# Contents

**Chapter 1: Building block basics**  
Building Block Structure ................................................................. 1

**Chapter 2: How the building block works**  
Batch data input ..................................................................................... 2
Batch processing ..................................................................................... 2
Batch data output ..................................................................................... 3
Example of process that generates interactive statements in batch ................................. 3

**Chapter 3: Implementing the user stories**  
Create a process for generating interactive statements in batch mode ............................... 5
Best practices ......................................................................................... 5

**Chapter 4: Understanding the building block assets**  
Building block services ........................................................................... 8

**Chapter 5: Troubleshooting**  
Batch Job Completion ........................................................................... 9
Chapter 1: Building block basics

The Batch Processor building block invokes an Adobe® LiveCycle® Enterprise Suite (ES2.5) service for each item in a large input data set. The Batch Processor building block provides the following features:

- Runs batch jobs.
- Distributes the processing over multiple threads on a server and across nodes in a cluster to increase processing throughput.
- Allows you to start and in the event of a failure, restart batch jobs.

Building Block Structure

The Batch Processor building block includes a component and API that lets you run batch jobs for generating interactive statements. The Batch Processor building block components are installed in the following folder:

C:\Adobe\Adobe LiveCycle ES2\sa_resources\SA_SDK_9.5\.

<table>
<thead>
<tr>
<th>Component or API and location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\BuildingBlocks\BatchProcessor\dsc\adobe-batchprocessor-dsc.jar</td>
<td>This building block processes large data sets in a batch.</td>
</tr>
<tr>
<td>\BuildingBlocks\BatchProcessor\lib\adobe-batchprocessor-client.jar</td>
<td>The client API used for invoking batch jobs from a remote Java client.</td>
</tr>
</tbody>
</table>
Chapter 2: How the building block works

The Batch Processor service reads input items from a data set that a business process generates externally. The service uses the values for each input item as parameters to invoke a LiveCycle service. That LiveCycle service performs processing specific to the application, such as document generation. The output parameters of the LiveCycle service are returned to the Batch Processor service where they can optionally act as inputs to another LiveCycle batch process or an external business process.

The Batch Processor service is a standard LiveCycle ES2.5 service. You can access the Batch Processor service in Adobe LiveCycle Workbench 9.5 or develop using the APIs to build a solution.

- For the operations in Workbench, see the Services reference in the LiveCycle Workbench 9.5 Help.
- For information about the Flex APIs, see the ActionScript 3.0 Reference for the Adobe Flash Platform.
- For information about the Java APIs, see the Solution Accelerators API Reference.

Batch processor service component

Batch data input

An external business process generates the batch data inputs from sources such as billing systems or customer databases. The inputs for batch processing can come from comma-delimited text files, XML files, or by way of JDBC queries to a database.

Batch processing

The Batch Processor service reads items from the batch data input and uses them as parameters for invoking a LiveCycle service, for example, a process that generates an interactive statement. The process generates one interactive statement for each item in the batch data input.
Batch data output

The parameters that are output from the batch processor service can optionally be written to an output file or database table. A processing step that runs after the batch job completes might consume the output file. For example, a service that sends the generated interactive statements to an email service for delivery. If you don’t need to write the batch output, you can omit the parameters.

Example of process that generates interactive statements in batch

A typical batch process is a single-step process that uses the Batch Processor service to invoke a LiveCycle service. In the following example process, the batch process reads each item in the batch data input and invokes a process that assembles an interactive statement. The batch data input is an XML file.

Here is an example of a simple batch XML file:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <item>
    <data>
      <accountHolder>
        <name>Akira Tanaka</name>
      </accountHolder>
    </data>
    <packageDefinition>
      <package locale="en_US" source="file://c:/Adobe/MyNavigator.nav" type="nav"
xmlns="http://ns.adobe.com/livecycle/icomm/assetplacement/model"/>
    </packageDefinition>
  </item>
  <item>
    <data>
      <accountHolder>
        <name>John Jacobs</name>
      </accountHolder>
    </data>
    <packageDefinition>
      <package locale="en_US" source="file://c:/Adobe/MyNavigator.nav" type="nav"
xmlns="http://ns.adobe.com/livecycle/icomm/assetplacement/model"/>
    </packageDefinition>
  </item>
</root>
```

Each item node results in one invocation of the process that assembles the interactive statement. The child nodes of item are transformed to the correct type and are passed as parameters to the service that assembles the interactive statement. For example, the child nodes data and packageDefinition are passed as document parameters.
For example, this batch input might be used to invoke a process that uses the Asset Placement service to generate a document. For each input item, the data and packageDefinition elements are transformed to XML document values that are passed to input process variables with the same name. When the Asset Placement service operation is invoked, it uses these process variables as inputs. The operation generates a document that is used to set the value of the assembleStatement process variable. This process might use this generated document in additional process steps, such as writing it to a file, and/or it can be returned as an output parameter that can be written with the batch output item. For more information about the Batch Processor service, see the Services reference in the LiveCycle Workbench 9.5 Help.
Chapter 3: Implementing the user stories

You can implement elements of the following stories using the Batch Processor building block.

Create a process for generating interactive statements in batch mode

Typically, a process developer creates a batch process that invokes a LiveCycle process that generates an interactive statement.

Create a batch process

- Create an input data set that defines the input parameters for the interactive statement you want to generate in a job run.
- Set the identification property for the job run. For example, enter: Run batch #1 for the east coast on Thursdays.
- Set the service operation properties.
- Set the static service parameter properties. For example, static parameters values that are the same for each item in the batch. For example, batchScopeId specifies cache for a specific batch job.
- Set the batch input and output properties.
- Set the optimization properties. For example, how to partition the job into smaller pieces that can be processed in parallel.
- Create a LiveCycle process for the Batch Process to invoke.

Best practices

Keep the following best practices in mind when creating batch processes.

General

The Batch Processor service manages a series of short transactions that can be completed within the transaction timeout interval. When invoking a batch process:

- Invoke the Batch Processor component within a long-lived process.
- Set the transaction propagation setting to NEVER for the process invoking the Batch Processor service. Make sure that the transaction setting for the long-lived process does not cause a transaction to start. For large input sources, any transaction is likely to time out before the Batch Processor invocation completes. In Workbench, you configure the propagation properties for the process invoked from the Batch Processor service. For the operations in Workbench, see the Services reference in the LiveCycle Workbench 9.5 Help.
Transaction configuration of invoked services
The transaction configuration of the service being invoked affects the performance and correct operation of the batch process:

- Make sure that the service that the Batch Processor service invokes is short lived. The Batch Processor service always invokes the service synchronously.
- Make sure that the service that the Batch Processor service invokes participates in the enclosing transaction and that it does not start a new transaction. In Workbench, you configure the transaction in properties for the process invoked from the Batch Processor service. For more information about the Batch Processor service, see the Services reference in the LiveCycle Workbench 9.5 Help.

Make sure that any processing done in the service that the Batch Processor service invokes is transactional. This ensures that any processing done by the service is coordinated with the state tracking that the Batch Processor manages. It also ensures a clean restart of a partially completed batch job. If the processing writes a document to the file system, the file system write is not transactional. This is not a problem. If a restart overwrites an existing file (for example, without adding a unique suffix).

Global transactions
The Batch Processor service uses global (XA) transactions to ensure that the persistent state is coordinated between the LiveCycle persistent storage, the batch persistent storage, as well as with any transaction batch input and output data sources, for example, JDBC. Follow these guidelines for global transactions:

- Make sure that the application server is configured to support global transactions.
- Typically, application servers allow one non-XA resource to participate in a global transaction. If the LiveCycle IDP_DS data source uses the default configuration, the resource participating in the transaction is treated as the one non-XA resource allowed. All other resources must be capable of XA transactions.
- Configure the Batch Processor service to use an XA enabled data source. On some application servers (for example, WebSphere), the database connection to the LiveCycle database is not automatically shared between the Batch Processor and internal LiveCycle processing. This forces the use of XA transactions for all invocations of the Batch Processor service. In this case, you need to configure the Batch Processor service to use a copy of the IDP_DS data source that uses an XA-capable JDBC driver. The solution accelerator install creates the AdobeDefaultSA_DS data source that you can use for this purpose.
- Make sure that JDBC batch input or output sources are capable of XA transactions.

Performance considerations
When a service is being called repeatedly, you can choose to cache data across invocations. Implementing a caching system is important when you plan to process many items. The Batch Processor assists with creating a caching system by automatically providing a value for an input parameter named batchScopeId to any service it calls. The value of this parameter is the same for each item within a batch job, and different between batch jobs. For example, the Asset Placement service in the interactive statements solution defines this parameter so that it caches content across the items processed within a batch job.

When running a batch job that uses the Asset Placement service, specify the Document template argument in the package definition (not as a literal parameter to Asset Placement). This allows the Asset Placement service to cache the document template document. See “Reference Information” in the Personalization Manager Building Block 9.5 Technical Guide.
Partitioned Processing

When processing large number of batch items, the input items can be partitioned into smaller subsets that are processed in parallel. The processing of partitions is automatically distributed across available threads within the server, and if the server is part of a cluster, across nodes in the cluster.

Partitioned processing can allow the batch job to complete processing much faster than if it was run without partitioning. However, partitioning a job will result in higher usage of processing resources while the job is running, which could potentially interfere with other processing taking place in the server. The choice of whether to use partitioning will depend on the volume of items to process and the intended use of the server.

For example, if the primary use for a server is to provide high-volume batch processing, the partitioning strategy used would typically be to use all available threads/nodes for batch processing. In this case, the partitioning strategy should be set to "The input items are split into a specified number of partitions" and the partitioning strategy parameters might be the number of processors available on the server (or across the cluster).

If the LiveCycle server is intended to handle other requests concurrently with batch processing, the partitioning strategy can be configured to use fewer partitions (or no partitioning at all) so that resources are available to process those requests.
Chapter 4: Understanding the building block assets

The Batch Processor building block provides the following assets that you use to build a solution:

- Batch Processor service and associated operations available in Workbench
- Flex and Java APIs

For additional information about the Batch Processor service, see these resources:

- For the operations in Workbench, see the Services reference in the LiveCycle Workbench 9.5 Help.
- For information about the Flex APIs, see the ActionScript 3.0 Reference for the Adobe Flash Platform.
- For information about the Java APIs, see the Solution Accelerators API Reference.

Building block services

The Batch Processor building block includes the Batch Processor service. It invokes a LiveCycle service for each item in a large input data set.

Batch Processor service

The Batch Processor service implements batch processing. Each operation implements a different type of input and output for the batch items. Four operations are available for batch processing:

- Run Flat File Job (runFlatFileJob)
- Run JDBC Cursor Job (runJDBCCursorJob)
- Run JDBC Paging Job (runJDBCPagingJob)
- Run XML File Job (runXmlFileJob)

The runJDBCCursorJob operation is more efficient than the runJDBCPagingJob operation and supports selection using a stored procedure as well as SQL statements, and we recommend its use. However, if you are using a transactional data source with some combinations of the WebLogic Server or WebSphere Application Server and JDBC drivers, the runJDBCCursorJob operation is not supported. Instead, use the runJDBCPagingJob operation. See the Services reference in the LiveCycle Workbench 9.5 Help.
Chapter 5: Troubleshooting

You can use the *LiveCycle ES2.5 Error Code Reference* to troubleshoot errors you may encounter when working with the Interactive Statements Solution Accelerator. Errors may be written to log files when you are installing, configuring, or running LiveCycle ES2.5. For each error code, there is a probable cause and an action you can take to resolve the error. Error codes are grouped according to component or activity. See the *LiveCycle ES2.5 Error Code Reference*.

You can also consult the guide *Troubleshooting LiveCycle ES2.5* for information about how to troubleshoot installation, configuration and administration issues that may arise within a LiveCycle ES(ES2.5) production environment. See *Troubleshooting LiveCycle ES2.5*.

**Batch Job Completion**

Each Batch Processor operation returns a string value that indicates whether the job completed successfully. A batch job can terminate processing early if a failure that it cannot recover from occurs, or if the operator explicitly interrupts the job.

For a large job run that takes a long time to complete, running the job from the start may not be practical due to time constraints, or because processing the same input items again causes duplicated processing. The work done by a batch job is tracked as it is executed, so it may be possible to restart the job and have it automatically resume processing from where the previous partially complete job run ended.

Restarting a job is done by executing the job using the identical set of parameters that were used for some previous partial job run. The Batch Processor operation will automatically continue processing starting after the last chunk of inputs items that were known to have been successfully processed by the previous partial job run.

The following table describes the possible result codes that can be returned by a batch job and what each outcome means.

<table>
<thead>
<tr>
<th>Batch Processing Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPLETED</td>
<td>The job finished processing successfully.</td>
</tr>
<tr>
<td>FAILED</td>
<td>The job finished processing with an error. The system log can be consulted to determine the cause of the failure. If the cause for the job failure has been resolved (e.g., a required resource that was temporarily unavailable is available again), then it may be possible to restart the job.</td>
</tr>
<tr>
<td>STOPPED</td>
<td>The job was interrupted by the operator. The job is in a known state and it may be possible to restart the job.</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>The job failed, and it is in an unknown state. Restarting the job is not safe since it cannot be determined whether all input items will in fact be processed. The system log can be consulted to determine the cause of failure. A job in this state will typically require manual intervention (e.g., clearing any partially completed outputs), and starting the job again with a different run identifier.</td>
</tr>
<tr>
<td>Batch Processing Result</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>ALREADY_RUNNING</td>
<td>Another instance of the same job with identical parameter values is already running. The attempt to run a duplicate job was ignored.</td>
</tr>
<tr>
<td>RESTART_FAILED</td>
<td>A job instance with identical parameter values has already finished in an UNKNOWN state. The attempt to restart the job was rejected because it is not safe to continue processing.</td>
</tr>
<tr>
<td>ALREADY_COMPLETE</td>
<td>A job instance with identical parameter values has already finished in a COMPLETE state. The attempt to run a duplicate job was ignored.</td>
</tr>
</tbody>
</table>