Adobe

Batch Sequences

1/27/21 Adobe, Inc.

Contents

List of Examples

1 Using Batch Sequences

A batch sequence performs repetitive operations on a collection of *selected files*. Batch processing sequences, like all JavaScript code, are executed as a result of a particular event. During the event, the script is repeatedly executed on each of the selected files.

This chapter addresses some of the principles of writing a batch sequence that uses the Execute JavaScript batch command. With these principles in mind, and a knowledge of the JavaScript API for Acrobat DC, you will be able to write your own batch sequences.

Creating and running a batch sequence

A batch sequence is created through the UI of Acrobat Pro DC. In version 8, the batch sequence feature is accessed by selecting Advanced > Document Processing > Batch Sequences. The following batch sequence files are shipped with Acrobat Pro DC:

```
Embed Page Thumbnails.sequ
Open All.sequ
Print All.sequ
Save All as RTF.sequ
Fast Web View.sequ
Print 1st Page of All.sequ
Remove File Attachments.sequ
Set Security to No Changes.sequ
```

The batch sequence is saved to a text file with an extension of `sequ`. Acrobat Help covers how to use the batch processing feature, and, in particular, how to use these sequences.

You are more interested in writing your own, so let's walk through the creation of a simple, yet useful, batch sequence.

Task: For each of the selected files, set the `disclosed` property to `true`. (The `disclosed` property of the Doc object is very important to certain document processing operations. When `disclosed` is set to `true`, the `app.activeDocs` and `app.openDoc` methods return Doc objects rather than `null`. See the JavaScript for Acrobat API Reference for documentation on these two methods.)

The procedure below steps you through the creation of a batch sequence. The sequence inserts the script `this.disclosed = true` in each of the selected documents as document JavaScript.

Create a batch sequence that sets the disclosed property to true:

1. Select Tools Pane > Action Wizard > Create New Action.
2. Click New Sequence, and type Make disclosed in the Name Sequence dialog box.
3. In the Edit Batch Sequence dialog box, click the Select Command button to get to the Edit Sequence dialog box.
4. In the Edit Sequence dialog box, double-click Execute JavaScript to move it to the right side.

5. Click the Edit button to open the JavaScript editor.

6. Type the following code:

```
/* Make disclosed */
this.addScript("This Doc is Disclosed", "this.disclosed = true;");
```

7. Close all the dialog boxes.

`Make disclosed` now appears in the list of actions.

To run this new script you must first select the collection of files that the batch sequence will process. The general procedure below assumes the Edit Sequence dialog box is not open. Follow the procedure for the new Make disclosed batch sequence.

To select files:

1. Choose Tools Pane > Action Wizard.

2. In the Batch Sequences dialog box, highlight one of the listed sequences, and click Edit Sequence.

3. In the Edit Batch Sequence dialog box, select one of the following items from the Run Commands On list:
   - Selected Files
   - Selected Folders
   - Ask When Sequence is Run
   - Files Open in Acrobat

4. Close all the dialog boxes.

Finally, run your new batch sequence.

To run a batch sequence:

1. Choose **Advanced** > **Document Processing** > **Batch Processing**.

2. In the **Batch Sequences** dialog box, highlight one of the listed sequences, and click **Edit Sequence**.

3. In the **Edit Batch Sequence** dialog box, verify that the files selected are the ones you want to process. Click **OK**.

4. In the **Batch Sequences** dialog box, click **Run Sequence**. In the **Run Sequence Confirmation** dialog box, click **OK**.

5. After processing is over, close the **Batch Sequences** dialog box.

When creating a batch sequence to perform a task, you can use one or more of the batch commands listed in the Edit Sequence dialog box in combination with the Execute JavaScript batch command; for example, the sample sequence described in Sign all documents uses Execute JavaScript to sign the document, then uses the Security batch command to set the security of the document.

Batch processing objects

When you run a batch sequence that contains an Execute JavaScript batch command, a *batch event* occurs and an `event` object is created. This object contains the following properties:

`target`	During a batch sequence, the `target` property of the `event` object points to the Doc object of the file currently being processed. For example, `event.target.path` is the device-independent path of the file currently being processed. **Note:** The `this` object also points to the Doc object of the file currently being processed in the batch. See the following code lines in the section Using the this object.
`rc`	The `rc` property contains a return code for each application of the sequence. Setting `rc` to `false` aborts the batch process, so that no further files are processed (if any remain in the selected files collection).

Aborting a script

Setting `event.rc = false` aborts the batch process, but does not abort the script itself. The remaining code in the program flow is executed. One strategy to avoid the continuation of the flow is to throw an exception, and set `event.rc = false` with the catch:

```
try {
   ... JS script for batch processing .....
   if ( somethingGoesWrong ) throw "Aborting Process"
   ....JS script for batch processing ....
   } catch (e)
   {
   ... clean up code lines ....
   event.rc = false;
   }
```

Using the this object

The this object points to the Doc object, for example, the following script gathers all annotations in the document currently being processed, sorts them by author, and returns an array of annotation objects for further manipulation:

```
var annots = this.getAnnots({nSortBy: ANSB_Author});
```

Alternatively, you can also use `event.target`:

```
var annots = event.target.getAnnots({nSortBy: ANSB_Author});
```

Global variables

Variables that must hold their values across document processing must be declared as global.

```
global.report = new Report()
global.counter = 0;
```

This declaration causes the `global.report` object to be available for each application of the script to a selected file. For example, the script that processes a file might contain the following code:

```
global.counter++
global.report.writeText("Report on File \#"+global.counter);
```

At the end of the batch job, any global variables can be removed. For example:

```
delete global.counter;
```

Beginning and ending a batch job

More complicated batch jobs that involve cross-document reporting may need some start-up, or *Begin Job* code, to initialize global variables before processing begins. After the files are processed, you can use *End Job* code to clean up or write a report.

The batch feature of Acrobat Pro DC does not have a built-in capability for *Begin Job* and *End Job* code. The approach the sample sequences takes for detecting the begin and end of job is to use two global variables, `global.counter` and `global.FileCnt`.

- `global.counter`: This variable detects the beginning of the job, counts the files as they are processed and determines when the end of the job occurs. When the batch is run, the variable is initialized to zero and incremented with each iteration of the script, or is initialized to the number of files to be processed and decremented with each iteration. The *End Job* code runs when the variable reaches the number of files being processed, or 0.
- `global.FileCnt`: Most of the work is done by `global.counter`; however, you do need to know how many files are to be processed. This is the role the `global.FileCnt` plays. The value of `global.FileCnt` can be manually set (by executing `global.FileCnt = 5` in the console, for example) or programmatically by another batch sequence. The batch sequence Count PDF files sets the value of `global.FileCnt` to the number of files selected.

Example: *Specify the number of files to process*

The following code is a general outline of how you can insert *Begin Job* and *End Job* code: it uses `global.counter` to detect the beginning of the job and uses another global variable `global.FileCnt`, which is set earlier by another batch sequence that counts the number of files to be processed, see Count PDF files to detect the end of the job.

```
// Begin job
if ( typeof global.counter == "undefined" ) {
     console.println("Begin Job Code");
     global.counter = 0;
     // insert beginJob code here
     .................
}
// Main code to process each of the selected files
try {
  global.counter++
    console.println("Processing File #" + global.counter);
    //  insert batch code here.
    ...............
  } catch(e) {
    console.println("Batch aborted on run #" + global.counter);
    delete global.counter;    // Try again, and avoid End Job code
    event.rc = false;         // Abort batch
  }
  // End job
  if ( global.counter == global.FileCnt ) {
    console.println("End Job Code");
    // Insert endJob code here
    .................
    // Remove any global variables used in case user wants to run
    // another batch sequence using the same variables
    delete global.counter;
  }
```

Debugging and testing tips

The batch feature is very useful for performing a number of tasks on large scale. However, it poses its own dangers. Any automated utility that saves files to a hard drive can potentially create havoc on that drive.

Here are some debugging and testing tips:

- When testing the script, use test documents rather than real documents.
- Initially, work with a small number of test documents.
- Use test directories containing files that can be safely overwritten.
- Make sure your paths exist. If an `app.openDoc` is executed with a path that does not exist, an exception is thrown.
- Use `try/catch/throw` to exit gracefully from a batch if something goes wrong (such as bad paths), and to better control the flow of the code. (See the code snippet in Aborting a script.)
- Use `console.println()` to write information to the console for feedback as to the value of different variables as the batch runs.
- When potentially executing an infinite loop, during the testing phase limit the loop to a finite number of executions until you are sure the code is correct. (See the batch sequence described in Populate a form from a database and save it.)
- If you use the *Begin Job/End Job* technique illustrated above, make sure the global counter variable is undefined before the job runs.