HTTP request (GET or POST).

Here an example of HTTP binding configuration

```
<binding xmlns="http://www.ivitechnologies.com/pipeline/server">
    <port_manager>
        <port name="HTTP Request" protocol="http"/>
        <port name="HTTP Response" protocol="http"/>
    </port_manager>
</binding>
```

Once a pipeline input port is bound to an HTTP end point you can start a pipeline execution just using a browser. The following example shows how to execute a pipeline named mypipeline.pipeline

http://localhost/rest/pipelineRun?pipelineId=prj\mypipeline.pipeline

Using parameter ContentType you can specify the pipeline output content type which is very handy when a pipeline outputs HTML

http://localhost/rest/pipelineRun?pipelineId=prj\mypipeline.pipeline&ContentType=text/html

You can also pass parameters to the pipeline for example an order id

http://localhost/rest/pipelineRun?pipelineId=prj\mypipeline.pipeline&orderID=10

XML Pipeline Server passes the URL parameters to the pipeline input port using the following XML format **when method is GET or, when method is POST and the content-Type header is set to "application/x-www-form-urlencoded"**. In all other cases the message payload is passed as is the pipeline input port.

```
<form>
 <parameters>
  <parameter name="ContentType">text/html</parameter>
  <parameter name="pipelineId">prj\mypipeline.pipeline</parameter>
  <parameter name="orderID">10</parameter>
 </parameters>
</form>
```

Starting with build 170 is now possible to capture all form fields from muti-part form-data request.

To enable the new behavior pipeline biding property “enable_all_multipart_form_data_fields” needs to be set to true in the pipeline binding file as following:

```
<binding xmlns="http://www.ivitechnologies.com/pipeline/server">
 <port_manager>
  <port name="input_http" protocol="http"/>
  <port name="output_http" protocol="http"/>
 </port_manager>
 <http_properties enable_all_multipart_form_data_fields="true"/>
</binding>
```

Here is an example of simple HTML form

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head><title>Test EDI Conversion</title></head>
    <body>
        <h2>X12 Validation Form</h2>
        <form action="http://localhost:81/rest/pipelineRun"
              method="post"
              enctype="multipart/form-data">

         <input  type="text"
                 style="display:none"
                 name="pipelineId"
                 value="echo\echo.pipeline"/>
         <input type="text"
                style="display:none"
                name="ContentType"
                value="text/xml"/>

          <p>Select the EDI file you want to convert</p>
          <input type="text" name="first"/>
          <p><input type="file" name="edi"/></p>
          <p><input type="submit" value="Submit"/></p>
        </form>
    </body>
</html>
```

The pipeline input looks like the following. The file field values are encoded as base64.

```
<form>
 <parameters>
  <parameter name="pipelineId">echo\echo.pipeline</parameter>
  <parameter name="ContentType">text/xml</parameter>
  <parameter name="first">ivan</parameter>
  <parameter name="edi"
             file="TR.Z10530N81396.834FC.W.230920032549.001.x12"
             content-type="application/octet-stream">....</parameter>
 </parameters>
</form>
```

The HTTP parameters are accessible to the pipeline operations (XQuery and XSLT only). The operation needs to define a global parameter named "http-request" which is bound to an XML fragment formatted as the document above.

```
declare variable $http-request as document-node(element(*, xs:untyped)) external;

declare variable $customerID :=
       $http-request/form/parameters/parameter[@name="CustomerID"]/string();
```

It also possible to access the HttpServletRequest object using the following approach:

```
declare namespace hr = "ddtekjava:javax.servlet.http.HttpServletRequest";
declare namespace e = "ddtekjava:java.util.Enumeration";
declare namespace sb = "ddtekjava:java.lang.StringBuffer";
declare namespace xps = "ddtekjava:com.ivitechnologies.pipeline.web.HTTPContext";


declare function xps:getHttpServletRequest($sessionid as xs:string) as ddtek:javaObject external;

declare function e:hasMoreElements($this as ddtek:javaObject) as xs:boolean external;
declare function e:nextElement($this as ddtek:javaObject) as xs:string external;

declare function sb:toString($this as ddtek:javaObject) as xs:string external;

declare function hr:getRequestURI($this as ddtek:javaObject) as xs:string? external;
declare function hr:getRequestURL($this as ddtek:javaObject) as ddtek:javaObject external;
declare function hr:getRemoteUser($this as ddtek:javaObject) as xs:string? external;
declare function hr:getQueryString($this as ddtek:javaObject) as xs:string? external;
declare function hr:getRequestedSessionId($this as ddtek:javaObject) as xs:string? external;
declare function hr:getHeaderNames($this as ddtek:javaObject)as ddtek:javaObject external;
declare function hr:getHeader($this as ddtek:javaObject, $name as xs:string) as xs:string?
external;

declare function local:getHeaders(
 $httpServletRequest as ddtek:javaObject,
 $enum as ddtek:javaObject) as xs:string*
{
    if(e:hasMoreElements($enum)) then (
        e:nextElement($enum),
        local:getHeaders($httpServletRequest, $enum)
    )
    else ()
};



declare variable $http-request-id as xs:string external;
declare variable $httpServletRequest := xps:getHttpServletRequest($http-request-id);

declare option ddtek:serialize "indent=yes";


<HttpServletRequest>
  <RequestURI>{ hr:getRequestURI($httpServletRequest) }</RequestURI>
  <RequestURL>{ sb:toString(hr:getRequestURL($httpServletRequest)) }</RequestURL>
  <RemoteUser>{ hr:getRemoteUser($httpServletRequest) }</RemoteUser>
  <RequestedSessionId>{ hr:getRequestedSessionId($httpServletRequest) }</RequestedSessionId>
  <QueryString>{ hr:getQueryString($httpServletRequest) }</QueryString>
  <headers>{
     let $enum := hr:getHeaderNames($httpServletRequest)
     for $headerName in local:getHeaders($httpServletRequest, $enum)
     return
        let $headerValue := hr:getHeader($httpServletRequest, $headerName)
        return
        <header name="{$headerName}">{$headerValue}</header>
  }</headers>
</HttpServletRequest>
```

The Rest API supports upload of large files using RFC 1867, "Form-based File Upload in HTML". Here an example of a simple HTML form that posts an EDI file to the XML pipeline "X12 831 Example.pipeline"

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>Test EDI Conversion</title>
    </head>
    <body>
        <h2>EDI to XML Conversion</h2>
        <form
             action="http://localhost:81/rest/pipelineRun"
             method="post"
             enctype="multipart/form-data">

            <input
               type="text"
               style="display:none"
               name="pipelineId"
               value="X12 831 Example\X12 831 Example.pipeline"/>

            <input type="text" style="display:none" name="ContentType" value="text/xml"/>

            <p>Select the EDI file you want to convert</p>
            <p><input type="file" name="edi"/></p>
            <p><input type="submit" value="Submit"/></p>
        </form>
    </body>
</html>
```

HTTP Runtime Errors

When a runtime occurs while executing a pipeline bound to HTTP endpoints, the error is collected and send into the HTTP response. To disable this behavior and return an empty HTTP response set binding property push_error_on_http_output to false.

```
<binding xmlns="http://www.ivitechnologies.com/pipeline/server">
    <port_manager>
        <port name="HTTP_INPUT" protocol="http"/>
        <port name="HTTP_OUTPUT" protocol="http"/>
    </port_manager>
 <http_properties push_error_on_http_output="false"/>
</binding>
```

HTTP Headers

The following binding file shows how to configure extra http headers in the HTTP response.

```
<binding xmlns="http://www.ivitechnologies.com/pipeline/server">
    <port_manager>
        <port name="HTTP_INPUT" protocol="http"/>
        <port name="HTTP_OUTPUT" protocol="http">
```

```
            <header name="Control-Allow-Origin">*</header>
        </port>
    </port_manager>
</binding>
```

Logging HTTP Request Response

In some situations, it is useful to log all HTTP request response to investigate why a web service end point is not working properly.

XML Pipeline Server bundles a Servlet Filter implementation for this purpose, to enable the filter simply uncomment the following highlighted lines in <install-folder>\tomcat\webapp\WEB-INF\web.xml.

The HTTP traffic is stored in the tomcat standard output files, for example <install-folder>\tomcat\logs\ivixmlpipelineserver64-stdout.2016-10-12.log

Such filter may degrade the web service performance therefore it should be used only for diagnostic purpose and then disabled.

```
<web-app>
	<display-name>XML Pipeline Server</display-name>
	<servlet>
		<servlet-name>XML Pipeline Server</servlet-name>
		<servlet-class>com.ivitechnologies.pipeline.web.XMLPipelineServlet</servlet-class>
		<init-param>
			<param-name>root</param-name>
			<param-value>..</param-value>
		</init-param>
		<load-on-startup>1</load-on-startup>
	</servlet>
	<servlet-mapping>
		<servlet-name>XML Pipeline Server</servlet-name>
		<url-pattern>/rest/*</url-pattern>
		<url-pattern>/REST/*</url-pattern>
	</servlet-mapping>

<!--
	<filter>
	  <filter-name>Diagnostic Filter</filter-name>
	  <filter-class>com.ivitechnologies.pipeline.web.filter.ServletLogFilter</filter-class>
	</filter>
	<filter-mapping>
	  <filter-name>Diagnostic Filter</filter-name>
	  <url-pattern>/rest/*</url-pattern>
	  <url-pattern>/REST/*</url-pattern>
	</filter-mapping>
-->
</web-app>
```

Managing Token ID in Web applications

Some web application require to implement a token based authentication, XPS provides a token map built-in implementation for this purpose. In the following example, you see an XQuery program which implements a login functionality.

The query check if the user is already associated to a token ID calling tm:getToken.

If the token ID is not found user/password are searched in the database.

If user/password are found a new token is created calling tm:addToken and the token ID is returned to the caller.

Token IDs by default expire after 5 minutes. To change the time to live use function setTokenTimeToLiveInMilliseconds.

In some situation it may be useful to extend the token life as long the user is active on the service for such purpose call function extendTokenLife which resets the token internal counter.

```
declare namespace tm = "ddtekjava:com.ivitechnologies.pipeline.ext.web.TokenMap";

declare function tm:getToken($key as xs:string) external;
declare function tm:getKeyFromToken($key as xs:string) as xs:string? external;
declare function tm:addToken($key as xs:string) as xs:string external;
declare function tm:tokenExists($key as xs:string) as xs:boolen external;

declare function tm:setTokenTimeToLiveInMilliseconds($TimeToLive as xs:long) as xs:boolean
external;

declare function tm:removeTokenByID($token as xs:string) as xs:boolean external;
declare function tm:extendTokenLife($token as xs:string) as xs:boolean external;
declare function tm:getTokenAge($token as xs:string) as xs:long external;

declare variable $user := /Login/user;
declare variable $pwd := /Login/password;
declare variable $token := tm:getToken($user);


<LoginResponse>{
 if($token) then
  <tokenID new="false" user="{ tm:getKeyFromToken($token)}">{$token}</tokenID>
 else
  if( exists(collection("users")/users[user=$user and password= $pwd]) ) then
   <tokenID new="true">{tm:addToken($user)}</tokenID>
  else
   <error>Invalid Username or Password</error>
}</LoginResponse>
```

Disable End-Point

Sometimes it may be useful to shut down a particular end point without stopping the entire pipeline. To disable an end-point (input or output) use binding attribute active=false. Binding files can be update at any time. Here an example. When an input end point is disable it no longer triggers pipeline executions. When output end point is disable it no longer generates output. In both case a warning entry is logged by the server.

```
<binding xmlns="http://www.ivitechnologies.com/pipeline/server">
 <port_manager>
 <port
   name="Pipeline Input"
   protocol="file"
   folderFullPath="c:\temp\InputPortTest"
   active="false"/>
 </port_manager>
</binding>
```

Encryption

XML Pipeline Server supports encrypting and decrypting data using RSA PGP key pairs. All end points support a set of binding properties for PGP.

The following example is a binding definition for a file system input port which decrypts the incoming file before passing the data to the pipeline. Notice that URL for the private key can be expressed relative to the pipeline location or in absolute URL form file:///c:/mypgpkeys/secret.bpg

```
<port
    name="inEncryptedFiles"
    protocol="file"
    pgp="true"
    pgp_private_key_url="../../secret.bpg"
    pgp_key_passphrase="secret!"
/>
```

The next example is a binding definition for a file system output port which encrypts the data before storing the result on disk.

```
<port
    name="outEncryptedFiles"
    protocol="file"
    pgp="true"
    pgp_do_integrety_check="false"
    pgp_do_armor="false"
    pgp_public_key_url="../../pub.bpg"
/>
```

The same set of properties can be use with SFTP, FTP and Emails bindings

XQuery and Database Connections

When an XQuery operation which access a relational data-source is added to an XML pipeline, Stylus Studio registers the database connection into the operation's properties.

In the example below, you can see a fragment of a pipeline file. Notice that the password property, highlighted in yellow, is encrypted by Stylus Studio and it cannot be edit.

```
 <properties>
  <property name="Name" value="db.xquery"/>
  <property name=".xquery file" isURL="true" value="db.xquery"/>
  <property name="DB Connections">
  <xquery-collection
      name="localhost_1433"
      value="jdbc:I9:sqlserver:// localhost:1433;password=0C223C;DatabaseName=
test;xmlforest=true;user=xps;urltype=.xml"/>
  </property>
 </properties>
```

By default, database connections are created using the JDBC Driver Manager connection pooling. The default connection pooling can be disabled using the following pipeline binding property.

```
<?xml version="1.0"?>
<binding xmlns="http://www.ivitechnologies.com/pipeline/server">
 <xquery_properties use_built_in_database_connection_pooling="false" />
</binding>
```

In some situation, it may be necessary to override the database connection settings stored in the pipeline file to point to a different database server. The following example shows a pipeline binding file which defines an XQJConnection element.

The XQJConnection element takes precedence on the database connections defined in the pipeline file. The XML syntax for the XQJ element can be found at "Elements of the Source Configuration File"

Notice that database password is stored in clear using this approach.

```
<?xml version="1.0"?>
<binding xmlns="http://www.ivitechnologies.com/pipeline/server">
    <XQJConnection xmlns="http://www.datadirect.com/xquery">
        <JDBCConnection name="example_connection_name">
            <url>jdbc:I9:sqlserver://host2:1433</url>
            <user>xps</user>
            <password>xps</password>
        </JDBCConnection>
    </XQJConnection>
</binding>
```

Server Extension Functions

A variety of useful extension functions implemented in Java can be invoked from XQuery to perform tasks where the order of execution needs to be deterministic.

Zipping files

The following example show how to split a large file into a set of small files and move them into a zip archive.

```
declare namespace xps_zip = "ddtekjava:com.ivitechnologies.pipeline.ext.Zip";

declare function xps_zip:zip_files_to_url($fileNames as xs:string*, $zipURI as xs:string,
$moveFiles as xs:boolean) as xs:boolean external;

declare variable $zip_file := resolve-uri("books.zip",fn:static-base-uri());
declare variable $moveFilesToZip := fn:true();

let $xml_files :=
    (:Save each book on a separate file:)
    for $book at $pos in /books/book
        ,$fileName in concat($pos, ".xml")
        ,$url in resolve-uri($fileName ,fn:static-base-uri())
    return ( $url ,ddtek:serialize-to-url($book, $url,""))

return xps_zip:zip_files_to_url($xml_files,$zip_file,$moveFilesToZip)
```

Build 139 adds extension functions for zipping/unzipping strings

```
declare namespace ex="ddtekjava:com.ivitechnologies.pipeline.ext.Zip";
declare function ex:zipString($str as xs:string) as xs:base64Binary external;
declare function ex:unzipString($bytes as xs:base64Binary) as xs:string external;

declare option ddtek:serialize "indent=yes";

let $str := "XML XML XML XML XML XML XML XML XML XML XML XML XML XML XML XML XML "
let $bytes := ex:zipString($str)
let $zippedStr := xs:string($bytes)
let $unzippedStr := ex:unzipString($bytes)
return
   <root>
    <input_string>{$str}</input_string>
    <input_string_length>{fn:string-length($str)}</input_string_length>
    <zipped_string>{$zippedStr}</zipped_string>
    <zipped_string_length>{fn:string-length($zippedStr)}</zipped_string_length>
   </root>
```

Build 157 adds support of zipping entire folders:

```
declare function xps_zip:zip_files_to_url(
 $fileNames as xs:string*,
 $zipURI as xs:string,
 $moveFiles as xs:Boolean,
 $zipOnlyFolderContent as xs:boolean) as xs:boolean external;

declare function xps_zip:list_zip_content($zipURI as xs:string) as xs:string external;

declare variable $zip_file := resolve-uri("tempFolder.zip",fn:static-base-uri());
declare variable $moveFilesToZip := fn:false();
declare variable $zipOnlyFolderContent := fn:false();
declare variable $folders := ("file://c:/temp");

return
 let $success :=
    xps_zip:zip_files_to_url($folders, $zip_file,$moveFilesToZip,$zipOnlyFolderContent)
 return
    if($success) then
       xps_zip:list_zip_content($zip_file)
    else ()
```

To run the above query in Stylus Studio, make sure that the Project Classpath includes the following server jar files from the servlet lib folder

xmlpipelineserver.jar
commons-compress-1.12.jar

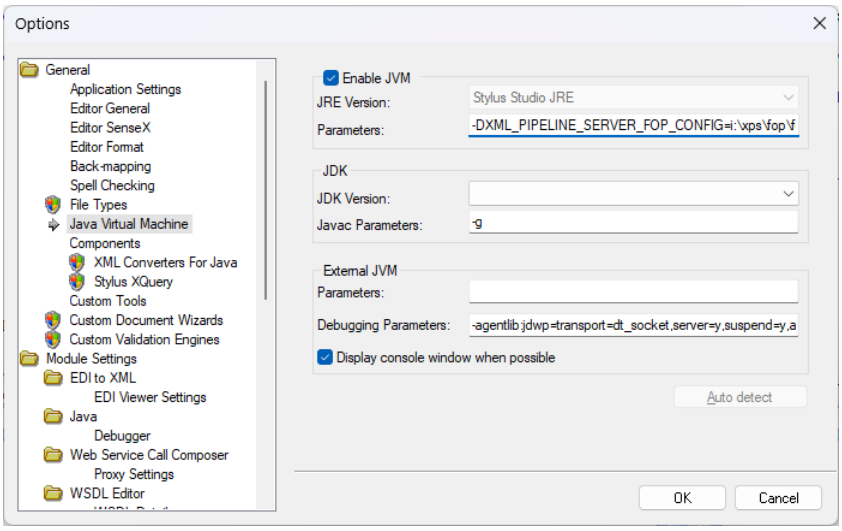
Executing FO Processor

The following examples shows how to execute Apache FOP from XQuery using FOP server extension function.

Please notice the FO extension functions require the Apache FOP configuration file, by default such file is located at `<XML Pipeline Server folder>\fop\fop_config.xml`

If you are testing your code in Stylus Studio and you have copied some of Java libraries but not the entire XML Pipeline Server distribution you need to set system property XML_PIPELINE_SERVER_FOP_CONFIG to the full path where your FOP configuration is located.

`-DXML_PIPELINE_SERVER_FOP_CONFIG=``i:\xps\fop\fop_config.xml`




```
declare namespace xps_pdf = "ddtekjava:com.ivitechnologies.pipeline.ext.fo.FOProcessor";
declare function xps_pdf:run($fo_root as node(), $pdfFilePath as xs:string) as xs:boolean
external;
declare function xps_pdf:run2($foFileURL as xs:string, $pdfURL as xs:string) as xs:boolean
external;

declare variable $pdf_url := fn:resolve-uri("hello.pdf", fn:static-base-uri());
declare variable $fo_document :=
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
    <fo:layout-master-set>
        <fo:simple-page-master
        master-name="default-page" page-height="11in" page-width="8.5in"
        margin-left="0.6in" margin-right="0.6in" margin-top="0.79in" margin-bottom="0.79in" >
```

```
            <fo:region-body/>
        </fo:simple-page-master>
    </fo:layout-master-set>
    <fo:page-sequence
    master-reference="default-page">
        <fo:flow flow-name="xsl-region-body">
            <fo:block>Hellow World</fo:block>
        </fo:flow>
    </fo:page-sequence>
</fo:root>;

xps_pdf:run($fo_document, $pdf_url)
```

To run the above query in Stylus Studio, make sure that the Project Classpath includes the following server jar files from the servlet lib folder

xmlpipelineserver.jar
avalon-framework-4.2.0.jar
fop.jar
avalon-framework4.2.0.jar
batik-all-1.8.jar
commons-io-2.4.jar
xmlgraphics-commons-2.1.jar
commons-logging-1.0.4.jar

Extracting Form Fields from PDF documents

The following examples shows how to execute the PDF form field extractor extension function.

```
declare namespace pdf = "ddtekjava:com.ivitechnologies.pipeline.ext.pdf.PDF";

declare function pdf:getFormFieldsFromPDFasBase64( $pdfDocAsBase64 as xs:string, $filterRegEx as
xs:string) as element() external;

declare function pdf: getFormFieldsFromPDFasFilePath( $pdfFilePathOrLocalUrl as xs:string,
$filterRegEx as xs:string) as element() external;


declare option ddtek:serialize "indent=yes";

pdf:getFormFieldsFromPDFasBase64(/pdf/string(), "LAST_NAME")
```

To run the above query in Stylus Studio, make sure that the Project Classpath includes the following server jar files from the servlet lib folder

pdfbox-2.0.13.jar
fontbox-2.0.13.jar
xmlpipelineserver.jar
commons-logging-1.2.jar
json-20090211.jar

Sending and Reading Emails

It is also possible to send and to read emails using the following server extension function

```
declare namespace es="ddtekjava:com.ivitechnologies.pipeline.ext.email.EmailSession";
declare namespace em="ddtekjava:com.ivitechnologies.pipeline.ext.email.Email";
declare function es:EmailSession( $settings as element(*, xs:untyped)) as ddtek:javaObject external;
declare function es:sendEmail($this as ddtek:javaObject, $email as ddtek:javaObject )  as xs:boolean external;
declare function em:Email($sender as xs:string, $recipients as xs:string, $subject as xs:string,
$body as xs:string, $message_mime_type as xs:string, $atatchments as xs:string*) as ddtek:javaObject external;
declare function em:setHeader($name as xs:string, $value as xs:string) as xs:string external;

let $e =<email
         mail.smtp.host=""
         mail.smtp.port=""
         mail.smtp.user=""
         mail.smtp.password=""
         mail.smtp.auth=""
         mail.smtp.starttls.enable=""
        read_emails_n_retries="3"
        read_emails_wait_before_retry_in_milliseconds="60000"
         send_email_n_retries="1"
         send_email_wait_before_retry_in_milliseconds="10000"
/>

let $session := es:EmailSession($e)
let $attachments := ("file:///c:/files/Users_Guide.pdf")
let $email := em:Email("ivanpedruzzi@gmail.com", "ivanpedruzzi@hotmail.com", "Subject", "Body", "text",
$attachments)
(:let $previuosHDvalue := em:setHeader("custom_header", "custom_header"):)
return es:sendEmail($session, $email)
```

Starting with version 111 it is possible to download the message body only when accessing its parts, setting attribute “download_parts_on_demand” to 'true', which makes downloading email much faster if you are only interested in the primary properties like subject, sender, sent, etc. **When such feature is enable it is mandatory to call ems:closeFolder to close the IMAP connection.**

To improve performance change the default fetch size from 16 KB to 4 MB and disable partial message fetching, in the EmailSession profile.

```
mail.imap.fetchsize="4194304"
mail.imap.partialfetch="false"
```

Here an example for reading emails.

```
declare namespace es="ddtekjava:com.ivitechnologies.pipeline.ext.email.EmailSession";
declare namespace ems="ddtekjava:com.ivitechnologies.pipeline.ext.email.Emails";
declare namespace em="ddtekjava:com.ivitechnologies.pipeline.ext.email.Email";
declare namespace emp="ddtekjava:com.ivitechnologies.pipeline.ext.email.EmailMessagePart";


declare function es:EmailSession($settings as element(*, xs:untyped)) as ddtek:javaObject external;
declare function es:readEmails($this as ddtek:javaObject, $filter as element()) as ddtek:javaObject external;

declare function ems:size($this as ddtek:javaObject) as xs:int external;
declare function ems:get($this as ddtek:javaObject, $index as xs:int) as ddtek:javaObject external;
declare function ems:closeFolder($this as ddtek:javaObject) as empty-sequence() external;

declare function em:getSubject($this as ddtek:javaObject) as xs:string external;
declare function em:getUID($this as ddtek:javaObject) as xs:long external;
declare function em:getSender($this as ddtek:javaObject) as xs:string external;
declare function em:getSentISODate($this as ddtek:javaObject) as xs:string external;
```

```
declare function em:getRecipients($this as ddtek:javaObject) as xs:string external;
declare function em:getRecipients_CC($this as ddtek:javaObject) as xs:string external;
declare function em:getRecipients_BCC($this as ddtek:javaObject) as xs:string external;

declare function em:getPart($this as ddtek:javaObject, $index as xs:int) as xs:string external;
declare function em:getPartObject($this as ddtek:javaObject, $index as xs:int) as ddtek:javaObject external;
declare function em:getPartObjectByFileName($this as ddtek:javaObject, $FileName as xs:string) as
ddtek:javaObject external;

declare function emp:getFileName($this as ddtek:javaObject) as xs:string external;
declare function emp:getContentType($this as ddtek:javaObject) as xs:string external;
declare function emp:getDescription($this as ddtek:javaObject) as xs:string external;
declare function emp:getDisposition($this as ddtek:javaObject) as xs:string external;
declare function emp:geContent($this as ddtek:javaObject) as xs:base64Binary* external;
declare function emp:toFileOrUrl($this as ddtek:javaObject, $FileName as xs:string) as xs:boolean external;

declare function em:getParts($this as ddtek:javaObject) as document-node()? external;
declare function em:saveAttachmentToFile($this as ddtek:javaObject, $attachmentName as xs:string, $fileOrUri as
xs:string) as xs:boolean external;


declare option ddtek:serialize "indent=yes";

let $settings := <email
    mail.imap.host=""
    mail.imap.port=""
    mail.imap.user=""
    mail.imap.password=""
    mail.imap.ssl.enable="true"

    mail.imap.fetchsize="4194304"
    mail.imap.partialfetch="false"

    read_emails_n_retries="3"
    read_emails_wait_before_retry_in_milliseconds="60000"
/>

let $session := es:EmailSession($settings)

let $nStartSearchDaysBack := '1'
let $duration := xs:dayTimeDuration(concat('P',$nStartSearchDaysBack,'D'))
let $startingFrom:= ddtek:format-date(fn:current-date() - $duration, '[Y0001]-[M01]-[D01]')

let $filter := <filter
                    folder='inbox'
                    subjectFilter='XML Pipeline Server Download for'
                    fromFilter='website@ivitechnologies.com'
                    startingFrom='{$startingFrom}'
                    download_parts_on_demand='true'/>

let $emails := es:readEmails($session, $filter)
return (
<emails>{
    for $i in 0 to ems:size($emails) - 1
        ,$email in ems:get($emails,xs:int($i) )
    return
        <email>
            <uuid>{em:getUID($email)}</uuid>
             <subject>{em:getSubject($email)}</subject>
             <sender>{em:getSender($email)}</sender>
             <recipients>{em:getRecipients($email)}</recipients>
             <recipients_CC>{em:getRecipients_CC($email)}</recipients_CC>
             <recipients_BCC>{em:getRecipients_BCC($email)}</recipients_BCC>
            {em:getParts($email)}
            <body>{ddtek:convert-to-xml(em:getPart($email, xs:int(0)), "HTML:")}</body>
            <downloadAttachment>{
                em:saveAttachmentToFile($email, "fo.pdf",
                    resolve-uri("fo_downloaded_attachment.pdf", fn:static-base-uri())
                )
            }</downloadAttachment>
             <emailMessageAsXML>{em:getParts($email)}</emailMessageAsXML>
```

```
        </email>
}</emails>

 , ems:closeFolder();
)
```

It is also possible to call a different override for extension function readEmails that takes an XML element for configuring the query filters. The following example set the lower and upper limit dates to restrict the result. Both limits are inclusive

```
declare namespace es="ddtekjava:com.ivitechnologies.pipeline.ext.email.EmailSession";
declare namespace ems="ddtekjava:com.ivitechnologies.pipeline.ext.email.Emails";
declare namespace em="ddtekjava:com.ivitechnologies.pipeline.ext.email.Email";

declare function es:EmailSession($settings as element(*, xs:untyped)) as ddtek:javaObject external;
declare function es:readEmails($this as ddtek:javaObject, $settings as element(*, xs:untyped));

let $settings := <email
   mail.imap.host=""
   mail.imap.port=""
   mail.imap.user=""
   mail.imap.password=""
   mail.imap.ssl.enable="true"
   read_emails_n_retries="3"
   read_emails_wait_before_retry_in_milliseconds="60000"
/>
let $session := es:EmailSession($settings)
let $filters := < filters folder="inbox" startingFrom="2015-06-01" endingTo="2015-06-02"/>
return es:readEmails($session, $filters)
```

Moving emails on a different folder

```
declare namespace es="ddtekjava:com.ivitechnologies.pipeline.ext.email.EmailSession";

declare function es:EmailSession(
 $settings as element(*, xs:untyped)) as ddtek:javaObject external;

declare function es:moveEmails(
 $session as ddtek:javaObject,
 $settings as element(*, xs:untyped),
 $targetFolder as xs:string) as xs:boolean external;


let $settings := <email mail.imap.host="" mail.imap.port="" mail.imap.user=""
mail.imap.password="" mail.imap.ssl.enable="true"/>

let $session := es:EmailSession($settings)

let $filters := < filters
folder="inbox"
subjectFilter="XML Pipeline Server Download for"
fromFilter="website@ivitechnologies.com"
startingFrom="2015-06-01"/>

return es:moveEmails($session, $filters, "Deleted Items")
```

Starting from build 166 the email session allows to enable a file system based cache which allows skip emails already downloaded. The following example shows the new attributes to enable the cache.

When property emailCacheDeleteAfterDuration is set the fingerprint files older than the duration indicated will be deleted, in the example below files older than 30 day are deleted.

```
<email
 mail.imap.host="imap.gmail.com"
 mail.imap.port=""
 mail.imap.user=""
 mail.imap.password=""
 mail.imap.ssl.enable="true"
 emailCacheFolderLocalFilePath="c:\xps\EmailCache"
 emailCacheDeleteAfterDuration="P0Y0M30DT0H0M0S"
 emailCacheDeleteFilter=".*"
/>
```

To run the above query in Stylus Studio, make sure that the Project Classpath includes the following server jar files from the servlet lib folder.

xmlpipelineserver.jar
javax.mail.jar

Add Email Attachments as BASE64

Starting with build 244, it possible to create email attachments using base64 string. The attachment mime type is autodetected from the file name extension, using the Java platform default mime type mapping.

```
declare namespace es="ddtekjava:com.ivitechnologies.pipeline.ext.email.EmailSession";
declare namespace em="ddtekjava:com.ivitechnologies.pipeline.ext.email.Email";
declare function es:EmailSession($settings as element(*, xs:untyped)) as ddtek:javaObject
external;
declare function es:sendEmail($this as ddtek:javaObject, $email as ddtek:javaObject ) as empty-
sequence() external;
declare function em:Email($sender as xs:string, $recipients as xs:string, $subject as xs:string,
$body as xs:string, $message_mime_type as xs:string, $atatchments as xs:string*) as
ddtek:javaObject external;
declare function em:addBase64Attachment($this as ddtek:javaObject, $filename as xs:string,
$base64 as xs:string) as xs:boolean external;

let $settings := doc("email.xml")
let $uri := document-uri($settings)
let $session := es:EmailSession($settings/email)
let $attachmentFileName := "onepixel.png"
let $attachmentBase64 :=
"iVBORw0KGgoAAAANSUhEUgAAAAEAAAABCAIAAACQd1PeAAAACXBIWXMAAAsTAAALEwEAmpwYAAAADElEQVQImWP4w8YGAAMK
AQkLTCt0AAAAAElFTkSuQmCC"
let $subject := "Email from Java Extension - base64 attachment"
let $from := "stylussupport@ivitechnologies.com"
let $to := "stylussupport@ivitechnologies.com"
let $email := em:Email($from, $to, $subject, $subject, "text", ())
return
 if(em:addBase64Attachment($email, $attachmentFileName, $attachmentBase64) = fn:false()) then
  fn:error((), "addBase64Attachment failed")
```

```
else
 es:sendEmail($session, $email)
```

IMAP and SMTP OAUTH2 Authentication

Starting year 2022, cloud email server provides like Google Gmail and Microsoft Office365 have blocked plain text authentication and forced email client applications to switch to OAUTH2.

XML Pipeline Server IMAP client supports OAUTH2 authentication starting with build 136.

The following configurations show the properties necessary for connecting to Office365.

Your Azure/Exchange administrator will need to register a Client Application in Azure Active Directory and provide you with the following information

Email address you want to access
Azure tenant id
Azure Client Application client id
Azure Client Application client secret

```
<imap
 mail.store.protocol="imap"
 mail.imap.host="outlook.office365.com"
 mail.imap.port="993"
 mail.imap.ssl.enable="true"
 mail.imap.starttls.enable="true"
 mail.imap.auth="true"
 mail.imap.auth.mechanisms="XOAUTH2"
 mail.imap.user="{email address}"

 xps_oauth2_url="https://login.microsoftonline.com/{tenantid}/oauth2/v2.0/token"
 xps_oauth2_client_id="37b……."
 xps_oauth2_client_secret="Mfg……"
 xps_oauth2_grant_type="client_credentials"
 xps_oauth2_scope="https://outlook.office365.com/.default"
 />
```

Starting build 139, the same properties can be used with SMTP.

As things stand on September 2022, Microsoft Office365 SMTP does not support yet OAUTH authentication with the "client credential" flow.

The following configurations show the properties necessary for connecting to Gmail

Your Google administrator will need to register a Client Application in Goggle Admin Console and provide you with the following information

Email address you want to access
Client Application client id
Client Application client secret
Refresh Token

```
<imap
 mail.store.protocol="imap"
 mail.imap.host="imap.gmail.com"
 mail.imap.port="993"
 mail.imap.ssl.enable="true"
 mail.imap.starttls.enable="true"
 mail.imap.auth="true"
 mail.imap.auth.mechanisms="XOAUTH2"
 mail.imap.user="{email address}"

 xps_oauth2_url="https:// oauth2.googleapis.com/token"
 xps_oauth2_client_id="37b……"
 xps_oauth2_client_secret="Mfg……"
 xps_oauth2_grant_type="refresh_token"
 xps_oauth2_refresh_token="1//0……"
 />
```

File Operations

The following example shows how to copy, move and delete files

```
declare namespace xps_file = "ddtekjava:com.ivitechnologies.pipeline.ext.io.FileOperations";

declare function xps_file:copy(
 $uriSourceFile as xs:string, $uriTargetFile as xs:string) as xs:boolean external;

declare function xps_file:move(
 $uriSourceFile as xs:string, $uriTargetFile as xs:string) as xs:boolean external;

declare function xps_file:delete(
 $uriLocalFile as xs:string, $deleteIfExists as xs:boolean) as xs:boolean external;

declare function xps_fle:createDirectory($uriLocalFile as xs:string, $doNotThrowIfAlreadyExists
as xs:boolean) as xs:boolean external;
declare function xps_file:deleteTree($uriLocalFile as xs:string) as xs:boolean external;
declare function xps_file:urlToLocalPath($url as xs:string) as xs:string external;

declare function xps_file:fileExists($uriFile as xs:string) as xs:boolean external;


declare variable $source_url := fn:document-uri(/);
declare variable $target_url := fn:concat($source_url,".copy");
declare variable $target2_url := fn:concat($target_url,".move");
declare variable $delete_only_if_exists as xs:boolean := fn:false();
```

```
xps_file:copy($source_url, $target_url),
if(xps_file:fileExists($target_url) = fn:false())
then "xps_file:copy failed"
else (
    xps_file:move($target_url, $target2_url)
  ,xps_file:delete($target2_url, $delete_only_if_exists)
)
```

The following example illustrates how to list files in a folder:

```
import module namespace file = "ddtekjava:com.ivitechnologies.pipeline.ext.io.FileOperations" at
"../../xquery_lib/xps/file/file_lib.xquery";

declare option ddtek:serialize "indent=yes";

let $file := fn:tokenize(fn:static-base-uri(), "/")[last()]

let $folder := fn:substring(
 fn:static-base-uri(),
 1,
 fn:string-length(fn:static-base-uri()) - fn:string-length($file)-1)

return file:listFiles($folder, "books*\.xml", fn:true())
```

The above query returns the following xml:

```
<files folder="F:\builds\PipelineServer\tests\FileOperations"
       folderURL="file:/F:/builds/PipelineServer/tests/FileOperations/">
   <file date="2021-02-20T00:44:11.437279Z"
         dateMilliseconds="1613781851437"
         name="books.xml"
         size="2788"/>
   <folder folder="F:\builds\PipelineServer\tests\FileOperations\subfolder"
           folderURL="file:/F:/builds/PipelineServer/tests/FileOperations/subfolder/"
           name="subfolder">
      <file date="2021-02-20T00:44:11.437279Z"
            dateMilliseconds="1613781851437"
            name="books.xml"
            size="2788"/>
   </folder>
</files>
```

The following example illustrates how to append data to an existing file

```
declare function xps_file:appendFileToFile($sourceUriOrLocalPath as xs:string,
$targetUriOrLocalPath as xs:string, $deleteSource as xs:boolean)as xs:boolean external;

declare function xps_file:appendDataToFile($data as xs:string, $targetUriOrLocalPath as
xs:string)as xs:boolean external;



let $sourceText1 := resolve-uri("text1.txt", fn:static-base-uri())
let $sourceText2 := resolve-uri("text2.txt", fn:static-base-uri())
let $targetText := resolve-uri("text_append.txt", fn:static-base-uri())
return(
     xps_file:appendFileToFile($sourceText1, $targetText, fn:false())
    ,xps_file:appendDataToFile("&#10;&#13;", $targetText)
    ,xps_file:appendFileToFile($sourceText2, $targetText, fn:false())
)
```

The following example illustrates how to read and to write text to and from files

```
declare namespace xps_file = "ddtekjava:com.ivitechnologies.pipeline.ext.io.FileOperations";
declare function xps_file:readAll($uriSourceFile as xs:string) as xs:string external;
declare function xps_file:writeAll($uriSourceFile as xs:string, $data as xs:string ) as xs:boolean external;
declare variable $source_url := fn:document-uri(/);
declare variable $target_url := fn:concat($source_url,".copy");
declare option ddtek:serialize "method=text";

let $data := xps_file:readAll($source_url)
return xps_file:writeAll($target_url, $data)
```

Delete Old Files

There are situations in which files are accumulated over time for logging purpose but files older than a threshold are no longer relevant, form example a pipeline may want to keep only files created in the last 30 days. The value for parameter “olderThan” must be expressed as string representation, "PnYnMnDTnHnMnS", as defined in XML Schema 1.0 section 3.2.6.1.

```
declare namespace xpsfiles = "ddtekjava:com.ivitechnologies.pipeline.ext.io.FileOperations";

declare function xpsfiles:deleteOldFilesFromFolder(
    $folder as xs:string,
    $file_filter_regex as xs:string,
    $olderThan as xs:string) as xs:boolean external;

declare function xpsfiles:fileExists($uriFile as xs:string) as xs:boolean external;

declare variable $folder := "file:///c:/logs";

xpsfiles:deleteOldFilesFromFolder($folder,".*\\.log", "P0Y0M30DT0H0M0S")
```

Synchronize Folder Content

Extension function synchFiles allows to copy files from a source folder to a target folder that are new or do not exist in the target folder. The function returns a list of URL for the files that have been copied.

```
declare namespace xps_file = "ddtekjava:com.ivitechnologies.pipeline.ext.io.FileOperations";


declare function xps_file:syncFiles(
  $uriOrPathSourceFolder as xs:string
 ,$uriOrPathTargetFolder as xs:string
 ,$filter as xs:string) as xs:string* external;


let $source := fn:resolve-uri("source", fn:static-base-uri())
let $target := fn:resolve-uri("target", fn:static-base-uri())
return

 fn:string-join(
  xps_file:syncFiles($source, $target, "*.*")
 ,"&#10;")
```

In order to run the above query in Stylus Studio make sure that the Project Classpath includes the following server jar files from the servlet lib folder

xmlpipelineserver.jar

Execute External Processes

The following example shows how to execute a batch file from an XQuery program

```
declare namespace processBuilder =
"ddtekjava:com.ivitechnologies.pipeline.ext.execution.SimpleProcessBuilder";
declare namespace xps_fo =
"ddtekjava:com.ivitechnologies.pipeline.ext.io.FileOperations";

declare function
processBuilder:startAndWaitForCompletion($workingFolderOrFileInWorkingFolder as
xs:string, $fullPathToCommand as xs:string, $args as xs:string*, $waitInMillisecond as
xs:long) as xs:int external;
declare function xps_fo:urlToLocalPath($url as xs:string) as xs:string external;

let $workingFolder := xps_fo:urlToLocalPath(fn:static-base-uri())
let $cmd := xps_fo:urlToLocalPath(fn:resolve-uri("mycopy.bat", fn:static-base-uri()))
let $arg := xps_fo:urlToLocalPath(document-uri(/))
let $T10Seconds := xs:long(10000)
return
    processBuilder:startAndWaitForCompletion(
         $workingFolder
        ,$cmd
        ,$arg
        ,$T10Seconds
    )
```

To run the above query in Stylus Studio, make sure that the Project Classpath includes the following server jar files from the servlet lib folder

xmlpipelineserver.jar

FTP Operations

```
declare namespace xps_ftp = "ddtekjava:com.ivitechnologies.pipeline.ext.net.FTP";
declare function xps_ftp:sendFile($element as element(), $uriLocalFile as xs:string, $targetPath
as xs:string) as xs:boolean external;
declare function xps_ftp:sendString($element as element(), $data as xs:string, $targetPath as
xs:string) as xs:boolean external;
declare function xps_ftp:deleteFile($element as element(),$targetPath as xs:string) as xs:boolean
external;
declare function xps_ftp:renameFile($element as element(),$sourcePath as xs:string,$targetPath as
xs:string) as xs:boolean external;
declare function xps_ftp:makeFolder($element as element(),$targetPath as xs:string) as xs:boolean
external;
declare function xps_ftp:deleteFolder($element as element(),$targetPath as xs:string) as
xs:boolean external;
declare function xps_ftp:fileExists($element as element(),$targetPath as xs:string) as xs:boolean
external;
```

```
xps_sftp:sendFile(
    <root host='127.0.0.1' port='21' path='/' user='ftp' password='ftp'/>
    ,resolve-uri("../../831.xml", fn:static-base-uri())
    ,"831.xml")
```

The following configuration properties instructs function sendFile to wait until the target file on the remote FTP server is deleted.

Property Name	Values	Default
wait_until_target_file_exists	true, false	true
wait_until_target_file_exists_sleeping_interval_in_milliseconds	number	10000
wait_until_target_file_exists_timeout_in_milliseconds	number	3600000

The following example shows how to download files from a FTP server. Such approach can be used instead of binding an input port when the application needs to download a set files in a single pass and process them as group.

```
declare namespace ftp = "ddtekjava:com.ivitechnologies.pipeline.ext.net.FTP";

declare function ftp:listFiles($config as element()) as element() external;
declare function ftp:get($config as element(), $sourcePath as xs:string, $urlOrLocalPathTarget as
xs:string) as xs:boolean external;

declare variable $ftp := <root host='127.0.0.1' port='21' path='/' user='ftp' password='ftp'/>;
declare variable $downloadFolder := resolve-uri("downloaded/", static-base-uri());


for $fileName in ftp:listFiles($ftp)/file/@name
return
 ftp:get($ftp, $fileName, resolve-uri($fileName, $downloadFolder)
```

Function listFiles returns an XML structure like in the following example

```
<files>
 <file date="2017-08-23 17:10:00" dateMilliseconds="1503522600000" name="books.xml" size="2788"/>
</files>
```

In order to run the above query in Stylus Studio make sure that the Project Classpath includes the following server jar files from the servlet lib folder

xmlpipelineserver.jar
commons-net-3.6.jar

SFTP Operations

```
declare namespace xps_sftp = "ddtekjava:com.ivitechnologies.pipeline.ext.net.SFTP";
declare function xps_sftp:sendFile($element as element(), $uriLocalFile as xs:string, $targetPath
as xs:string) as xs:boolean external;
```

```
declare function xps_sftp:sendString($element as element(), $data as xs:string, $targetPath as
xs:string) as xs:boolean external;
declare function xps_sftp:deleteFile($element as element(),$targetPath as xs:string) as
xs:boolean external;
declare function xps_sftp:renameFile($element as element(),$sourcePath as xs:string,$targetPath
as xs:string) as xs:boolean external;
declare function xps_sftp:makeFolder($element as element(),$targetPath as xs:string) as
xs:boolean external;
declare function xps_sftp:deleteFolder($element as element(),$targetPath as xs:string) as
xs:boolean external;
declare function xps_sftp:fileExists($element as element(),$targetPath as xs:string) as
xs:boolean external;
```

```
xps_sftp:sendFile(
    <root host='127.0.0.1' port='21' path='/' user='sftp' password='sftp'/>
    ,resolve-uri("../../831.xml", fn:static-base-uri())
    ,"831.xml")
,
xps_sftp:deleteFile(
    <root host='127.0.0.1' port='21' path='/' user='sftp' password='sftp'/>
   ,"831.xml")
,
xps_sftp:renameFile(
    <root host='127.0.0.1' port='21' path='/' user='sftp' password='sftp'/>
   ,"831.xml", "831_new.xml")
,
xps_sftp:makeFolder(
    <root host='127.0.0.1' port='21' path='/' user='sftp' password='sftp'/>
   ,"MyNewFolder")

,
xps_sftp:deleteFolder(
    <root host='127.0.0.1' port='21' path='/' user='sftp' password='sftp'/>
   ,"MyNewFolder")
```

The following configuration properties instructs function sendFile to wait until the target file on the remote FTP server is deleted.

Property Name	Values	Default
wait_until_target_file_exists	true, false	true
wait_until_target_file_exists_sleeping_interval_in_milliseconds	number	10000
wait_until_target_file_exists_timeout_in_milliseconds	number	3600000

The following example shows how to download files from a SFTP server. Such approach can be used instead of binding an input port when the application needs to download a set files in a single pass and process them as group.

```
declare namespace sftp = "ddtekjava:com.ivitechnologies.pipeline.ext.net.SFTP";

declare function sftp:listFiles($config as element()) as element() external;
declare function sftp:get($config as element(), $sourcePath as xs:string, $urlOrLocalPathTarget
as xs:string) as xs:boolean external;

declare variable $ftp := <root host='127.0.0.1' port='21' path='/' user='sftp' password='sftp'/>;
declare variable $downloadFolder := resolve-uri("downloaded/", static-base-uri());
```

```
for $fileName in sftp:listFiles($ftp)/file/@name
return
 sftp:get($ftp, $fileName, resolve-uri($fileName, $downloadFolder)
```

Function listFiles returns an XML structure like in the following example

```
<files>
 <file date="2017-08-23 17:10:00" dateMilliseconds="1503522600000" name="books.xml" size="2788"/>
</files>
```

To run the above query in Stylus Studio make sure that the Project Classpath includes the following server jar files from the servlet lib folder

xmlpipelineserver.jar
jsch-0.2.24.jar

HTTP Operations

In order to overcome the limitation of ddtek:http-post a new extension function was added in XPS build 167.

Function HTTP.VERB supports the following HTTP verbs: POST,PUT, PATCH.GET,HEAD,OPTIONS,TRACE,DELETE.

Element option is fully compatible with ddtek:http-* functions

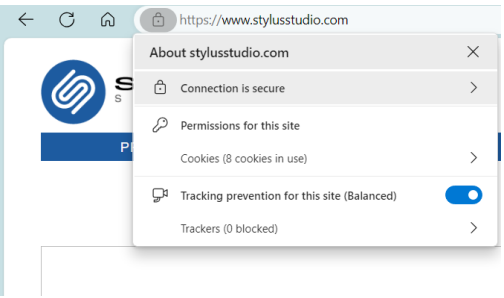
Attributes supported by element “options”:

Name	Default Value	Possible values	Optional
username			Yes
password			Yes
force_authentication		BASIC, DIGEST	Yes
ssl-common-name-whitelist			Yes
realm			Yes
Nonce			Yes
proxy-host			Yes
proxy-port			Yes
proxy-username			Yes
proxy-password			Yes

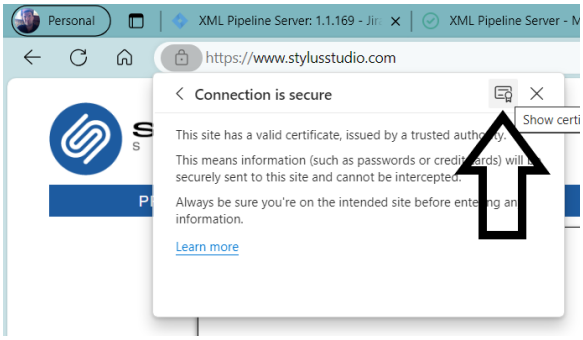
connection-timeout			Yes
connection-request-timeout			Yes
Retries	1		Yes
wrap-exception	False	false, true	Yes

Attrbute “ssl-common-name-whitelist” is used to accept specific Common Name from the SSL certificate without the need to install the service.

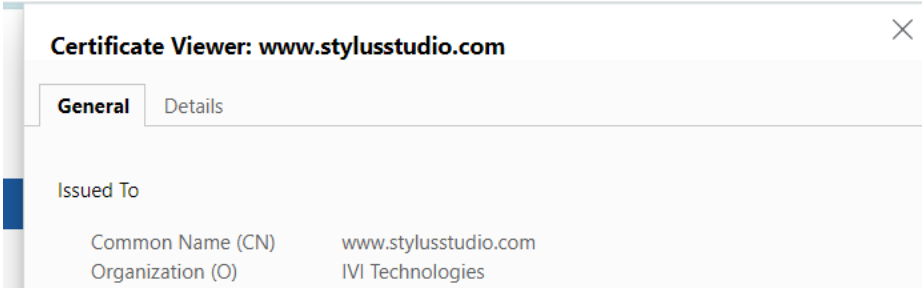
To get the common name (CN) in the SSL certificate, open the URL in the browser and click on the lock icon



Click Connection is Secure



Click certificate icon, the common name (CN) is the first field



Attribute supported by element “request/parameters/parameter”:

Name	Default Value	Possible values	Optional
name			No
filename			Yes
type	text	text, file, base64	No

content-type	When type=base64 or type=file
	`application/octet-stream`
	
	When type=text
	`text/plain`
			Yes

Attribute supported by element “request/form/parameters/parameter”:

Name	Default Value	Possible values	Optional
Name			No

The following attributes are supported: username, password, proxy-host, proxy-port, proxy-username, proxy-password, connection-timeout, connection-request-timeout, socket-timeout, retries, wrap-exception.

The function return value is also compatible with ddtek:http-* functions. Here an example

```
<response http-version="HTTP/1.1" status-code="100" reason="OK">
 <response-header>
  <header name="Content-Type" value="application/json"/>
 </response-header>
 <response-body>…</response-body>
</response>
```

The following example shows how to a HTTP POST as form-data. When the parameter type is set to “file” the parameter value must point to a local file.

```
declare namespace xps_http = "ddtekjava:com.ivitechnologies.pipeline.ext.net.HTTP";
declare function xps_http:VERB(
   $verb as xs:string, $url as xs:string,
   $options as document-node()?,$payload as document-node()?,
   $output_URI as as:AnyURI?) external;

declare variable $url := "http://myhost/maypath";
declare variable $VERB_OPTIONS_POST :=
 document{
  <options username="*****" password="*****">
   <headers>
      <header name="SomeCustomHeader">SomeCustomValue</header>
   </headers>
  </options>
 };

declare variable $payload :=
 document{
  <request>
    <parameters>
     <parameter
      name="file"
      content-type="text/csv"
      type="file">c:\myfolder\myfile.csv</parameter>
    </parameters>
  </request>
};
xps_http:VERB('POST',$url,$VERB_OPTIONS_POST,$payload, ())
```

The following payload shows how to post data as x-www-form-urlencoded.

```
declare variable $payload :=
 document{
  <request>
   <form>
    <parameters>
     <parameter name="first">Ivan</parameter>
     <parameter name="last">Pedruzzi</parameter>
    </parameters>
   </form>
  </request>
};
```

The following payload shows how to post JSON, setting attribute “sendRequestBodyTextOnly” to true instructs the http client to send only the text embedded in the request element.

```
declare variable $VERB_OPTIONS_POST :=
 document{
  <options>
   <headers>
      <header name="Content-Type">application/json</header>
   </headers>
  </options>
 };


declare variable $payload :=
 document{
  <request sendRequestBodyTextOnly="true">{id:0}</request>
};
```

The following example shows how download a file using HTTP GET

```
let $url_invoice := "https://mysite/myinvoices/123456"
let $invoice_number := "123456"
let $filename := concat("invoice_",$invoice_number,".pdf")
let $output_URI := fn:resolve-uri($filename, fn:static-base-uri())

xps_http:VERB('GET',$url_invoice,$VERB_OPTIONS_POST,(), $output_URI)
```

To run the above query in Stylus Studio, make sure that the Project Classpath includes the following server jar files from the servlet lib folder:

xmlpipelineserver.jar
json-20090211.jar
httpcore5-5.2.jar
httpclient5-5.2.1.jar
slf4j-api-1.7.36.jar
httpcore5-h2-5.2.jar

Processing ICAL and xCAL calendars

The following program illustrates how to scan email-box for emails with attachment's content type is "text/calendar" and transforms into xCal, an XML based standard format which is easy to manipulate in XQuery.

```
declare namespace jcal = "ddtekjava:com.ivitechnologies.pipeline.ext.ical.ICALParser";
declare namespace es="ddtekjava:com.ivitechnologies.pipeline.ext.email.EmailSession";
declare namespace ems="ddtekjava:com.ivitechnologies.pipeline.ext.email.Emails";
declare namespace em="ddtekjava:com.ivitechnologies.pipeline.ext.email.Email";

declare function es:EmailSession($settings as element()) as ddtek:javaObject external;
declare function es:readEmails($this as ddtek:javaObject, $folder as xs:string, $subjectFilter as xs:string,
$fromFilter as xs:string, $dateFilter as xs:string) as ddtek:javaObject external;
declare function ems:size($this as ddtek:javaObject) as xs:int external;
declare function ems:get($this as ddtek:javaObject, $index as xs:int) as ddtek:javaObject external;
declare function em:Email($sender as xs:string, $recipients as xs:string, $subject as xs:string, $body as
xs:string, $message_mime_type as xs:string, $atatchments as xs:string*) as ddtek:javaObject external;
declare function em:getParts($this as ddtek:javaObject) as document-node()? external;
declare function em:getPartsCount($this as ddtek:javaObject) as xs:int external;
declare function jcal:convert_ICAL_to_xCAL($ICAL as xs:string) as document-node() external;

declare option ddtek:serialize "method=xml,indent=yes";

let $settings := <email mail.imap.host="" mail.imap.port="" mail.imap.user=""  mail.imap.password=""
mail.imap.ssl.enable="true"/>

let $session := es:EmailSession($settings)
let $emailFolder := "inbox"
let $subjectFilter := "Calendar Test "
let $fromFilter := ""
let $receivedFilter := ""
let $emails := es:readEmails($session, $emailFolder, $subjectFilter, $fromFilter, $receivedFilter)
return
for $i in 0 to ems:size($emails) - 1
,$email in ems:get($emails,xs:int($i))
,$ical in
em:getParts($email)/parts/part[contains(string(contentType),'text/calendar')]/content_as_text/string()
return
  jcal:convert_ICAL_to_xCAL($ical)
```

To run the above query in Stylus Studio, make sure that the Project Classpath includes the following server jar files from the servlet lib folder

xmlpipelineserver.jar
biweekly-0.6.3.jar
vinnie-2.0.2.jar
jackson-core-2.9.7.jar
jackson-databind-2.9.7.jar

PGP Encryption Operations

The following query shows how to encrypt a file using a PGP key

```
declare namespace pgp = "ddtekjava:com.ivitechnologies.pipeline.ext.security.PGP";

declare function pgp:encryptFile(
   $encryptedOutputURL as xs:string,$inputURL as xs:string,$publicKeyURL as xs:string,
    $armor as xs:boolean, $withIntegrityCheck as xs:boolean) as xs:boolean external;

declare variable $encryptedOutputURL :=
   fn:resolve-uri("out/books_encrypted.xml", fn:static-base-uri());

declare variable $inputURL := fn:document-uri(.);
declare variable $publicKeyURL := fn:resolve-uri("../../pub.bpg", fn:static-base-uri());
declare variable $armor := fn:false();
declare variable $withIntegrityCheck := fn:false();

pgp:encryptFile($encryptedOutputURL,$inputURL,$publicKeyURL,$armor,$withIntegrityCheck)
```

Here the opposite operation to decrypt a file using PGP private key

```
declare namespace pgp = "ddtekjava:com.ivitechnologies.pipeline.ext.security.PGP";

declare function pgp:decryptFile(
    $encryptedInputURL as xs:string, $decryptedOutputURL as xs:string,
    $passphrase as xs:string,  $privateKeyURL as xs:string) as xs:boolean external;

declare variable $encryptedInputURL :=
   fn:resolve-uri("out/books_encrypted.xml", fn:static-base-uri());

declare variable $decryptedOutputURL :=
   fn:resolve-uri("out/books_decrypted.xml", fn:static-base-uri());

declare variable $passphrase := "secret!";

declare variable $privateKeyURL :=
   fn:resolve-uri("../../secret.bpg", fn:static-base-uri());

pgp:decryptFile($encryptedInputURL,$decryptedOutputURL,$passphrase,$privateKeyURL)
```

In order to run the above query in Stylus Studio make sure that the Project Classpath includes the following server jar files from the servlet lib folder

xmlpipelineserver.jar
bcmail-jdk15on-152.jar
bcpg-jdk15on-152.jar
bcpkix-jdk15on-152.jar
bcprov-jdk15on-152.jar

Nonce Encryption

Nonce Authentication uses a one-time signature that is valid only for a web service specific request and for a brief period. The following query shows how to create a Nonce signature

```
declare namespace nonce = "ddtekjava:com.ivitechnologies.pipeline.ext.security.Nonce";

declare function nonce:calculateSignature($value as xs:string, $seed as xs:string) as xs:string
external;

let $value_to_encode := "XML Pipeline Server"
let $signature_seed  := "001-002-003-003"
let $value_encoded   := nonce:calculateSignature($value_to_encode, $signature_seed )
return $value_encoded
```

To run the above query in Stylus Studio, make sure that the Project Classpath includes the following server jar files from the servlet lib folder

xmlpipelineserver.jar
commons-codec-1.9.jar

RSA Encryption and Decryption

RSA extension function can be used for encryption/decryption of strings.

Function encryptString returns a base-64 encoded string. PublicKeyURL is expected to point to a local file containing the RSA public key

function decryptString expects a base-64 encoded string as source. PrivateKeyURL is expected to point to a local file containing the RSA private key

```
declare namespace rsa = "ddtekjava:com.ivitechnologies.pipeline.ext.security.RSA";
declare function rsa:encryptString( $source as xs:string, $publicKeyURL as xs:string) as
xs:string external;
declare function rsa:decryptString( $encryptedSourceBase64 as xs:string, $privateKeyURL as
xs:string) as xs:string external;


declare variable $string := "Hello World";
declare variable $publicKeyURL := fn:resolve-uri("public.pem", fn:static-base-uri());
declare variable $privateKeyURL := fn:resolve-uri("private.pem", fn:static-base-uri());
declare variable $encrypted := rsa:encryptString($string,$publicKeyURL);
declare variable $decrypted := rsa:decryptString($encrypted,$privateKeyURL);

declare option ddtek:serialize "indent=yes";

<root>
 <source_string>{$string}</source_string>
 <encrypted>{$encrypted}</encrypted>
 <decrypted>{$decrypted}</decrypted>
</root>
```

To run the above query in Stylus Studio, make sure that the Project Classpath includes the following server jar files from the servlet lib folder

xmlpipelineserver.jar
json-20240303.jar
bcpg-jdk18on-1.79.jar
bcprov-jdk18on-1.79.jar
bcutil-jdk18on-1.79.jar
bctls-jdk18on-1.79.jar
bcpkix-jdk18on-1.79.jar

AES Encryption and Decryption

EAS extension function can be used for encryption/decryption of strings.

Function encryptAESString returns a base-64 encoded string. privateKeyURL is expected to point to a local file containing the private key.

function decryptAESString expects a base-64 encoded string as source. PrivateKeyURL is expected to point to a local file containing the private key

```
declare namespace rsa = "ddtekjava:com.ivitechnologies.pipeline.ext.security.RSA";

declare function rsa:decryptAESString( $encryptedSourceBase64 as xs:string, $privateKeyURL as
xs:string) as xs:string external;
declare function rsa:encryptAESString( $encryptedSourceBase64 as xs:string, $privateKeyURL as
xs:string) as xs:string external;

declare variable $string := "Hello World";
declare variable $privateKeyURL := fn:resolve-uri("private-key.txt", fn:static-base-uri());
declare variable $encrypted := rsa:encryptAESString($string,$privateKeyURL);
declare variable $decrypted := rsa:decryptAESString($encrypted,$privateKeyURL);

declare option ddtek:serialize "indent=yes";

<root>
 <source_string>{$string}</source_string>
 <encrypted>{$encrypted}</encrypted>
 <decrypted>{$decrypted}</decrypted>
</root>
```

T run the above query in Stylus Studio, make sure that the Project Classpath includes the following server jar files from the servlet lib folder

xmlpipelineserver.jar
json-20240303.jar
bcpg-jdk18on-1.79.jar
bcprov-jdk18on-1.79.jar
bcutil-jdk18on-1.79.jar
bctls-jdk18on-1.79.jar
bcpkix-jdk18on-1.79.jar

JSONWebToken

JSON Web Token is a JSON-based open standard for creating access tokens that assert some number of claims more information at https://jwt.io/.

The following program shows how to create a JSONWebToken using server extension function passing the header and body as XML elements and how to parse the resulting base64 signature as XML.

Public and private keys must be RSA256 format.

If IAT is missing the function defaults it to current time.
If EXP is missing the function defaults to 1 hour from the current time.

```
declare namespace jwt = "ddtekjava:com.ivitechnologies.pipeline.ext.security.JSONWebToken";


declare function jwt:createToken( $header as element(), $body as element(), $privateKeyURL as
xs:string) as xs:string external;

declare function jwt:parseToken( $token as xs:string, $publicKeyURL as xs:string, $privateKeyURL
as xs:string) as element() external;


declare variable $publicKeyURL  := fn:resolve-uri("public.pem", fn:static-base-uri());
declare variable $privateKeyURL := fn:resolve-uri("private.pem", fn:static-base-uri());
declare variable $header := <root typ='JWT' alg='RS256'/>;
declare variable $body := <root iss='iss' sub='sub' exp='exp' aud='aud' scope='scope'/>;

declare variable $token := jwt:createToken( $header, $body, $privateKeyURL);

declare option ddtek:serialize "indent=yes";

jwt:parseToken($token, $publicKeyURL, $privateKeyURL)
```

To run the above query in Stylus Studio, make sure that the Project Classpath includes the following server jar files from the servlet lib folder

xmlpipelineserver.jar	objenesis-2.6.jar
bcmail-jdk15on-152.jar	mockito-core-2.18.3.jar
bcpg-jdk15on-152.jar	jackson-annotations-2.9.0.jar
bcpkix-jdk15on-152.jar	jackson-core-2.9.7.jar
bcprov-jdk15on-152.jar	jackson-databind-2.9.7.jar
commons-codec-1.9.jar	java-hamcrest-2.0.0.0.jar
	java-jwt-0.0.1-SNAPSHOT.jar

XML Schema Validation

Class XSDValidation exposes high performance XML Schema validation. The function returns the validation report as XML document which contains all validation errors found. If parameter “ReportErroLocation” is set to true, the line and column where the error was detected are reported.

Parameter “bUseSchemaCahe” instructs the schema processor to cache the Schema object which speeds up the validation. If the schema URL is file based the schema processor can detect a change in the last modification date and reloads the schema automatically even if, bUseSchemaCache is set to true.

```
declare namespace xsdv = "ddtekjava:com.ivitechnologies.pipeline.ext.xml.XSDValidation";

declare function xsdv:validate(
  $node as node(), $schema_uri as xs:string,
  $bReportErroLocation as xs:boolean, $bUseCache as xs:boolean) as node() external;

declare variable $schema_url := resolve-uri("books.xsd", fn:static-base-uri());
declare variable $bReportErroLocation as xs:boolean :=fn:true();
declare variable $bUseSchemaCache as xs:boolean :=fn:true();
```

```
declare variable $validationReport := xsdv:validate(/, $schema_url, $bReportErroLocation, $bUseSchemaCache);


if($validationReport/validation-result/error) then
 ddtek:serialize-to-url($validationReport, $validatioReportURL, "indent=yes")
else ()
```

To run the above query in Stylus Studio, make sure that the Project Classpath includes the following server jar files from the servlet lib folder

xmlpipelineserver.jar

The validation report is structured as follow

```
<validation-result>
    <error>
        <message>cvc-complex-type.2.4.b: The content of element 'book' is not complete. One of '{title}' is expected.</message>
        <systemId/>
        <publicId/>
        <lineNumber>3</lineNumber>
        <columnNumber>9</columnNumber>
    </error>
</validation-result>
```

Checking for Well-formed XML

The following example shows how to check if the query input is well-formed XML using extension function isWellFormed.

Notice that you must bind the query input to an external variable with type xs:string.

```
declare namespace xsdv = "ddtekjava:com.ivitechnologies.pipeline.ext.xml.XSDValidation" ;
declare function xsdv:isWellFormed($xml as xs:string) as document-node() external;

declare variable $data as xs:string external;

xsdv:isWellFormed($data)
```

When the above query external variable “data” is bound to text “<root>” which is not a well-formed XML, the result is the following.

```
<result wellFormed="false">
    <error column="7" line="1">XML document structures must start and end within the same entity.</error>
</result>
```

ICalender

ICalender extension functions allows to convert from XCalender to ICalender format and vice versa.

```
declare namespace jcal = "ddtekjava:com.ivitechnologies.pipeline.ext.ical.ICALParser";
declare namespace es="ddtekjava:com.ivitechnologies.pipeline.ext.email.EmailSession";
declare namespace ems="ddtekjava:com.ivitechnologies.pipeline.ext.email.Emails";
declare namespace em="ddtekjava:com.ivitechnologies.pipeline.ext.email.Email";

declare function es:EmailSession($settings as element()) as ddtek:javaObject external;
declare function es:readEmails($this as ddtek:javaObject, $folder as xs:string, $subjectFilter as
```

```
xs:string, $fromFilter as xs:string, $dateFilter as xs:string) as ddtek:javaObject external;
declare function ems:size($this as ddtek:javaObject) as xs:int external;
declare function ems:get($this as ddtek:javaObject, $index as xs:int) as ddtek:javaObject
external;
declare function em:Email($sender as xs:string, $recipients as xs:string, $subject as xs:string,
$body as xs:string, $message_mime_type as xs:string, $atatchments as xs:string*) as
ddtek:javaObject external;
declare function em:getParts($this as ddtek:javaObject) as document-node()? external;
declare function em:getPartsCount($this as ddtek:javaObject) as xs:int external;
declare function jcal:convert_ICAL_to_xCAL($ICAL as xs:string) as document-node() external;
declare function jcal:convert_xCAL_to_iCAL($XCAL as document-node()) as xs:string external;


declare option ddtek:serialize "method=xml,indent=yes";

let $settings := <email
  mail.imap.host=" "
  mail.imap.port="993"
  mail.imap.user=" "
  mail.imap.password=" "
  mail.imap.ssl.enable="true"/>

let $session := es:EmailSession($settings)
let $emailFolder := "inbox"
let $subjectFilter := "TEST - Force 5 Calendar Test for Pipeline Server ICAL integration"
let $fromFilter := ""
let $receivedFilter := ""
let $emails := es:readEmails($session, $emailFolder, $subjectFilter, $fromFilter,
$receivedFilter)
return
 for $i in 0 to ems:size($emails) - 1
               ,$email in ems:get($emails,xs:int($i))
               ,$ical in em:getParts($email)/parts/part[contains(string(contentType),
'text/calendar')]/content_as_text/string()
 return
  jcal:convert_ICAL_to_xCAL($ical)
```

To run the above query in Stylus Studio, make sure that the Project Classpath includes the following server jar files from the servlet lib folder

xmlpipelineserver.jar
biweekly-0.6.3.jar

SQLCommand

SQLCommand.exeeute is a function extension designed to execute arbitrary SQL to any JDBC driver

SQLCommand can be used for:
- Querying database that are not directly supported by the XQuery engine
- Executing DDL instructions like CREATE TABLE, INSERT, etc.
- Call store procedures

The result set returned by the SQL is transformed to XML according to the SQLXML mapping specification. For example, the following statement the output is the XML below

SELET * FROM USERS

```
<table>
 <row>
  <id>393</id>
```

```
  <first>IVAN</first>
  <last>PEDRUZZI</last>
 </row>
</table>
```

If the SQL statement is an update the output reports the number of rows affected.

INSERT INTO TEST_TABLE (id) VALUES (0);

```
<table>
 <row>
  <rows_affected>1</rows_affected>
 </row>
</table>
```

The function takes 1 parameter which is profile in XML format, here an example

profile.xml----------

```
<SQLCommand
 driverClass="com.mysql.jdbc.Driver"
 user="Admin"
 password="Admin"
 url="jdbc:mysql://localhost:3306/mydatabase">
 <SQL>SELECT * FROM user</SQL>
</SQLCommand>
```

Profile can be provided as URL or as XML document. The following XQuery example shows how to invoke SQLCommand.

```
declare namespace ex="ddtekjava:com.ivitechnologies.pipeline.ext.jdbc.SQLCommand";

declare function ex:execute($profile as xs:anyURI) as document-node() external;
declare function ex:execute($profile as document-node()) as document-node() external;

declare variable $profile := fn:resolve-uri("profile.xml", fn:static-base-uri());

declare option ddtek:serialize "indent=yes";

ex:execute($profile)
```

To run the above query in Stylus Studio, make sure that the Project Classpath includes the following server jar files from the servlet lib folder

xmlpipelineserver.jar

Encoding and Decoding text from and to QR Code

Sometimes is useful to encode information into QR code images that can be easily acquire by mobile applications.

The following example shows how to encode text into a QR code image

```
declare namespace qr="ddtekjava:com.ivitechnologies.pipeline.ext.qrcode.QrCodeManager"

declare function qr:encode_qrcode($value as xs:string, $bCompress as xs:boolean,
 $imageFormat as xs:string, $outputUrl as xs:string) as xs:boolean external;
declare function qr:encode_qrcode_base64($value as xs:string, $bCompress as xs:boolean,
 $imageFormat as xs:string) as xs:string external;
declare function qr:decode_qrcode_base64($qrcodeAsBase64 as xs:string, $bDecompress as xs:boolean) as xs:string external;

declare variable $value := "MIDWAY upon the journey of our life
I found myself within a forest dark,
For the straightforward pathway had been lost.";

declare variable $bCompress := fn:true();
declare variable $imageFormat := "png";
declare variable $qrcode_image_url := fn:resolve-uri("qrcode.png", fn:static-base-uri());

(:Generate QR code as base-64:)
let $qrcodeAsBase64 := qr:encode_qrcode_base64($value, $bCompress, $imageFormat)

return (
  (:Generate QR code to file:)
  if(qr:encode_qrcode($value, $bCompress, $imageFormat, $qrcode_image_url) = fn:false())
  then fn:error((), "encode_qrcode failed") else ()

  (:decode QR code from base-64:)
  ,qr:decode_qrcode_base64($qrcodeAsBase64, $bCompress)
)
```

the image generated “qrcode.png” looks like this

The following example shows how to decode QR code image into text

```
declare namespace qr="ddtekjava:com.ivitechnologies.pipeline.ext.qrcode.QrCodeManager";
declare function qr:decode_qrcode_base64($qrcodeAsBase64 as xs:string, $bDecompress as xs:boolean) as xs:string
external;

declare variable $bCompress := fn:true();
declare variable $qrcode_image_url := fn:resolve-uri("qrcode.png", fn:static-base-uri());
declare variable $url := concat("converter:base64?", $qrcode_image_url);
declare variable $qrcodeAsBase64 := fn:replace(doc($url),"&#10;", "");
```

qr:decode_qrcode_base64($qrcodeAsBase64, $bCompress)

LDAP extension function

LDAP server can also be query using extension function as follows:

```
declare namespace ldap = "ddtekjava:com.ivitechnologies.pipeline.ext.ldap.LDAPSession";
declare function ldap:search_xmlreader($settings as element()) as document-node() external;
declare option ddtek:serialize "indent=yes";
declare variable $settings :=
 <port
        url="ldap://localhost:10389"
        dn="dc=example,dc=com"
        Context.SECURITY_AUTHENTICATION="simple"
        Context.SECURITY_PRINCIPAL="uid=admin,ou=system"
        Context.SECURITY_CREDENTIALS="secret"
        property_list_to_omit_separated_by_comma="objectGUID,objectSid"
        objectClass="*"
        read_timeout_in_milliseconds="1000"
        connection_timeout_in_milliseconds="5000"
        sortBy="cn"
        paginationTrueFalse="true"
        pageSize="500"
      />;

ldap:search_xmlreader($settings)
```

Microsoft Graph extension function

Microsoft Graph can also be query using extension function as follows:

```
declare namespace msg =
"ddtekjava:com.ivitechnologies.pipeline.ext.microsoftgraph.MicrosoftGraph";

declare function msg:search_xmlreader($element as element()) as document-node() external;

declare variable $settings := doc($url)/root;
declare variable $msgraph  := msg:search_xmlreader($settings);

declare option ddtek:serialize "indent=yes";

$msgraph
```

Running XSLT from XQuery

Starting with build 244, it's possible to execute XSLT transformations from XQuery. The following example illustrates extension function transformToString can be used to generate HTML using XSLT to populate an email. Check the most recent xslt_lib XQuery module for the list of functions available.

<xps folder>xquery_lib\xps\xslt\xslt_lib.xquery

If the XSLT URL is file based, the XSLT will be automatically cached and reload when the file last modified date changes.

```
import module namespace xps_xslt = "ddtekjava:com.ivitechnologies.pipeline.ext.xslt.XSLT" at
"xslt_lib.xquery";
import module namespace em = "ddtekjava:com.ivitechnologies.pipeline.ext.email.Email" at
"email_message_lib.xquery";
import module namespace es = "ddtekjava:com.ivitechnologies.pipeline.ext.email.EmailSession" at
"email_session_lib.xquery";

declare variable $xslt := fn:resolve-uri("html.xsl" , fn:static-base-uri());
declare variable $input := /;


let $settings := doc(fn:resolve-uri("email.xml" , fn:static-base-uri()))
let $session := es:EmailSession($settings/email)
let $from := xs:string($settings/email/@mail.smtp.user)
let $to := $from
let $subject := xs:string($settings/email/@subject)
let $message := xps_xslt:transformToString($xslt, $input, ())
let $email := em:Email($from, $to, $subject, $message, "html", ())
return es:sendEmail($session, $email)
```

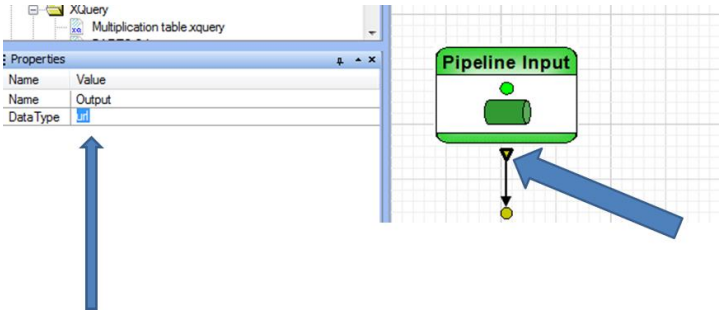
To run the above query in Stylus Studio, make sure that the Project Classpath includes the following server jar files from the servlet lib folder

xmlpipelineserver.jar
saxon-ee-12.5.jar
xmlresolver-5.2.2.jar
xmlresolver-5.2.2-data.jar
json-20240303.jar

Data Types, File Types

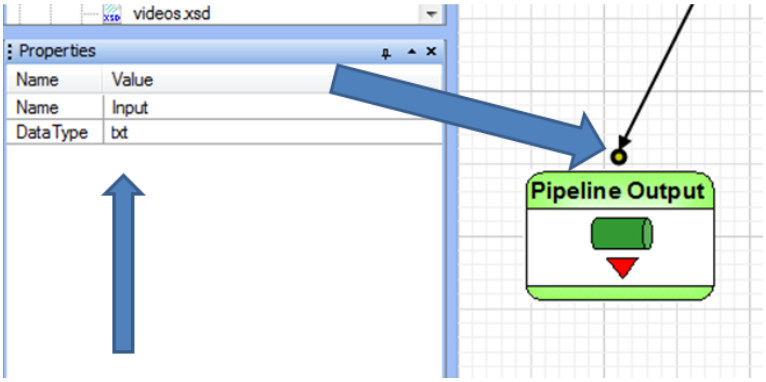
By default an input port reads the data from the end-point and passes it to the operation linked to it.

In some cases it is useful to pass the URL where the data is located and let the operation do the reading. To enable this, set the DataType property on the output handle of your Input port to "url". See this in the following screenshot in the Stylus Studio Pipeline Designer:



Change Output File Extensions

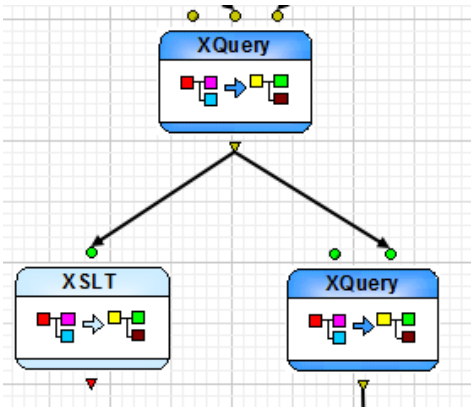
You can force the file extension for a given output port by simply setting the DataType property on the input handle of your output port. In the following example, you can see how to set the output file extension to be always .txt:



Parallel Execution and Streaming

XML Pipeline Server has been designed to execute tasks in parallel and streaming data from one operation to another. XML Pipeline Server can truly take advantage of multicore systems.

When a pipeline branches, the source operation broadcasts the data to the receiving target operations which run on separate threads. This allows scaling performance linearly when the load can be distributed on separate cores.



XML Pipeline Server locks the processing until the output ports consume the data in their caches. This helps to avoid overloading the system for excessive throughput.

Server Information

The HTTP server interface exposes the following open endpoints:

`/rest/version` returns the product version

`/rest/ping` returns OK if the server is running.

`/rest/serverInfo` returns the license information.

Server Configuration

Logging

By default, the logging is set to finer granularity (level = ALL). The logging level can be changed editing the attribute "level" in the server configuration (server.xml).

The log files are organized in rotating set, by default the set is made two files, 10 Mb each. You can change policy using the properties maximum_number_of_log_files and maximum_size_in_bytes_for_log_file.

Passwords for connecting to SFTP and FTP are by default masked but if necessary, can be displayed in clear.

In addition, you can setup an email notification which will trigger an email for each server log entry. The email notification settings have an optional attribute "level" which is useful if you want notifications only for few events like SEVERE.

To prevent sending error email notification too frequently, set property buffer_errors_before_t_in_seconds to a value greater than 0. For example, when the value is set to 60, email notifications will not be sent more frequently than 60 seconds. The errors occurring inside the interval will be accumulated and send with the first email after the interval expires.

Use property max_errors_buffered to configure the maximum number of errors buffered in memory. The default and minimal value are 10, the maximum value is 1,000.

```
<logging
level="INFO"
hide_passwords_from_log="true"
maximum_number_of_log_files="1"
maximum_size_in_bytes_for_log_file="10048576">

        <!--
        Level can bet set to the following values

        SEVERE          highest value logs only severe errors
        WARNING
        INFO
        CONFIG
        FINE
        FINER
```

```
            FINEST
            ALL                     lowest value logs everything [default]
            -->
            <!--<email
                  level="SEVERE"
                   mail.smtp.host="smtp.gmail.com"
                   mail.smtp.port="587"
                   mail.smtp.user=""
                   mail.smtp.password=""
                   mail.smtp.auth="true"
                   mail.smtp.starttls.enable="true"
                   mail.smtp.from=""
                   recipients=""
                   subject="XML Pipeline Server - Error"
                   buffer_errors_before_t_in_seconds="60"
                   max_errors_buffered="100"
                   send_email_n_retries="1"
                   send_email_wait_before_retry_in_milliseconds="10000"
            />-->
      </logging>
```

Thread Manager

The thread_manager allows to tune the thread pool manager behavior.

corePoolSize set the number of initial threads the pool manager creates.
maximumPoolSize set the maximum number of threads the manager can create.
keepAliveTimeInMilliseconds set number of milliseconds the thread stays alive after entering in idle state.

```
<thread_manager
      corePoolSize="0"
      maximumPoolSize="50"
      keepAliveTimeInMilliseconds="2000"/>
```

The timeOutInMilliseconds defines in the operation_runner set the number of milliseconds an operation will wait to obtain a thread from the pool after which the execution will be interrupted.

Such setting is useful to set how fast a pipeline should fail safe when running a scenario in which a large number of operations have been started and the thread pool is exhausted.

```
      <operation_runner timeOutInMilliseconds="60000"/>
```

Bridge Input Output Source

In some peculiar use-cases it may be necessary to increase the bridge data pipe buffer to improve processing speed. For example, processing large XML documents may gain speed by several folds. The default buffer size is 128 Kbytes.

```
     <bridge_output_input_source pipe_buffer_size_in_bytes="128000"/>
```

Pipeline Loader

The following element allows to configure the delay before loading a new pipeline. This is useful when copying/deploying large project and the pipeline file can be detected before all files referenced by it have been copied

```
<pipeline_loader pipeline_folder_inspection_delay_in_seconds="5"/>
```

File System Service

The following element configure the behavior for the file system service.

pullEventIntervalInMilliseconds configures the frequence to pull event from the File System Evet Queue

execute_pipeine_asynchronus allows to fire pipeline execution asynchronously as soon as the input file is ready without forcing sequencing. Default is off, execution is synchronous.

execute_pipeine_synchronous_timeout_millisecond allows to set the time out for synchronous execution

```
<file_system_service
 pullEventIntervalInMilliseconds="1000"
 execute_pipeine_asynchronus="false"
 execute_pipeine_synchronous_timeout_millisecond="-1"/>
```