Changes to LEADTOOLS JavaScript Library

Projects Hierarchy

The library projects have been merged into a single ASP.NET Web API project.

Documents Library

July 2015 Release

Item	Location	Description
Web Service	Examples\REST\Leadtools.Documents.Service	Web service for accessing the native .NET Documents Library classes from the web application.
Web Service Host	Examples\REST\DocumentsServiceHost	WCF Host for Leadtools.Documents.Service
Web Service Deployment	Services\LeadtoolsServicesHostManager	Tool for generating IIS virtual directory and application domains used by the service.
Cache Utility	Examples\REST\LTCachePurge	Command line tool for purging expired items from the cache
Demo	Examples\JS\DocumentViewerDemo	Client-side JavaScript demo

November 2015 Release

All the items above have been consolidated into a single ASP.NET MVC Web API solution located at

Examples\JS\DocumentViewer

Item	Location	Description
Web Service	DocumentViewer\WebApp\Controllers	ASP.NET Controllers for accessing the native .NET Documents Library classes from the web application.
Web Service Host	N/A	The solution can be self-hosted inside IIS Express
Web Service Deployment	N/A	The solution can be deployed using standard Visual Studio web deployment support.
Cache Utility	api/factory/purge	Web method to purge the cache directly
Demo	DocumentViewer\WebApp\App	Client-side JavaScript demo

Leadtools.Documents Library Breaking Changes

The following lists of classes/functions/properties that have been removed in the November release and their new counterparts:

Removed Member	New Member
DocumentFactory.serviceAppName	DocumentFactory.servicePath and DocumentFactory.serviceHost
DocumentFactory.documentServiceName	DocumentFactory.serviceApiPath
DocumentConverterJobErrorMode.continue	DocumentConverterJobErrorMode.resume
ICommand, IStatus, SimpleCommand	Commands are removed and not used, refer to TODO ADD JQUERY LINK in the documentation
CommandError	ServiceError
LoadDocumentCommand	DocumentFactory.loadFromUri or loadFromCache
UploadDocumentCommand	DocumentFactory.uploadFile or DocumentFactory.beginUpload/uploadDocument/abortUploadDocument
GetAnnotationsCommand	DocumentAnnotations.getAnnotations or DocumentPage.getAnnotations
SetAnnotationsCommand	DocumentAnnotations.setAnnotations or DocumentPage.setAnnotations
ParseStructureCommand	DocumentStructure.parse
DecryptDocumentCommand	Document.decrypt
GetTextCommand	DocumentPage.getText
ParseTextCommand	DocumentPageText.buildText
ConvertCommand	Document.convert

In short, the old commands system has been removed and instead, the Documents toolkit uses methods in the classes themselves to perform the actions. Each of these methods returns a jquery promise that can be resolved to the method result. Porting the code should be as easy as finding the equivalent method from the above table and modifying the call.

For example, this is the old code to load a document:

```
// Create a LoadDocumentCommand
var url = "http://demo.leadtools.com/images/pdf/leadtools.pdf";
var loadDocumentCommand = lt.Documents.LoadDocumentCommand.create();
// Set some options
loadDocumentCommand.name = "mydocument";
// Run the command to load it
loadDocumentCommand.run()
   .done(function (document) {
      // Ready, use document
      alert("Finished loading:" + document.name);
   })
   .fail(function (error) {
      console.log("Failed: " + error);
   });
```

And the new code:

```
// Create document load options objects
var url = "http://demo.leadtools.com/images/pdf/leadtools.pdf";
var options = new lt.Documents.LoadDocumentOptions();
// Set some options
options.name = "mydocument";
// Load it
lt.Documents.DocumentFactory.loadFromUri(url, options)
   .done(function (document) {
      // Ready, use document
      alert("Finished loading:" + document.name);
   })
   .fail(function (jqXHR, statusText, errorThrown) {
      console.log("Failed: " + errorThrown);
   });
```

So, instead of creating a command, we just call the method directly. The options that were part of the command are converted into either parameters or as in the case above, part of a dedicated options object passed as a parameter.

The **done** and **fail** promises works like before with **fail** now returning the AJAX raw results for further inspection (removes the need for the old CommandError class).

Similarly, this is the old code for getting text from a page:

```
// Create a get text command for the first page in the document
var getTextCommand = lt.Documents.GetTextCommand.create(document.pages[0]);
// Run it
getTextCommand.run()
   .done(function (documentPageText) {
      console.log("Text was read succesfully:");
      // Build the text
      documentPageText.buildText();
      // Show the result as a string
      console.log(documentPageText.text);
   });
```

New code:

```
// Get the text command for the first page in the document
document.pages[0].getText(lt.LeadRectD.empty)
   .done(function (documentPageText) {
      console.log("Text was read succesfully:");
      // Build the text
      documentPageText.buildText();
      // Show the result as a string
      console.log(documentPageText.text);
   });
```

These new features are added to the Documents library with this release:

- User data for all calls (DocumentFactory.serviceUserData and Document.serviceUserData)
- Fully customizable AJAX calls – to add support for authentication headers for instance
- Barcodes reading
- DocumentViewer demo support for loading/saving documents from/to Microsoft One Drive, Microsoft SharePoint and Google Drive
- Display resample interpolation for color as well as Black/White images
- New HTML elements mode in the viewer for memory optimization as well as CSS transitions and animations

Imaging Library

July 2015 Release

Item	Location	Description
Web Service	Examples\REST\Leadtools.RESTServices	Web service for accessing the native .NET Imaging and Documents Library classes from the web application.
Web Service Host	Examples\REST\RESTServicesHost	WCF Host for Leadtools.RESTServices
Web Service Deployment	Services\LeadtoolsServicesHostManager	Tool for generating IIS virtual directory and application domains used by the service.
Demo	Examples\JS\AnnotationsDemo	JavaScript Demo
Demo	Examples\JS\BarcodeDemo	JavaScript Demo
Demo	Examples\JS\ImageProcessingDemo	JavaScript Demo
Demo	Examples\JS\MRTDReaderDemo	JavaScript Demo
Demo	Examples\JS\OcrDemo	JavaScript Demo
Demo	Examples\JS\SvgDemo	JavaScript Demo
Demo	Examples\JS\ThumbnailsDemo	JavaScript Demo
Demo	Examples\JS\ViewerDemo	JavaScript Demo

November 2015 Release

All the items above have been consolidated into a single ASP.NET MVC Web API solution located at

Examples\JS\Demos

Item	Location	Description
Web Service	Demos\WebApp\Controllers	ASP.NET Controllers for accessing the native .NET Image and Documents Library classes from the web application. The old service contracts have been ported

		into controllers using the same name
Web Service Host	N/A	The solution can be self-hosted inside IIS Express
Web Service Deployment	N/A	The solution can be deployed using standard Visual Studio web deployment support.
Demo	Demos\WebApp\Apps\Annotations	JavaScript Demo
Demo	Demos\WebApp\Apps\BankCheckReader	JavaScript Demo
Demo	Demos\WebApp\Apps\Barcode	JavaScript Demo
Demo	Demos\WebApp\Apps\ DriverLicenseFormsRecognition	JavaScript Demo
Demo	Demos\WebApp\Apps\ImageProcessing	JavaScript Demo
Demo	Demos\WebApp\Apps\ImageViewerStyles	JavaScript Demo
Demo	Demos\WebApp\Apps\MRTDReader	JavaScript Demo
Demo	Demos\WebApp\Apps\OCR	JavaScript Demo
Demo	Demos\WebApp\Apps\Thumbnails	JavaScript Demo
Demo	Demos\WebApp\Apps\Viewer	JavaScript Demo