Easy PDF

Developer Guide

About this Document

This guide describes the Easy PDF API, usage, and samples.

For additional description of product features please see:

Administrator Guide Getting Started Series	Registration, Setup and User Management Product Guides
Available on the website at:	https://easypdf365.com/guides
API Reference	https://easypdf365.com/developer/doc

For questions, assistance, product demonstration, or anything else...
Please don't hesitate to contact us at:

By email: support@easypdf365.com
On the web: https://easypdf365.com/support
By phone: +1 (407) 260-0834 (8-5pm EST)

About the API

Easy PDF is all about delivering messages from Business Central.
A message is any form of digital correspondence – print, email, or fax. (purposely a flexible definition)
Easy PDF has a built-in workflow for preparing and delivering a message.

The Easy PDF API exposes events and procedures to allow you to customize the content and behavior of the message generation workflow.

About the Workflow

Easy PDF is designed to fit (as seamlessly as possible) into the built-in BC messaging workflow.
In a nutshell, the Easy PDF messaging workflow entails:

- Initiating message delivery
  (usually from an action on a document card, e.g., Posted Sales Invoice, ez|Send by Email)
- Generating message attachment content
  (usually by running/rendering a BC report into a PDF and/or appending additional content)
- Generating message body content
  (usually by rendering a message body template into HTML)
- Collecting recipient information
  (e.g., harvesting email addresses when sending email)
- Assembling the above bits for delivery
  (e.g., creating an email message when sending email)
- Sending the message
- Logging the result

Pretty much every messaging scenario follows that flow.
Easy PDF taps into that flow for its out-of-the-box messaging features and provides the Easy PDF API to let you customize the process.

About the State Machine

There is a fundamental concept that you should bear in mind when developing with the Easy PDF API:

It is state dependent.

As Easy PDF works through the messaging flow it relies upon an inherent state machine to manage its behavior. The state machine is defined by the variables in the codeunits that it invokes whilst processing the message.

A slightly more detailed description:

When sending a message (for instance selecting the 'ez|Send by Email' action on the sales order page) an Easy PDF codeunit is invoked. The codeunit is handed details about the message being sent, such as the document type and document record. It stores that information in global variables and records. It then collects and stores additional related information such as: customer/vendor record, user record, Easy PDF configuration records, etc. The totality of that data is the "Message State", or just "State". After collecting State, it generates the content for the message including the message attachment(s) and

message body (email). It then delivers the message (print, email, or fax). Lastly, it records the activity in the Easy PDF History.

During that flow it raises timely events to allow you to manipulate the state.
Via that mechanism you can manage the state (and hence the result) of the messaging.

About the Players

In order to play the Easy PDF API game you need to know the players…

EZP_Document – the Easy PDF Document record
This is the main player.
It is the record that defines how a message is processed.
We'll cover specifics later – just know that when a message is processed an EZP_Document is in play.

EZP_Setup – the Easy PDF General Settings record
This one contains settings of a general nature that govern general behaviors. (pretty general eh?)
Things like your preferred language, etc.

EZP_MailSetup – the Easy PDF Mail Account record
This one contains the settings that determine what type of email to send and how to send it.
(i.e., account details and preferences).

EZP_Attachment – the Easy PDF Attachment (printed document) record
Contains the details of the attachment(s) to being sent (or printed) – one per attachment.
This is the rendered content (e.g., a PDF file).

EZP_Address – the Easy PDF Address Book entry record
Contains the details of message recipients (name, address) in the Address Book.

EZP_API_Events – codeunit
Houses the integration events that you can hook into.

EZP_API_Document – codeunit
Used for registering custom document types (more on that later).

EZP_API_Send – codeunit
The entry points for invoking Easy PDF code.
(You call these functions from your custom code to tell Easy PDF to send a message)

Scope – Scope is essentially a privacy setting. It is a string value containing either the value "SHARED" (meaning available to anyone) or a User ID (obtained by calling the system UserId() procedure). When Scope = UserID it is synonymous with private (to that user).

Easy PDF API Documentation – API documentation (generated from code comments).
https://easypdf365.com/developer/doc

A super simple sample

TL;DR; show me..

Here is probably one of the simplest customizations.
In it, we will tell Easy PDF to use a specific report when printing the attachment for the Sales Order.

Note: You could do this without code in the Easy PDF Document Setup configuration or in the Delivery Methods configuration – but hey, that's boring and when you're a carpenter you use a hammer – so we'll do it in code..

```
[EventSubscriber(ObjectType::Codeunit, Codeunit::EZP_API_Events, 'OnGetReportId', true, true]
internal procedure OnGetReportId(EZPDocumentRec: Record EZP_Document; CompoundKey: Text; ExternalDocumentVar: Variant; var
ReportId: Integer)
begin
    case EZPDocumentRec.Code of
        'SALES ORDER':
            begin
                ReportId := 1305;
            end;
    end;
end;
```

Here we hooked into the OnGetReportId event and forced the issue – we told it to always use report# 1305 when processing the "SALES ORDER" document type.
So, now, when Easy PDF formulates the state for the message content, it has been forced to use report id 1305 when it prints the attachment for the message (regardless of the current configuration in the setup tables).

OK, that is a ridiculously simple example, but it provides an opportunity to make a few points:

1. By setting the ReportId, you are modifying Easy PDF state.
   Remember, Easy PDF collects its state before acting on things – here you have altered that state so now it will behave differently.
2. There are a few important parameters in the event signature that you will find in most events raised by Easy PDF – we'll describe them next.

Parameters—
EZPDocumentRec – this is a record from the EZP_Document table.
It is passed to most integration events and contains the state of the Easy PDF Document currently being processed (the document state is a subset of the current message state).
CompoundKey – this is a string parameter that you have control over. You can put pretty much anything you want into it (see the procedures in the API_Send codeunit). It will resurface in the downstream events so you can use it as needed. Most times you will use it as an identifier to know that your code initiated the flow and hence, you know and can decode its content.
ExternalDocumentVar – this is a Variant. Similar to CompoundKey, you have control over this parameter. You can tuck any variable into this Variant and it will resurface in the downstream events. This parameter is handy when you are defining your own Easy PDF Document types based on custom code. You can tuck a record, codeunit or whatever into this parameter and then have access to it during subsequent event processing.
Note: this parameter is now obsolete –use the Set/GetAssociatedRecord('Document' … API instead.

Change report layout sample

Ok, here is a slightly more useful sample.
We will tell Easy PDF to use Word report# 1305 with the RED layout when sending the SALES ORDER – but only when sent from our customization.

This sample is available in the sample repo – ez-api-sample-changelayout
First, we define a PageExtension for the Sales Order and add an action to send with Easy PDF.
(For brevity I only show relevant lines of code – see the sample project for a working implementation)

```
EzpApiSend: Codeunit EZP_API_Send;
EzpApiSend.SendByEmail('SALES ORDER', Rec."No.", 'EZPX;1305;MS-1305-RED', false);
```

Here we invoke the SendByEmail() API for a sales order record -- telling Easy PDF to do its thing.
Note that we are passing a CompoundKey string that starts with 'EZPX;' – so later on we'll look for that key to know that the flow was initiated from our code. Also note that we have tucked the report Id and report layout Id into the CompoundKey – we'll add code to decode and use those when printing the report.

Now add a couple of event subscribers to do the work.

First set the report Id:

```
EventSubscriber 'OnGetReportId'
internal procedure OnGetReportId(EZPDocumentRec: Record EZP_Document; CompoundKey: Text; ExternalDocumentVar: Variant; var
ReportId: Integer)
begin
    if not CompoundKey.StartsWith('EZPX;') then
        exit;

    case EZPDocumentRec.Code of
        'SALES ORDER':
            Evaluate(ReportId, CompoundKey.Split(';').Get(2));
    end;
end;
```

Then set the report layout code:

```
EventSubscriber 'OnGetCustomReportLayoutCode'
internal procedure OnGetCustomReportLayoutCode(EZPDocumentRec: Record EZP_Document; CompoundKey: Text; ExternalDocumentVar:
Variant; ReportId: Integer; var ReportLayoutCode: Code[20]);
begin
    if not CompoundKey.StartsWith('EZPX;') then
        exit;

    case EZPDocumentRec.Code of
        'SALES ORDER':
            ReportLayoutCode := CompoundKey.Split(';').Get(3);
    end;
end;
```

Now your sales order prints with 1305 in the RED layout.
Sweet.

… add a more complete API example here …

About the Event Model

Now that you have a feel for the Easy PDF programming style, (i.e., state dependent, event driven) you will need a bit of insight as to which API events to hook into to implement your customization.

So, here is a timing diagram of the event model for a simple Send-By-Email flow:

Flow	Purpose	Event
Initialization	Sales Order action – ez|Send by Email	OnBeforeAction
	Allow send override	OnBeforeSend
	Prepare the SALES ORDER Document record	OnInitializeRecord
	Retrieve the RecordRef of the associated record (Sales Header)	OnGetRecordVariables
	Retrieve the recipient information	OnGetRecipientDetails
	Retrieve the preferred language	OnGetPreferredLanguage
Print the report	Retrieve the ID of the report to print	OnGetReportId
	Retrieve the XML for the report	OnGetReportParametersXml
	Retrieve a report selection if one is defined	OnGetCustomReportSelectionKeys
	Retrieve a custom report layout if one is defined	OnGetCustomReportLayoutCode
Create the message content	Retrieve the message template Id to use for the email body	OnGetTemplateId
	Retrieve the message subject line	OnGetSubject
	Iterate and expand tokens in the subject and template content	OnBeforeExpandToken
	Allow custom predefined tokens	OnGetPredefinedToken
Collect addresses	Retrieve addresses from the Easy PDF Address Book (if any)	OnGetAddressBookAddresses
	Retrieve the address from the document	OnGetDocumentEmailAddress
	Allow override of addresses	OnGetRecipientAddresses
Get the mail account used for sending	Retrieve the mail account used for sending the email	OnGetMailSetup
Show the email preview dialog	Allow suppression of the email dialog	OnQueryPreview
Send the email	Send the message	OnBeforeSendMessage
Add to history	Add the message to history	OnAfterAddHistoryEntry
Finish		OnAfterAction

Let's pick through that list to shed some light…

Initialization:

OnBeforeAction – standard BC action event, in this example, the 'ez|Send by Email' action was invoked.

OnBeforeSend – raise by Easy PDF to allow you to override sending a document

OnInitializeRecord – raised after Easy PDF has executed its standard initialization logic.

This is usually the first bit of Easy PDF state logic to be hit. Easy PDF needs to collect state for the messaging flow, so it initializes an EZP_Document record based on the BC record being operated on (in this case the Sales Header record).

By hooking this event you can influence the underlying records that are associated with the flow.

Developer Note: If coding a Custom Easy PDF Document Type then you must implement this event.
Developer Note: Variables prefixed with 'g' are Easy PDF state variables.
Here is a sanitized version of the InitializeRecord() logic for the Sales Order.

```
case DocumentCode of
'SALES ORDER':
begin
    gEzpDocumentRec.Get('SALES ORDER', Scope);
    gSalesHeaderRec.Get("Document Type"::Order, DocumentNo);

    gRecipientType := EZP_RecipientType::Customer;
    gCustomerRec.Get(gSalesHeaderRec."Bill-to Customer No.");              // or Sell-to depending on config
    gContactRec.GetBySystemId(gCustomerRec."Contact ID");                  // alsso dependent on config
    gSalespersonPurchaserRec.Get(gCustomerRec."Salesperson Code");
```

```
    gShiptoAddressRec.Get(gSalesHeaderRec."Sell-to Customer No.", gSalesHeaderRec."Ship-to Code");
    gLocationRec.Get(gSalesHeaderRec."Location Code");

    gEzpDocumentRec.SetAssociatedRecordId('Document', gSalesHeaderRec.RecordId);
    gEzpDocumentRec.SetAssociatedRecordId('Customer', gCustomerRec.RecordId);
    gEzpDocumentRec.SetAssociatedRecordId('Contact', gContactRec.RecordId);
    gEzpDocumentRec.SetAssociatedRecordId('SalespersonPurchaser', gSalespersonPurchaserRec.RecordId);
    gEzpDocumentRec.SetAssociatedRecordId('Shipto', gShiptoAddressRec.RecordId);
    gEzpDocumentRec.SetAssociatedRecordId('Location', gLocationRec.RecordId);
end;

ApiEventsCu.OnInitializeRecord(DocumentCode, DocumentNo, gCompoundKey, gExternalDocumentVar, gEzpDocumentRec);
```

So, in a nutshell, InitializeRecord() is all about collecting database records for later use.
The OnInitializeRecord event is raised to allow you to override the default state.

OnGetRecordVariables – raised to retrieve a RecordRef and Field record for the (primary) BC record in the flow.

```
gRecordRef.GetTable(gSalesHeaderRec);
gFieldRec.SetRange(TableNo, DATABASE::"Sales Header");

ApiEventsCu.OnGetRecordVariables(gEzpDocumentRec, gCompoundKey, gExternalDocumentVar, gRecordRef, gFieldRec);
```

These variables are used when expanding tokens in the message template, subject or filenames.

OnGetRecipientDetails – raised to retrieve details (name, number, type) of the recipient of the message.
OnGetPreferredLanguage – raised to retrieve the preferred language for the recipient of the message.

Used when printing the report, and when generating the message body from a template

Printing:

OnGetReportId – raised to retrieve the ID of the BC report to print (and attach if email or fax).

OnGetReportParametersXml – raised to retrieve the XML parameters and filters for the report request page. The content format is the same as returned from system function Report.RunRequestPage(). We supply XML utilities in the ApiTools codeunit to facilitate generation of the content in code.

OnGetCustomReportSelectionKeys – raised to retrieve the keys (Usage, Source Type, Source No.) used to find a report when an Easy PDF template is configured to use a BC report as the source.

OnGetCustomReportLayoutCode – raised to retrieve the custom report layout code when an Easy PDF template is configured to use a BC report as the source.

Message Generation:

OnGetTemplateId – raised to retrieve the ID of the Easy PDF template to use when generating the message body.

OnGetTemplateId – raised to retrieve the text for the subject line of the message (Tokens in the returned text will be expanded before use).

OnGetTemplateId – raised to retrieve the text for the subject line of the message (Tokens in the returned text will be expanded before use).

OnBeforeExpandToken – raised to retrieve allow override of token expansion.

OnGetPredefinedToken – raised during token expansion to allow custom predefined tokens.

Address Harvesting:

OnGetAddressBookAddresses – raised to allow custom logic when retrieving addresses from the Easy PDF Address Book for the current document. At the time this is invoked Easy PDF has already retrieved the addresses using standard logic. This provides an opportunity to override that logic.

OnGetDocumentEmailAddress – raised to retrieve the email address from the current document. Ditto previous comments.

OnGetRecipientAddresses – raised after all addresses have been harvested but before they are added to the message. Use this event to modify the addresses prior to adding them to the message.

Mail Account:

OnGetMailSetup – raised to retrieve the Easy PDF Mail Account to use for delivery. At the time this event is raised, Easy PDF will have retrieved the mail account using standard logic. Use this event to customize that logic.

OnQueryPreview – raised to determine if the message should be previewed (displayed to the user). Only raised in situations where Preview is possible (e.g., not during a batch or job).

OnBeforeSendEmail – raised prior to sending an email message. At the time of the event, all message state has been collected and is passed in the event. Use this event to perform alternate delivery of the message (e.g., a custom mail provider).

OnAfterAddHistoryEntry – raised after a history entry has been added.

-- End of Document --