DEVELOPMENT WITH OPENPRODOC V 1.0

Joaquín Hierro - 2020

Index

1 Introduction

This document describes the various alternatives to develop and parametrize OpenProdoc. We must distinguish three large blocks, in turn divided into two or three each:

1- Use or integration of OpenProdoc
   a. By API Java
   b. By REST services
2- Extension/plugins of OpenProdoc
   a. Repositories
   b. Authenticators
   c. Tasks
3- Parametrizations
   a. Ribbon/Toolbars
   b. Reports
   c. OPAC (Online Public Access Catalog)
   d. OPAD (Online Public Access Donations, Collaborations and Contributions)

The first ones are designed for the integration of different applications and user interfaces with OpenProdoc. The Java API is the most powerful and offers ALL the functionality of OpenProdoc, not only the management of documents and folders but the management of users, object definitions, groups, ACL, etc. The REST API it is a simpler one and offer the most used functions needed for managing documents, folders and thesauri and, being a pure REST API, can be called from any language (JavaScript, Python, php, etc.) including the user interface.

The use of the API Java or REST allows developing projects such as creating a new user interface (for example Widgets that present some specific elements such as a list of newly received documents), processes that carry out massive operations (updating a set of documents from a list of data received) or integration with other applications (for example when registering a client in an accounting system, automatically create a folder in OpenProdoc with its data).

Extensions allow you to expand the internal operation of OpenProdoc. You can define new types of repository (to save documents in a destination other than those already included in OpenProdoc), create connectors to authenticate users against your own system or create new tasks to be run when an event (as uploading or deleting a document) triggers the task or scheduled with a specified frequency and time.

The parametrizations do not require programming, simply the definition of some text files using specific expressions and reserved words, and allow the creation of searches, reports and data extraction from documents and folders.

The Reports allow to present metadata of sets of documents, in any “text” format (understood “text” as any editable format with a text editor, which includes html, xml, txt, etc.).

OPACs are customized forms to personalize, simplify and guide the search of documents or folders to users, even without having to be registered in the system, which allows publishing information to the world maintaining security at the same time.

OPADs are customized forms to personalize, simplify and guide the donation and contribution of documents and folders to external users, allowing the collaboration of internal users with external users without the need of being everybody OpenProdoc users, but maintaining the security

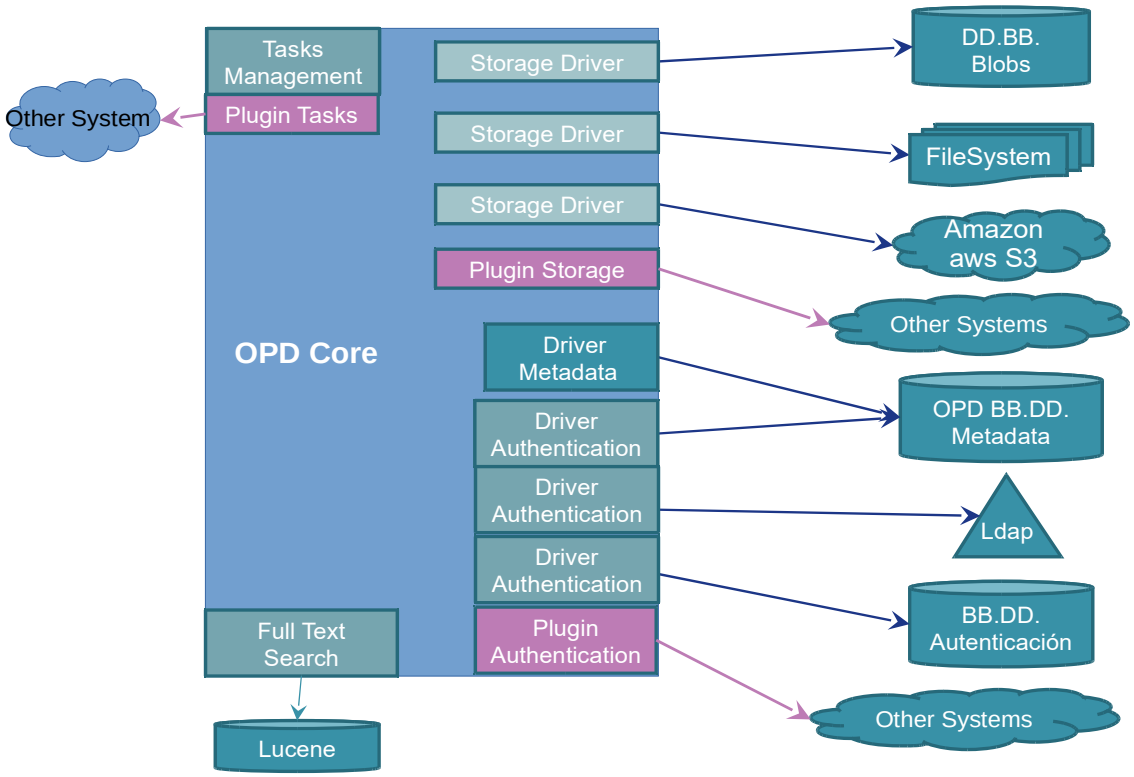
The following sections detail the different alternatives and include examples.

A complete development package containing all the documentation (including this document), specifications, examples and Javadoc can be downloaded in a compressed file from: OpenProdoc Git.

This documentation for development requires OpenProdoc v 3.0.2 or later.

2 Architecture

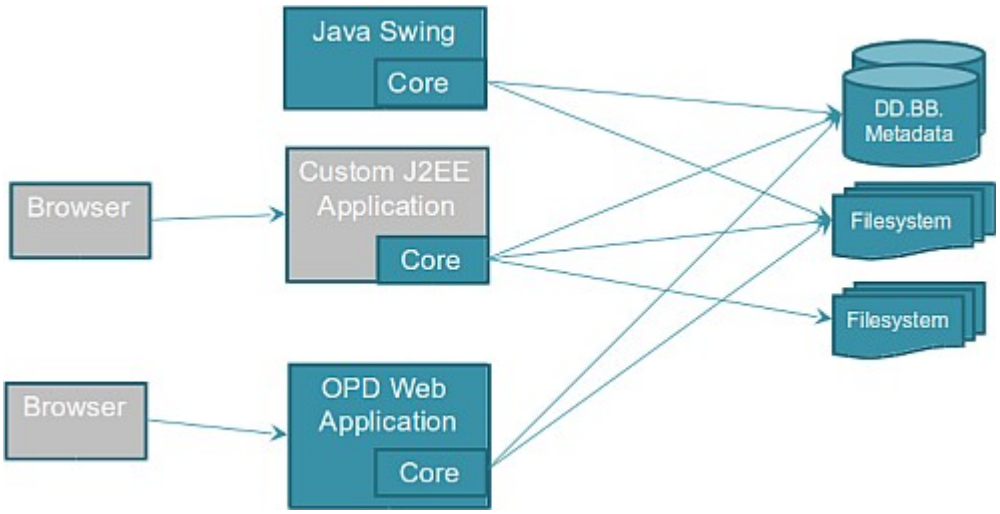
Previously to any development, it’s important to understand the internal architecture of OpenProdoc in order to know where and when every operation is run.



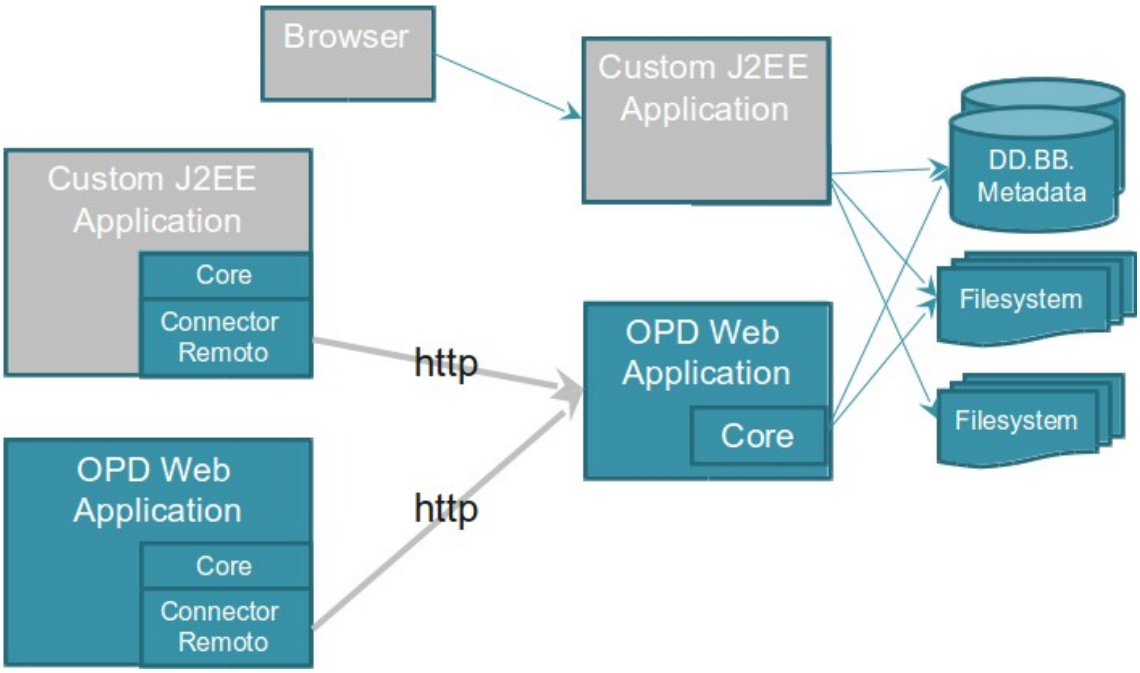
The OpenProdoc architecture is based in a Core component (Prodoc.jar) that is responsible of all functionality. This component is also a Java API that publish all the

functionality of OpenProdoc (with the exception of the user interface) and is used by the thick client, the web application and the REST API. The core is responsible of managing the security, documents storage, metadata, etc. Internally is structured in a set of connectors/plugins of different families (storage, security, etc.)

The core can be embedded in a Java application (as in the Swing OpenProdoc Client). Also it can be embedded in a J2EE application (as in the OpenProdoc Web Client). So more than a traditional API that calls to a remote server, with the core you can embed a complete "OpenProdoc server" in your application.



However the usual way to use the core will be connecting to a remote server.



The way in which the core works depend on the Prodoc.properties file. If the kind of connection defined in the properties file is REMOTE, the core will try to connect to an OpenProdoc Server publishing its internal "proxy" by http. Example:

```
# The Kind of connection to data (JDBC, Remote, ...)
PD.DATA_TYPE=Remote
# URL form conection
PD.DATA_URL=http://localhost:8080/ProdocWeb2/Oper
# connection user
PD.DATA_USER=
# connection password
PD.DATA_PASSWORD=
```

If the connection in the properties is JDBC the core will connected DIRECTLY with the database containing the metadata of the documents and with the repository (or repositories) containing the document binaries. Example:

```
# The Kind of connection to data (JDBC, Remote, ...
PD.DATA_TYPE=JDBC
# URL form conection
PD.DATA_URL=jdbc:hsqldb:file:DB/OPD;hsqldb.default_table_type=cached;
# connection user
PD.DATA_USER=Prodoc
# Encrypted connection password to database
PD.DATA_PASSWORD=0612071149111211
```

In a development using the core (Java API) the code will be running in the computer where the development is deployed. However, a plugin (in example a repository plugin) will run always in the OpenProdoc server.

Additionally, the Prodoc.properties will allow to activate locally the execution of certain tasks so if a core found in the properties file the parameters:

```
# Category of task to generate and execute in this computer
PD.TaskCategory=Export
# pooling frecuency for generation in miliseconds
PD.TaskSearchFreq=300000
# pooling frecuency for execution  in miliseconds
PD.TaskExecFreq=300000
```

Will run locally the tasks of type Export (or all the tasks if TaskCategory is equal to character “*”). If the tasks are of type “custom”, will download the jar from the server before starting the tasks (See Tasks).

3 Development Environment

The best way for developing with OpenProdoc is to download the portable version from the web (http://jhierrot.github.io/openprodoc/) and do all the tests in same computer used for developing.

That way you can have complete independency of others projects or developers doing changes and accessing to OpenProdoc, can review the logs and results of just your actions and control completely the environment. If you need to share definitions and documents, you can export them from one OpenProdoc installation and import your local copy. And being a portable

system, if you need to start with a clean environment, or to create a second environment for comparing behaviour, just unzip again the installation pack.

Of course you can access to a common server of OpenProdoc, shared with other developers and departments. Each project and company can have different needs and an optimal way to work. OpenProdoc just offer different alternatives.

The documentation included in the installation package details how to manage all the functionality of OpenProdoc (start, stop, user management, document and folders management, etc.).

Except for developing with the API REST, you must include the file Prodoc.jar (included in the *".\webapps\ProdocWeb2\WEB-INF\lib*" folder) as dependency for compilation for any other kind of development.

Regarding the running and deployment, for integration projects using the OpenProdoc API you must include, besides Prodoc.jar, all the files included in the *".\webapps\ProdocWeb2\WEB-INF\lib*" folder.

For extension projects, as they run in the OpenProdoc server, you don't need to include any OpenProdoc jar.

4 Integration using the APIs

The API REST is part of the OpenProdoc server, can be deployed in one or more nodes (for High Availability) and publish a set of methods/services that can be called from any language supporting the REST standard (JavaScript, Java, Python, php, .Net, etc.). All the usual needs (CRUD) related to documents, folders and Thesauri-controlled metadata are included.

The API Java (the OpenProdoc core) includes all the functionality available in OpenProdoc (including managing of users, groups, repositories, object definition, tasks, etc. All the functionality of any user interface of OpenProdoc has been developed using the Java API, even the installation of the product.

4.1 API REST

The API REST is a set of services/methods following the standard REST. It is structured in four groups of services:

1- Login/session Services
2- Folder Services
3- Document Services
4- Thesauri Services

The Login Services include the methods:

- **Login**: that must be called before any other method, in order to authenticate, create a session and return a JWT token for later calls and reconnection in the same or other node.
- **Logout**: for release of resources and connections when the session has finished. In order to improve the performance and stability, it's recommended to use this method although the API REST releases sessions automatically after some time without use.

The Folder Services include the methods:

- **Insert**: Creates a folder of the specified Folder Type, with metadata indicated and under the folder specified.
- **UpdateById**: Updates a Folder specifying the unique identifier of the folder (and the new metadata)
- **UpdateByPath**: Updates a Folder specifying the path of the folder in OpenProdoc (and the new metadata)
- **GetById**: Returns the metadata of a Folder specifying the unique identifier of the folder.
- **GetByPath**: Returns the metadata of a Folder specifying the path of the folder in OpenProdoc.
- **DeleteById**: Deletes a Folder specifying the unique identifier of the folder.
- **DeleteByPath**: Deletes a Folder specifying the path of the folder in OpenProdoc.
- **GetSubFoldersById**: Returns the metadata of the list of subfolders of a Folder specifying the unique identifier of the folder.
- **GetSubFoldersByPath**: Returns the metadata of the list of subfolders of a Folder specifying the path of the folder in OpenProdoc.
- **GetContainedDocumentsById**: Returns the metadata of the list of documents contained in a Folder specifying the unique identifier of the folder.
- **GetContainedDocumentsByPath**: Returns the metadata of the list of documents contained in a Folder specifying the path of the folder in OpenProdoc.
- **Search**: Returns the specified metadata of the folders of the specified type that match the specified criteria, using a subset of SQL.

The Document Services include the methods:

- **Insert**: Creates a document with metadata indicated, the binary content included and under the folder specified.
- **UpdateById**: Creates a new version of the with metadata indicated, the binary content included and the version label specified
- **MetadataGetById**: Returns the metadata of a document specifying the unique identifier of the document.

- **ContentGetById**: Returns the content/binary of a document specifying the unique identifier of the document.
- **DeleteById**: Deletes a document specifying the unique identifier of the document.
- **Search**: Returns the specified metadata of the document that match the specified criteria, using a subset of SQL.

Thesauri Services include the methods:

- **Insert**: Creates a Thesaurus term with metadata indicated and under term specified.
- **UpdateById**: Updates a thesaurus term specifying the unique identifier of the term (and the new metadata)
- **GetById**: Returns the metadata of a term specifying the unique identifier of the term.
- **DeleteById**: Deletes a thesaurus term specifying the unique identifier of the folder.
- **GetNarrowTermsById**: Returns the metadata of the list of Narrow (sub terms) of a term specifying the unique identifier.
- **Search**: Returns the specified metadata of the terms that match the specified criteria, using a subset of SQL.

Of course, all the methods are restricted by the ACLs (authorization) assigned to the object (or related objects) and the logged user, so you can't delete a document if the user of the session doesn't have delete permissions over the document or you can´t insert a document or folder under a parent folder if the user of the session doesn't have write permissions on the parent folder.

A formal definition of the API REST in Postman format is included in the downloaded development package with the name "*OPD API REST.postman_collection.json*".

4.1.1 Development

It is possible to develop an application that uses the OpenProdoc API REST with any language/tool (Java, JavaScript, .Net, php, Python, etc.) with capabilities for calling REST methods. Some examples are included in this documentation.

Usually the application will call to the Login method for authentication and creation of a "session", The Login method will return a token that can be used for next calls. The token also allows, if the installation has several nodes/J2EE servers with the REST API deployed, to reconnect in another node if the assigned (by using Sticky Sessions) node stops working.

It's recommended to use Sticky sessions because due the nature of the document management, the system needs to load a lot the information when the user do the login. The reason is that the most of the laws and recommendations about document management

requires the operation to use a nominal/real user, not a generic user, and each user can have permissions different over EACH document and EACH folder.

After login, using the returned token as authorization header, the application can call different methods, and when the user disconnects, it is recommended to call the logout method, in order to free resources. Although the API REST will free automatically the resources after some time, the performance of the complete system will be better.

Even if the user has activity, in order to improve the security, the returned token will expire automatically after less than 24 hours and a new login it's required for creating a new token.

Usually, all the methods return OK=200 status for a success operation, with additional information when required (Id of new element, required metadata, etc.). Any answer different from 200 should be managed as an error.

When the user is not logged, the API will return a 401 Status.

A malformed (by syntax or required elements) Json entry will be responded with a 400 status.

Any other error will return a 500 error, with a message describing the error.

```
{
   "Res": "KO",
   "Msg": "Text of Error"
}
```

4.1.1.1 Login/session Services

Login	
HTTP Method	PUT
Url	http://server:port/ProdocWeb2/APIRest/session
Headers	Content-Type: application/json Accept: application/json

Body	{
	"Name": "*User Name*",
	"Password": "*Password*"
	}

Status Response	200 = OK, 401 = Unauthorized, 500 = Internal error

Body Response	{
	"Res": "OK",
	"Token": "*Created JWT Token*"
	}
	Or
	{
	"Res": "KO",
	"Msg": "Unauthorized"

	}
Comments	

Logout	
HTTP Method	DELETE
Url	http://server:port/ProdocWeb2/APIRest/session
Headers	Accept: application/json Authorization: Bearer *Token_received_in_login*
Body	N/A
Status Response	200 = OK, 401 = Unauthorized, 500 = Internal error

Body Response	{
	"Res": "OK",
	"Msg": "Closed"
	}

Comments	

4.1.1.2 Folder Services

Insert Folder	
HTTP Method	POST
Url	http://server:port/ProdocWeb2/APIRest/folders
Headers	Content-Type: application/json Accept: application/json Authorization: Bearer *Token_received_in_login*

Body	{
	"Id": "*Optional Id*",
	"Name": "*Name of new Folder*",
	"ACL": "*Optional ACL*",
	"Idparent": "*Id of Parent Folder*",
	"Type": "*Folder Type*",
	"ListAttr":[
	{"Name":"*Attribute1*", "Type": "*Physical type*", "Values":["*Value1*"]},
	{"Name":"*Attribute2*", "Type": "*Physical type*", "Values":["*Value2*", "*Value 3*"]},
	. . .
	{"Name":"*AttributeN*", "Type": "*Physical type*", "Values":["*Value N*"]}
	]
	}

Status Response	200 = OK, 401 = Unauthorized, 500 = Internal error

Body Response	{
	"Res": "OK",
	"Msg": "Created=*IdNewFolder*"
	}

Comments	• The allowed types of metadata/attribute are: ○ String, Date, Integer, TimeStamp, Decimal, Boolean, Thesaur. • ListAttr should include all metadata defined for the Folder type. For PD_FOLDERS,

	no other information is needed. • For Thesaur fields, the Id of Term is needed. • For date, timestamp, the allowed input/output formats are: o "yyyy-MM-dd", "yyyy-MM-dd HH:mm:ss"

Update Folder By Id	
HTTP Method	PUT
Url	http://server:port/ProdocWeb2/APIRest/folders/ById/*{IdUpdatedFolder}*
Headers	Content-Type: application/json Accept: application/json Authorization: Bearer *Token_received_in_login*

Body	{
	"Name": "*Name of Updated Folder*",
	"ACL": " *Optional Updated ACL*",
	"ListAttr":[
	{"Name":"*Attribute1*", "Type": "*Physical type*", "Values":["*Value1*"]},
	{"Name":"*Attribute2*", "Type": "*Physical type*", "Values":["*Value2*", "*Value 3*"]},
	. . .
	{"Name":"*AttributeN*", "Type": "*Physical type*", "Values":["*Value N*"]}
	]
	}

Status Response	200 = OK, 401 = Unauthorized, 500 = Internal error

Body Response	{
	"Res": "OK",
	"Msg": "Updated=*IdUpdatedFolder*"
	}

Comments	• The allowed types of metadata/attribute are: o String, Date, Integer, TimeStamp, Decimal, Boolean, Thesaur. • ListAttr should include all metadata to be updated defined for the Folder type. For PD_FOLDERS, no other information is needed. • For Thesaur fields, the Id of Term is needed. • For date, timestamp, the allowed input/output formats are: o "yyyy-MM-dd", "yyyy-MM-dd HH:mm:ss"

Update Folder By Path	
HTTP Method	PUT
Url	http://server:port/ProdocWeb2/APIRest/folders/ByPath/*{PathUpdatedFolder}*
Headers	Content-Type: application/json Accept: application/json Authorization: Bearer *Token_received_in_login*
Body	{ "Name": "*Name of Updated Folder*", "ACL": "*Optional Updated ACL*", "ListAttr":[{"Name":"*Attribute1*", "Type": "*Physical type*", "Values":["*Value1*"]}, {"Name":"*Attribute2*", "Type": "*Physical type*", "Values":["*Value2*", "*Value 3*"]},

	. . . {"Name":"*AttributeN*", "Type": "*Physical type*", "Values":["*Value N*"]}] }
Status Response	200 = OK, 401 = Unauthorized, 500 = Internal error

Body Response	{
	"Res": "OK",
	"Msg": "Updated=*IdUpdatedFolder*"
	}

Comments	• The {path} is the absolute path to the folder: Example: o http:// localhost:8080/ProdocWeb2/APIRest/folders/ById/***System/OPAC/Temp*** • The allowed types of metadata/attribute are: o String, Date, Integer, TimeStamp, Decimal, Boolean, Thesaur. • ListAttr should include all metadata to be updated defined for the Folder type. For PD_FOLDERS, no other information is needed. • For Thesaur fields, the Id of Term is needed. • For date, timestamp, the allowed input/output formats are: o "yyyy-MM-dd", "yyyy-MM-dd HH:mm:ss"

Get Metadata Folder By Id	
HTTP Method	GET
Url	http://server:port/ProdocWeb2/APIRest/folders/ById/*{IdFolder}*
Headers	Accept: application/json Authorization: Bearer *Token_received_in_login*
Body	N/A
Status Response	200 = OK, 401 = Unauthorized, 500 = Internal error
Body Response	
Comments	• For date, timestamp, the allowed input/output formats are: o "yyyy-MM-dd", "yyyy-MM-dd HH:mm:ss"

Get Metadata Folder By Path	
HTTP Method	GET
Url	http://server:port/ProdocWeb2/APIRest/folders/ByPath/*{PathFolder}*
Headers	Accept: application/json Authorization: Bearer *Token_received_in_login*
Body	N/A
Status Response	200 = OK, 401 = Unauthorized, 500 = Internal error
Body Response	
Comments	• The {path} is the absolute path to the folder: Example: o http:// localhost:8080/ProdocWeb2/APIRest/folders/ById/***System/OPAC/Temp*** • For date, timestamp, the allowed input/output formats are: o "yyyy-MM-dd", "yyyy-MM-dd HH:mm:ss"

Delete Folder By Id

HTTP Method	DELETE
Url	http://server:port/ProdocWeb2/APIRest/folders/ById/*{IdFolder}*
Headers	Accept: application/json Authorization: Bearer *Token_received_in_login*
Body	N/A
Status Response	200 = OK, 401 = Unauthorized, 406=No permissions, 500 = Internal error

Body Response	{
	"Res": "OK",
	"Msg": "Deleted=*IdDeletedFolder*"
	}
	Or
	{
	"Res": "KO",
	"Msg": "User_without_permissions_over_folder"
	}

Comments	

Delete Folder By Path	
HTTP Method	DELETE
Url	http://server:port/ProdocWeb2/APIRest/folders/ByPath/*{PathFolder}*
Headers	Accept: application/json Authorization: Bearer *Token_received_in_login*
Body	N/A
Status Response	200 = OK, 401 = Unauthorized, 406=No permissions , 500 = Internal error

Body Response	{
	"Res": "OK",
	"Msg": "Deleted=*IdDeletedFolder*"
	}

Comments	• The {path} is the absolute path to the folder: Example: ○ http:// localhost:8080/ProdocWeb2/APIRest/folders/ById/***System/OPAC/Temp***

Get SubFolders of Folder By Id	
HTTP Method	GET
Url	http://server:port/ProdocWeb2/APIRest/folders/SubFoldersById/*{IdFolder}*?Initial=0&Final=200
Params	Initial: First Element of the list to be returned (Included) [Optional] Final: Last Element to be returned (Excluded) [Optional]
Headers	Accept: application/json Authorization: Bearer *Token_received_in_login*
Body	N/A
Status Response	200 = OK, 401 = Unauthorized, 406=No permissions, 500 = Internal error
Body Response	
Comments	

Get SubFolders of Folder By Path	
HTTP Method	GET
Url	http://server:port/ProdocWeb2/APIRest/folders/SubFoldersByPath/*{PathFolder}*?Initial=0&Final=2
Params	Initial: First Element of the list to be returned (Included) [Optional] Final: Last Element to be returned (Excluded) [Optional]
Headers	Accept: application/json Authorization: Bearer *Token_received_in_login*
Body	N/A
Status Response	200 = OK, 401 = Unauthorized, 406=No permissions , 500 = Internal error
Body Response	
Comments	• The {path} is the absolute path to the folder: Example: ○ http:// localhost:8080/ProdocWeb2/APIRest/folders/ById/***System/OPAC/Temp***

Get Contained Documents of Folder By Id	
HTTP Method	GET
Url	http://server:port/ProdocWeb2/APIRest/folders/ContDocsById/*{IdFolder}*?Initial=0&Final=200
Params	Initial: First Element of the list to be returned (Included) [Optional] Final: Last Element to be returned (Excluded) [Optional]
Headers	Accept: application/json Authorization: Bearer *Token_received_in_login*
Body	N/A
Status Response	200 = OK, 401 = Unauthorized, 406=No permissions, 500 = Internal error
Body Response	
Comments	

Get Contained Documents of Folder By Path	
HTTP Method	GET
Url	http://server:port/ProdocWeb2/APIRest/folders/ContDocsByPath/*{PathFolder}*?Initial=0&Final=2
Params	Initial: First Element of the list to be returned (Included) [Optional] Final: Last Element to be returned (Excluded) [Optional]
Headers	Accept: application/json Authorization: Bearer *Token_received_in_login*
Body	N/A
Status Response	200 = OK, 401 = Unauthorized, 406=No permissions , 500 = Internal error
Body Response	
Comments	• The {path} is the absolute path to the folder: Example: ○ http:// localhost:8080/ProdocWeb2/APIRest/folders/ById/***System/OPAC/Temp***

Search Folders	
HTTP Method	POST
Url	http://server:port/ProdocWeb2/APIRest/folders/Search

Headers	Content-Type: application/json Accept: application/json Authorization: Bearer *Token_received_in_login*

Body	{
	"Query": "*OpenProdoc Query*",
	"Initial": "*Optional Initial entry to be returned (included)*",
	"Final": "*Optional Final entry to be returned (excluded)*"
	}

Status Response	200 = OK, 401 = Unauthorized, 500 = Internal error
Body Response	List of Metadata in JSON format
Comments	• The allowed types of metadata/attribute are: ○ String, Date, Integer, TimeStamp, Decimal, Boolean, Thesaur. • For Thesaur fields, the VALUE of Term is returned. • For date, timestamp, the allowed input/output formats are: ○ "yyyy-MM-dd", "yyyy-MM-dd HH:mm:ss"

4.1.1.3 Document Services

Insert Document	
HTTP Method	POST
Url	http://server:port/ProdocWeb2/APIRest/documents
Headers	Content-Type: application/json Accept: application/json Authorization: Bearer *Token_received_in_login*

Body	Multipart Form-data: with two “fields”:
	• Metadata: Jason with the metadata of Document
	• Binary: File to upload:
	
	Metadata:
	{
	"Id": "*Optional Id*",
	"Title": "*Title of new Document*",
	"ACL": "*Optional ACL*",
	"Idparent": "*Id of Parent Folder*",
	"Type": "*Document Type*",
	“DocDate”: “*Date of Document*”,
	"ListAttr":[
	{"Name":"*Attribute1*", "Type": "*Physical type*", "Values":["*Value1*"]},
	{"Name":"*Attribute2*", "Type": "*Physical type*", "Values":["*Value2*", “*Value 3*”]},
	. . .
	{"Name":"*AttributeN*", "Type": "*Physical type*", "Values":["*Value N*"]}
	]
	}

Status Response	200 = OK, 401 = Unauthorized, 500 = Internal error
Body Response	{

	"Res": "OK", "Msg": "Created=*IdNewDocument*" }
Comments	• The allowed types of metadata/attribute are: o String, Date, Integer, TimeStamp, Decimal, Boolean, Thesaur. • ListAttr should include all metadata defined for the Folder type. For PD_FOLDERS, no other information is needed. • For Thesaur fields, the Id of Term is needed. • For date, timestamp, the allowed input/output formats are: o "yyyy-MM-dd", "yyyy-MM-dd HH:mm:ss"

Update Document	
HTTP Method	PUT
Url	http://server:port/ProdocWeb2/APIRest/documents/ById/*{IdDocument}*
Headers	Content-Type: application/json Accept: application/json Authorization: Bearer *Token_received_in_login*

Body	Multipart Form-data: with two “fields”:
	• Metadata: Jason with the metadata of Document
	• Binary: File to upload:
	
	Metadata:
	{
	"VerLabel": "*Required Version Label (Ex. 1.2, 2.0,..)*",
	"Title": "*Title of new Document*",
	"ACL": "*Optional ACL*",
	“DocDate”: “*Date of Document*”,
	"ListAttr":[
	{"Name":"*Attribute1*", "Type": "*Physical type*", "Values":["*Value1*"]},
	{"Name":"*Attribute2*", "Type": "*Physical type*", "Values":["*Value2*", “*Value 3*”]},
	. . .
	{"Name":"*AttributeN*", "Type": "*Physical type*", "Values":["*Value N*"]}
	]
	}

Status Response	200 = OK, 401 = Unauthorized, 500 = Internal error

Body Response	{
	"Res": "OK",
	"Msg": "Updated=*IdNewDocument*"
	}

Comments	• The method will do a CheckOut, Update and CheckIn of the document, creating a new version. • The allowed types of metadata/attribute are: o String, Date, Integer, TimeStamp, Decimal, Boolean, Thesaur. • ListAttr should include all metadata defined for the Folder type. For PD_FOLDERS, no other information is needed. • For Thesaur fields, the Id of Term is needed. • For date, timestamp, the allowed input/output formats are: o "yyyy-MM-dd", "yyyy-MM-dd HH:mm:ss"

Get Metadata Document By Id	
HTTP Method	GET
Url	http://server:port/ProdocWeb2/APIRest/documents/ById/*{IdDoc}*
Headers	Accept: application/json Authorization: Bearer *Token_received_in_login*
Body	N/A
Status Response	200 = OK, 401 = Unauthorized, 500 = Internal error
Body Response	Metadata of Document
Comments	• For date, timestamp, the allowed input/output formats are: ○ "yyyy-MM-dd", "yyyy-MM-dd HH:mm:ss"

Get Content of Document By Id	
HTTP Method	GET
Url	http://server:port/ProdocWeb2/APIRest/documents/ContentById/*{IdDoc}*
Headers	Accept: application/octet-stream Authorization: Bearer *Token_received_in_login*
Body	N/A
Status Response	200 = OK, 401 = Unauthorized, 500 = Internal error
Body Response	Binary element of the document
Comments	The method will return the binary content, the actual mimetype and a suggested file name.

Delete Folder By Id	
HTTP Method	DELETE
Url	http://server:port/ProdocWeb2/APIRest/documents/ById/*{IdDoc}*
Headers	Accept: application/json Authorization: Bearer *Token_received_in_login*
Body	N/A
Status Response	200 = OK, 401 = Unauthorized, 406=No permissions, 500 = Internal error

Body Response	{
	"Res": "OK",
	"Msg": "Deleted=*IdDeletedDocument*"
	}
	Or
	{
	"Res": "KO",
	"Msg": "User_without_permissions_over_Document"
	}

Comments	

Search Documents	
HTTP Method	POST
Url	http://server:port/ProdocWeb2/APIRest/documents/Search
Headers	Content-Type: application/json Accept: application/json Authorization: Bearer *Token_received_in_login*

Body	{
	"Query": "*OpenProdoc Query*",
	"Initial": "*Optional Initial entry to be returned (included)*",
	"Final": "*Optional Final entry to be returned (excluded)*"
	}

Status Response	200 = OK, 401 = Unauthorized, 500 = Internal error
Body Response	List of Metadata in JSON format
Comments	• The allowed types of metadata/attribute are: ○ String, Date, Integer, TimeStamp, Decimal, Boolean, Thesaur. • For Thesaur fields, the VALUE of Term is returned. • For date, timestamp, the allowed input/output formats are: ○ "yyyy-MM-dd", "yyyy-MM-dd HH:mm:ss"

4.1.1.4 Thesauri Services

Insert Term	
HTTP Method	POST
Url	http://server:port/ProdocWeb2/APIRest/thesauri
Headers	Content-Type: application/json Accept: application/json Authorization: Bearer *Token_received_in_login*

Body	{
	"Name": "*Name of Term*",
	"Descrip": "*Description of Term* ",
	"Lang": "*Language Code (EN, ES, PT,..)*",
	"SCN": "*Optional Scope Note*",
	"ParentId": "*Id of Parent Term (or Thesaurus for top terms*"
	}

Status Response	200 = OK, 401 = Unauthorized, 500 = Internal error

Body Response	{
	"Res": "OK",
	"Msg": "Created=*IdNewTerm*"
	}

Comments	

Update Term	
HTTP Method	PUT
Url	http://server:port/ProdocWeb2/APIRest/thesauri/ById/{IdTerm}

Headers	Content-Type: application/json Accept: application/json Authorization: Bearer *Token_received_in_login*

Body	{
	"Name": "*Name of Term*",
	"Descrip": "*Description of Term* ",
	"Lang": "*Language Code (EN, ES, PT,..)*",
	"SCN": "*Optional Scope Note*",
	}

Status Response	200 = OK, 401 = Unauthorized, 500 = Internal error

Body Response	{
	"Res": "OK",
	"Msg": "Updated=*IdUpdTerm*"
	}

Comments	

Delete Term	
HTTP Method	DELETE
Url	http://server:port/ProdocWeb2/APIRest/thesauri/ById/{IdTerm}
Headers	Accept: application/json Authorization: Bearer *Token_received_in_login*
Body	N/A
Status Response	200 = OK, 401 = Unauthorized, 500 = Internal error

Body Response	{
	"Res": "OK",
	"Msg": "Deleted=*IdDelTerm*"
	}

Comments	

Get Metadata Term	
HTTP Method	GET
Url	http://server:port/ProdocWeb2/APIRest/thesauri/ById/{IdTerm}
Headers	Accept: application/json Authorization: Bearer *Token_received_in_login*
Body	N/A
Status Response	200 = OK, 401 = Unauthorized, 500 = Internal error
Body Response	JSON with metadata
Comments	

Get SubTerms of Term	
HTTP Method	GET
Url	http://server:port/ProdocWeb2/APIRest/thesauri/SubThesById/{IdTerm}?Initial=0&Final=2
Params	Initial: First Element of the list to be returned (Included) [Optional] Final: Last Element to be returned (Excluded) [Optional]
Headers	Accept: application/json

	Authorization: Bearer *Token_received_in_login*
Body	N/A
Status Response	200 = OK, 401 = Unauthorized, 500 = Internal error
Body Response	JSON with metadata
Comments	

Search Terms	
HTTP Method	POST
Url	http://server:port/ProdocWeb2/APIRest/ thesauri/Search
Headers	Content-Type: application/json Accept: application/json Authorization: Bearer *Token_received_in_login*

Body	{
	"Query": "*OpenProdoc Query*",
	"Initial": "*Optional Initial entry to be returned (included)*",
	"Final": "*Optional Final entry to be returned (excluded)*"
	}

Status Response	200 = OK, 401 = Unauthorized, 500 = Internal error
Body Response	List of Metadata in JSON format
Comments	

4.1.2 Examples

4.1.2.1 Examples in JavaScript:

Login:

```
var data = JSON.stringify({
  "Name": "User",
  "Password": "Password"
});
var xhr = new XMLHttpRequest();
xhr.withCredentials = true;
xhr.addEventListener("readystatechange", function () {
  if (this.readyState === 4) {
    console.log(this.responseText);
  }
});
xhr.open("PUT", "http://localhost:8080/ProdocWeb2/APIRest/session");
xhr.setRequestHeader("Content-Type", "application/json");
xhr.setRequestHeader("Accept", "application/json");
xhr.setRequestHeader("cache-control", "no-cache");
xhr.send(data);
```

Logout:

```
var data = null;
```

```
var xhr = new XMLHttpRequest();
xhr.withCredentials = true;
xhr.addEventListener("readystatechange", function () {
  if (this.readyState === 4) {
    console.log(this.responseText);
  }
});
xhr.open("DELETE",
"http://localhost:8080/ProdocWeb2/APIRest/session");
xhr.setRequestHeader("Accept", "application/json");
xhr.setRequestHeader("Authorization", "Bearer eyJhbiOiiJ9.eyJzdiJBb");
xhr.setRequestHeader("cache-control", "no-cache");
xhr.send(data);
```

Insert Folder:

```
var data = JSON.stringify({
  "Id": "",
  "Name": "New Course Maths",
  "ACL": "Public",
  "Idparent": "16ac20d9415-3fedb58bdb94afd4",
  "Type": "Course",
  "ListAttr": [
    {"Name": "Classroom",
      "Type": "String",
      "Values": ["Aula 1"]},
    {"Name": "StartDate",
      "Type": "Date",
      "Values": ["1980-11-12"]},
    {"Name": "Subject",
      "Type": "String",
      "Values": ["Maths"]},
    {"Name": "Teacher",
      "Type": "String",
      "Values": ["Ana María"]}
  ]
});
var xhr = new XMLHttpRequest();
xhr.withCredentials = true;
xhr.addEventListener("readystatechange", function () {
  if (this.readyState === 4) {
    console.log(this.responseText);
  }
});
xhr.open("POST", "http://localhost:8080/ProdocWeb2/APIRest/folders");
xhr.setRequestHeader("Content-Type", "application/json");
xhr.setRequestHeader("Accept", "application/json");
xhr.setRequestHeader("Authorization", "Bearer eyJhbiOiiJ9.eyJzdiJBb");
xhr.setRequestHeader("cache-control", "no-cache");
xhr.send(data);
```

Delete Folder by Path:

```
var data = null;
var xhr = new XMLHttpRequest();
xhr.withCredentials = true;
xhr.addEventListener("readystatechange", function () {
  if (this.readyState === 4) {
    console.log(this.responseText);
  }
});
xhr.open("DELETE",
"http://localhost:8080/ProdocWeb2/APIRest/folders/ByPath/2019/Maths/")
;
xhr.setRequestHeader("Accept", "application/json");
xhr.setRequestHeader("cache-control", "no-cache");
xhr.send(data);
```

Search Folder:

```
var data = JSON.stringify({
  "Query": "Select PDId, Title from Course2 where Teacher='Ana
María'",
  "Initial": "0",
  "Final": "100"
});
var xhr = new XMLHttpRequest();
xhr.withCredentials = true;
xhr.addEventListener("readystatechange", function () {
  if (this.readyState === 4) {
    console.log(this.responseText);
  }
});
xhr.open("POST",
"http://localhost:8080/ProdocWeb2/APIRest/folders/Search");
xhr.setRequestHeader("Content-Type", "application/json");
xhr.setRequestHeader("Authorization", "Bearer eyJhbiOiiJ9.eyJzdiJBb");
xhr.setRequestHeader("Accept", "application/json");
xhr.setRequestHeader("cache-control", "no-cache");
xhr.send(data);
```

Insert Doc:

```
var data = new FormData();
data.append("Metadata", "{\"Title\":\"New
Document\",\"ACL\":\"Public\",\"Idparent\":\"15db8ae908c-
3fe9b47f5f572394\",\"Type\":\"ECM_Standards\",\"DocDate\":\"1999-09-
22\",\"ListAttr\":
[{\"Name\":\"Author\",\"Type\":\"String\",\"Values\":[\"Committee on
Descriptive Standards\"]},
{\"Name\":\"CreativeCommons\",\"Type\":\"Boolean\",\"Values\":
[\"0\"]},{\"Name\":\"DocScope\",\"Type\":\"String\",\"Values\":
[\"World\"]},{\"Name\":\"DocCode\",\"Type\":\"String\",\"Values\":
[\"ISBN 0-9696035-5-X\"]},
{\"Name\":\"Keywords\",\"Type\":\"String\",\"Values\":[\"Archival
materials--Standards\",\"International Council on Archives\"]}]}");
```

```
data.append("Binary", "/D:/Upload/Document.pdf");
var xhr = new XMLHttpRequest();
xhr.withCredentials = true;
xhr.addEventListener("readystatechange", function () {
  if (this.readyState === 4) {
    console.log(this.responseText);
  }
});
xhr.open("POST",
"http://localhost:8080/ProdocWeb2/APIRest/documents");
xhr.setRequestHeader("Content-Type", "application/json");
xhr.setRequestHeader("Authorization", "Bearer eyJhbiOiiJ9.eyJzdiJBb");
xhr.setRequestHeader("Accept", "application/json");
xhr.setRequestHeader("cache-control", "no-cache");
xhr.send(data);
```

Download Document:

```
var data = null;
var xhr = new XMLHttpRequest();
xhr.withCredentials = true;
xhr.addEventListener("readystatechange", function () {
  if (this.readyState === 4) {
    console.log(this.responseText);
  }
});
xhr.open("GET",
"http://localhost:8080/ProdocWeb2/APIRest/documents/ContentById/15db8a
e914a-3fc4a9313b06b040");
xhr.setRequestHeader("Accept", "application/octet-stream");
xhr.setRequestHeader("Authorization", "Bearer eyJhbiOiiJ9.eyJzdiJBb");
xhr.setRequestHeader("cache-control", "no-cache");
xhr.send(data);
```

4.1.3 Deployment

Being a REST services you need to deploy your developed application with the tools specific for the selected language/technology, to have installed the OpenProdoc Server application in one or more nodes and to point your application to the url in which the nodes with OpenProdoc Server are running.

If you want to install two or more nodes of OpenProdoc Server for High Availability and scalability, you will need some kind of load balancer. For better performance it’s recommended to define Sticky Sessions, so always the same node answer to the same client/user. This saves resources by reusing the previous session and data. Anyway, the OpenProdoc API REST will reconnect in another node (using the JWT token) if one node falls.

4.2 API Java

This section explains the general concepts for developing using OpenProdoc Java API.

For development of extensions/plugins, although the general concepts are common, you must check the specific needs in Extensions/Plugins.

A formal definition of the API Java in Javadoc format is included in the downloaded development package in the folder “OpenProdoc Javadoc”.

Although the OpenProdoc API (Prodoc.jar) can be used as a “traditional” API in a “client-server” mode, it must be noted that the API itself is a complete system that can be used for creating a complete server, needing only access to a database (for storing metadata) and a filesystem) for storing the documents. As an example, after “initiated” the API can start several automatics tasks (depending on the values in configuration file), both internals or extensions.

In order to understand how OpenProdoc works, as well as the options included in the properties file, it’s important to review the Architecture section.

The API itself is multithread safe and can be used in a multithread/server environment. However, not all the objects are multithread. In example two thread can ask for a session at the same time and will obtain different session. But each session can’t be shared between different threads.

4.2.1 Development

4.2.1.1 Start & stop OpenProdoc

The first steep when developing with OpenProdoc is to “start” the framework/API, calling the static method:

```
ProdocFW.InitProdoc(“PD”, Path of Prodoc.properties file);
```

The method will read the configuration file, will open the internal number of session defined in the configuration and will start, if activated in the properties file, the automatic tasks, or some of them.

It is allowed to call several times to the InitProdoc method, even at the same time (is internally synchronized). Only the first one will be effective.

When the application ends (or the application server stops), in order to close and free resources and sessions, the application should call the static method:

```
ProdocFW.ShutdownProdoc(“PD”);
```

If not called, anyway OpenProdoc will detect the JVM shutdown and will disconnect and free all the resources.

4.2.1.2 Login and sessions

After starting OpenProdoc the next step will be to obtain a user session, with the static method:

```
DriverGeneric OPDSess = ProdocFW.getSession(“PD”, User, Password);
```

OpenProdoc will read the configuration of the user. If it is active, will read the authenticator assigned (that can be internal one or an authenticator extension) and will call the authenticator.

If the authenticator fails (incorrect user/password, inexistent user, locked user, etc.), it will throw an exception, otherwise will return a session for that user with all the information of the user (role, permissions, ACL, etc.).

It possible to obtain information of the user object from the session with the method:

```
PDUser SessUser=OPDSess.getUser();
```

The user object contains all the information, as name, description, role, etc.

```
System.out.println("User Name:"+SessUser.getName());
System.out.println("User Description:"+SessUser.getDescription());
System.out.println("User Role Name:"+SessUser.getRole());
```

After logged, the user will be able to manipulate the type of elements (documents, user, groups, folders, thesauri, etc.) defined in his role. Additionally, for Documents and Folders the permissions have a smaller granularity, so even if the user is granted in his role the “delete of documents”, he will be able to delete some documents but not others (see the ACL description of the help). If the role or the ACLs change while the user is logged, the new permissions will be effective AFTER the next login.

```
PDRoles UsrRol=SessUser.getRol();
System.out.println("Can create docs:"+UsrRol.isAllowCreateDoc());
System.out.println("Can Mod docs:"+UsrRol.isAllowMaintainDoc());
System.out.println("Can create Folds:"+UsrRol.isAllowCreateFolder());
```

If the user is an “application user”, not a real/human user, several sessions for the same user can be established. Each session will have its own information and transactions.

When the user ends his work, the session must be released with the method.

```
ProdocFW.freeSesion(“PD”, OPDSess);
```

So another user (or instance of the same user in a server) can use the connection.

Is the session is unused too much time, it will be automatically released and can throw an exception if the application try to use it.

The number of simultaneous user in each OpenProdoc node is limited by the parameter of the Prodoc.properties file:

```
# Maximum number of sessions for metadata
PD.DATA_MAX=100
```

.The maximum of concurrent user in all the nodes will be limited by the sum of all the parameters in all the nodes or the limit established in the database.

4.2.1.3 Manipulating objects

In OpenProdoc the API and the functional model is object oriented, with strong use of inheritance and polymorphism. All the elements that can be manipulated are subtypes of a parent class ObjPD and inherit most of the methods and behaviour.

The classes that can be managed are:

- PDDocs: The main class, for managing documents.
- PDFolders: For managing the folders containing documents
- PDThesaur: Storing thesauri and list of terms used in metadata of folders and documents
- PDMimeType: Mime types of the documents uploaded and downloaded.
- PDUser: managing users.
- PDRoles: Roles/permissions of the users.
- PDACL: For managing permissions over the objects for users/groups.
- PDAuthenticators: For authenticating users
- PDGroups: Groups of users and other groups.
- PDObjDefs: Definitions of Documents and folders.
- PDReport: Subtype of Documents used for reporting
- PDRepository: Classes responsable of storing the binary of the documents.

Review the help of the OpenProdoc application for a complete description of use and metadata of each class, as well as limitations in values. The Javadoc describes all the classes and methods in a systematic way.

The way of managing all the objects is consistent:

- The objects can be instantiated or loaded in memory for using their values and calling their methods.
- After assigning values, for saving them the method insert() will store in the database (if the user is granted enough permissions and there are no other limitations, as duplicated indexes, required values or referential integrity).
- After changing values, the method update() will save the changes (with similar restrictions to the insert). ALL the fields in memory are stored with the current values, so if you don’t load and only assign new values to an empty object, non-informed fields will be cleared in database.

- It is possible to delete an object with the delete method (with restrictions similar to insert or update, including referential integrity).
- When there is need of retrieving a set of objects, some methods create a Cursor that retrieve Records that can be used for managing values or instantiating new objects.
- By traceability, all the objects have two fields added automatically: PDAutor and PDDate, that store the name of the session user that do the last insert or update of an object and the timestamp of the event.

Some examples:

Creating a new one:

Instance the object, assigning a session:

```
PDRoles NewRole=new PDRoles(OPDSess);
//Assign values:
// The Name is always identifier/unique key in all the classes
NewRole.setName("SecurityResp");
// other fields will have different restrictions
NewRole.setDescription("Security Responsible");
NewRole.setAllowCreateUser(true);
NewRole.setAllowMaintainUser(true);
NewRole.setAllowCreateGroup(true);
NewRole.setAllowMaintainGroup(true);
NewRole.setAllowCreateAcl(true);
NewRole.setAllowMaintainAcl(true);
// other permissions non set will be default value (false)
//Insert:
NewRole.insert();
```

Modifying an existing one:

Instance the object, assigning a session:

```
PDRoles UpdRole=new PDRoles(OPDSess);
```

Load the current values using the identifier name:

```
UpdRole.Load("SecurityResp");
```

Modify other elements:

```
UpdRole.setAllowCreateRole(true);
UpdRole.setAllowMaintainRole(true);
```

Update

```
UpdRole.update();
```

Deleting an existing one:

```
PDRoles DelRole=new PDRoles(OPDSess);
```

Load the current values using the identifier name:

```
DelRole.Load("SecurityResp");
```

Delete:

```
DelRole.delete();
```

4.2.1.4 Records and Attributes

As a way to generalize the use of objects, and to allow to manage new classes of documents and folders defined for projects, OpenProdoc has Records and Attributes.

An Attribute is class representing metadata, including a Name, a Description, physical type, value and other characteristics.

A Record is a class containing a collection of Attributes.

A Record can be used to manage in a generic way the metadata of any object, to assign metadata to an object, to export information or to manage the results of a search. It's similar to a "row" in JDBC.

You can obtain a Record from a loaded object:

```
PDUser User=new PDUser(OPDSess);
```

Load the current values using the identifier name:

```
User.Load("User1");
```

Retrieve a record:

```
Record Rec= User.getRecord();
```

Assign all the values of a Record to an Object:

```
PDUser User2=new PDUser(OPDSess);
User2.assignValues(Rec);
User2.setname("User1Copy");
User2.insert();
```

Retrieve a specific Attribute of a Record by its Name:

```
Attribute Attr=Rec.getAttr("email");
```

And manipulate de Attribute:

```
String AttrName=Attr.getDescription(); // "eMail"
Attr.setValue("user1@mydomain.com");
System.out.println("Attribute="+Attr);
```

The Records can be managed in a generic mode by:

```
Rec.initList();
for (int i = 0; i < Rec.NumAttr(); i++)
      {
      Attribute Attr=Rec.nextAttr();
      System.out.println("Attribute["+i+"]="+Attr);
      }
```

An Attribute has this main fields with its "getter" and "setter":

```
// Technical/Internal name of the Attribute used in tables, xml,..
private String Name;
// Public name of the Attribute used by user and displayed in forms
private String UserName;
// Description and comments for the Attribute used in tooltips
private String Description;
// Physical type (String, Integer, Date,..)
private int Type;
/* When true, the Attribute must be filled before saving to any Object
(Doc, Folder, User, etc.) */
private boolean Required = false;
// Value of the Atribute
private Object Value = null;
//Max Length for String or number of Thesarus for type tTHES
private int LongStr=0;
// When true the Attribute has unique value in the database
private boolean Unique=false;
/* When true the Attribute can be modified after inserted, otherwise
becomes fixed */
private boolean ModifAllowed=true;
/** When true the Attribute allows to add multiple values (Keywords,
authors, etc.) */
private boolean Multivalued=false;
```

And can have this “physical” type:

```
public static final int tINTEGER  =0;
public static final int tFLOAT    =1; // BigDecimal actually
public static final int tSTRING   =2;
public static final int tDATE     =3; // java.util.date (hour ignored)
public static final int tBOOLEAN  =4;
public static final int tTIMESTAMP=5; // java.util.date
public static final int tTHES     =6; // reference to a thesaurus term
```

The setValue method accepts any object of a subclass of Object as parameter and internally checks the type using the Type of Attribute.

For multivalued Attributes (that is Attributes that allow to manage several values like, as an example, an Attribute “Keywords”), the class offer methods as:

```
AddValue(Object pValue)
TreeSet getValuesList()
```

Usually the Records should be used for generic managing of objects, as import, export, display or cursor result sets managing. For predefined classes, there are getters and setters, however for document types and folder types, the new metadata defined in a project doesn’t have getter and setter and the only way is by means of Records.

Usually when you retrieve a Record and an Attribute from any object, it is a copy, so for changing the original value you should use something like:

```
User.getRecord().getAttr("email").setValue(“user1@mydomain.com”);
```

A Record is just a collection of Attributes and besides the methods showed previously, some important methods are:

- Copy(): Returns a new Record with a copy of all the Attributes
- CopyMono():Returns a new Record with a copy of all the monovalued Attributes
- toXML(): Returns an String of XML representing the Record.
- CreateFromXML(Node AttrsNode): Creates a new Record from a XML org.w3c.dom.Node.

Please check the Javadoc for more information and a complete list of methods.

4.2.1.5 Cursors

Some methods used for searching information return objects of class Cursor. A Cursor is similar to a database cursor (and actually in some scenarios is internally associated a database cursor). With the obtained Cursor, you can travel the set of values, which are returned as Records.

Example:

```
PDUser User=new PDUser(OPDSess);
Cursor ListUsers=null;
Vector<Record> ListRes=new Vector<Record>();
try {
ListUsers=User.SearchLike("Pete"); // Cursor with users names Pete*
Record NextRec= OPDSess.NextRec(ListUsers);
while (NextRec!=null) // travel until NextRec returns null
    {
    // Use the returned values:
    // storing . . .
    ListRes.add(NextRec);
    // or creating a new object
    PDUser UserT=new PDUser(OPDSess);
    UserT.assignValues(NextRec);
    // and use it
    UserT.setRole("Guest");
    UserT.update();
    // . . .
    NextRec= OPDSess.NextRec(ListUsers);
    }
} catch (Exception Ex)
      {
      // Trace and manage exception
      If (PDLog().Error("Process X Error:"+Ex.getLocalizaedmessage());
      }
finally // ALWAYS CLOSE THE CURSOR FOR RELEASING RESOURCES
    {
    if (ListUsers !=null)
       OPDSess.CloseCursor(ListUsers);
    }
```

The structure of the returned Record depends on the object that creates the Cursor and of the specified list of metadata (when using some methods, as SQL search). By default, all the monovalued fields of the object type are returned. When the multivalued values, or other information, is needed, you can assign the retrieved record, or just the identifier (Name for most objects and PDId for Documents and folders), and `Load` the object (`LoadFull` for Documents and folders).

Some of the most used methods that create Cursors are:

- ObjPD.SearchLike(String Name): Creates a Cursor with all the Attributes of all objects whose name is “similar” to Name. Allows wildcards “*”.
- ObjPD. SearchSelect(String SQL): Creates a Cursor according to the received OpenProdoc SQL syntax.
- PDDocs. Search(..): Creates a Cursor with documents Attributes according to the search parameters received.
- PDFolders. Search(..): Creates a Cursor with folders Attributes according to the search parameters received.

Some Cursor methods have “brother” methods with the same name and an additional “V” with the same parameters that return a Vector of Records. Those methods internally call the Cursor method but they go through the Cursor and fill the Vector

4.2.1.6 Folders

The folders are virtual containers of other folders or documents. They are “virtual” because the relationship is implemented by means of some tables in the metadata database and has no relation with the physical way in that the documents are stored in the one or more repositories or Filesystems.

The folders are managed by the class PDFolders, which publish (directly or by inheritance of parent class) all the methods needed for creating, searching or managing folders and its contained elements.

Each folder MUST be contained in a “parent” folder, except a special folder “RootFolder”, which is the “top level” and has no parent. The folders are referenced by its unique identifier PDId and, in some methods can be referenced by its path, with a syntax similar to Linux (that is “/System”, “/Users/root”,...). Two methods: `getIdPath(String FoldName)` and `getPathId(String Id)` are provided for converting from Path to Id and to the contrary.

OpenProdoc includes out-of-the-box one folder type PD_FOLDERS, but it is possible to create all the folder types needed, each one with its own metadata. All the folders are managed with the same Java class PDFolders.

If you didn’t specify a type, the default PD_FOLDERS is implicit:

```
PDFolders Fold=new PDFolders(OPDSess);
```

And you will be able to assign the values using the getters and setters:

```
Fold.setTitle("Prueba");
Fold.setParent("RootFolder");
Fold.insert();
// if PDId is not assigned, a new PDId is created
String NewId=Fold.getPDId();
PDFolders Child=new PDFolders(OPDSess);
Child.setTitle("Child of Prueba");
Child.setParent(Fold.getPdId());
Child.insert();
// once created and loaded, we can modify a value and update
Child.SetTitle("Child of Prueba modificado");
Child.update();
// or we can delete it
Child.delete();
```

When you delete a folder, ALL the subfolders to ANY level are deleted, and ALL the documents contained in ANY subfolder are sent to the paper bin (from where they can be undeleted or purged definitively). All the operation is run in a transactional way and is limited by the permissions of the current user and the ACL assigned to each folder and document. If the current user hasn't DELETE permissions in any object (folder or document), an exception will be thrown and the operation cancelled.

If you want to create or look for a folder of a specific type (defined previously with the administration interface of OpenProdoc), you must create an instance of the class specifying the folder type:

```
PDFolders Fold=new PDFolders(OPDSess, "FolderType");
```

So for inserting a folder of a specific type, the process would be:

```
PDFolders Fold=new PDFolders(OPDSess, "FolderType1");
Record Rec=Fold.getRecSum();//returns ALL attributes of "FolderType1"
// set values to each Attribute of the type
. . . .
// and reassign to the folder
Fold.assignValues(Rec);
Fold.insert(); /* will throw PDException in any error as: required
field, duplicated index, lack of permissions in the parent folder,..*/
```

Once the folder is instantiated (with any folder type or the default one), if you load a specific folder (of any type) with the method Load(), all the previous information is cleared (including the folder type) and the COMMON data of the folder (that is the metadata defined in PD_FOLDERS, including the folder type, its metadata and the contained elements) is loaded.

```
PDFolders Fold=new PDFolders(OPDSess);
Fold.Load("12345");
Fold.getFolderType();
Record Rec=Fold.getRecSum();//returns common attributes of folder
```

If you want to retrieve ALL the metadata (including the metadata defined for the specific folder type), the method LoadFull() is provided.

```
PDFolders Fold=new PDFolders(OPDSess);
Fold.LoadFull("12345");
Fold.getFolderType();
Record Rec=Fold.getRecSum();//returns ALL attributes of folder "12345"
```

If the folder is of type PD_FOLDERS, both Load and LoadFull will have equivalent results.

Any operation over a folder is limited by the ACLs, so if the user of session doesn't have WRITE permissions over the parent, the insert operation will fail, and the same restriction apply for updating the folder. If the user doesn't have ANY permission over the folder, a call to Load or any other operation will fail with a message of "*User without permissions over folder*".

For retrieving the contained elements, there are available the methods:

- HashSet<String> getListDirectDescendList(String PDId), that return the Id of folders contained DIRECTLY in the specified folder Id.
- HashSet<String> getListDescendList(String PDId), that returns ALL the folders contained (directly or under subfolders up to any level).

For searching folders, it is possible to use any of the methods:

- public Cursor Search(String FolderType, Conditions AttrConds, boolean SubTypes, boolean SubFolders, String IdActFold, Vector Ord) throws PDException
- public Vector<Record> SearchV(String FolderType, Conditions AttrConds, boolean SubTypes, boolean SubFolders, String IdActFold, Vector Ord) throws PDException
- public Cursor SearchSelect(String SQL) throws PDException
- public Vector<Record> SearchSelectV(String SQL) throws PDException

Where the methods ending in 'V' are equal to the others with the same name, but they go through the Cursor and fill the Vector.

4.2.1.7 Documents

Managing documents is the core of any DMS. As with folders, you can define new classes and subclasses of document types, with new attributes and behaviour.

OpenProdoc includes out-of-the-box two document types PD_DOCS (the base type, managed by the java class PDDocs) and PD_REPORT (subtype of PD_DOCS and managed by java class PDReport for creating OpenProdoc Reports), but it is possible to create all the document types needed, each one with its own metadata. The new document types will be a subtype of PD_DOCS and will be managed in Java by PDDocs.

Each document MUST be contained in a “parent” folder. The documents are referenced by its unique identifier PDId and, when needed, by its version.

A document consist of a binary “content” or file, a set of attributes, a folder where it’s stored and a security (ACL). When a document is created, all that information must be assigned. The metadata can be managed as a Record and, for the common metadata of PDDocs and subclasses, using the getters and setters. The ACL and parent folder can be managed just as another metadata.

The binary can be assigned as a local path to a file or as InputStream. Related to the binary is the Attribute mime type, which references the IANA media types. The mime type is always required and, when not informed, will be calculated using the file extension. OpenProdoc will store the mime type of a document using the extension as a reference to the PDMimeType objects. When the document is downloaded in a browser, OpenProdoc will inform the mime type so the browser can use the application, codec or plugin for that kind of file. If the extension is not included in the lists managed by the PDMimeType class, the document will have a generic mime application/octet-stream.

The insertion is similar to other OpenProdoc objects:

```
PDDocs Doc=new PDDocs(OPDSess, DocType);
Record Rec=Doc.getRecSum();
// fill Rec Attributes for DocType. . .
Doc.assignValues(Rec);
Doc.setParentId("54545454-545454");
Doc.SetACL(“PrivateDocs”);
Doc.setFile("/tmp/example/ImageExample.jpg");
//MimeType calculated from extension
Doc.insert();
InputStream Is=ObtainStream(); // open in your method or received
// default DocType
PDDocs Doc2=new PDDocs(OPDSess);
Doc2.setParentId("6776878-aaabb");
Doc2.setTitle("Monthly Report");
Date D=ObtainDateOfDocument(); //Day-moth-year of the document
Doc2.setDocDate(D);
Doc2.SetACL(“PublicDocs”);
Doc2.setStream(Is);
Doc2.setMimeType(“pdf”);//MimeType MUST be assigned for InputStream
Doc2.insert();
```

And the delete is also similar:

```
PDDocs Doc=new PDDocs(OPDSess);
Doc.setPDId(“6927875-ab-c4-5”);
Doc.delete();
```

However, when a document is deleted, it is not actually deleted but hide and marked as deleted. The document can’t me managed or retrieved but will be in the paper bin until it is

purged from the repository by an automatic process configured for periodically empty the paper bin, or by calling the method `Purge(String DocTypename, String Id).`

On the other hand, the update of a document is very different. As in most DMS, OpenProdoc maintains every version/change of a document and it's not possible to call directly to update.

First it's necessary the lock the document so no other people or process change the document.

```
PDDocs Doc=new PDDocs(OPDSess);
Doc.setPDId("111f37875-ab-c4-5");
Doc.CheckOut();
```

After locking the document, only the user which locks the document can update it using the usual method:

```
PDDocs Doc2=new PDDocs(OPDSess);
Doc.setPDId("111f37875-ab-c4-5");
Doc.LoadFull();
Doc2.setTitle("Monthly Report updated");
Date D2=ObtainDateOfDocument(); //Day-moth-year of the document
Doc2.setDocDate(D2);
Doc2.setFile("/tmp/UpdatedReport.doc");
Doc2.update();
```

This will create a Private Working Copy or Draft for the user. It's possible to update several times the Draft. Only the last version will be maintained. When the document is correct, the user owner cam approve and publish using Checkin();

```
PDDocs Doc2=new PDDocs(OPDSess);
Doc.setPDId("111f37875-ab-c4-5");
Doc.CheckIn("2.0"); // Version Number
```

If by some reason the Draft must be discarded, the method CancelCheckout() deletes the Draft and returns to the previous version.

```
PDDocs Doc2=new PDDocs(OPDSess);
Doc.setPDId("111f37875-ab-c4-5");
Doc.CanCelCheckout(); // undo changes
```

A list of version (with its metadata) can be retrieved with the method:

```
Cursor ListVersions(String DocTypename, String Id)
```

4.2.1.8 Transactions

In order tom maintain a coherence of information, it's possible to start transactions in OpenProdoc. The available methods of the class DriverGeneric, are:

```
void IniciarTrans() // Start a transaction
void CerrarTrans()  //  Commit a transaction
void AnularTrans()  // Rollback a transaction
```

4.2.2 Deployment

The application developed must include in the classpath (or lib folder) the libraries:

- Prodoc.jar
- commons-fileupload-1.3.3.jar
- commons-net-3.6.jar
- lucene-analyzers-common-7.3.1.jar
- lucene-core-7.3.1.jar
- lucene-queryparser-7.3.1.jar
- tika-app-1.18.jar
- jsqlparser-2.0.jar
- jjwt-0.9.1.jar

A Java 1.8 (or higher) is needed in order to run the application.

Additionally, a Prodoc.properties is needed.

Depending on the mode of connection (direct or remote) specified in the properties file, an OpenProdoc server (for remote mode) or a jdbc driver (for direct mode to database) will be needed. See Architecture.

4.3 OpenProdoc SQL

OpenProdoc uses for searching a Subset of SQL with a structure and methods similar to the SQL of the CMIS standard. The general structure is:

```
SELECT Columns_List
FROM Object_Name
WHERE Search_Condition
ORDER BY Sort_Description
```

A more detailed BNF description is:

General

```
Select_Expression ::= SELECT Columns_List FROM Object_Name [ WHERE
Search_Condition ] [ ORDER BY Sort_Description ]
Columns_List ::= * | ColumnName [ { , ColumnName } ]
Object_Name ::= this | FolderTypeName [,SUBTYPES]
Search_Condition ::= [ NOT ] [ ( ]Bool_Term | Search_Condition OR
Bool_Term [ ) ]
Bool_Term ::= Bool_Factor | Bool_Term AND Bool_Factor
Bool_Factor ::= Exp_Comp | Exp_In
Exp_Comp ::= FieldName Comparator [ FieldName | Value ]
Comparator ::= = | <> | <= | >= | > | <
Value ::= String | Date | Integer | TimeStamp | Boolean | Decimal
Exp_In ::= FieldName IN In_List
In_List ::= (Value [ { , Value } ]) | Select_Expression
Sort_Description ::= FieldName ASC | DESC
```

For Folders

```
Select_Expression ::= SELECT Columns_List FROM Object_Name [ WHERE
Search_Condition ] [ ORDER BY Sort_Description ]
Columns_List ::= * | ColumnName [ { , ColumnName } ]
Object_Name ::= this | FolderTypeName [,SUBTYPES]
```

```
Search_Condition ::= [ NOT ] [ ( ]Bool_Term | Search_Condition OR
Bool_Term [ ) ]
Bool_Term ::= Bool_Factor | Bool_Term AND Bool_Factor
Bool_Factor ::= Exp_Comp | Exp_In | Exp_Func
Exp_Comp ::= FieldName Comparator [ FieldName | Value ]
Comparator ::= = | <> | <= | >= | > | <
Value ::= String | Date | Integer | TimeStamp | Boolean | Decimal
Exp_In ::= FieldName IN In_List
In_List ::= (Value [ { , Value } ]) | Select_Expression
Exp_Func ::= IN_TREE( FolderId ) | IN_FOLDER( FolderId )
Sort_Description ::= FieldName ASC | DESC
```

For documents

```
Select_Expression ::= SELECT Columns_List FROM Object_Name [ WHERE
Search_Condition ] [ ORDER BY Sort_Description ]
Columns_List ::= * | ColumnName [ { , ColumnName } ]
Object_Name ::= this | FolderTypeName [,SUBTYPES]
Search_Condition ::= [ NOT ] [ ( ]Bool_Term | Search_Condition OR
Bool_Term [ ) ]
Bool_Term ::= Bool_Factor | Bool_Term AND Bool_Factor
Bool_Factor ::= Exp_Comp | Exp_In | Exp_Func
Exp_Comp ::= FieldName Comparator [ FieldName | Value ]
Comparator ::= = | <> | <= | >= | > | <
Value ::= String | Date | Integer | TimeStamp | Boolean | Decimal
Exp_In ::= FieldName IN In_List
In_List ::= (Value [ { , Value } ]) | Select_Expression
Exp_Func ::= CONTAINS(`FullText_Search`) | IN_TREE( FolderId ) |
IN_FOLDER( FolderId )
Sort_Description ::= FieldName ASC | DESC
FullText_Search ::= Expression following the Lucene Syntax. See Lucene Syntax.
```

The SQL is the more powerful way to search in OpenProdoc. However it must be used carefully to avoid syntax errors, too much results or too complex queries that can cause problems in the servers or timeout in the applications.

It must be noted that additionally to the created conditions, internally OpenProdoc can add additional columns (in example it will add always the PDId) and will add additional security conditions and filters so a user will never retrieve document he is not allowed to view, even if the documents match the conditions initially defined in the query.

The SQL can be used in the REST API and in some search methods of the PDFolders and PDDocs classes. There are other ways of searching or loading objects, that doesn't use SQL.

Besides the usual expressions in SQL, the only special functions are referenced as <Exp_Func> and all of them return a set of identifiers, that is are equivalent to an expression "PDId in (list of documents of folders PDIds that match the function)"

`CONTAINS(`'<FullText_Search>'`)`: List of Id of documents that match the fulltext Lucene query. See Lucene Syntax

`IN_TREE( <FolderId> ):` List of Id of documents or folder contained AT ANY LEVEL OF SUBFOLDERS under the specified Folder identified by it Id

`IN_FOLDER( <FolderId> ):` List of Id of documents or folder contained DIRECTLY under the specified Folder identified by it Id

4.4 Lucene Syntax

The Fulltext search allow to find document by its text besides it metadata or other criteria. This is possible by means of the Apache libraries Tika, which extracts the text content from different file formats and Lucene, that analyses and index all the words of the text. The index are stored in a disk folder (for what it must be created a special repository) and are updated when documents are inserted, updated or deleted (for what it must be created some indexing tasks).

The Fulltext syntax can be used in the CONTAINS method of OpenProdoc SQL and in some search methods of PDDocs.

A complete reference of the syntax can be found in: Lucene. It must be noted that in OpenProdoc the indexation is made over all the content of the document, so fields' references of the syntax don't apply.

The expressions used usually for searching are:

- **Word**: The search will return the documents containing the word.
- **Some words**: The search will return the documents containing ANY of the words.
- **"Some words"**: The search will return the documents containing exactly the expression between quotes.
- **+Word**: The word MUST be included in all the documents.
- **-Word**: The word CAN'T be included in any the documents.
- **Word***: The search will return the documents that include the words starting with the root defined.

4.4.1 Indexing optimization by selecting language and stop words

In the full-text search, since version 2.3 of OpenProdoc, some improvements have been introduced, such as the possibility of choosing the language or being able to define a dictionary of stop words, two measures that improve the quality of the searches results as well as performance.

Choosing a language activates the stemming for that language that is the conversion from the words to their "root" before indexing. In this way, when searching, it is indifferent to enter "Document" or "Documents". Logically the stemming rules are different by language (for example in English the suffix "ing" of the gerunds will be eliminated). Therefore, the appropriate

language must be chosen to match the language of documents to be searched. If the documents can be of several languages, it is possible to keep the language unspecified. It is generally not convenient to use a different language than the language of the documents, since the application of rules designed for another language may cause the quality of the results to decrease instead of increasing.

Regarding the dictionary of stop words, includes words that are not significant for a search, either because they are "meaningless" particles (articles, prepositions, pronouns ...) or because they will appear in almost all documents (for example the word "ecology" in documentation of an environmental organization) and therefore the Search for those terms will return almost all documents, which does not add any value. The inclusion of words in the dictionary of empty words, on one hand saves space in the files of search indexes by full text and provides more speed of search and indexing, and on the other hand facilitates the search, since their appearances are ignored and it focuses on the significant terms. For example, you can find documents where "pollutant discharges into the river" appear and "some pollutant has been spilled on the right bank of the river" if they are empty words: "the, in, the, has, the, margin, right ", since the terms associated with the document will be: *dumping*, *contaminant*, *river* (where stemming has also been used to remove plurals).

To choose the language or the list of empty words, the following procedure should be followed:

1- Create a text file (.TXT) where all the empty words to be used are entered, each of them in a line. Lines that start with the # character will be ignored and can be used to include comments and explanations.
2- The file must be incorporated into OpenProdoc, assigning it any type of document. After the insertion, the unique identifier of the document (PDId) must be saved. It is recommended to insert it in the "System" folder, although it is not essential.
3- Subsequently, the full text repository (which has the name "PD_FTRep") must be modified in the list of repositories. The system detects that it is a full text repository and presents a "W" button that when pressed shows a form that allows editing the language (between the supporters by OpenProdoc: ES, EN, PT, CT) and enter the Identifier (PDId) of the document with the list of stop words.
4- After saving the modifications, the program will use the new parameters for the following indexing and searching processes. The new configuration may take some time to be used (because the information is cached to increase performance). To force the use of the new configuration, it is best to restart the server.
5- You can choose not to inform the language (by choosing the value "*") or not to enter a dictionary of empty words (leaving the identifier empty).
6- The document with the stop words can be versioned, like any other OpenProdoc document. The program will use the latest version, although it may take some time to

update it (because the information is cached to increase performance). To force the use of the new version, it is better to restart the server.

5 Extension/Plugins

There are 3 kinds of extensions:

1- Repository Extensions, for storing and retrieving documents from your own storage systems.
2- Extensions of Tasks, for executing periodically or each time an event associated to a document or folder occurs.
3- Authentication Extensions, for validating users against proprietary systems.

All developments and methods must throw an OpenProdoc exception (PDException) in case of error and release the resources and elements that were reserved for the operation. The recommended way is to structure it as follows:

```
try {
. . . .
} catch (Exception Ex)
    {// method that traces the error and also throws it.
    PDException.GenPDException ("Error reading" + getServer (),
    Ex.getLocalizedMessage ());
    }
finally
    {// Free resources and close connections
    . . .
    }
```

The trace must be done using the OpenProdoc trace singleton: PDLog, checking the trace level before generating the corresponding message:

```
if (PDLog.isDebug ()
    PDLog.Debug ("Starting connection:" + getServer ());
. . .
if (PDLog.isInfo ()
    PDLog.Info ("Deleted document:" + IdDoc);
. . .
if (PDLog.isError ()
    PDLog.Error ("Error in operation XXX:" + Ex.getLocalizaedMessage
    ());
```

This avoids generating unnecessary calls and instantiation of string, which will never written and only delay the work of the Logger.

Each type of extension has a form of development and deployment, which is detailed below.

It is important to emphasize that the **developments must be multi-threaded** always, since different users or sessions can concurrently insert or retrieve documents stored in the

same repository or manage authentication calls, so that simultaneously different instances of the same class will be using the different functions of the Extension at the same time in different threads.

For more information, review API Java.

5.1 Repositories

The purpose of Repository Extensions is to store and retrieve documents from different media or byte storage systems (byte files, ftp, blob, Amazon S2, Atmos, Dropbox, Google Drive, etc.). For example, a driver could be created to store documents in the Apache Hadoop file system: HDFS (Hadoop Distributed File System).

For this, a class could be developed that will be a subtype of the OpenProdoc StoreGeneric class and that will have to overload, at least the abstract methods, although generally it will have to overload other methods and create new private methods for the functions that are necessary.

If the development requires other libraries, they must be installed or deployed in the corresponding environment and added to the CLASSPATH, so that the Repository Extension can find them when invoked by OpenProdoc. The easiest way can be to copy the jars to the folder “OPD_PortableWeb/webapps/ProdocWeb2/WEB-INF/lib/”

The extension itself is automatically downloaded and deployed, so it does not need to be included in the CLASSPATH.

OpenProdoc provides to the Extension, in addition to a series of standard parameters (Url/URI, User, Password and Additional Parameters), of a property file that each Extension can define according to its needs and parameters.

The life cycle of extensions and storage classes for OpenProdoc is as follows:

1- When it is necessary to use it for the first time, the constructor will be invoked
2- The Connect () connection method may be invoked periodically
3- After that, one or more of the methods of handling the contents of the documents will be invoked: Insert, delete or recover. (Insert (), Delete (), Retrieve ())
4- Finally, a Disconnect () disconnection method will be invoked.

5.1.1 Development

The methods to be implemented generally are:

```
public Constructor(String pServer, String pUser, String pPassword,
String pParam, boolean pEncrypt) throws PDException
protected void Connect() throws PDException
protected void Disconnect() throws PDException
```

```
protected int Insert(String Id, String Ver, InputStream Bytes, Record
Rec, String OPDPath) throws PDException
protected InputStream Retrieve(String Id, String Ver, Record Rec)
throws PDException
protected void Delete(String Id, String Ver, Record Rec) throws
PDException
```

The defined contract and the expected behaviour of each method is as follows

5.1.1.1 Constructor

The constructor (whose name will logically be that of the class to be built, eg: MiDriver) receives the parameters:

● String pServer: Uri / URI and in general reference to the “address” of the server (it can be a string, a http reference, a local folder, etc.) that should be interpreted by the Extension.

● String pUser: Connection user, which could be empty if it were for example a local folder, or could be used as part of a system of public / private keys or certificates together with pPassword.

● String pPassword: Connection password, which could be empty if it were for example a local folder, or could be used as part of a public / private key system or certificates together with pUser.

● String pParam: Additional parameter that each Extension can interpret as deemed necessary. You can access at any time through getParam ().

● boolean pEncrypt: Indicates that documents must be encrypted in the repository, so that if someone has access to that repository (for example a file system or a Dropbox account), they cannot see the documents if it is not through OpenProdoc. Its implementation is not mandatory, the parameter can be ignored, but if implemented, it must be a transparent process, that is, it is encrypted in real time when inserted and decrypted upon return.

Those parameters are assigned in the default constructor of the parent class and can be read (not changed) later by means of the corresponding getter.

In the most general case, the implementation can be as simple as:

```
public MiDriver(String pServer , String pUser, String pPassword,
String pParam, Boolean pEncrypt) throws PDExceptionFunc
{
super(pServer, pUser, pPassword, pParam, pEncrypt);
}
```

That is, delegate the construction class to the parent class. But of course it could be used to initialize all the elements, both of the object and static elements of the whole class, for example:

```
static private Started = false;
```

```
public MiDriver(String pServer , String pUser, String pPassword,
String pParam, boolean pEncrypt) throws PDExceptionFunc
{
super(pServer, pUser, pPassword, pParam, pEncrypt);
if (!Started)
      StartStaticElements();
StartElementsInstance();
}
```

5.1.1.2 Connect

The connection method allows the class to connect (if necessary) to the destination where the documents will be stored. To do this, it will have the server address (getServer()), and the user (getUser()) and password (getpassword ()) informed in the constructor.

To optimize performance, if the repository to use has some type of connection pool, it is recommended to use it in the implementation.

Although a connection should not be made as such, the method to verify access to the resource (existence of the folder, device or resource, connectivity, write permissions, etc.) can be used.

Possibly this method after connection should save in the object some type of variable of the type of storage to be used (BBDD session, ftp, network, etc ...)

In a trivial case that does not require connection (Example a folder in a local filesystem), it can be as simple as:

```
protected void Connect() throws PDException
{
}
```

5.1.1.3 Disconnect

This method will disconnect (or free a session) from the repository. If an object variable has been created, a null value may have to be assigned. Any resources related to the session must be deleted.

In a trivial case that does not require connection, it can be as simple as:

```
protected void Disconnect() throws PDException
{
}
```

5.1.1.4 Insert

The Insert method is the method in charge of storing the document content, received as InputStream, in the repository or storage system that will manage the Extension.

Receive the parameters:

● String Id: Unique document identifier

● String Ver: Unique version identifier. The Id + Ver combination will be unique, but each of the values can be repeated.

● InputStream Bytes: Document content. The extension is responsible for closing the InputStream when the process of saving in the repository ends.

● Record Rec: Set of metadata of the document to be inserted, including: document type, user who performs the operation, title, mime type, etc.). It will not be retrieved from the repository and generally not be used but can be used to segment or adjust storage, add additional security to what OpenProdoc provides, etc.

● String OPDPath: Full path (which you can change without reinserting the document into the repository) where the document is referenced within OpenProdoc. In principle the storage place in the repository is totally disconnected from the place in OpenProdoc, however in some cases it can be useful as an aid to implement or optimize the structure of the repository.

The method must store, uniquely identified by the combination of Id and Ver (and optionally with the help of any of the other data received) the content received and return the total number of bytes of the content.

The InputStream is binary, so no interpretation or modification regarding page codes, etc. should be applied. It should be stored as is, or by applying a fully reversible operation (compression, encryption, etc.) that when retrieved with the Retrieve method returns exactly the same set of bytes.

If you have chosen to implement the Encrypted mode, the insert must encrypt the document when it is stored.

5.1.1.5 Retrieve

The recovery method allows you to load the bytes of the document.

Receive the parameters:

● String Id: Unique document identifier

● String Ver: Unique version identifier. The Id + Ver combination will be unique, but each of the values can be repeated.

● Record Rec: Set of metadata of the document to be recovered, including: documentary type, user who performs the operation, title, mime type, etc.). It will generally not be useful but can be used to segment, adjust storage, add additional security to what OpenProdoc provides, etc.

The method must recover, uniquely identified by the combination of Id and Ver (and optionally with the help of any of the other data received) the content previously received and return a Content InputStream.

The InputStream is closed by the OpenProdoc core itself when the recovery is finished, so the extension of the repository does not need to take care of it, unless an error occurs during the execution of the method, in which case before throwing an exception you must make sure the InputStream closes.

The InputStream is binary, so no interpretation or modification regarding page codes, etc. should be applied. It should be stored as is, or by applying a fully reversible operation (compression, encryption, etc.) that when retrieved with the Retrieve method returns exactly the same set of bytes.

If you have chosen to implement the Encripted mode, the recovery must decrypt the document transparently upon return.

5.1.1.6 Delete

This method will completely remove the contents of the repository where it is stored.

A temporary logical deletion may be permissible (that is to say that the method temporarily marks the content as "erasable") provided that a process specific to the development of the extension (or the platform itself used) completely eliminates the content (or makes it inaccessible for Nobody whatever your profile) in a short time (less than 1 day).

Receive the parameters:

● String Id: Unique document identifier

● String Ver: Unique version identifier. The Id + Ver combination will be unique, but each of the values can be repeated.

● Record Rec: Set of metadata of the document to be recovered, including: documentary type, user who performs the operation, title, mime type, etc.). It will generally not be useful but can be used to segment, adjust storage, add additional security to what OpenProdoc provides, etc.

5.1.2 Configuration

Optionally, the Repository Extensions can have configuration files in order to have more parameters in addition to the 4 standard parameters received in the constructor.

To do this, they can define a properties file (with the standard format of properties files) and assign it during deployment. OpenProdoc will load the properties file after instantiating the object and assign it to it, so that when any of the methods is used, an object of type properties can be accessed through the getProp () method and retrieve the required property.

When it is not necessary to use a properties file, it is enough not to include that option in the deployment, in which case, if the getProp () method is invoked, an empty properties object will be returned.

The number and meaning of the properties will be specific to each Extension and does not require following any fixed format.

```
protected void Delete(String Id, String Ver, Record Rec)
throws PDException
{
String Bucket = getProp().GetProperty("BUCKET");
. . .
}
```

5.1.3 Example

An example of driver (similar to the internal one for folders):

```
/*
 * Example of OpenProdoc Driver for storage of documents in a Folder
 * (similar to the internal one in OpenProdoc for folders)
 */
package driverexample;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.InputStream;
import prodoc.PDException;
import prodoc.PDExceptionFunc;
import prodoc.PDLog;
import prodoc.Record;
import prodoc.StoreCustom;

/**
 * Example of developed driver.
 * It MUST extend StoreCustom repository
 * @author Joaquín Hierro
 */
public class DriverExample extends StoreCustom
{
static final String SEP="v";

public DriverExample(String pServer, String pUser, String pPassword,
String pParam, boolean pEncrypt) throws PDExceptionFunc
{
super(pServer, pUser, pPassword, pParam, pEncrypt);
if (PDLog.isDebug())  //recommended way of trace
    PDLog.Debug("Created Driver:"+DriverExample.class+"/"+pServer);
// if required and configured, a properties file is available through
the parent method:
// Properties getProp()
// example String MyProp=getProp().getProperty("MyProp1")
}
//---------------------------------------------------------------------
/**
```

```
 * The create method could be called from the user interface
(Repositories Management) so each repository creates in its own way
 * @throws PDException in any error
 */
@Override
protected void Create() throws PDException
{
File BasePath=new File(getServer()); // each storage driver can use
the Server in a way (url, path, UNC, code,..)
if (BasePath.isDirectory())
    return;
try {
BasePath.mkdirs();
} catch (Exception e)
    {// this method logs the exception AND throws again

PDException.GenPDException("Error_creating_folder_for_StoreFS",getServ
er()+"="+e.getLocalizedMessage());
    }
}
//----------------------------------------------------------------------
/**
 * The Delete method is not actually called by the risk of deleting
thousands of documents
 * @throws PDException in any error
 */
@Override
protected void Destroy() throws PDException
{
File BasePath=new File(getServer());
if (!BasePath.isDirectory())
    return;
BasePath.delete();}
//----------------------------------------------------------------------
/**
 * It is called allways BEFORE using the repository.
 * Could be dummy, connect, create sockets, authenticate, etc.
 * @throws PDException In any Error
 */
@Override
protected void Connect() throws PDException
{
if (PDLog.isDebug())  //recommended way of trace
    PDLog.Debug("Connected
Driver:"+DriverExample.class+"/"+getServer()+"-"+getUser());
}
//----------------------------------------------------------------------
/**
 * It is called allways AFTER using the repository.
 * Could be dummy, connect, create sockets, authenticate, etc.
 * @throws PDException In any Error
 */
@Override
```

```
protected void Disconnect() throws PDException
{
if (PDLog.isDebug())  //recommended way of trace
    PDLog.Debug("DisConnected
Driver:"+DriverExample.class+"/"+getServer()+"-"+getUser());
}
//----------------------------------------------------------------------
/**
 * Do the actual insertion of the document binary.
 * Always MUST close the input stream to avoid problems
 * and degradation.
 * @param Id OpenProdoc identifier (PDId) of the document to store
 * @param Ver Versión Label of the Document
 * @param Bytes Input stream to store
 * @param Rec   Record with Metadata of the document (just in case it
used)
 * @param OPDPath Path in OpenProdoc of the document (just in case it
used)
 * @return the size of the file
 * @throws PDException In any error
 */
@Override
protected int Insert(String Id, String Ver, InputStream Bytes, Record
Rec, String OPDPath) throws PDException
{
VerifyId(Id);
FileOutputStream fo=null;
int Tot=0;
try {
File Path=new File(getServer());
if (!Path.isDirectory())
    Path.mkdirs();
fo = new FileOutputStream(getServer()+Id+SEP+Ver);
int readed=Bytes.read(Buffer);
while (readed!=-1)
    {
    if (isEncript())
       EncriptPass(Buffer, readed);  // another stronger algorithm for
encription can be used
    fo.write(Buffer, 0, readed);
    Tot+=readed;
    readed=Bytes.read(Buffer);
    }
Bytes.close();
fo.close();
if (PDLog.isDebug())  //recommended way of trace
    PDLog.Debug("Inserted Driver:"+DriverExample.class+"/"+getServer()
+"-"+Id+SEP+Ver);
} catch (Exception e)
    {

PDException.GenPDException("Error_writing_to_file",Id+"/"+Ver+"="+e.ge
tLocalizedMessage());
```

```
    }
finally
    {
    try {
    if (fo!=null)
        fo.close();
    Bytes.close();
    } catch (Exception e)
        {
        }
    }
return(Tot);
}
//----------------------------------------------------------------------
/**
 * Deletes the binary
 * @param Id Identifier of document
 * @param Ver Identifier of version
 * @param Rec Record with Metadata of the document (not used in
Filesystem storage)
 * @throws PDException In Any Error
 */
@Override
protected void Delete(String Id, String Ver, Record Rec) throws
PDException
{
VerifyId(Id);
File f=new File(getServer()+Id+SEP+Ver);
f.delete();
if (PDLog.isDebug())  //recommended way of trace
    PDLog.Debug("Deleted Driver:"+DriverExample.class+"/"+getServer()
+"-"+Id+SEP+Ver);
}
//----------------------------------------------------------------------
/**
 * Returns the binary as InputStream
 * @param Id Identifier of document
 * @param Ver Identifier of version
 * @return an InputStream with the content
 * @throws PDException in any error
 */
@Override
protected InputStream Retrieve(String Id, String Ver, Record Rec)
throws PDException
{
VerifyId(Id);
FileInputStream in=null;
try {
in = new FileInputStream(getServer()+ Id+SEP+Ver);
} catch (FileNotFoundException ex)
    {
```

```
PDException.GenPDException("Error_retrieving_file",Id+"/"+Ver+"="+ex.g
etLocalizedMessage());
    }
return(in);
}
//---------------------------------------------------------------
/**
 * Changes the name of e binary. Used for CheckInCheckOut,..
 * @param Id1 Identifier of original document
 * @param Ver1 Identifier of original version
 * @param Id2 Identifier of Target document
 * @param Ver2 Identifier of Target version
 * @throws PDException
 */
@Override
protected void Rename(String Id1, String Ver1, String Id2, String
Ver2) throws PDException
{
VerifyId(Id1);
File f=new File(getServer()+Id1+SEP+Ver1);
File f2=new File(getServer()+Id2+SEP+Ver2);
f.renameTo(f2);
}
//---------------------------------------------------------------
}
```

5.1.4 Deployment

To use a Repository Extension, the following steps must be verified:

1- Ensure that, if third-party libraries are required in addition to the Extension's own development, these are installed and configured on all the machines where the repository will be used. OpenProdoc dynamically loads and configures the Extension, but NO additional libraries.

2- Incorporate to OpenProdoc, with any document type, the jar with the development of the extension. It should be noted that when the Extension is used, at least the first time, it must be downloaded, so it must be accessible in read mode for all users who require it. The PdId of this document should be saved, which will then be used as a reference in the definition of the repository.

3- Optionally, add the properties file to OpenProdoc, with any document type. It should be noted that when the Extension is used, at least the first time, the properties file must be downloaded, so it must be accessible in read mode for all users who require it. The PdId of this document should be saved, which will then be used as a reference.

4- Define a CUSTOM type repository with the required Server, User and Password parameters (if necessary) and with the following Param value:

```
Jar PDId | Package name + class | Optional PDId of properties file.
```

The last parameter can be ignored. For example, you can define:

```
12342343-543fgae454|mipackage.MyDriver|ab5455c-4343ghde
```

or

```
565df365afbc34fe65-65defc3456ad|otherdriver.Repo2
```

After that you can define document types that have the newly defined repository.

You can create several instances of the same type of repository with different properties.

It should be noted that documents containing the code and properties may be versioned and modified following the same procedure as with any other document. When you start OpenProdoc you will always use the latest published version of any of them.

However, if the version of the jar or the properties is updated and that driver has ALREADY been used, since a previous version is loaded and instantiated in the JVM, that new version will NOT be used. The JVM must be restarted so that the Java classloader can use the new version.

5.2 Authenticators

The purpose of Authenticator Extensions is to verify the correct identity of users, that is, to authenticate them against a system/technology different from the included in OpenProdoc. In example, you can need to authenticate using an oAuth token or JWT token or an application or security service of your company/institution.

For this, a class must be developed that will be a subtype of the OpenProdoc AuthGeneric class and that will have to overload, at least the abstract methods, although generally it will have to overload other methods and create new private methods for the functions that are necessary.

If the development requires other libraries, they must be installed or deployed in the corresponding environment and added to the CLASSPATH, so that the Repository Extension can find them when invoked by OpenProdoc. The easiest way can be to copy the jars to the folder “OPD_PortableWeb/webapps/ProdocWeb2/WEB-INF/lib/”

The extension itself is automatically downloaded and deployed, so it does not need to be included in the CLASSPATH.

OpenProdoc provides the Extension, in addition to a series of standard parameters (Url / URI, User, Password and Additional Parameters), of a property file that each Extension can define according to its needs and parameters.

The life cycle of authenticator classes for OpenProdoc is as follows:

1- When it is necessary to use it for the first time, the constructor will be invoked

2- The Authenticate () method will be called for the users that have associated that kind of authentication.

5.2.1 Development

The methods to be implemented generally are:

```
public Constructor(String pServer, String pUser, String pPassword,
String pParam) throws PDException
public void Authenticate(String User, String Pass) throws PDException
```

The defined contract and the expected behaviour of each method is as follows

5.2.1.1 Constructor

The constructor (whose name will logically be that of the class to be built, eg: MiAuthentic) receives the parameters:

● String pServer: Uri / URI and in general reference to the “address” of the server (it can be a string, an http reference, a local folder, etc.) that should be interpreted by the Extension. In example, for a Ldap or Active Directory server, the url of server.

● String pUser: Connection user if needed. Can be empty but some systems .could use, as should be a Human Resources system where the plugin connect and check information of user.

● String pPassword: Connection password or certificate or any other element used by the plugin.

● String pParam: Additional parameter that each Extension can interpret as deemed necessary. The plugin can access this information at any time through getParam().

5.2.1.2 Authenticate

The Authenticate method will verify the identity of the user trying to do login in OpenProdoc with the method selected and using the information/configuration received in the constructor, mainly the reference to the server.

This method receive the parameters:

● String User: Name of the user to authenticate.

● String Password: Password (in text/clear) of the user to authenticate.

To do this, it will have the server address (getServer ()), and, if needed, the user (getUser()) and password (getPassword()) informed in the constructor. It must be noted that the User and password defined with the authenticator are not the user to authenticate but other OPTIONAL user that can be need depending on the kind of authenticator. Example: the authentication can be defined against an internal Human Resources application and the user

defined in the constructor is an administrator granted with permissions for checking users in that system.

If the authentication is correct, the system will not return anything, otherwise, an exception must be thrown (with the usual text “user/password wrong” or other messages as “Server unavailable”, “User disabled”... when apply).

5.2.2 Configuration

Optionally, the Authenticator Extensions can have configuration files in order to have more parameters in addition to the 4 standard parameters received in the constructor.

To do this, you can define a properties file (with the standard format of properties files) and assign it during deployment. OpenProdoc will load the properties file after instantiating the object and assign it to it, so that when any of the methods is used, an object of type properties can be accessed through the getProp () method and retrieve the required property.

When it is not necessary to use a properties file, it is enough not to include that option in the deployment, in which case, if the getProp () method is invoked, an empty properties object will be returned.

The number and meaning of the properties will be specific to each Extension and does not require following any fixed format.

```
public void Authenticate(String User, String Pass) throws PDException
{
String CipherType = getProp().GetProperty(“CipherType”);
. . .
}
```

5.2.3 Example

This trivial example verify users against a file with users-passwords pairs:

```
/*
 * Example of authentication system
 * JUST AN EXAMPLE, NOT AN USEFUL NOR SECURE SOLUTION
 */
package authexample;

import prodoc.PDException;
import prodoc.PDExceptionFunc;
import prodoc.PDLog;
import prodoc.security.AuthGeneric;

/**
 * Example of Authentication system that check the user/password
 * against a list of user/passwords defined in a properties files
 * JUST AN EXAMPLE, NOT AN USEFUL NOR SECURE SOLUTION
 * @author jhier
 */
public class AuthExample extends AuthGeneric
```

```
{
/**
 * Constructor
 * @param pServer Server to authenticate
 * @param pUser User of connection to server (if need, NOT the user to
authenticate)
 * @param pPassword Password of connection to server (if need, NOT the
user to authenticate)
 * @param pParam Aditional param
 */
public AuthExample(String pServer, String pUser, String pPassword,
String pParam)
{
super(pServer, pUser, pPassword, pParam);
if (PDLog.isDebug())  //recommended way of trace
    PDLog.Debug("Created Auth:"+AuthExample.class+"/"+pServer);
// if required and configured, a properties file is available through
the parent method:
// Properties getProp()
// example String MyProp=getProp().getProperty("MyProp1")
}
//---------------------------------------------------------------
/**
 * Authenticates the user against the server defined and thows an
exception
 * if the authetication fails.
 * @param pUser user to authenticate
 * @param pPass Password (in clear)of the user
 * @throws PDException if the user is nos authenticated
 */
@Override
public void Authenticate(String pUser, String pPass) throws
PDException
{
String StoredPass=getProp().getProperty(pUser);
if (StoredPass!=null && StoredPass.equals(pPass))
    return;
// this method logs the exception AND throws again
PDExceptionFunc.GenPDException("Error
authenticating:"+AuthExample.class+"/"+getServer(), "User="+pUser);
}
//---------------------------------------------------------------
}
```

and the properties file will have the structure:

```
JohnSmith=Pass1
Pocahontas=Pass2
```

5.2.4 Deployment

To use an Authenticator Extension, the following steps must be verified:

1- Ensure that, if third-party libraries are required in addition to the Extension's own development, these are installed and configured on all the machines where the Extension will be used. OpenProdoc dynamically loads and configures the Extension, but NO additional libraries.
2- Incorporate to OpenProdoc, with any document type, the jar with the development of the extension. The PdId of this document should be saved, and will then be used as a reference.
3- Optionally, add the properties file to OpenProdoc, with any document type. The PdId of this document should be saved, and then be used as a reference.
4- Define a CUSTOM type Authenticator with the required Server, User and Password parameters (if necessary) and with the following Param value:

```
Jar PDId | Package name + class | Optional PDId of properties file .
```

The last parameter can be ignored. For example, you can define:

```
12342343-543fgae454|mipackage.MyAuthent|ab5455c-4343ghde
```

or

```
565df365afbc34fe65-65defc3456ad|otherauth.Auth
```

After that you can define users that authenticate against the new system.

You can create several instances of the same type of authenticator with different properties.

It should be noted that documents containing the jar and properties may be versioned and modified following the same procedure as with any other document. When you start OpenProdoc you will always use the latest published version of any of them.

However, if the version of the jar or the properties is updated and that authenticator has ALREADY been used, since a previous version is loaded and instantiated in the JVM, that new version will NOT be used. The JVM must be restarted so that the Java classloader can use the new version.

5.3 Tasks

The purpose of Tasks Extensions is to run (periodically or when an event occurs) some process besides the already defined in OpenProdoc. In example, you can run every Friday night a process for creating a report with statistics of all the documents received during the week, or when you insert a document of a specific kind, like a bill, to connect to the accounting system and add an entry. Check the OpenProdoc help for more information.

For extending tasks, a class must be developed that will be a subtype of the OpenProdoc CustomTask class and that will have to overload some methods, although generally it will have to overload other methods and create new private methods for the functions that are necessary.

If the development requires other libraries, they must be installed or deployed in the corresponding environment and added to the CLASSPATH, so that the Repository Extension can find them when invoked by OpenProdoc. The easiest way can be to copy the jars to the folder “OPD_PortableWeb/webapps/ProdocWeb2/WEB-INF/lib/”

The extension itself is automatically downloaded and deployed, so it does not need to be included in the CLASSPATH.

There are two main classes of tasks: Event triggered Tasks and Scheduled tasks. In turn, the former has two subtypes: Events triggered by documents and events triggered by folders.

It’s important to note that the custom tasks (Event triggered of scheduled) will be run always in non transactional (delayed) way. So, they can be used only for insert or update events, because after de delete event, the object didn’t exist.

All the classes developed must extend the CustomTasks class. However, they must overload/implement different methods that will be called by OpenProdoc:

- Event (both):

```
boolean CustomMeetsReqRec(String param, String param2, String param3,
String param4, Record Rec, DriverGeneric Drv)
```

  - Document:

```
void ExecuteEventDoc(String Param1,String Param2,String Param3,String
Param4, PDDocs Doc)
```

  - Folder:

```
void ExecuteEventFold(String Param1, String Param2,String Param3,
String Param4, PDFolders Fold)
```

- Scheduled:

```
Cursor CursorCustom(DriverGeneric Drv, String ObjectType, String
Filter, String param, String param2, String param3, String param4)
throws PDException
void CustomCronTask(DriverGeneric Drv, String objType, String
objFilter, String param, String param2, String param3, String param4)
throws PDException
```

The life cycle of Event tasks classes for OpenProdoc is as follows:

1- When it is necessary to use it for the first time, the constructor will be invoked

2- When the event associated occurs, the method CustomMeetsReqRec will be called in order to check if the object metadata meets the criteria defined in Tasks.

3- If the object meets the criteria, the method ExecuteEventDoc (or ExecuteEventFold) will be called later (and perhaps with a different instance of the class, so you shouldn’t assume reusing the same object, although it’s possible).

The life cycle of Scheduled tasks classes for OpenProdoc is as follows:

1- When it is necessary to use it for the first time, the constructor will be invoked

2- When an administrator needs to check the documents or folders that will meet the criteria of the tasks, OpenProdoc will call the method CursorCustom that will create a Cursor containing all the objects.

3- When the scheduled time is meet, OpenProdoc will call the method CustomCronTask, that must call the method CursorCustom for obtaining the objects, and then execute the tasks over them.

5.3.1 Development

For tasks triggered by events, you must implement, at least, the methods:

- ExecuteEventDoc (for event triggered by Documents)
- ExecuteEventFold (for event triggered by Folders)

And optionally the methods:

- CustomMeetsReqRec (if you want to check values before running it, otherwise it will be always run)

For tasks scheduled, you must implement, at least, the methods:

- CursorCustom
- CustomCronTask

5.3.1.1 Constructor

The constructor (whose name will logically be that of the class to be built, eg: MiTask) will have no parameters, and can be used for initialize any element needed.

5.3.1.2 CustomMeetsReqRec

This method must check if the received metadata (Record Rec) of the object that triggers the event, meets the criteria defined by the parameters. The parameters can be understood by the tasks as criteria for selecting the object or as a parameters of the tasks.

When the tasks is defined in OpenProdoc, a document type (or folder type) is assigned and also an event (Insert, Update, Delete), so some tasks can have enough information and return true always. Other developments will need to check the values and return true or false. If the task returns false, Openprodoc will never call the Execute method,

This method receive the parameters:

- String Param: parameter that the tasks can use as preferred.
- String Param2: parameter that the tasks can use as preferred.
- String Param3: parameter that the tasks can use as preferred.

● String Param4: parameter that the tasks can use as preferred.

● Record Rec: A record containing the attributes of the object (Document or folder) that triggers the event.

● DriverGeneric Drv: A session for any access to the OpenProdoc repository that the tasks can need.

The method should return true if the object meets the criteria defined and false otherwise. If the tasks mus be executed always for any values it's not necessary to implement it.

5.3.1.3 ExecuteEventDoc

This method must do the actual process on the document that triggers the event. The parameters can be understood by the tasks as criteria for selecting the object or as a parameters of the tasks.

This method receive the parameters:

● String Param: parameter that the tasks can use as preferred.

● String Param2: parameter that the tasks can use as preferred.

● String Param3: parameter that the tasks can use as preferred.

● String Param4: parameter that the tasks can use as preferred.

● PDDocs Doc: The document that triggers the event.

5.3.1.4 ExecuteEventFold

This method must do the actual process on the folder that triggers the event. The parameters can be understood by the tasks as criteria for selecting the object or as a parameters of the tasks.

This method receive the parameters:

● String Param: parameter that the tasks can use as preferred.

● String Param2: parameter that the tasks can use as preferred.

● String Param3: parameter that the tasks can use as preferred.

● String Param4: parameter that the tasks can use as preferred.

● PDFolders Fold: The folder that triggers the event.

5.3.1.5 CursorCustom

This method must use the parameters received and create a cursor including all the elements that meet the criteria.

This method receive the parameters:

- DriverGeneric Drv: OpenProdoc session for access elements and creating cursor.
- String ObjectType: parameter that the tasks can use as preferred.
- String Filter: parameter that the tasks can use as preferred.
- String Param: parameter that the tasks can use as preferred.
- String Param2: parameter that the tasks can use as preferred.
- String Param3: parameter that the tasks can use as preferred.
- String Param4: parameter that the tasks can use as preferred.

5.3.1.6 CustomCronTask

This method do the actual process over the elements of the cursor created. The method must call the CursorCustom method for creating the cursor, process the elements and close ALWAYS the cursor

This method receive the parameters:

- DriverGeneric Drv: OpenProdoc session for access elements.
- String ObjectType: parameter that the tasks can use as preferred.
- String Filter: parameter that the tasks can use as preferred.
- String Param: parameter that the tasks can use as preferred.
- String Param2: parameter that the tasks can use as preferred.
- String Param3: parameter that the tasks can use as preferred.
- String Param4: parameter that the tasks can use as preferred.

5.3.2 Examples

Event task for documents, to be used when it's inserted, that is event INSert:

```
/*
 * Example that export to a folder the documents of a specific
mimetype/extension
 */
package eventdocexample;

import prodoc.CustomTask;
import prodoc.DriverGeneric;
import prodoc.PDDocs;
import prodoc.PDException;
import prodoc.PDLog;
import prodoc.Record;
```

```
/**
 *
 * @author jhier
 */
public class EventDocExample extends CustomTask
{

//---------------------------------------------------------------------
----
/**
 * Example of checking if a document meets the requirements. In ths
example, the document must be a PDf
 * @param Param1 Parameter 1 of the configured Event (in this example,
the extension that must have the doc)
 * @param Param2 Parameter 2 of the configured Event (In this example,
path to export to the document)
 * @param Param3 Parameter 3 of the configured Event
 * @param Param4 Parameter 4 of the configured Event
 * @param Rec  Metadata of the document that triggers the event
 * @param Drv OPD Session for retrieving additional information
 * @return true if the document meets the expected
 */
@Override
protected boolean CustomMeetsReqRec(String Param1, String Param2,
String Param3, String Param4, Record Rec, DriverGeneric Drv)
{
if (PDLog.isDebug())
    PDLog.Debug("EventDocExample.CustomMeetsReqRec.Param1=["+Param1+"]
Param2=["+Param2+"] Param3=["+Param3+"] Param4=["+Param4+"]
Rec=["+Rec+"]");
if
(((String)Rec.getAttr(PDDocs.fMIMETYPE).getValue()).equalsIgnoreCase(P
aram1))
    return(true);
else
    return(false);
}
//-----------------------------------------------------------------
/**
 *
 * @param Param1 Parameter 1 of the configured Event (in this example,
the extension that must have the doc)
 * @param Param2 Parameter 2 of the configured Event (In this example,
path to exporto to the document)
 * @param Param3 Parameter 3 of the configured Event
 * @param Param4 Parameter 4 of the configured Event
 * @param Doc    PDDocs Document that triggers the Event
 * @throws PDException In any Error
 */
@Override
protected void ExecuteEventDoc (String Param1,String Param2,String
Param3,String Param4, PDDocs Doc)  throws PDException
{
```

```
if (PDLog.isInfo())
    PDLog.Info("EventDocExample.ExecuteEventDoc.Param1=["+Param1+"]
Param2=["+Param2+"] Param3=["+Param3+"] Param4=["+Param4+"]
Doc=["+Doc.getRecSum()+"]");
Doc.ExportXML(Param2, false);
}
//-----------------------------------------------------------------
----
}
```

Event task for Folders, to be used, in example, when it's updated:

```
/*
 * Example that export to a folder all the documents to a Folder
 */
package eventdocexample;

import prodoc.CustomTask;
import prodoc.DriverGeneric;
import prodoc.PDException;
import prodoc.PDFolders;
import prodoc.PDLog;
import prodoc.Record;

public class EventFoldExample extends CustomTask
{
//---------------------------------------------------------------
/** Method that check if the object meets  the process
 * Check if the folder is under another folder
 * @param Param1 Parameter 1 of the configured Event
 * @param Param2 Parameter 2 of the configured Event
 * @param Param3 Parameter 3 of the configured Event
 * @param Param4 Parameter 4 of the configured Event
 * @param Rec with the data
 * @param Drv generic driver
 * @return true if the record attributes of Folder meets the
conditions
 * @throws prodoc.PDException in any error
 */
@Override
protected boolean CustomMeetsReqRec(String Param1, String Param2,
String Param3, String Param4, Record Rec, DriverGeneric Drv)  throws
PDException
{
PDFolders Fold=new PDFolders(Drv);
String IdUnder=Fold.getIdPath(Param1);
Fold.setPDId((String)Rec.getAttr(PDFolders.fPDID).getValue());
if (!Fold.IsUnder(IdUnder))
    return(true);
else
    return(false);
}
//-------------------------------------------------------------
/** Method that do the process
```

```
 * Example that exports the metadata of all the contained documents to
path Param1
 * @param Param1 Parameter 1 of the configured Event
 * @param Param2 Parameter 2 of the configured Event
 * @param Param3 Parameter 3 of the configured Event
 * @param Param4 Parameter 4 of the configured Event
 * @param Fold   PDFolders Folder object that triggers the Event
 * @throws PDException In any Error
 */
@Override
protected void ExecuteEventFold(String Param1, String Param2,String
Param3, String Param4, PDFolders Fold) throws PDException
{
if (PDLog.isInfo())
    PDLog.Info("EventFoldExample.ExecuteEventFold.Param1=["+Param1+"]
Param2=["+Param2+"] Param3=["+Param3+"] Param4=["+Param4+"]
Doc=["+Fold.getRecSum()+"]");
Fold.ExportDocs(Param2);
}
//--------------------------------------------------------------
}
```

Scheduled task:

5.3.3 Deployment

To use a Tasks Extension, the following steps must be verified:

1- Ensure that, if third-party libraries are required in addition to the Extension's own development, these are installed and configured on all the machines where the Extension will be used. OpenProdoc dynamically loads and configures the Extension, but NO additional libraries.

2- Incorporate to OpenProdoc, with any document type, the jar with the development of the extension. The PdId of this document should be saved, and will then be used as a reference.

3- Define a CUSTOM type Tasks with the required parameters and with the following Description value:

```
Jar PDId | Package name + class
```

Example:

```
1234fdge-567aabb5534|mipackage.MyTask
```

After that you can define run the tasks.

You can create several instances of the same type of tasks with different parameters.

It should be noted that documents containing the jar and properties may be versioned and modified following the same procedure as with any other document. When you start OpenProdoc you will always use the latest published version of any of them.

However, if the version of the jar or the properties is updated and that tasks has ALREADY been used, since a previous version is loaded and instantiated in the JVM, that new version will NOT be used. The JVM must be restarted so that the Java classloader can use the new version.

6 Parametrizations

6.1 Ribbon/Toolbar

Although OpenProdoc will display in the menu and ribbon only the functions allowed to the user, in order to optimize the interface and display in different order and size the buttons, it's possible to change the buttons of the toolbar and adapt it to the needs and work of each user.

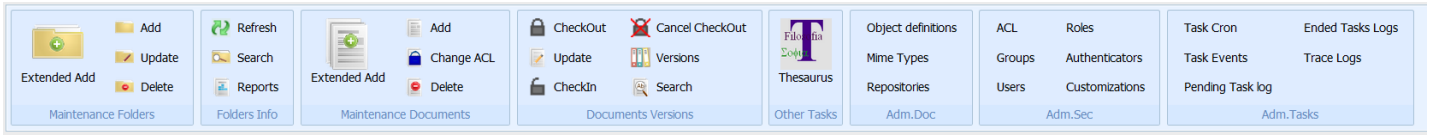
For changing the toolbar, it's only needed to create a document in the personal folder of the user (Ex.: */User/JohnSmith*) with the Title "Ribbon" and as content, an XML file.

The XML should be a modification of the default XML:

```
<?xml version='1.0' encoding='UTF-8'?>
<ribbon>
    <item id='Folders' type='block' mode='cols' text='Maintenance Folders'>
        <item id='AddExtF' type='button' isbig='true' text='Extended Add'
img='img/FoldAdd.png' />
        <item id='AddFold' type='button' text='Add' img='img/Fold.png' />
        <item id='ModExtF' type='button' text='Update' img='img/FoldEdit.png' />
        <item id='DelFold' type='button' text='Delete' img='img/FoldDel.png' />
    </item>
    <item id='Folders2' type='block' mode='cols' text='Folders Info'>
        <item id='RefreshFold' type='button' text='Refresh'
img='img/refresh.png' />
        <item id='SearchFold' type='button' text='Search' img='img/FoldSearch.png'
/>
        <item id='FoldReports' type='button' text='Reports'
img='img/Reports.png' />
    </item>
    <item id='Documentos' type='block' mode='cols' text='Maintenance Documents'>
        <item id='AddExtDoc' type='button' text='Extended Add' isbig='true'
img='img/DocAdd.png' />
        <item id='AddDoc' type='button' text='Add' img='img/Doc.png' />
        <item id='ChangeACL' type='button' text='Change ACL'
img='img/DocACL.png' />
        <item id='DelDoc' type='button' text='Delete' img='img/DocDel.png' />
    </item>
    <item id='Documentos2' type='block' mode='cols' text='Documents Versions '>
        <item id='CheckOut' type='button' text='CheckOut' img='img/checkout.png' /
>
        <item id='ModExtDoc' type='button' text='Update' img='img/DocEdit.png' />
```

```
        <item id='CheckIn' type='button' text='CheckIn' img='img/checkin.png' />
        <item id='CancelCheckOut' type='button' text='Cancel CheckOut'
img='img/cancelcheckout.png' />
        <item id='ListVer' type='button' text='Versions' img='img/ListVers.png' />
        <item id='SearchDoc' type='button' text='Search' img='img/DocSearch.png'/>
    </item>
    <item id='Other' type='block' mode='cols' text='Other Tasks'>
        <item id='Thesaurus' type='button' text='Thesaurus' isbig='true' img='img/
Thesaurus.png' />
    </item>
    <item id='AdminDoc' type='block' mode='cols' text='Adm.Doc'>
        <item id='ObjDef' type='button' text="Object definitions"/>
        <item id='MimeTypes' type='button' text='Mime Types'/>
        <item id='Repositories' type='button' text='Repositories'/>
    </item>
    <item id='AdminSec' type='block' mode='cols' text='Adm.Sec'>
        <item id='ACL' type='button' text='ACL'/>
        <item id='Groups' type='button' text='Groups'/>
        <item id='Users' type='button' text='Users'/>
        <item id='Roles' type='button' text='Roles'/>
        <item id='Authenticators' type='button' text="Authenticators"/>
        <item id='Customizations' type='button' text='Customizations'/>
    </item>
    <item id='AdminTasks' type='block' mode='cols' text='Adm.Tasks'>
        <item id='TaskCron' type='button' text='Task Cron'/>
        <item id='TaskEvents' type='button' text='Task Events'/>
        <item id='PendTasklog' type='button' text='Pending Task log'/>
        <item id='EndTasksLogs' type='button' text='Ended Tasks Logs'/>
        <item id='TraceLogs' type='button' text='Trace Logs'/>
    </item>
</ribbon>
```

that creates this Ribbon:



The structure of the xml is:

```
<ribbon>
  <item id='IdBlock' type='block' mode='cols' text='text of Block'>
   <item id='IdButton' type='button' text='Text of Button big image' isbig='true'
img='img/Thesaurus.png' />
   <item id='IdButton2' type='button' text='Text of Button no image'/>
   <item id='IdButton3' type='button' text='Text of Button small image'
img='img/checkin.png' />
  </item>
</ribbon>
```

You can change the order and items of blocks and the order of items. Also you can change the elements in “**bold**”. That is, it’s possible to change:

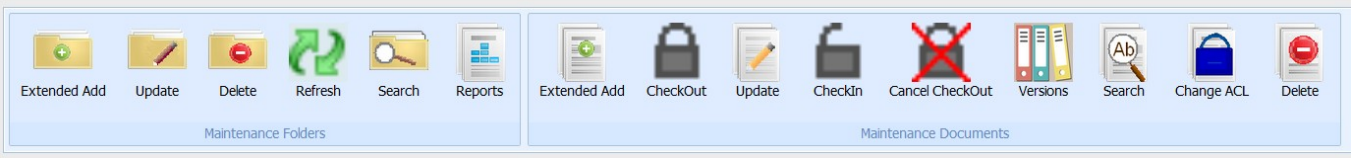
- Text of a block of buttons
- Text of each button
- Change the image
- Use a small or big image

Example:

This XML:

```
<?xml version='1.0' encoding='UTF-8'?>
<ribbon>
    <item id='Folders' type='block' mode='cols' text='Maintenance Folders'>
        <item id='AddExtF' type='button' text='Extended Add'  isbig='true'
img='img/FoldAdd.png' />
        <item id='ModExtF' type='button' text='Update' isbig='true'
img='img/FoldEdit.png' />
        <item id='DelFold' type='button' text='Delete' isbig='true'
img='img/FoldDel.png' />
        <item id='RefreshFold' type='button' text='Refresh' isbig='true'
img='img/refresh.png'  />
        <item id='SearchFold' type='button' isbig='true' text='Search'
img='img/FoldSearch.png'  />
        <item id='FoldReports' type='button' text='Reports' isbig='true'
img='img/Reports.png' />
    </item>
    <item id='Documentos' type='block' mode='cols' text='Maintenance
Documents'>
        <item id='AddExtDoc' type='button' text='Extended Add' isbig='true'
img='img/DocAdd.png' />
        <item id='CheckOut' type='button' text='CheckOut' isbig='true'
img='img/checkout.png' />
        <item id='ModExtDoc' type='button' text='Update' isbig='true'
img='img/DocEdit.png' />
        <item id='CheckIn' type='button' text='CheckIn' isbig='true' img='img/
checkin.png'  />
        <item id='CancelCheckOut' type='button' text='Cancel CheckOut'
isbig='true' img='img/cancelcheckout.png'  />
        <item id='ListVer' type='button' text='Versions' isbig='true'
img='img/ListVers.png'  />
        <item id='SearchDoc' type='button' text='Search' isbig='true'
img='img/DocSearch.png' />
        <item id='ChangeACL' type='button' text='Change ACL' isbig='true'
img='img/DocACL.png' />
        <item id='DelDoc' type='button' text='Delete' isbig='true'
img='img/DocDel.png' />
    </item>
</ribbon>
```

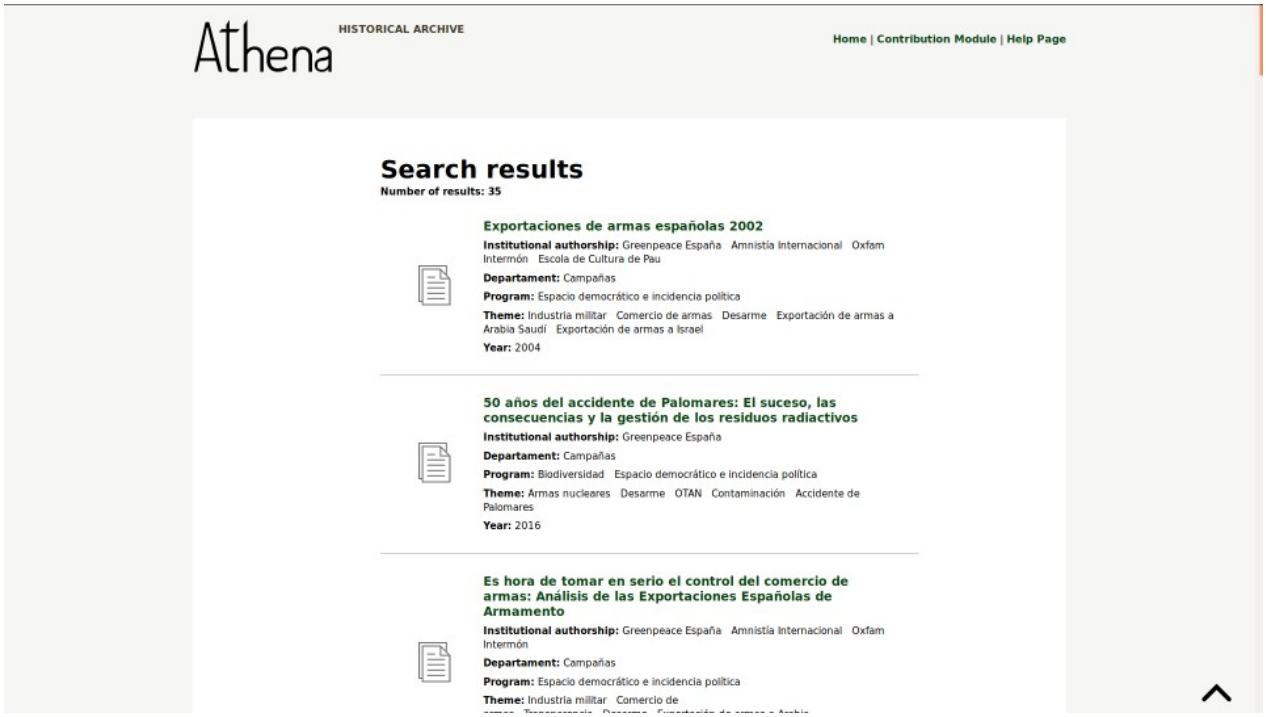
Will create this Ribbon:



6.2 Reports

The "Reports" function is developed for showing or exporting metadata from documents and folders in the desired format. The reports cover both functions because depending on the report file is possible to generate an html document formatted for good visibility or documents in CSV or XML format for exporting to other systems.

With the "Reports" function it's possible to obtain pages as:



The operation is as follows:

- A template document whose usual format is text (HTML, XML, CSV, TXT...) must be created. "Reports" generated have the same extension / mime type that the document template. That is a template with html extension will generate HTML documents that must meet that standard in their internal structure.
- The document must follow the syntax detailed below (Parametrization). Syntax basically involves combining:
    - Literals that will be displayed as introduced,
    - Variables representing metadata and will be replaced by the value stored in the metadata.

    - and control elements that repeat the elements contained within the boundaries to make nested loops over
        - the records/objects in the list
        - and, for each object, a loop for all the metadata.
        - and, for each metadata, a loop for all the values.
- Templates are not necessary specific to a document or folder type and can be applied to various types. Depending on the use of specific metadata, or the use of a loop for all metadata, the report may be applied to a document type, a type and all its subtypes or is valid for all types of objects (including both folders and documents).
- Once created the template, it must be stored in OpenProdoc, preferably in the "/System" folder (where already some examples are included) as PD_REPORT document type.
- This document type includes 2 additional metadata, the "number of documents per page" and the "number of pages per file." The reporting function will write the "report header" (content elements between the beginning of the document and the start of the loop of records), then repeat loop region record as many times as "documents per page" are defined, then write the report footer (content elements between the end of the loop of records and the end of the document) and then return to start another "page" with the same procedure. When you have write "number of pages per file", the file will be closed and a new one will be created, and so on until you have write all records from the list. Some examples are included at the end of this page.
- With the template created and stored in OpenProdoc, you can access the functions of reports from several points:
    - Folders Menu
    - Documents Menu
    - Results of Folders Search
    - Results of Documents Search
    - From the API Java, generating a Report with the cursor/list of documents/folders.
- At each point, the report function will receive a list of items to generate the report (taking into account the user's access permissions on documents).
- As answer to a request the generation a report from a template, the report will be composed and a reference to the outcome or results will be returned. If multiple files are generated with reports, in Web client a compressed file with all reports you will be returned.

6.2.1 Parametrization

The syntax of the OPD reports is as follows:

- If the line starts with the character **"#"** is considered a comment and content of that line is totally ignored.
- Before evaluating the line, trailing spaces are removed.
- If the line starts with the character **"+"**, the line will be added to the previous one **after** evaluation, deleting the character **“+”**, otherwise it's evaluated and write in a new line. Evaluation is made, in both cases, following the next lines.
- If line starts by **“@OPD”**, the line is evaluated as a "reserved word" from the next list, otherwise it is evaluated as a literal string and write to the file "as is".
- The reserved words list is:
  - **@OPD_DOCSLOOP_S**: Starts the records loop. All content from the beginning to the end (OPD_DOCSLOOP_E) of records loop is repeated for all objects of the list, whether literal or reserved words. After @OPD_DOCSLOOP_S the character **"-"** can appear followed by a list of names of types of documents or Folders separated by the **","** (eg "OPD_DOCSLOOP_S -Contract, Passport "). Such objects included in the list are ignored and will not be write on the report and the loop will not be repeated for them.
  - **@OPD_DOCSLOOP_E**: Ends the Records loop
  - **@OPD_ATTRLOOP_S**: Starts the Metadata loop. All content from the beginning to the end (@OPD_ATTRLOOP_E) metadata loop will be repeated for all metadata/attributes of each document/folder, whether literal or reserved words. After that the character can appear **"-"** followed by a list of metadata names separated by the **","** (eg "@OPD_ATTRLOOP_S -PDID, LockedBy, ParentId"). The metadata included in the list are ignored and will not be write in the report and the loop will not be repeated for them. Following @OPD_ATTRLOOP_S and before **"-"** 2 operator be included: **"*"** and **"?"**. The **"*"** (eg. "@OPD_ATTRLOOP_S*") indicates that OPD must retrieve all the metadata of the document or folder, not just those obtained in the search and returned in the list. This can occur if you search documents of a document type and its subtypes. In that case only metadata of document type father will be returned so that the structure is homogeneous. The **"?"** indicates that there should be NOT included in the metadata loop the empty metadata.
  - **@OPD_ATTRLOOP_E**: Ends the metadata loop
  - **@OPD_VALLOOP_S**: Values loop start (for multivalued attributes). All content (whether literal or reserved words) from the beginning to the end (@OPD_VALLOOP_E) of the values loop will be repeated for all values of metadata.
  - **@OPD_VALLOOP_E**: Ends the values loop (for multivalued attributes)
  - **@OPD_GLOBPARENT**: This expression will be replaced by the full path of the containing folder on which the search or list of items has started (ex. "/Files")
  - **@OPD_PARENT**: This expression will be replaced by the full path of the parent folder of the document or folder within the current list (ex.

"/Contracts/S-12345/Application", "/Contracts/Z-67896/Approved"). This variable can be different on each item if the report is generated after a search (which can locate elements at different levels) and will be equal when the items shown are in the same folder.

- **@OPD_NAME_ATTR**: Shows the internal technical name of an attribute within the attributes loop. It can be expressed as @OPD_NAME_ATTR_* (for all attributes) or @OPD_NAME_ATTR_NombreInternoAtributo (ex. @OPD_NAME_ATTR_TITLE). It can followed by **":"** and an integer that indicates the length of the text. If the value is less, it is truncated, otherwise spaces will be added. (Eg. @OPD_NAME_ATTR _*:20).
- **@OPD_UNAME_ATTR**: Shows the user/visible name of an attribute within the attributes loop. It can be expressed as @OPD_UNAME_ATTR_* (for all attributes) or @OPD_UNAME_ATTR_NombreInternoAtributo (eg. @OPD_UNAME_ATTR_TITLE). It can followed by **":"** and an integer that indicates the length of the text. If the value is less, it is truncated, otherwise spaces will be added. (Ex. @OPD_UNAME_ATTR_*:20).
- **@OPD_VAL_ATTR**: Shows the value of an attribute within the attributes loop. It can be expressed as @OPD_VAL_ATTR_* (for all attributes) or @OPD_VAL_ATTR_NombreInternoAtributo (eg. @OPD_VAL_ATTR_TITLE). It can followed by **":"** and an integer that indicates the length of the text. If the value is less, it is truncated, otherwise spaces will be added. (Eg. @OPD_VAL_ATTR_*:20).
- **@OPD_REF_ATTR**: Attribute value or reference. The behavior and syntax is the same as in the case of @OPD_VAL_ATTR, but if the variable is thesaurus, reference to a mime type or to the containing folder, rather than showing the value of the variable (element identifier, Eg "12e434_43af43 ") will show the value of the referenced term (eg" Portugal ").
- **@OPD_RECCOUNT**: Number of records written to the report until the current record.
- **@OPD_TOTALREC**: Total number of records of the report (0 if the report was generated with Cursor, not with Vector).
- **@OPD_PAGCOUNT**: Number of pages written to the report until the current record.

When creating an html report, it’s possible to construct html pages with urls that reference other documents (or the content of the own document itself) however it must be noted that the security is still active, so if you download an html that reference other documents or contents and there is no session in the browser, you will be unable to access those documents.

6.2.2 Examples

Assuming two document types (simplified for clarity) with the metadata (in brackets "username" metadata):

- PD_DOCS
  - PDId (PDId)
  - Title (Document_Title)
  - DocDate (Document_Date)
- Dossier (subtipe of PD_DOCS):
  - PDId (PDId)
  - Title (Document_Title)
  - DocDate (Document_Date)
  - Author (Author name)
  - Keywords (Keywords)

And a result list:

- PD_DOCS:
  - PDId=1001
  - Title="Document 1"
  - DocDate=2015/02/15
- PD_DOCS:
  - PDId=1002
  - Title="Document 2"
  - DocDate=
- Dossier:
  - PDId=1003
  - Title="Document 3"
  - DocDate=2001/04/25
  - Autor="John Smith"
  - Keywords="Economy", "Documentation"

This report template:

```
# Example of report template. This comment will not be shown
================================================================================

Folder Content:
+@OPD_GLOBPARENT

# Records loop start
@OPD_DOCSLOOP_S
Doc:
+@OPD_RECCOUNT
--------------------------------------------------------------------------------
Identifier=
+@OPD_REF_ATTR_PDID
@OPD_UNAME_ATTR_TITLE
+=
+@OPD_REF_ATTR_TITLE
@OPD_UNAME_ATTR_DocDate
+=
+@OPD_REF_ATTR_DocDate
--------------------------------------------------------------------------------

# Records loop End
@OPD_DOCSLOOP_E
```

```
Total Docs=
+@OPD_RECCOUNT
+ Page:
+@OPD_PAGCOUNT

================================================================================
```

Will create this report:

```
================================================================================

Folder Content:/Report Test

Doc:1
--------------------------------------------------------------------------------
Identifier=1001
Document_Title=Document 1
Document_Date=2015-02-15
--------------------------------------------------------------------------------

Doc:2
--------------------------------------------------------------------------------
Identifier=1002
Document_Title=Document 2
Document_Date=
--------------------------------------------------------------------------------

Doc:3
--------------------------------------------------------------------------------
Identifier=1003
Document_Title=Document 3
Document_Date=2001-04-25
--------------------------------------------------------------------------------


Total Docs=3 Page:1

================================================================================
```

And the Report:

```
<!DOCTYPE html>
<html>
<head>
    <title>OpenProdoc Picture Html Report</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <link rel="shortcut icon" href="img/OpenProdoc.ico" type="image/x-icon"/>
<style>
body
{
background-color: #FFFFFF;
font-family: Tahoma,Helvetica;
font-size: 12px;
}
table
{
font-size: 12px;
border: 2px solid grey;
width: 100%;
}
td
```

```
{
border: 1px dotted grey;
padding: 2px;
}
td:first-child
{
width: 30%;
}
h3
{
font-family: Tahoma,Helvetica;
font-size: 16px;
font-weight: bold;
}
thead
{
background:lightgray;
padding: 2px;
}
.ExternTab
{
/* width: 100%; */
border-style: none;
}
.ExternCol:first-child
{
width: 70%;
}
.ImgThumb
{
max-height: 250px;
max-width: 80%;
display: block;
margin-left: auto;
margin-right: auto;
/* border-style: outset; */
border-style: inset;
border-width: 6px;
border-color: lightgrey;
}
</style>
</head>
<body>
<img alt="Logo OpenProdoc" src="……. %3D%3D"/>
<H3>OpenProdoc Picture Html Report</H3>
@OPD_DOCSLOOP_S
<table class="ExternTab">
<tr><td class="ExternCol">
<table>
<thead>
<tr><td  style="text-align:center;">
+@OPD_VAL_ATTR_DocType
</td><td></td></tr>
<tr><td>Folder:</td><td>
+@OPD_PARENT
</td></tr>
</thead>
```

```
@OPD_ATTRLOOP_S?- PDId,DocType,Title, PurgeDate, Name, Reposit, Status, ParentId,
PDAutor, PDDate, Version
<tr><td><b>
+@OPD_UNAME_ATTR_*
</b></td><td>
@OPD_VALLOOP_S
+@OPD_REF_ATTR_*
+<br>
@OPD_VALLOOP_E
</td></tr>
@OPD_ATTRLOOP_E
</table>
</td>
<td>
<a href="SendDoc?Id=
+@OPD_VAL_ATTR_PDId
+" target=\"_blank\">
<img class="ImgThumb" src="SendDoc?Id=
+@OPD_VAL_ATTR_PDId
+"/>
</a>
</td>
</tr>
</table>
@OPD_DOCSLOOP_E
</body>
</html>
```

Will generate:

OpenProdoc Picture Html Report

Picture	
Folder:	/Tmp2
Document_ACL	Public
Author - Autor	Joaquin Hierro
Document_Date	2004-12-05
Keywords - Palabras Clave	Mountains Pandebano Picos de Europa
MimeType	image/jpeg
PixHeight - Altura	600
PixWide - Anchura	800

Picture	
Folder:	/Tmp2
Document_ACL	Public
Author - Autor	Joaquin Hierro
Document_Date	2005-11-09
Keywords - Palabras Clave	Evolution Fossils Technology
MimeType	image/jpeg
PixHeight - Altura	800
PixWide - Anchura	600

6.2.3 Deployment

The reports are deployed in OpenProdoc repository as documents of type: PD_REPORTS and downloaded when needed.

Being “normal” documents, they can be versioned and the access to them follow the same security model that any other document, by means of the ACL. When versioned,

OpenProdoc will use the last version, however, due the cache behaviour, the use of the last version can be delayed.

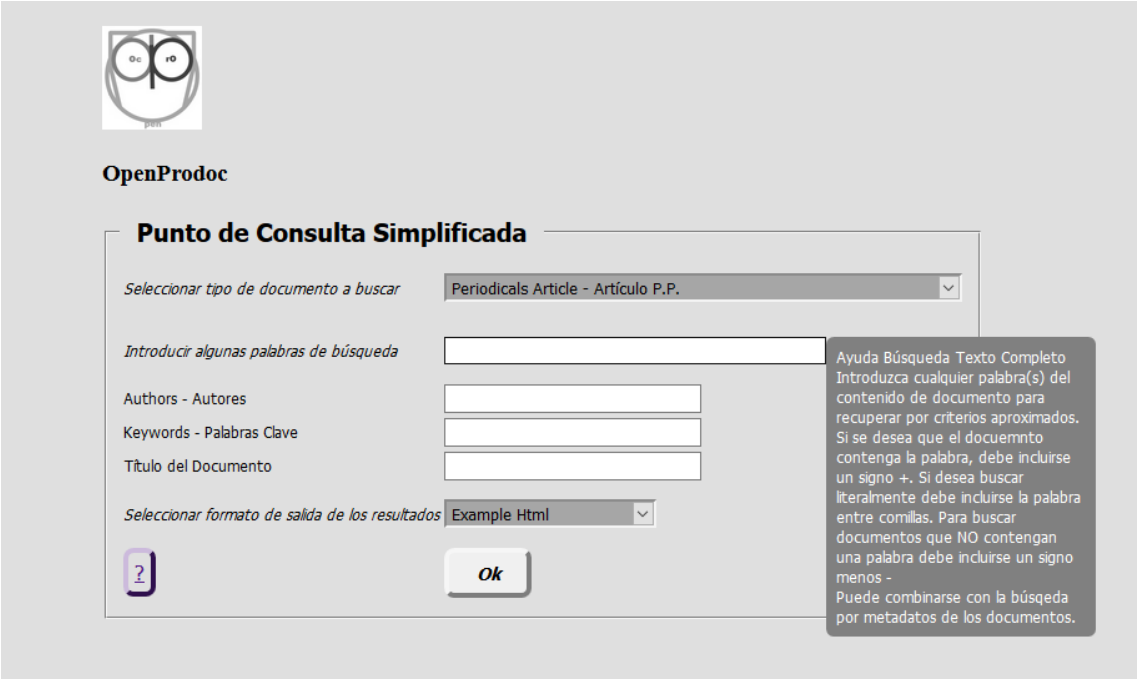
The Reports can be stored in any folder, however it's recommended to store then in the "System" folder. As they can be used combined with other elements (OPAC, Extensions, etc.) it's recommended to create a folder for the "project". The use of a folder allows also to export and import the complete folder and its contained document (the project/"package") between environments/installations just from the Web user interface.

6.3 OPAC

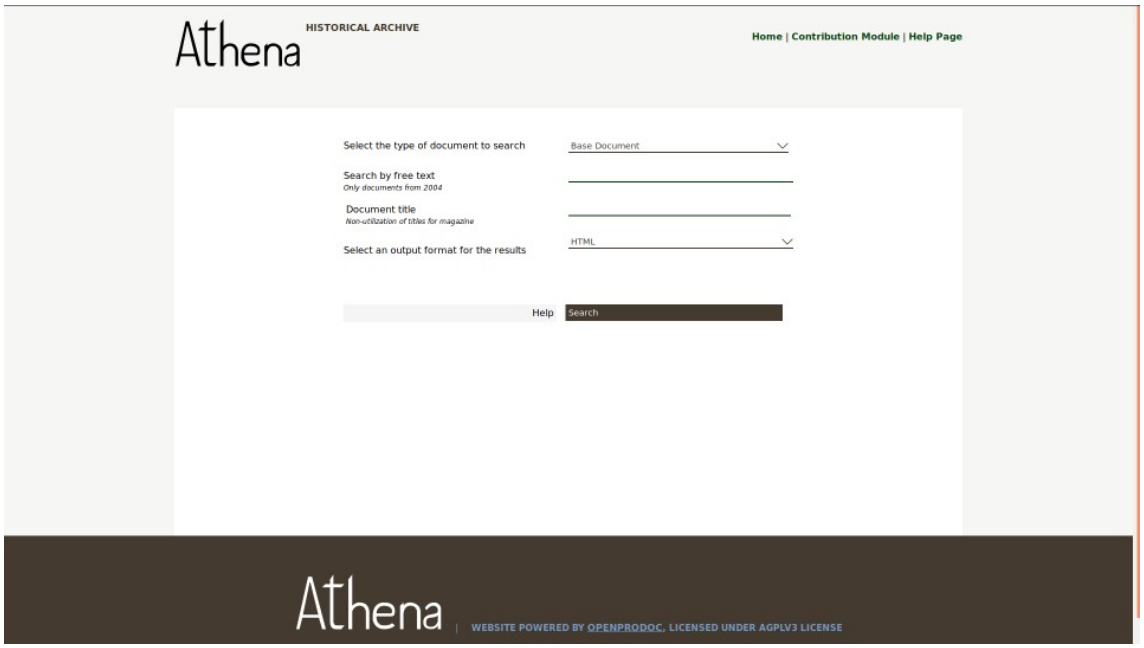
The OpenProdoc Public Access Catalog (sometimes referred in this documentation with the traditional term OPAC), allows people external to the company institution (or internal with a simplified interface and without login) to look for in an easy way the documents (or folders) stored in OpenProdoc without the need of a user. The OPAC allows to select the document (or folder) type to search for, to key in the search criteria and to select the format of Report to list the results.

The use of the OPAC requires to adjust parameters for an easy use and also for adding security, because we are allowing access to unknown people that could access to confidential documentation or damage something by error or intentionally. The parameters are defined by means of a properties file with the elements defined in the next list.

With the "OPAC" functionality it's possible to create search forms as:



Or



The process of parameterizing the OPAC is to create a text file with the indicated parameters (you can use as base the included example and modify it), entering the appropriate values to the installation of OpenProdoc.

It's necessary to modify the configuration file (Prodoc.properties or the name used) and add two lines:

User=User1

Pass=PassUsr1

Where User1 y PassUsr1 would be the data of a user with access to the configuration documents (OPAC y OPAC_CSS) used for creating the OPAC. It's recommended that the user used had a limited role and a minimum of permissions, because it's only needed read access to the configuration files. As any other change in the configuration file (Prodoc.properties), the server MUST be restarted so it can read the new configuration.

Next, you must create (or modify the included example) the CSS to adapt it to the style of the corresponding institution or company (or reference the official one in any url). Then add the CSS to the OpenProdoc repository and note the generated PdId. That PdId is the one that should be referenced as parameter FormSearchCSS in the OPAC configuration file. You should then upload that OPAC configuration file.

With the generated ID of the OPAC configuration file, the OpenProdoc url for the OPAC can be published: with the form:

http://localhost:8080/ProdocWeb2/OPAC?Id= + the new document ID uploaded,

In example:

http://localhost:8080/ProdocWeb2/OPAC?Id=566b6464a654-9696e68d686.

For searching folders use the OpenProdoc url for folders:

*http://localhost:8080/ProdocWeb2/OPAC**f**?Id= + the new document ID uploaded,*

In example:

*http://localhost:8080/ProdocWeb2/OPAC**f**?Id=566b6464a654-9696e68d686*

OpenProdoc will create and show a page according to the parameters defined in that file, including style sheet, literals, document types, etc.

In that page the users, without entering access login data (user / password), can choose one of the document types they want to search for, enter words from the content or known metadata, choose the format of the results and search.

When the search button is clicked, OpenProdoc will connect using the user indicated in the parameterization of OPAC, will search according to the included criteria and return the results in the chosen format of Report. The format can be html to present with a more aesthetic format, txt or csv for an automatic treatment, or xml to be able to interchange or to process.

Since all appearance is parametrized, including typologies of documents, metadata, etc., and is aimed at sporadic users, it is reasonable to assume that a personalized help is needed. For that you have a button that will open the html page whose url is indicated in a configuration. It could be an external page or an html document stored in OpenProdoc.

It should be noted that the configuration file, like any OpenProdoc document can be edited and versioned, always being used to compose the OPAC the last published version. However the upgrade may take some time, because to improve performance the configuration is not updated immediately.

Since the parameterization is based on one OpenProdoc document and the style in another, it is possible to have several parameterizations simultaneously, presenting different document sets, different interface language, different style or different query user. Providing each group of users with the appropriate url (which will only differ in the Document Identifier) many different OPACSs can be simultaneously offered with very little effort.

If you need to modify the default html template for OPAC more than just using CSS, or to adapt the html to different browsers, it's possible to use different html templates. For using this functionality you must include additional entries in the OPAC confguration file.

If the User-Agent of the browser do not contains any of the texts of the entries, then OpenProdoc will use the internal html template

6.3.1 Parametrization

This file follows the standard format of the properties files (label/element+"="+Value), with comments (the lines starting with the character '#'). The meaning of each entry (that can't be repeated) is:

- **DocTipesList**: Names of the Document types by which the user is allowed to search. It must be noted that if the param **Inheritance** is enabled (equal to 1), automatically all the subtypes of any of the selected document types will be included in the search. For searching folder (using Url OPACf), you must include folders types.
- **FieldsToInclude**: Names (Internal names, not the user names) of the metadata/fields (of any type included in the previous list) by which the user is allowed to search. In the form will be showed (and searched for) the subset of the listed metadata of the type selected. That is each time you select a type, the form will be updated showing the subset of fields of the type included in the list. This list has no relation with the fields shown in the results form, which depend on the reports selected in **ResultForm**.
- **FieldsComp**: Search / compare operators for each metadata. By default it is equality (EQ) i.e. it searches for all documents (or folders) whose value of that metadata is equal to the one entered. The possible values are 2 letter characters: EQ (=), NE (<>), GT (>), GE (>=), LT (<), LE (<=), CT (Contains , Like). Different values can be assigned such that for example a date-type metadata is searched for values greater than that entered using GE. The CT operator allows you to search for metadata that CONTAINS the entered value, which facilitates the search but slows it down and can generate too many results.
- **BaseFolder**: Folder tree below which searches will be performed. The rest of the documents stored outside that tree will not be returned.
- **Inheritance**: If this parameter is enabled (1) the search will include all subtypes of documents of the selected document type. If it is disabled, it will only include the selected type. This option does NOT affect performance, so the criteria for using it is purely a functional one.
- **ResultForm**: List of identifier codes (PDId) of documents of type Reports (PD_REPORTS) that will be used to present the results, separated by character “|”. At least 1 Id must be included.
- **MaxResults**: Maximum number of results to be returned (0 = "no limit", actually 1 million).
- **FormSearchCSS**: Identifier code (PDId) of the CSS file to be used for the query form, or url to an external file, starting with http. The predefined styles on the page are detailed in OPAC CSS Styles

- **FormSearchLogo**: URL of the logo to be presented in the query form. It can be external (ex: "http://intranet.empres.com/img/Logo.jpg") or internal (ex "SendDoc?Id = 44345543-757656") using as the value of the Id parameter, the unique code PDId of an image in OpenProdoc. It should be verified that this document is visible to all users.
- **User**: User with which the OpenProdoc queries will be performed. It should be checked that this user has access to the documents but no permissions of another type, since the Query Point makes a real connection that could allow you to perform operations. The recommendation is to create a specific user to query and assign a Role without any permission (with which he can only search)
- **Pass**: Password (clear) of the query user.
- **Title**: Header of the search form.
- **DTLabel**: Label of the drop-down list of document types (or folder types) on which the search can be performed.
- **FTLabel**: label of the full text search field for documents. (Only for searching documents)
- **FormatLabel**: Label of the results Reports drop-down list (defined in the ResultForm parameter)
- **HelpForDocType**: Text of the pop-up help that will be displayed when you are in the document (or folder) type selection combo box. Can contain html control characters.
- **HelpForFullText**: Text of the pop-up help in the field of full text search of the documents. Can contain html control characters.(Only for searching documents)
- **HelpForFormatType**: Text of the pop-up help that will be displayed when you are in the reports format selection combo box. Can contain html control characters.
- **UrlHelp**: Url of the personalized help page for the OPAC that will be presented to the users by clicking the help button.
- **NumHtmlOpac**: Number of alternatives HTML for supporting specific needs or formats of browsers or just for using a different template form the internal one. After this parameter, you must include the same numbers defined in NumHtmlOpac of pairs of the next parameters.
- **ListAgent**[i]=Text1|Text2|...: You must include as many entries as defined in **NumHtmlOpac** starting in 0, that is: ListAgent0, ListAgent1, ListAgent2,.. . If the description of "User-Agent" of the browser CONTAINS any of the text "Text1", "Text2"... then the html template for the OPAC will be a document stored in OpenProdoc with PDID specified in **HtmlAgent**[i]. If the Text is *, then OpenProdoc will not check other values and will return the html specified in **HtmlAgent**[i] for any browser.

- **HtmlAgent**[i]=PDID Document: Identifier of the html document to use of template uf the User-Agent contains any of the text specified in **ListAgent**. The numbers must be start in 0, that is: HtmlAgent0, HtmlAgent1, HtmlAgent2...

6.3.2 Examples

```
##############################################
####               OPAC                   ####
##############################################
# Names of the types of documents a user could search for
DocTipesList=Article|ECM_Standards|InternetProfile|MusicRecords|Picture
# Metadata/Fields (of any of the document types) internal names that a
# user can search for
FieldsToInclude=Author|Authors|Keywords|Player|Title|Country|CreativeCommons
# Search / comparison operators. 1 for each metadata. Default is EQ equality
# Possible values = EQ, <> NE, > GT, >= GE, < LT, <= LE, Contains CT
FieldsComp=EQ|EQ|EQ|EQ|CT|EQ|EQ
# Folder below which searches will be performed
BaseFolder=/Examples - Ejemplos
# Search Extended to subtypes of selected document types
Inheritance=1
# Identification codes (PDId) of documents of type PD_REPORTS that will be used
to
# display the results. Must be at least 1.
ResultForm=150c9be080c-3fe46f69eb1b2cb7|150c9be8462-3fd76612bb72fece
# Maximum number of results to be returned. (0 = unlimited)
MaxResults=0
# Code identifier (PDId) of the style sheet (CSS) to be used for the OPAC
FormSearchCSS=15db73b6628-3fee99cd40e27fee
# Url (internal or external) of the image used as logo
FormSearchLogo=img/LogoProdoc.jpg
# User with whom the query will be made
User=Invitado
# Login password of the user with whom the query will be made
Pass=PassInvit
# Search Form Header
Title=OpenProdoc OPAC Example
# Label of the drop-down list of document types to search
DTLabel=Select document type for search
# Full text search field label for documents.
FTLabel=Key in some words for search documents
# Label from the drop-down list of output formats.
FormatLabel=Select results format
# Text of the pop-up help that will be displayed when you are in the document
type
# selection combo box. Can contain html control characters.
HelpForDocType=List of document types you can look for
# Text of the pop-up help that will be displayed when you are in the fulltext
# search field. Can contain html control characters.
HelpForFullText=Help FullText<br>Please, wriet any word(s) that you now are
included in the docuemnts you are lookin for.
# Text of the pop-up help that will be displayed when you are in the report type
# selection combo box. Can contain html control characters.
HelpForFormatType=Select the tormat you want to receive the results of the
search.
# Url pf the complete help page for the configured OPAC
```

```
UrlHelp=help/EN/HelpIndex.html
#OPTIONAL, it is possible to use additional html templates for the OPAC as
# alternatie to the internal one. #Number of Htmls alternatives
NumHtmlOpac=2
#2 entries for each number of html, starting in 0
#------- 0 -
# When the client/agent contains this text
ListAgent0=Edge|Explorer
# The OPAC will use as base the html of this document (not the internal one)
HtmlAgent0=1632c1c280b-3fa2b6496a3c3250
#------- 1 -
# When the client/agent contains this text
ListAgent1=Firefox
# The OPAC will use as base the html of this document (not the internal one)
HtmlAgent1=1632c1dd6a2-3fe55f2e09ab3605
```

6.3.3 Deployment

The OPACs are deployed in OpenProdoc repository as documents of any type and downloaded when needed.

Being normal documents, they can be versioned and the access to them follow the same security model that any other document, by means of the ACL. When versioned, OpenProdoc will use the last version, however, due the cache behaviour, the use of the last version can be delayed.

The OPACs can be stored in any folder, however it's recommended to store then in the "System" folder. As they can be used combined with other elements (Reports, Extensions, etc.) it's recommended to create a folder for the "project". The use of a folder allows also to export and import the complete folder and its contained document (the project/"package") between environments/installations just from the Web user interface.

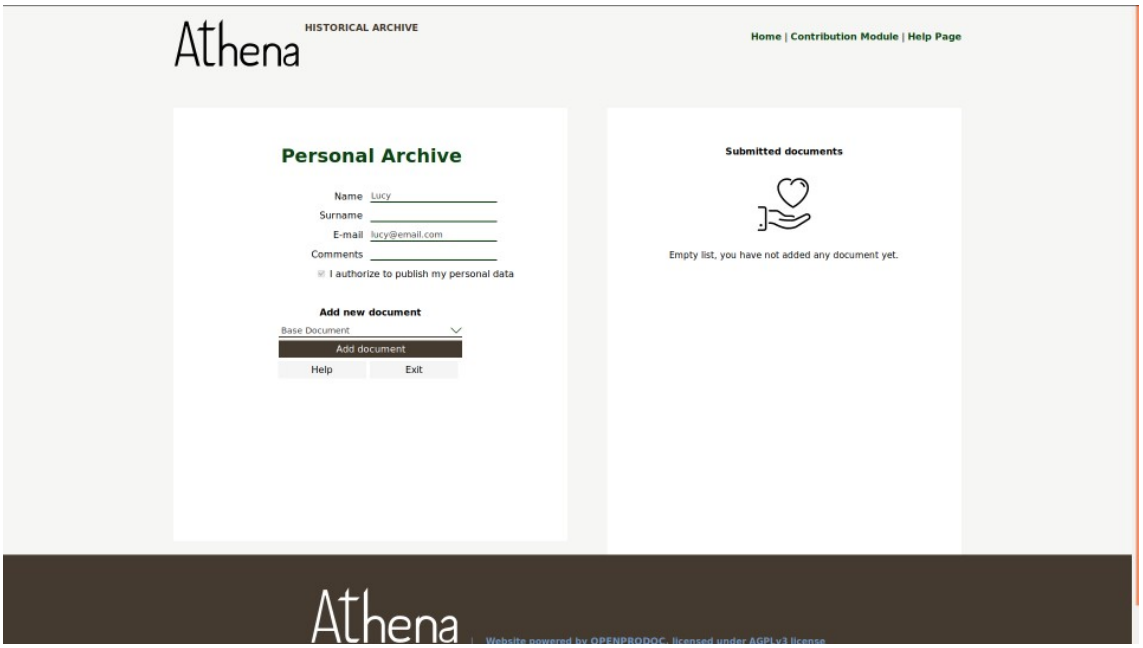
6.4 OPAD

The OpenProdoc Contribution module allows to insert documentation to users not registered in OpenProdoc by means of totally configurable forms, similar to the OPAC search module that allows to search documents and folders to external users.

This can facilitate users external to the entity, or internal but not registered and who will access punctually, being able to provide documentation without the need for training or knowledge of OpenProdoc, and avoids the management of users when it is necessary to deal with a large number of very specific collaborators, such as collaborators, partners or clients of an institution. As in the case of the OPAC, security is critical, since confidential information could be accessed, documents can be manipulated or files that are harmful or too large can be uploaded. To minimize risks, the contribution module has several forms of security and control that are detailed later.

With the function "Contribution Module" you can create forms like these:

Or



The operation of the "Contribution Module", which only requires parameterization, is as follows:

A folder should be chosen or created within OpenProdoc where the contributions of documents will be received and a type of folder should be created (chosen) to group those

documents into "cases" with the meaning that they want to give (donor, source, fund , ...), with all the metadata that is considered necessary to define the "case".

Within the total set of metadata, you can choose a subset of them that the external user can see, and enter if the system is defined as "open" (see the Security and Control section below), and some metadata that will be used to verify the identity of the external user if it is connected more than once or if the system is "closed".

In addition, the document types to be accepted, the metadata to be entered for each type of document and the file formats must be chosen.

When an external user connects, if the system is "open" it will present all metadata defined as "public" for the type of folder in the configuration. The user will fill in all the mandatory metadata (among which will be the "verification" metadata) and when accepting a folder will be created (contained in the main folder of contributions), within which all the documents that will be uploaded will be stored. Authorized users within OpenProdoc can modify the metadata entered or modify metadata of the "internal" use folder, adding, in example, notes.

Once the personal folder is accessed, the documents introduced up to that moment will be displayed and a new document type to be added can be chosen. The contribution module will present an input form with the metadata of the chosen documentary type, as well as a control to choose and upload the document itself. When accepting, the document will be incorporated into the folder, unless any restriction is breached, either by metadata (as mandatory, uniqueness, etc.) or by the document itself (extension not allowed, excessive size, ...) in which case error information will be presented. In another case, a confirmation screen will be presented and the content of the personal file will be presented again, with the documentation provided so far.

When the user wants to end the session, they can leave the system and return to the login screen.

If you want to provide documents in successive sessions you must enter the verification metadata so that the system can verify your identity and prevent a user from seeing the contents of a "case" that is not theirs. This metadata fulfils the authentication function that is usually fulfilled by the combination of username and password, so that no other external user can enter the personal folder (authorized internal users through ACL) can always enter and perform the operations that the ACL allows them).

If a system is closed, the difference is that only the verification metadata will be presented and that the folder MUST be already be created. In other words, the staff responsible for OpenProdoc will have created the appropriate folders previously and sent the access / verification information to the users who will be connected.

The "open" model can be applicable to a scenario in which a large number of unknown users will provide documentation, while the “closed” model would be applicable to a scenario

where an institution works with a small number of collaborators or external partners, with which wants to share information but without creating internal users of OpenProdoc. We must highlight the term "share", because in addition to uploading, all documents that are deposited in the folder case will be visible by the connected user. In other words, the institution can store documents that want the external user to see.

In addition we must bear in mind that in that folder documents could be inserted with another ACL, so that they cannot be seen by the external user but can be seen by internal staff.

Once the information has been entered, an internal user with permissions on the folder can review the documentation provided, modifying metadata if necessary (for which another version should be created) or reclassifying the documents (exporting and importing again) if the classification was wrong. Additionally, once the documentation is reviewed, it could be moved to another "public" folder from where it can be consulted through the OPAC or included in the corresponding document process.

6.4.1 Parametrization

The contribution module manages 4 pages to perform its functions. These pages, with a default structure already included in OpenProdoc, can be replaced by other pages (or even fragments of page or iframes if it is structured appropriately) indicating it in the parameters NumHtmlCont * and the rest of the related parameters.

The pages / steps that make up the contribution module are:

- **Login**: Entry form to request authentication data and file creation data if the system is open.
- **List**: Page that shows the data of the file and all the documents contained in it, and that allows to start the contribution of new documents.
- **Upload of documents**: Page that allows you to incorporate documents and fill in the metadata for the type of document chosen.
- **Confirmation**: Page showing the result (success or error) of the operation incorporating a document.

You can create as many contribution files as you want. Each one will represent a "contribution area" with different characteristics and interface. Although in principle they will be separated, CSS, internal user access and even the container folder could be shared. As in the case of the OPAC, it will be invoked by means of a call, including as a parameter the identifier of the contribution file.

The file follows the usual standard of property files (tag + "=" + Value), admitting comments (lines that begin with the character '#'). The meaning of each label (which cannot be repeated) is as follows:

- **LoginFolderType**: Type of folder that will be used to store and contain documents contributed by external users. A folder of this type will be created for each user. It can contain the number and type of metadata considered necessary, including access metadata, information provided by the external user and "internal" metadata not visible to external users. To avoid duplication, the type could include metadata with unique values. You could also define tasks associated with events to assign or change metadata values if desired. Ex *LoginFolderType=Donation*
- **LoginFields**: This entry contains the list of metadata of the type of folder that will be used to verify the identity of the user, separated by character “|”. If the system is closed, only those metadata will be presented and they will always be requested. If it is open all those included in **FieldsToInclude** will be presented and after the first time (in which the folder is created) the following times will only be required. The number of verification metadata can be variable. The minimum is one, but two, three, etc. can be included, according to the security and confidentiality required by the system. Keep in mind that these metadata, from the point of view of OpenProdoc are "normal" metadata and therefore are not ENCRYPTED, so it is not advisable to collect confidential information in them, as OpenProdoc users with access to these folders could see them. Ex: *LoginFields=Mail|Phone*
- **FieldsToInclude**: This entry contains all the metadata of the type of folder that will be allowed to view (always) or edit (if open) the user. The login metadata should be included in the list. Additional metadata can be defined for internal use by OpenProdoc users (Notes, codes, dates, etc.). Example: *FieldsToInclude=FirstName|LastName|Mail|Phone|Language*
- **DocTipesList**: This parameter contains the list of document types that the user can provide. Keep in mind that it only limits the types of documents to be contributed, but that in the case folder there may be other documents (introduced by an OpenProdoc internal user, for example, as an aid or to exchange documentation.) Example: *DocTipesList=PD_DOCS|Manual|Picture|Recordings*
- **Metadata list**: For each document type included in DocTipesList an entry with the name: **Fields_*NameType*** can be specified, including a list of the metadata of that document type that the user will be requested. Any metadata not included in the list will not be requested (although it may be presented if the Report used to show results contains it). If an entry is not included for any of the types included in DocTipesList, ALL metadata of that type will be presented. The internal metadata is always excluded from the list: ACL, Doctype, LockedBy, MimeType, Name, ParentId, PDAuthor, PDDate, PDId, Purgedate, Reposit, Status, Version .Ex. *Fields_Picture=Title|Author|Keywords|DocDate*

- **OpenContrib**: Indicates if the system is "open", that is, anyone can create folders or if it is closed and therefore the folders are already created and the external user that connects must know ALL the verification metadata to access the corresponding folder. Example: *OpenContrib=1*
- **BaseFolder**: Indicates the path of the folder within which all folders of the type indicated in **LoginFolderType** will be created. It must be a folder where the internal **User** used for the contribution has at least write and update permissions. Ex: *BaseFolder=/Archive/UserDonations*
- **User**: OpenProdoc user used internally to connect and create folders and documents. It must be a user with limited permissions on folders (ideally only on BaseFolder) and role (only inserting folders and documents) as much as possible to minimize security risks. Ex *User=guest1*
- **Pass**: Password in "clear" of the selected user. Example: *Pass=PassGuest1*
- **AllowedExt**: List of allowed extensions to upload. Any other extension will be rejected, avoiding the upload of dangerous files. Ex.: *AllowedExt=:doc|docx|xls|xlsx|ppt|pptx|txt|pdf|jpg|jpeg|tiff|tif|png|gif|odt*
- **MaxSize**: Maximum size in bytes of the files to be uploaded, in order to avoid overload of Filesystem. Ex.: *MaxSize=20000000*
- **ContribCSS**: Identifier of the css style sheet to be used. It can be an identifier of a file hosted in OpenProdoc or an external url. The predefined styles on the page are detailed in CSS Styles Contribution. Example: *ContribCSS=http://www.portalCorporativo.com/css/estandard.css* or *ContribCSS=16697ec3694-3fe7288b86493159.*
- **ContribLogo**: Url of the logo of the contribution screens. It can be an absolute url or a reference to an image hosted in OpenProdoc. Ex: *ContribLogo=http://www.portalCorporativo.com/imgs/Logo.jpeg* or *ContribLogo=SendDoc?Id=43436565-aefe43434*
- **Title**: Title to show in the contributions screen. Example: *Title=Input Donations Documentation*
- **TitleList**: Title to show on the list of documents contained in the folder. Ex: *TitleList=Documents contributed so far*
- **DocsReportId**: Identifier of the Report to be used to show documents contained in the folder. It should be a report with a fragment of the html page that can be "embedded" (that is not a complete html with head, body, …). Example: *DocsReportId=16654ff6af1-3f9b78099c0147a0*
- **UrlHelp**: Help page that explains the process in a general and complete way. Ex: *UrlHelp=http://www.portalCorporativo.com/aportaciones/HelpDon.html*
- **OKMsg**: Text to show when the document has been properly incorporated. Ex: *OKMsg=Document Loaded Correctly*

- **NumHtmlContLog**: Number of alternative login html pages. For each of the groups of agents that are listed, an identifier of an OpenProdoc document containing an html page must be defined. You must create as many pairs **ListAgentLog**[i], **HtmlAgentLog**[i] as indicated by this entry, starting at zero. Example: *NumHtmlContLog=2*
- **ListAgentLog**[i]: List of web agents for which the corresponding html of equal "subscript" must be returned. It is not necessary to include the full name of the agent, just a fragment of it, which may include name, version, etc. Ex: *ListAgentLog0=Firefox*
- **HtmlAgentLog**[i]: Identifier of the OpenProdoc document with the login html that must be returned for all the agents of the same subscript. Example: *HtmlAgentLog0=57576abf4-6565dde4*
- **NumColAgentLog**[i]: Indicates whether the table containing the metadata list of the access form has a column or two. If its value is 1, it contains a column. If it is zero or not informed, it will have two Ex.: *NumColAgentLog0 = 1*
- **NumHtmlContList**: Number of alternative html pages to present the list of documents. For each of the groups of agents that are listed, an identifier of an OpenProdoc document containing an html page must be defined. You must create as many pairs ListAgentList[i], HtmlAgentList[i] as indicated by this entry, starting at zero. Ex: *NumHtmlContList=1*
- **ListAgentList**[i]: List of web agents for which the corresponding html of equal "subscript" must be returned. It is not necessary to include the full name of the agent, just a fragment of it, which may include name, version, etc. Ex: *ListAgentList0=Firefox|Edge*
- **HtmlAgentList**[i]: Identifier of the OpenProdoc document with the document listing html to be returned for all the agents of the same subscript. Example:*HtmlAgentList0=574343abf4-86976ddaa3*
- **NumColAgentList**[i]: Indicates whether the table containing the metadata list of the folder listing form has a column or two. If its value is 1, it contains a column. If it is zero or not informed, it will have two Ex.: *NumColAgentList0=1*
- **NumHtmlContAdd**: Number of alternative html pages to incorporate documents. For each of the groups of agents that are listed, an identifier of an OpenProdoc document containing an html page must be defined. You must create as many pairs **ListAgentAdd**[i], **HtmlAgentAdd**[i] as indicated by this entry, starting at zero. Example: *NumHtmlContAdd=3*
- **ListAgentAdd**[i]: List of web agents for which the corresponding html of equal "subscript" must be returned. It is not necessary to include the full name of the agent, just a fragment of it, which may include name, version, etc. Example: *ListAgentAdd1=Edge|Firefox|Opera*

- **HtmlAgentAdd**[i]: Identifier of the OpenProdoc document with the document incorporation html that must be returned for all the agents of the same subscript. Example: *HtmlAgentAdd0=123456abf4-4477238dda3*
- **NumColAgentAdd**[i]: Indicates whether the table containing the metadata list of the Document Incorporation form has a column or two. If its value is 1, it contains a column. If it is zero or not informed, it will have two Ex.: *NumColAgentAdd0=1*
- **NumHtmlContRes**: Number of alternative html pages of results of the incorporation of documents. For each of the groups of agents that are listed, an identifier of an OpenProdoc document containing an html page must be defined. You must create as many pairs **ListAgentRes**[i], **HtmlAgentRes**[i] as indicated by this entry, starting at zero. Example: *NumHtmlContAdd=3*
- **ListAgentRes**[i]: List of web agents for which the corresponding html of equal "subscript" must be returned. It is not necessary to include the full name of the agent, just a fragment of it, which may include name, version, etc. Example: *ListAgentRes0=Chrome|Opera*
- **HtmlAgentRes**[i]: Identifier of the OpenProdoc document with the results html document incorporation that must be returned for all the agents of the same subscript. Example: *HtmlAgentRes0=1885ffeebf4-4900462aaf3*

If the browser User-Agent does not contain any of the indicated entries, then the internal templates included in OpenProdoc will be used. Alternative pages can be included only for any of the elements (login, list, ..), it is not necessary to do it for every kind of page.

The process of parameterizing a contribution module consists in creating a text file with the aforementioned content (the included example can be used as a base and modified), introducing the appropriate values to the specific OpenProdoc installation

Previously you must modify the OpenProdoc configuration file (Prodoc.properties or the name used) and add 2 lines:

User=User1

Pass=UserPass1

Where User1 and UserPass1 will be data of a user that has access to the different documents that configuration (Contrib and Contrib_CSS) that are described in this help. It is recommended to be a user with a limited role and with a minimum of permissions, since you should only be able to access the configuration files of the Contribution in read mode. As with any change in the configuration file (Prodoc.properties), the server must be restarted to be read again.

Next, the CSS must be created (or modified the example included) to adapt it to the style of the corresponding institution or company. Then you must add the CSS to the OpenProdoc repository and write down the generated PdId. This PdId is the one that should be referenced as

parameter ContribCSS: of the configuration file of the contribution module. Then that configuration file of the contribution module will be uploaded.

With the generated Id the OpenProdoc OPAD url should be invoked, that is:

http://localhost: 8080/ProdocWeb2/ContribLogin?Id=identifier of the document uploaded

for example

http://localhost:8080/ProdocWeb2/ContribLogin?Id=566b6464a654-9696e68d686

OpenProdoc will present a page according to the parameters defined in that file, including style sheet, literals, document types, etc.

Since all the appearance is parameterized, including document typologies, metadata, etc., and due it is oriented to sporadic users, it is reasonable to assume that a personalized help is necessary. For that there is a button that will open the html page whose url is indicated in the configuration. It could be an external page or an html document stored in OpenProdoc.

It should be noted that the configuration file, like any OpenProdoc document, can be edited and versioned, always being used the last published version to compose the contribution module. However, the update may take some time, since to improve performance, the configuration is not updated immediately because a cached copy is used.

Since the parameterization is based on one OpenProdoc document and the style on another, it is possible to have several settings simultaneously, which present different sets of documents, different interface language, different style or different user. Providing each user group with the appropriate url (which will only be differentiated in the Document Identifier), many different contribution modules can be offered with very little effort at the same time.

6.4.1.1 Automation

To facilitate the work of internal users, it is advisable to define automatic tasks that notify changes and normalize entries.

For example, if the folder has metadata Name, Surname and DNI, it may be advisable to create a task associated with the "Insertion" event of type "modify metadata", normalizing the title of the folder with the "formula"="Title = DNI + "-" + Surname + ", " + Name (the exact syntax is not this, but it is used for clarity). This ensures that the nomenclature is homogeneous regardless of who enters the data.

You could also create a scheduled task that sends a report every night to a certain group with the data of all the created or updated folders (that is, they have new documents) in the day, so that you can automatically know what new documentation has been received and files have to be reviewed without needing to review them one by one.

6.4.1.2 Security and Control:

Since external access to a document manager by unauthenticated external users may involve risks of various kinds, various measures have been introduced to minimize the risk:

The definition of a system as a closed system limits that only users who know information of the folders/ files can enter the system. Even if the system is open, once the file is created, only the user who knows the required access data can login in the system. For all purposes, the behaviour is similar to the creation of users, who must know the user and password to enter, requiring in the case of the contribution module, the knowledge of 2 or more fields in the file (ex NIF and telephone, name and code).

The user of the OpenProdoc application that is used internally to connect from the contribution module should be a user with the minimum permissions. Basically must have a role that only allows you to create documents (and create folders if the system is open, otherwise it is not necessary) and only have access to that folder. That will minimize the risk if somehow an external user could access with that internal user.

To prevent the introduction of damaging harmful files, the list of allowed extensions can be parameterized. If a document that you try to upload does not have any of the extensions included in the list, it will be rejected. This will prevent the incorporation of executable programs (exe, com, dll ...) or script (bat, sh, vbs...) that may contain viruses or harmful code.

To avoid overflow of the system, the maximum size of each file to be uploaded can be limited, so that multiple Gigabyte files cannot be incorporated to fill the file system and block operation.

The possibility of creating different configurations (configuration files) allows you to create separate areas where different types or groups of users can collaborate. Even if data could be obtained to access a contribution area, there would be no access to another area.

If the documentation contained in the repository includes confidential or especially sensitive documentation, to increase security the recommendation would be to have TWO installations, one dedicated solely to collecting the documentation displayed in the DMZ or cloud or an area visible from the Internet and another deployed in an internal environment where it resides finally. The communication between both can be automated by means of automatic tasks in the input repository that export the documentation (as soon as it is entered or periodically) and other automatic tasks that are imported by the destination repository.

6.4.2 Examples

```
#======= Document configuration ============================================
# Fields used for "login"/verifiation of identity
LoginFields=Correo|Telef
# Fields of the Foolder type to ask to be filled
FieldsToInclude=Nombre|Apellidos|Correo|Telef|Idioma
# Path of folder where folders will be created
```

```
BaseFolder=/Donaciones
# Document types allowed to be uploded
DocTipesList=PD_DOCS|Manual|Picture|Grabaciones
# Non included doc types show ALL fields
#Fields_PD_DOCS=
Fields_Manual=Title|DocDate
Fields_Picture=Title|Author|Keywords|DocDate
#Fields_Grabaciones=
#======= Security ============================================================
# Open (1) or closed (0)system. When closed, Folder MUST be created and login
information transmited to external user.
OpenContrib=0
# Folder type to use
LoginFolderType=Donaciones
# UserName and Password of the user that will do the actual insert in
openprodoc of Folders and docs.
# It is recmmended to be a user with a limiteed rol (only insert of folders
and docs) and permissions only in the Contribution folder
User=guest1
Pass=passguest1
# Allowed extensions to upload
AllowedExt=doc|docx|xls|xlsx|ppt|pptx|txt|pdf|jpg|jpeg|tiff|tif|png|gif|odt
# MaxSize upload (bytes)
MaxSize=20000000
#======= Interface ===========================================================
# Openprodoc identifier of CSS or http url of CSS
ContribCSS=16697ec3694-3fe7288b86493159
# url of logo. Can be a "local" url using the format /SendDoc?Id=Identifier of
doc
ContribLogo=img/LogoProdoc.jpg
# Title to be show in login
Title=Aportaciones
# Title to be show in content of folder
TitleList=Archivo personal
# Id of Report used for showing docs infolder
DocsReportId=16654ff6af1-3f9b78099c0147a0
# Url of help
UrlHelp=
#======= Alternative htmls ====================================================
# Alternative htmls depending on agent
#---------------------------------------------
# Num alternatives for login
NumHtmlContLog=1
# Agents for login
ListAgentLog0=Edge|Firefox
# html for each agent of login
HtmlAgentLog0=166a24cd914-3fee91e7fa2c96cc
#---------------------------------------------
# Num alternatives for Lista of docs
NumHtmlContList=1
# Agents for List
ListAgentList0=Chrome
# html for each agent of List
HtmlAgentList0=166a24d2fa1-3fe05747921f306c
#---------------------------------------------
# Num alternatives for adding docs
NumHtmlContAdd=1
```

```
# Agents for adding docs
ListAgentAdd0=Opera|Chrome
# html for each agent of adding docs
HtmlAgentAdd0=166a24d90f9-3fc08534bf7753e0
#-----------------------------------------
# Num alternatives for Results adding docs
NumHtmlContRes=1
# Agents for Results adding docs
ListAgentRes0=*
# html for each agent of Results adding docs
HtmlAgentRes0=166a24de462-3fc29665f8ea9ffc
#=================================================
```

6.4.3 Deployment

The OPADs are deployed in OpenProdoc repository as documents of any type and downloaded when needed.

Being normal documents, they can be versioned and the access to them follow the same security model that any other document, by means of the ACL. When versioned, OpenProdoc will use the last version, however, due the cache behaviour, the use of the last version can be delayed.

The OPADs can be stored in any folder, however it's recommended to store then in the "System" folder. As they can be used combined with other elements (Reports, OPAC, Extensions, etc.) it's recommended to create a folder for the project. The use of a folder allows also to export and import the complete folder and its contained document (the project/"package") between environments/installations just from the Web user interface.