# Migrating from PDFreactor 10.0 to PDFreactor 10.1

## SECURITY ENHANCEMENTS

PDFreactor now features configurable security behavior to prevent attacks in form of the injection of malicious code from potentially untrusted third parties that produce content which is converted by PDFreactor.

## UPDATED DEFAULT BEHAVIOR

Because of the default security settings, there are some behavior changes:

- When converting XML documents, PDFreactor will no longer automatically load external XML parser resources, such as DTDs, entities or XIncludes.
- Document resources can no longer be loaded from the server's file system by default.

Please refer to the PDFreactor manual for a detailed description of the security settings and the default behavior.

You can recreate the previous unsafe behavior by using the following security settings. This also works in the deprecated legacy API. Please note that this is strongly discouraged if you process content from potentially untrusted sources.

To Undo **Security Relevant**

```
config.setSecuritySettings(new SecuritySettings()
    .setAllowExternalXmlParserResources(true)
    .setDefaults(new SecurityDefaults()
        .setAllowFileSystemAccess(true)));
```

### PDFreactor Web Service

PDFreactor Web Service users can configure the security settings via server parameters. Please refer to the manual for more information.

# Migrating from PDFreactor 9 to PDFreactor 10

## IMPORTANT!

PDFreactor Web Service users: If you have customized the "start.ini" file of the Jetty server, please see section PDFreactor Web Service Jetty before installing PDFreactor 10.

## UPDATED DEFAULT BEHAVIOR

### CSS Validation

The CSS Validator is now enabled by default. In PDFreactor 9 the validation was disabled, thus invalid values could overwrite valid ones. Now with the validation enabled, invalid values are completely ignored, as if they were not in the style rule.

To disable the Validator, use the API method `setCssSettings`:

To Undo

```
config.setCssSettings(new CssSettings().setValidationMode(CssPropertySupport.ALL));
```

### Rasterization

The images that are produced when rasterizing SVGs and canvas elements have now a maximum size of 2 megapixels by default. If the image would be larger, the resolution is reduced. This way, the used memory and the size of the resulting PDF are reduced. The following snippet disables this limit:

To Undo

```
svg, canvas, img, object, embed {
    -ro-rasterization-max-size: none;
}
```

The property can also be used to increase the max-size to a certain megapixel value:

Example

```
svg, canvas, img, object, embed {
    -ro-rasterization-max-size: 3.5;
}
```

## MathML

Rendering MathML now requires the import of a MathML library. We recommend **MathJax**, as it creates better results than the lib PDFreactor used in previous versions.

If the input document can not be modified, the following user scripts can be defined to include and use MathJax. The first script consists of settings for the next one:

- "roMjPath" must be set to the URL or path to the file MathJax.js, excluding the filename itself.
- "roMjFile" specifies the name of the main MathJax file. It should should usually be left default.
- "roMjSvgBlacker" allows to optionally increase the thickness of the fonts used by MathJax.

Please see the comments in the snippet for example values:

```
roMjPath = "";              // default: "",
                            // examples: "MathJax/", "../../resource/js/mathjax/",
                            // "https://cdnjs.cloudflare.com/ajax/libs/mathjax/2.7.5/"
roMjFile = "MathJax.js";    // default: "MathJax.js",
                            // examples: "mathjax.js", "mathjaxmod.js"
roMjSvgBlacker = 0;         // default: 0,
                            // examples: 1, 2
```

The second script uses the values form the first one and inserts the required script elements into the document, so MathJax is loaded and processes all "math" elements. It does not have to be modified.

```
document.documentElement.firstElementChild.insertAdjacentHTML('beforeend',
    '\u003Cscript type="text/x-mathjax-config">MathJax.Hub.Config(' +
    JSON.stringify({
        jax: ["input/MathML", "output/SVG"],
        extensions: ["mml2jax.js"],
        MathML: { extensions: ["content-mathml.js"] },
        SVG: { blacker: (typeof window.roMjSvgBlacker == "number" &&
            window.roMjSvgBlacker > 0 ? window.roMjSvgBlacker : 0) }
    }) +
    ');\u003C/script>\n' +
    '\u003Cscript type="text/javascript" src="' +
    (window.roMjPath ? window.roMjPath : "MathJax/") +
    (window.roMjPath && !(window.roMjPath + "").endsWith("/") ? "/" : "") +
    (window.roMjFile ? window.roMjFile : "MathJax.js") +
    '">\u003C/script>'
);
```

### text-align

As defined by the CSS3 specifications, 'text-align' is now handled as a shorthand for 'text-align-all' and 'text-align-last'. This means that, if 'text-align-last' is defined **before** 'text-align', the last line alignment will be overridden.

This can be avoided by using 'text-align-all' instead of 'text-align'.

Furthermore 'text-align-last' now also works if the paragraph has not been set to 'justify'.

### Form Fields

The default styles for several form fields was updated to a more modern look. Authors that rely on these default styles should verify their documents. If necessary, the legacy styles can be recreated using CSS.

## API CHANGES

### General

**signPDF**

When digitally signing a PDF using the `signPDF` configuration property, PDFreactor will now throw an exception and thus fail the conversion if the PDF could not be signed. Appropriate log entries as well as exception messages will provide information as to the reason.

### Java (non-Web Service)

In previous versions, the "convert" methods were throwing various exceptions. In PDFreactor 10, this has bee reduced to only a single exception type—the `PDFreactorException`. Previously thrown exceptions are now wrapped in `PDFreactorException`s and are available as their cause. If you are trying to catch the older exception types in your integration, you will receive compile errors.

To resolve this, just remove the now superfluous catch blocks:

```
try {
    pdfReactor.convert(config);
} catch (PDFreactorException e) {
    // ...
} catch (IOException e) {
    // ...
} catch (SAXException e) {
    // ...
} catch (TransformerException e) {
    // ...
}
```

## PDFreactor Web Service Jetty

When upgrading to PDFreactor 10 via one of the installers, an existing "start.ini" file will now always be overwritten. If you have made any changes to it, it is highly recommended that you backup these changes before installing PDFreactor 10. The installer will also do a backup of an existing "start.ini" file.

From PDFreactor 10 onward, any customizations of Jetty should no longer be done in the "start.ini" file but instead in the new "main.ini" and other "ini" files located in the "PDFreactor/jetty/start.d" directory.

## JavaScript Wrapper

The JavaScript client API was changed. Instead of using **callback** parameters, the API methods now return **Promises**.

PDFreactor 9− example:

Legacy

```
pdfReactor.convert(config, function(result) {
    processResult(result);
}, function(error) {
    processError(error);
});
```

PDFreactor 10 example:

PDFreactor 10

```
try {
    const result = await pdfReactor.convert(config);
    processResult(result);
} catch (error) {
    processError(error);
}
```

For more information, please refer to the new JavaScript integration examples and the JavaScript API documentation.

## Java Wrapper

1. The class

   Legacy

   ```
   com.realobjects.pdfreactor.webservice.client.Log.LogRecord
   ```

   has been moved and renamed to

   PDFreactor 10

   ```
   com.realobjects.pdfreactor.webservice.client.Record
   ```

2. All methods that returned *lists* now return *arrays* instead.

## .NET Wrapper

1. The namespace of the class

   Legacy

   ```
   RealObjects.PDFreactor.Webservice.Client.Log.Record
   ```

   has been changed to

   PDFreactor 10

   ```
   RealObjects.PDFreactor.Webservice.Client.Record
   ```

2. The property

   Legacy

   ```
   ExceedingContent
   ```

   of the `Result` class has been renamed to

   PDFreactor 10

   ```
   ExceedingContents
   ```

# DEPRECATED API

## All APIs

The configuration properties `appendLog` and `enableDebugMode` are now obsolete. Use `debugSettings` instead.

| Old API | New API |
|---|---|
| **Enabling JavaScript** | |
| `config.setJavaScriptMode(true);` | `config.setJavaScriptSettings(`<br>`  new JavaScriptSettings().setEnabled(true));` |
| **Enabling debug mode** | |
| `config.setEnableDebugMode(true);` | `config.setDebugSettings(`<br>`  new DebugSettings().setAll(true));` |
| **Appending the log** | |
| `config.setAppendLog(true);` | `config.setDebugSettings(`<br>`  new DebugSettings().setAppendLogs(true));` |
| **Setting resource timeout** | |
| `config.setResourceRequestTimeout(`<br>`  Integer);` | `config.setResourceReadTimeout(Integer);`<br>`config.setResourceConnectTimeout(Integer);` |
| **Setting a color space** | |
| `config.setDEfaultColorSpace(`<br>`  ColorSpace);` | `config.setColorSpaceSettings(`<br>`  new ColorSpaceSettings().setTargetColorSpace(`<br>`  ColorSpace));` |

# CSS CHANGES

## string()

The String-function takes 2 parameters, with the second being a keyword to specify which value should be used, if there are multiple assignments for that named string on a single page.

In accordance with the CSS specifications, the identifier `last-except` has been renamed.

The keyword `last-except`

Legacy

```
content: string(namedString, last-except);
```

has been replaced with `first-except`

PDFreactor 10

```
content: string(namedString, first-except);
```

which has the same effect.

# Migrating from PDFreactor 8 to PDFreactor 9

## UPDATED DEFAULT HTML STYLES

With PDFreactor 9 some of the standard HTML styles have been edited to be more in line with the HTML5 specification and modern browsers. While most changes are cosmetic, some will influence the layouts of existing documents. When these have a negative impact on existing documents and adapting the print styles accordingly is to complex, the following snippets can be used to undo the most significant changes:

### Page Margin

The default page margins were increased from 1cm to 2cm.

To Undo

```
@page {
    margin: 1cm;
}
```

### Body Margin

In previous versions of PDFreactor the default margins of body elements were 8px, in accordance to the HTML specification. As these default margins are only useful in non-paged environments they were dropped in PDFreactor 9.

To Undo

```
body {
    margin: 8px;
}
```

For this to take effect in multi-page documents the following snippet has to be applied as well.

### Box Decoration Break

The default value of the property box-decoration-break is now slice, as per the specification. The margin, padding and border-width values around page break will be treated as 0.

To Undo

```
* {
    box-decoration-break: clone;
}
```

## Rounding Mode

The default value of the proprietary property -ro-rounding-mode is now floor. This avoids rare cases where accumulated rounding imprecisions could lead to unexpected layout results, but may lead to different line and page-breaks.

To Undo

```
html {
    -ro-rounding-mode: round;
}
```

## First Page Side

The default side of first pages has changed from left to recto (i.e. right, unless the document direction is right-to-left), as per the specification.

To Undo

```
@-ro-preferences {
    first-page-side: left;
}
```

## Margins after Breaks

The top margins of blocks at the start of pages and columns are no longer set to 0 for first pages and columns or after forced breaks. This matches the behavior of browsers and the requirements of the latest CSS specifications.

To Undo

```
html {
    -ro-truncate-margin-after-break: always;
}
```

However, in most cases it is advisable to apply non-proprietary styles, that result in the same improvement in browsers.

Example

```
h1 {
    break-before: page;
    margin-top: 0;
}

div.multiColumn > *:first-child {
    margin-top: 0;
}
```

# API CHANGES

The package of the `ExceedingContent` class has been changed from

```
com.realobjects.pdfreactor.exceedingcontent
```

to

```
com.realobjects.pdfreactor.contentobserver
```

# DEPRECATED API

## Java

The method `setDocument(Object)` is obsolete, use `setDocument(String)`, `setDocument(byte[])` or `setDocument(InputSource)` instead. This only affects the non-wrapper Java library.

## Java and Java Wrapper

Constructors of inner `Configuration` classes (except for the `KeyValuePair` class) that take multiple arguments have been deprecated and will be removed in a future version. Use the no-argument constructor in conjunction with individual setters instead. An exception is the `KeyValuePair` class whose multi-argument constructor is not deprecated. Setters now return an instance of the class, thus allowing you to chain multiple subsequent setter calls. The following example demonstrates using no-argument constructors and chainable setters by adding a user script and a user style sheet to a `Configuration` instance:

**Old API**

```
config.setUserScripts(new ScriptResource(null, "http://myScript.js", false));
config.setUserStyleSheets(new Resource("p { color: red; }", null));
```

**New API**

```
config
    .setUserScripts(new ScriptResource().setUri("http://myScript.js"))
    .setUserStyleSheets(new Resource().setContent("p { color: red; }"));
```

## .NET Wrapper

Multi-argument constructors are now obsolete in favor of object initializers.

**Old API**

```
config.UserScripts.Add(new ScriptResource(null, "http://myScript.js", false));
config.UserStyleSheets.Add(new Resource("p { color: red; }", null));
```

**New API**

```
config.UserScripts.Add(new ScriptResource { Uri = "http://myScript.js" });
config.UserStyleSheets.Add(new Resource { Content = "p { color: red; }" });
```

## PHP Wrapper

The PDFreactor PHP classes are now located in the namespace `com\realobjects\pdfreactor\webservice\client\`. To adjust your integration, just add appropriate `use` directives like this:

```
use com\realobjects\pdfreactor\webservice\client\PDFreactor as PDFreactor;
use com\realobjects\pdfreactor\webservice\client\LogLevel as LogLevel;
use com\realobjects\pdfreactor\webservice\client\ViewerPreferences as ViewerPreferences;
...
```

## All APIs

The configuration properties `mergeByteArray`, `mergeByteArrays`, `mergeURL` and `mergeURLs` are now obsolete. Use `mergeDocuments` instead. Note that `mergeDocuments` takes one or more `Resource` objects, so you have to use the appropriate properties of that object, either `data` for binary data or `uri` for URLs. Java example:

**Old API**

```
// merge single document from binary source
config.setMergeByteArray(byteArray1);
// merge multiple documents from binary source
config.setMergeByteArrays(byteArray1, byteArray2);
// merge single document from URL source
config.setMergeByteURL(url1);
// merge multiple documents from URL source
config.setMergeByteURLs(url1, url2);
```

**New API**

```
// merge single document from binary source
config.setMergeDocuments(new Resource().setData(byteArray1));
// merge multiple documents from binary source
config.setMergeDocuments(new Resource().setData(byteArray1),
        new Resource().setData(binary2));
// merge single document from URL source
config.setMergeDocuments(new Resource().setUri(url1));
// merge multiple documents from URL source
config.setMergeDocuments(new Resource().setUri(url1),
        new Resource().setUri(url2)));
```

The `ScriptResource` type is now deprecated, use `Resource` instead. This change only affects Java and .NET APIs.

# Migrating from PDFreactor 8.0 to PDFreactor 8.1

## JAVA AND .NET WRAPPERS

When using either the Java or .NET wrapper APIs, your integration has to be adjusted. These wrapper APIs are now based on the REST API rather than the SOAP API. This was done to provide an API that is more in line with the other wrapper APIs.

## PDFREACTOR WEB SERVICE SETTINGS

When using the PDFreactor Web Service with custom settings in the "pdfreactorwebservice.vmoptions" file, you have to migrate these settings to the "start.ini" located in the "PDFreactor/jetty" directory.

If your "pdfreactorwebservice.vmoptions" looked like this

```
-Xmx1024m
-Djava.awt.headless=true
```

you have to add the lines from your "pdfreactorwebservice.vmoptions" to the end of your "start.ini" file

```
--exec
-Dorg.apache.cxf.Logger=org.apache.cxf.common.logging.Slf4jLogger
-Dcom.realobjects.interceptConsoleOutput=true
# old pdfreactorwebservice.vmoptions settings
-Djava.awt.headless=true
-Xmx1024m
```

# Migrating from PDFreactor 7- to PDFreactor 8+

With PDFreactor 8 we are introducing the first major API change since PDFreactor 2. One major benefit of this change is that the new Java API is identical (with a few additions) to the newly introduced Java web service client API.

## USING THE LEGACY JAVA API

Java integrators can still use the old API, however we highly recommend to migrate to the new API as soon as possible, since the old one will be removed in a future release of PDFreactor. To continue using the old legacy API, just change the package from

```
com.realobjects.pdfreactor
```

to

```
com.realobjects.pdfreactor.legacy
```

## MIGRATING TO THE NEW API

### The Configuration Object

The most obvious change is the introduction of the `Configuration` object. In previous versions of PDFreactor, you invoked all API methods on the PDFreactor instance, however this had some disadvantages like making the PDFreactor instance non-reusable. In PDFreactor 8, you just have to create one instance of PDFreactor. The instance only has a few API methods, like `convert`.

All settings and options are properties of the configuration and have to be set there. While most methods are the same as in previous versions of PDFreactor, some have changed. For example, instead of add-methods, getters are used to retrieve lists on which new entries can be added. Make sure to consult the API documentation.

### Converting

To create a PDF or image, you now just have to call the `convert` method with the configuration as a single parameter, which also specifies the input document. PDFreactor automatically detects if you are converting an HTML string, a URL or binary data.

### Retrieving the Result

The new `convert(Configuration)` method no longer returns the PDF as binary directly. It returns a `Result` object which not only contains the PDF as binary, but also other useful data such as the log.

There are additional convert methods, such as `convert(Configuration, OutputStream)` which writes the PDF directly in the specified OutputStream instead of returning it or `convertAsBinary(Configuration)` which returns the binary data directly instead of a Result object. Please make sure to read the API documentation and the PDFreactor manual (Chapter "Integration").

## EXAMPLES

Below are simple examples in different programming languages that show how PDFreactor was used previously and how it is used now.

## Java

### Old API

```
PDFreactor pdfReactor = new PDFreactor();

// simple settings
pdfReactor.setAddBookmarks(true);
pdfReactor.setAddLinks(true);

// adding a user style sheet
pdfReactor.addUserStyleSheet("p { color: red }", null, null, null);

// create PDF and specify the document
byte[] pdf = pdfReactor.renderDocumentFromURL("https://www.realobjects.com");
```

### New API

```
PDFreactor pdfReactor = new PDFreactor();
Configuration config = new Configuration();

// specify the document
config.setDocument("https://www.realobjects.com");

// simple settings
config.setAddBookmarks(true);
config.setAddLinks(true);

// adding a user style sheet
config.getUserStyleSheets().add(new Resource("p { color: red }", null));

// create PDF
Result result = pdfReactor.convert(config);
byte[] pdf = result.getDocument();
```

## .NET

### Old API

```
PDFreactor pdfReactor = new PDFreactor();

// simple settings
pdfReactor.SetAddBookmarks(true);
pdfReactor.SetAddLinks(true);

// adding a user style sheet
pdfReactor.AddUserStyleSheet("p { color: red }", "", "", "");

// create PDF and specify the document
byte[] pdf = pdfReactor.RenderDocumentFromURL("https://www.realobjects.com");
```

### New API *(Changed in version 8.1)*

```
PDFreactor pdfReactor = new PDFreactor();
Configuration config = Configuration();

// specify the document
config.Document = "https://www.realobjects.com";

// simple settings
config.AddBookmarks = true;
config.AddLinks = true;

// adding a user style sheet
config.UserStyleSheets = new List<Resource> {new Resource("p { color: red }", "")};

// create PDF
Result result = pdfReactor.Convert(config);
byte[] pdf = result.Document;
```

## PHP

### Old API

```
$pdfReactor = new PDFreactor();

// simple settings
$pdfReactor->setAddBookmarks(true);
$pdfReactor->setAddLinks(true);

// adding a user style sheet
$pdfReactor->addUserStyleSheet("p { color: red }", "", "", "");

// create PDF and specify the document
$result = $pdfReactor->renderDocumentFromURL("https://www.realobjects.com");
```

### New API

```
$pdfReactor = new PDFreactor();
$config = array(
    // specify the document
    "document" => "https://www.realobjects.com",

    // simple settings
    "addBookmarks" => true,
    "addLinks" => true,

    // adding a user style sheet
    "userStyleSheets" => array(
        array(
            "content"=> "p { color: red }"
        )
    )
);

// create PDF
// ...as base64 encoded String
$result = $pdfReactor->convert($config);
$pdf = $result->document;

// ...as binary
$pdf = $pdfReactor->convertAsBinary($config);
```

*Note: To convert the base64 encoded document into binary data, you can do the following:*

```
echo base64_decode($result->document);
```

## Python

### Old API

```
pdfReactor = PDFreactor()

# simple settings
pdfReactor.setAddBookmarks(True)
pdfReactor.setAddLinks(True)

# adding a user style sheet
pdfReactor.addUserStyleSheet("p { color: red }", "", "", "")

# create PDF and specify the document
result = pdfReactor.renderDocumentFromURL("https://www.realobjects.com");
```

### New API

```
pdfReactor = PDFreactor()
config = {
    # specify the document
    'document': "https://www.realobjects.com",

    # simple settings
    'addBookmarks': True,
    'addLinks': True,

    # adding a user style sheet
    'userStyleSheets': [
        {
            'content': "p { color: red }"
        }
    ]
}

# create PDF
# ...as base64 encoded String
result = pdfReactor.convert(config)
pdf = result['document']

# ...as binary
pdf = pdfReactor.convertAsBinary(config)
```

*Note: To convert the base64 encoded document into binary data, you can do the following:*

```
import base64
print(base64.b64decode(result['document']))
```

## Ruby

### Old API

```
pdfReactor = PDFreactor.new();

# simple settings
pdfReactor.setAddBookmarks(true)
pdfReactor.setAddLinks(true)

# adding a user style sheet
pdfReactor.addUserStyleSheet("p { color: red }", "", "", "")

# create PDF and specify the document
result = pdfReactor.renderDocumentFromURL("https://www.realobjects.com")
```

### New API

```
pdfReactor = PDFreactor.new()
config = {
    # specify the document
    document: "https://www.realobjects.com",

    # simple settings
    addBookmarks: true,
    addLinks: true,

    # adding a user style sheet
    userStyleSheets: [
        {
            content: "p { color: red }"
        }
    ]
}

# create PDF
# ...as base64 encoded String
result = pdfReactor.convert(config)
pdf = result["document"]

# ...as binary
pdf = pdfReactor.convertAsBinary(config)
```

*Note: To convert the base64 encoded document into binary data, you can do the following:*

```
require "base64"
print Base64.decode64(result["document"])
```

## Perl

### Old API

```
my $pdfReactor = PDFreactor -> new();

# simple settings
$pdfReactor -> setAddBookmarks('true');
$pdfReactor -> setAddLinks('true');

# adding a user style sheet
$pdfReactor -> addUserStyleSheet("p { color: red }", "", "", "");

# create PDF and specify the document
$result = pdfReactor -> renderDocumentFromURL("https://www.realobjects.com");
```

### New API

```
my $pdfReactor = PDFreactor -> new();
$config = {
    # specify the document
    'document' => "https://www.realobjects.com",

    # simple settings
    'addBookmarks' => 'true',
    'addLinks' => 'true',

    # adding a user style sheet
    'userStyleSheets' => [
        {
            'content' => "p { color: red }"
        }
    ]
};

# create PDF
# ...as base64 encoded String
$result = $pdfReactor -> convert($config);
$pdf = $result->{'document'};

# ...as binary
$pdf = $pdfReactor -> convertAsBinary($config);
```

*Note: To convert the base64 encoded document into binary data, you can do the following:*

```
use MIME::Base64 ();
print MIME::Base64::decode($result->{'document'});
```