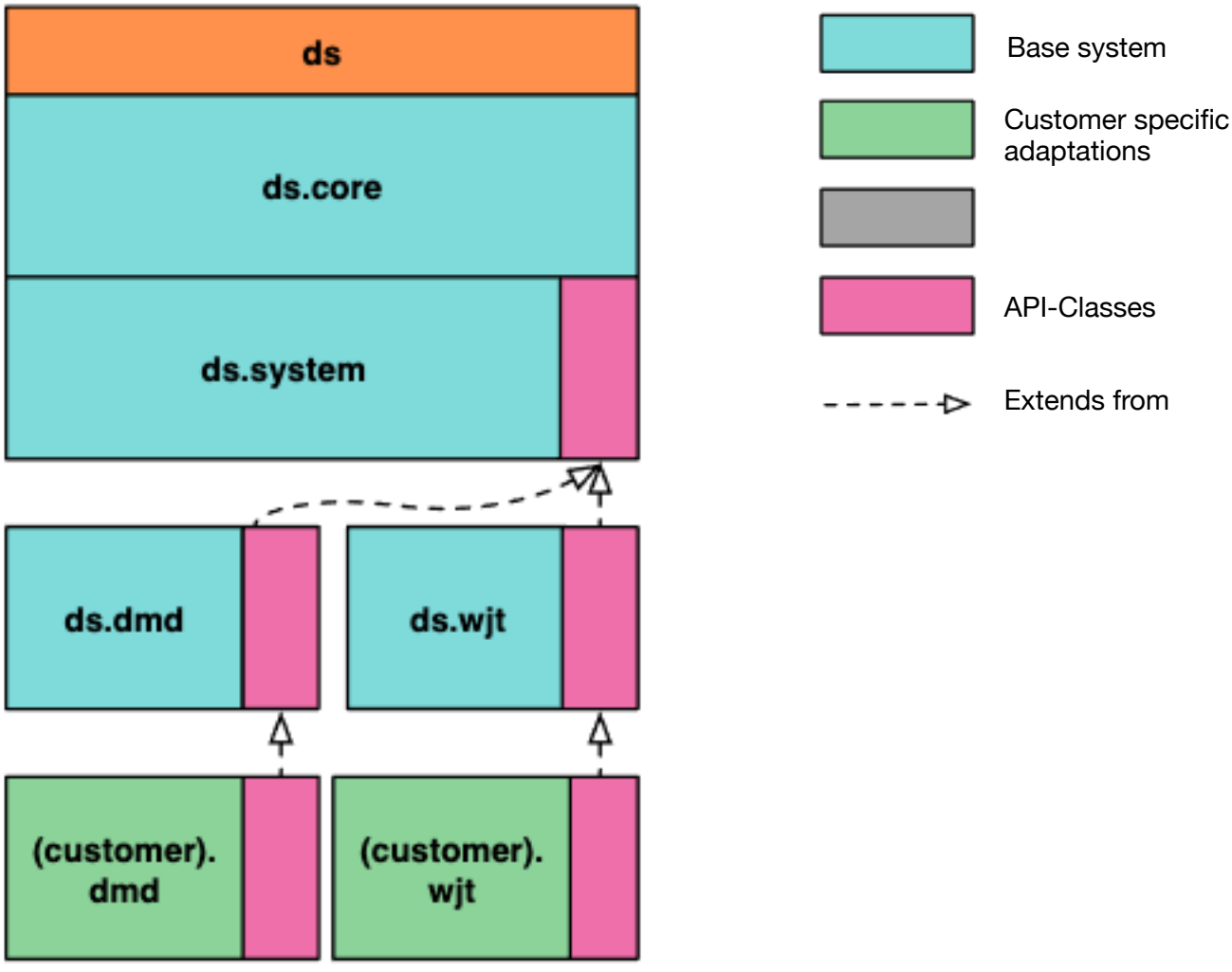
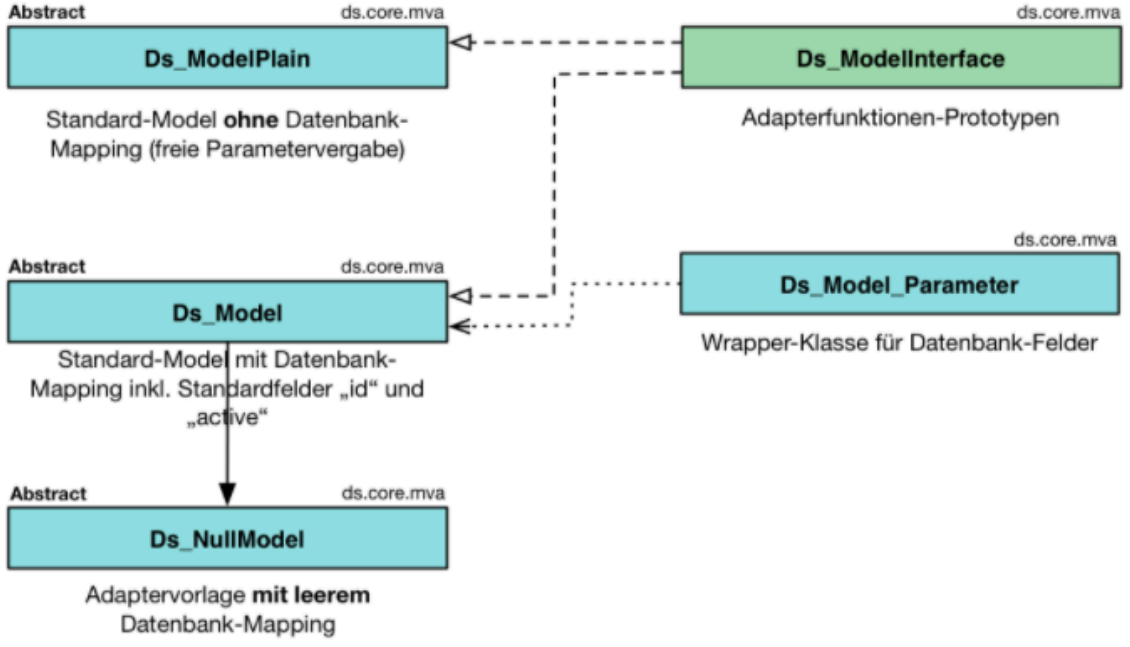
Brief Introduction


ds
ds.core
ds.system
ds.dmd
ds.wjt
(customer).
dmd
(customer).
wjt
Base system
Customer specific
adaptations
API-Classes
Extends from

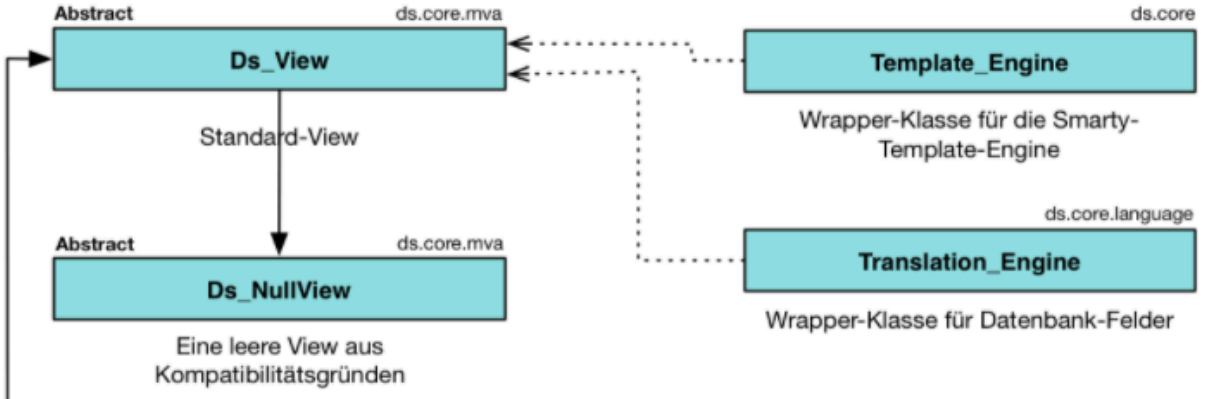
Core-Models

Ein Model gibt die grundlegende Datenstruktur vor mit der ein Adapter arbeitet. Das Ds_Model enthält ein Mapping zu einer bestimmten Datenbank-Tabelle und der Model-Parameter zu den jeweiligen Tabellen-Feldern.



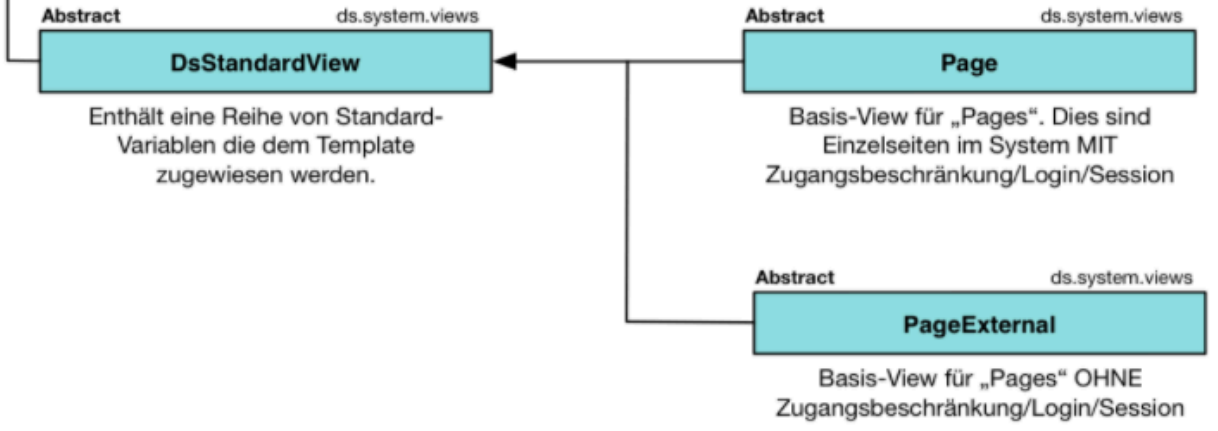
Core-Views

Eine View nimmt die Daten des Adapters entgegen, referenziert (mit Ausnahme der Ds_NullView) eine Template-Datei und rendert diese mit Hilfe der Zuweisung von Variablen.



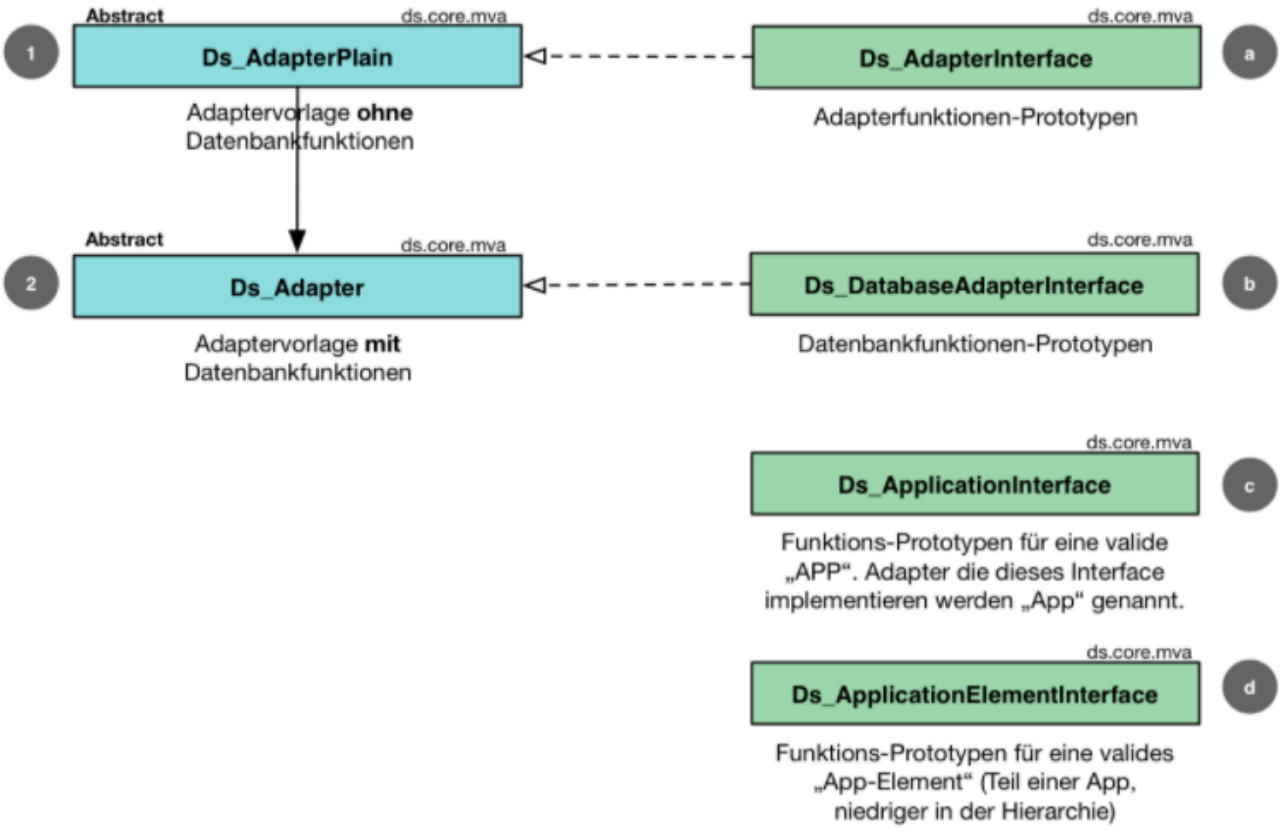
System-Views

Die Views der Systemebene geben mögliche Strukturen vor. Die Verwendung ist angeraten aber letzlich optional.

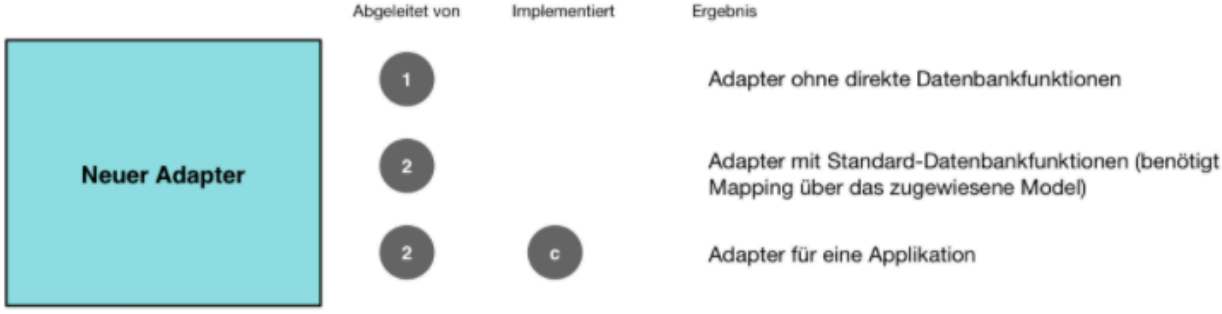


Core-Adapter

Bei der Entwicklung eines neuen Adapters sollte dieser, ja nach Anforderungen entsprechend von den Core-Klassen abgeleitet werden.



Beispiele



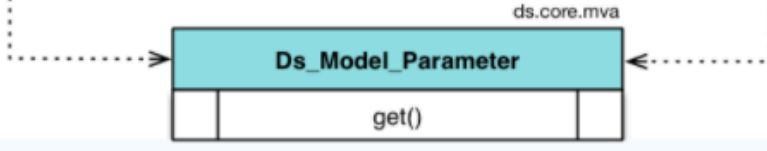
Beispiel

Ein Model gibt die grundlegende Datenstruktur vor mit der ein Adapter arbeitet. Das Ds_Model enthält ein Mapping zu einer bestimmten Datenbank-Tabelle und der Model-Parameter zu den jeweiligen Tabellen-Feldern.

Initialisierung der Basis-Parameter . Subklassen müssen diese Funktion implementieren, die Elternfunktion aufrufen und dort die Funktion setTableName verwenden um den Namen der Datenbanktabelle festzulegen.

```php
/**
 *  Initalize the var/sql column assignment table with global
 *  database related model class. Inject MANDATORY fields (for
 */
protected function _initParams(){
    $this->addColumnData(
        [
            "iId"           =>
                Ds_Model_Parameter::get(
                    'id',
                    null,
                    Ds_ModelInterface::PARAMTYPE_OMIT
                ),
            "sActive"       =>
                Ds_Model_Parameter::get(
                    'active',
                    Ds_ModelInterface::DATA_STATE_ACTIVE,
                    Ds_ModelInterface::PARAMTYPE_STRING
                ),
        ]
    )
    ;
}
```

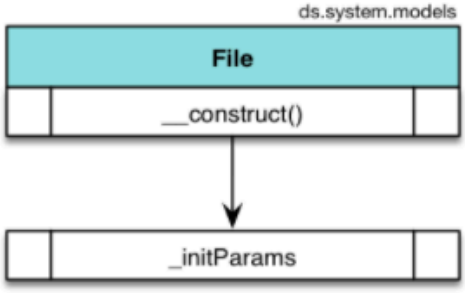
```php
/**
 * Basic model parameter assignment function
 */
protected function _initParams(){

    // Set table name
    $this->setTableName('ds_file');

    // Init standard params
    parent::_initParams();

    # Add specific fields belonging to this class
    $this->addColumnData(
        array (
            'sLabel'            => Ds_Model_Parameter::get('label',null),
            'iIdFiletype'       => Ds_Model_Parameter::get(
                'id_filetype',null,
                Ds_ModelInterface::PARAMTYPE_NUMERIC),
            'iIdResource'       => Ds_Model_Parameter::get(
                'id_ressource',null,
                Ds_ModelInterface::PARAMTYPE_NUMERIC),
            'iIdApp'            => Ds_Model_Parameter::get(
                'id_app',null,
                Ds_ModelInterface::PARAMTYPE_NUMERIC),
```




```php
/**
 * Get a Model_Parameter instance
 *
 * @param string $sName The name of the database table field
 * @param string $sValue The default value (set to NULL or an empty string if not used)
 * @param string $sModifier The field data type (@see Ds_ModelInterface)
 * @param null   $sTag  A optional tag parameter e.G. for additional flags
 * @return Ds_Model_Parameter
 */
public static function get($sName='',$sValue='',$sModifier=Ds_ModelInterface::PARAMTYPE_STRING,$sTag=NULL){
    return new Ds_Model_Parameter($sName,$sValue,$sModifier,$sTag);
}
```

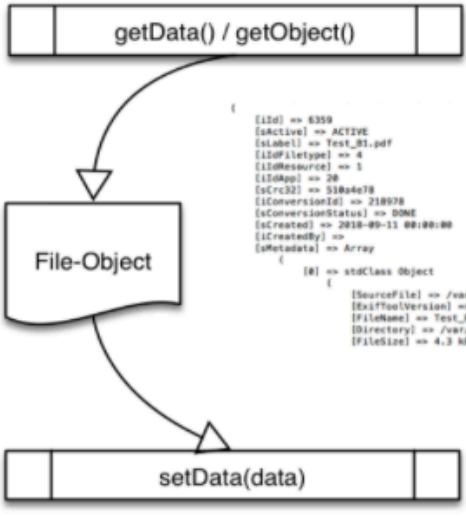
Beispiel

Ein Model gibt die grundlegende Datenstruktur vor mit der ein Adapter arbeitet. Das Ds_Model enthält ein Mapping zu einer bestimmten Datenbank-Tabelle und der Model-Parameter zu den jeweiligen Tabellen-Feldern.



Der Konstruktor ruft diese Funktion automatisch auf

```php
/**
 * Basic model parameter assignment function
 */
protected function _initParams(){

    // Set table name
    $this->setTableName('ds_file');

    // Init standard params
    parent::_initParams();

    # Add specific fields belonging to this class
    $this->addColumnData(
        array (
            'sLabel'            => Ds_Model_Parameter::get('label',null),
            'iIdFiletype'       => Ds_Model_Parameter::get(
                'id_filetype',null,
                Ds_ModelInterface::PARAMTYPE_NUMERIC),
            'iIdResource'       => Ds_Model_Parameter::get(
                'id_ressource',null,
                Ds_ModelInterface::PARAMTYPE_NUMERIC),
            'iIdApp'            => Ds_Model_Parameter::get(
                'id_app',null,
                Ds_ModelInterface::PARAMTYPE_NUMERIC),
```



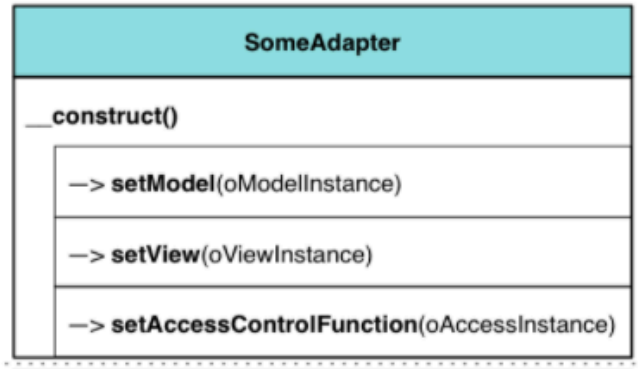
Generiert ein Objekt (Instanz von stdClass) welches alle Parameter mit den aktuellen Werten enthält.

```
(
    [iId] => 6359
    [sActive] => ACTIVE
    [sLabel] => Test_01.pdf
    [iIdFiletype] => 4
    [iIdResource] => 1
    [iIdApp] => 20
    [sCrc32] => 510a4e78
    [iConversionId] => 218978
    [sConversionStatus] => DONE
    [sCreated] => 2018-09-11 08:08:08
    [iCreatedBy] =>
    [sMetadata] => Array
        (
            [0] => stdClass Object
                (
                    [SourceFile] => /var/www/krueger/_storage/upload/Test_01.pdf
                    [ExifToolVersion] => 10.4
                    [FileName] => Test_01.pdf
                    [Directory] => /var/www/krueger/_storage/upload
                    [FileSize] => 4.3 kB
```

Nimmt das Objekt oder ein Assoziatives Array (Keys müssen den Parameternamen entsprechen) entgegen und weist die Werte der Model-Instanz zu.

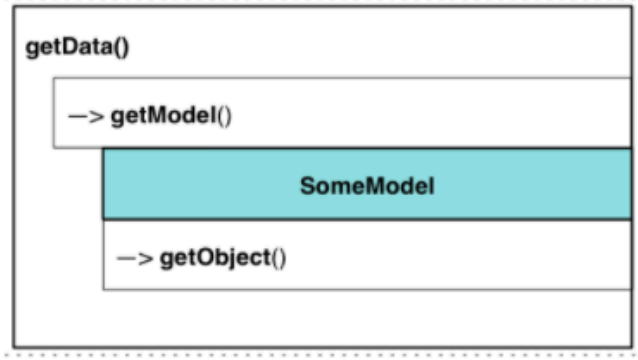
1. Init

Assignment of Model, View and mandatory access control instance.



2. Data acquisition

Standard behaviour is to fetch the data object from the model but the function can be rewritten to return more or different data.

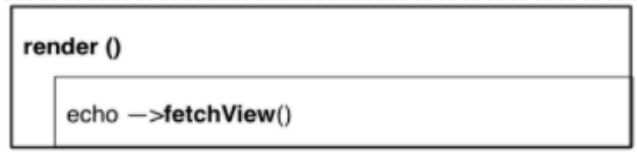


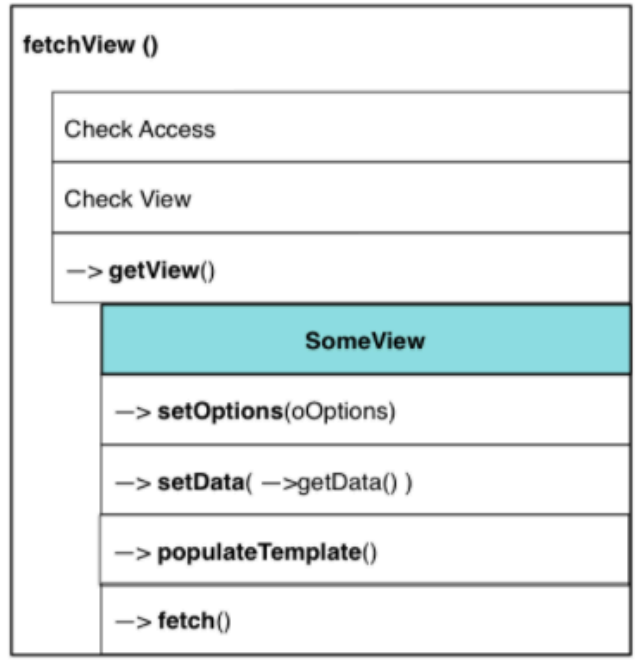
3. Rendering

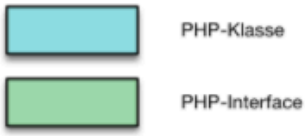
The fetchView() function feeds adapter options (if any) and data, returned by the adapter's getData() function to the View and then orders it to assign the data to its template.

When all these proceedings went fine, it orders the View to fetch the (then populated) template and returns the result.

The render() function simply echoes the fetched template content.





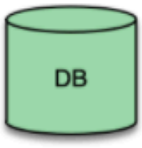


Fundamentals

Access to a specific entity within the DSFW ecosystem either can be *calculated* by evaluating all matching datasets in the ***ds_access*** table or by a specific procedure implementation.

Data structure

Each dataset (referred to a „Rule") contains **matching points**, an access-level value/dataset, and the DSFW standard dataset states („ACTIVE","INACTIVE" etc.). A NULL-Value means „matches any"



ds_access

id	id_application	id_element	id_node	id_user	id_usergroup	id_workflow_step	access	active
1165	10	6	NULL	NULL	11	27	a:1:{i:0;i:2;}	ACTIVE
1164	10	6	NULL	NULL	11	26	a:1:{i:0;i:2;}	ACTIVE
1163	10	6	NULL	NULL	11	25	a:1:{i:0;i:2;}	ACTIVE

Match fields

id_application	id_element	id_node	id_user	id_usergroup	id_workflow_step
Identifies the DSFW-Application for which this rule is a match	Identifies a „element" (abstract). e.G. a „Block" in the WJT	Identifies a „Node". Can be anything e.G. a single Button.	Identifies the user for which this rule is a match.	Identifies the usergroup for which this rule is a match.	Identifies the step id of the current workflow step for which this rule is a match.

Data fields

access	active
Serialised array of **access levels**. (Only one member legal at time of writing)	Dataset state ENUM. Only „ACTIVE" rules get evaluated

Access levels

A access level is a **signed integer** value. The following are considered legal :

DENIED	**-1**	Access is denied. A denial trumps all allowances.
UNSET	**0**	Access is neither denied nor granted.
READONLY	**1**	A readonly access is granted.
READWRITE	**2**	Full (readwrite) access is granted.

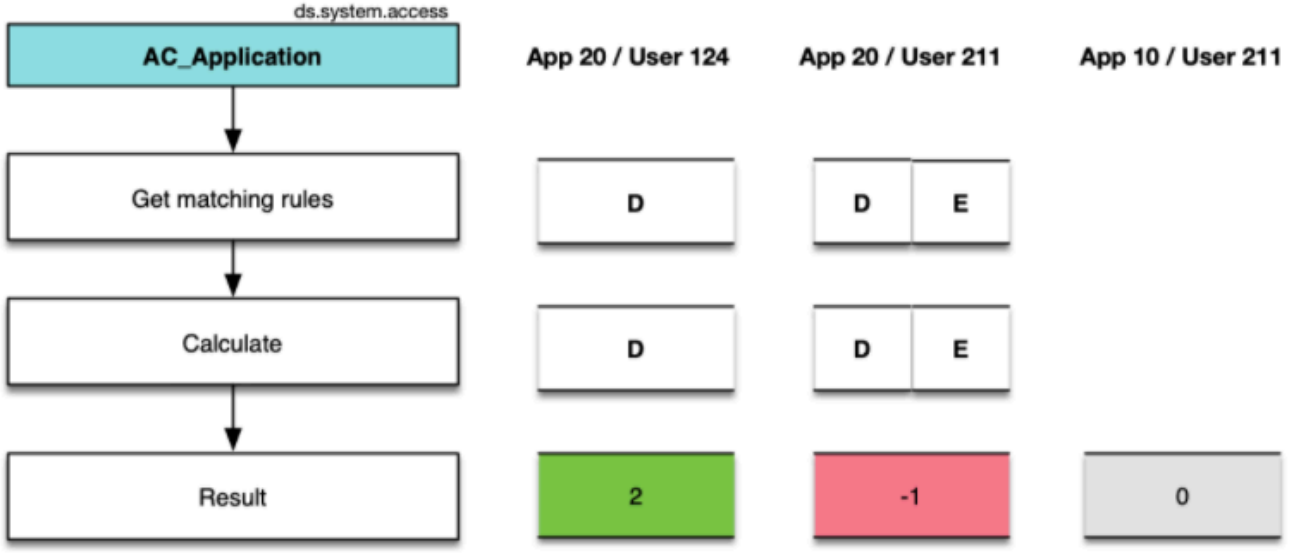
Calculation

Here are a few examples of how access level calculation works in the DSFW (matching point *id_workflow_step* has been left out deliberately here) :

#	id_application	id_element	id_node	id_user	id_usergroup	access	Description
A	10	5	NULL	NULL	11	2	Allow RW-Access to Element 5 of Application 10 when user is member of usergroup 11.
B	10	5	NULL	147	11	-1	Deny access to Element 5 of Application 10 when user id is 147 and is member of usergroup 11.
C	10	5	48	NULL	NULL	1	Allow readonly access to node 48 of Element 5 of Application 10 for every user.
D	20	NULL	NULL	NULL	11	2	Allow RW-Access to Application 20 for each member of usergroup 11.
E	20	NULL	NULL	211	NULL	-1	Deny access to Application 20 for user 211

Example: Application-Access

In the following example the Application-Access standard handler (ds.system.AC_Application) is to evaluate the access scenario for Application 20 (the DMD in most cases) and for a specific user.



Implementation

In order to implement your own access class you must extend it from ds.system.access.AccessControl (e.G. „AC_MyAccess"). By doing so you inherit the mandatory functions for the access class to work. All of them can be overwritten to match your use case.

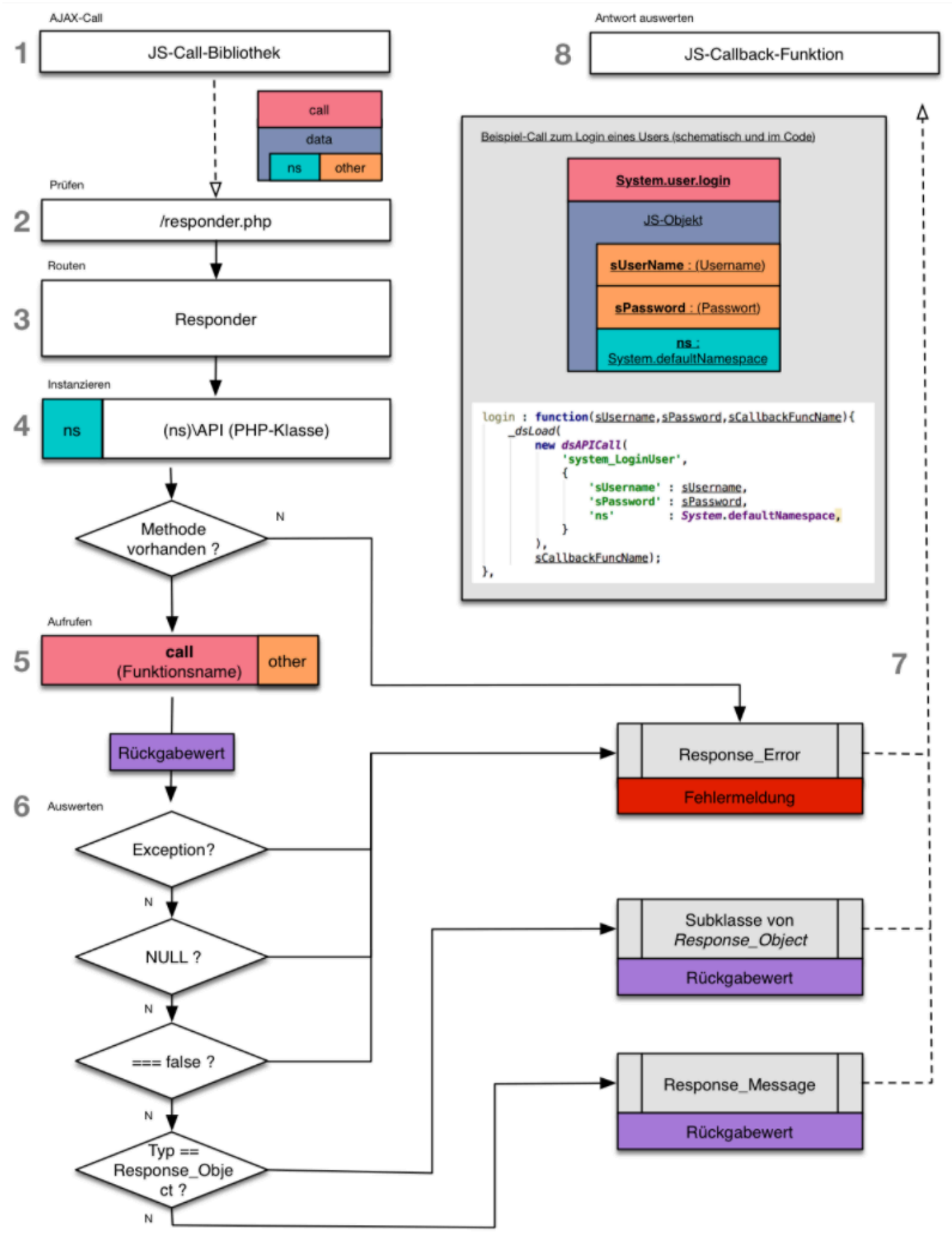
Abstract ds.system.access

AccessControl	
Rule matching/filter functions	
bool _ruleIsMatchingApplication(*AccessInterface*)	These functions perform a simple matching algo to tell whether a given Access-Object matches. The result will be used in the calculation You can override any of these in your „AC_MyAccess" if needed. In most cases this will not be necessary
bool _ruleIsMatchingElement(*AccessInterface*)	
bool _ruleIsMatchingNode(*AccessInterface*)	
bool _ruleIsMatchingOneOfUsersGroups(*AccessInterface*)	
bool _ruleIsMatchingUser(*AccessInterface*)	
bool _ruleIsMatchingWorkflowStep(*AccessInterface*)	

Rule prefilter functions	
bool _ruleIsRelevant(*AccessInterface*)	This was integrated because of performance reasons. It gets executed before the standard rule matching/filter functions and is meant to sort out irrelevant rules before the main evaluation (which is more costly). So if you for example evaluating Access to any Element/Node/… for AppId 20, you can return the result of a simple comparison of your desired ID 20 with the AppId of the current dataset.

Access level calculation	
int getAccessLevel()	Access level calculation. In some use cases it is feasible to shortcut that completely (see *ds.system.access.AC_Base* for example).

Access shorthand functions	
bool access()	Is the result of the access level evaluation >= 1 (READ) or the current user a Admin (those get an override)
bool hasReadAccess()	Is the result of the access level evaluation >= 1 (READ) or the current user a Admin (those get an override)
bool hasReadWriteAccess()	Is the result of the access level evaluation >= 2 (RW) or the current user a Admin (those get an override)
bool hasDeniedAccess()	Is the result of the access level evaluation < 0 (DENIAL) and the current user NOT a Admin (those get an override)


AJAX-Call
1
JS-Call-Bibliothek
call
data
ns
other
Prüfen
2
/responder.php
Routen
3
Responder
Instanzieren
4
ns
(ns)\API (PHP-Klasse)
Methode vorhanden ?
N
Aufrufen
5
call (Funktionsname)
other
Rückgabewert
6
Auswerten
Exception?
N
NULL ?
N
=== false ?
N
Typ == Response_Object ?
N
Antwort auswerten
8
JS-Callback-Funktion
Beispiel-Call zum Login eines Users (schematisch und im Code)
System.user.login
JS-Objekt
sUserName : (Username)
sPassword : (Passwort)
ns : System.defaultNamespace
login : function(sUsername,sPassword,sCallbackFuncName){
_dsLoad(
new dsAPICall(
'system_LoginUser',
{
'sUsername' : sUsername,
'sPassword' : sPassword,
'ns' : System.defaultNamespace,
}
),
sCallbackFuncName);
},
7
Response_Error
Fehlermeldung
Subklasse von Response_Object
Rückgabewert
Response_Message
Rückgabewert