

Wikiprint Book

Title: Rejstry

**Subject: eDokumenty - elektroniczny system obiegu dokumentów, workflow i CRM -
DeployerGuide/AdvancedConfiguration/CustomRegisters**

Version: 103

Date: 04/21/26 21:28:14

Table of Contents

| | |
|--|----|
| <i>Rejestry</i> | 3 |
| <i>Tworzenie rejestru</i> | 3 |
| <i>Tworzenie raportu</i> | 3 |
| <i>Sumowanie oraz grupowanie</i> | 3 |
| <i>Ważne tabele</i> | 3 |
| <i>Rejestr jako lista - zakładka</i> | 4 |
| <i>w dokumencie</i> | 4 |
| <i>w zadaniu</i> | 4 |
| <i>w sprawie</i> | 4 |
| <i>dodatkowe opcje</i> | 4 |
| <i>Podrejestr w rejestrze</i> | 4 |
| <i>Definicja rejestru - parametry</i> | 5 |
| <i>Walidacja wpisu w rejestrze</i> | 5 |
| <i>Konfiguracja wyglądu okna dialogowego</i> | 5 |
| <i>Definicje pól dla rejestru</i> | 5 |
| <i>Formatowanie pól i formatki</i> | 5 |
| <i>Akcje (javascript)</i> | 6 |
| <i>Walidacja wartości w polach</i> | 6 |
| <i>Ustawianie wartości domyślnych</i> | 6 |
| <i>Pole jako lista wyboru</i> | 7 |
| <i>Listy połączone</i> | 7 |
| <i>Pole tekstowe statyczne (tylko dla danych typu text). Pole nie musi mieć odwzorowania w bazie danych.</i> | 7 |
| <i>Pole tekstowe typu HTML</i> | 7 |
| <i>Pole tekstowe typu ComboBox</i> | 7 |
| <i>Pole typu Lookup</i> | 8 |
| <i>Pole typu wybór z drzewka</i> | 8 |
| <i>Pole jako status</i> | 9 |
| <i>Disablowanie pola</i> | 9 |
| <i>ToolBar</i> | 9 |
| <i>Filtrowanie listy</i> | 10 |
| <i>Liczniki</i> | 10 |
| <i>Modyfikacje JSON bezpośrednio w bazie danych</i> | 10 |
| <i>Uprawnienia do rejestrów</i> | 11 |
| <i>Migracja rejestrów z innej bazy</i> | 11 |
| <i>Indywidualna zakładka w Rejestrach</i> | 11 |
| <i>Auto uzupełnianie pól przy dodawaniu nowego rejestru</i> | 12 |
| <i>Edycja rejestru przez Excel</i> | 12 |
| <i>Przydatne konstrukcje i zapytania</i> | 13 |
| <i>Import z CSV</i> | 13 |

Rejestry

Tworzenie rejestru

Aby założyć rejestr w module rejestry musimy rozpocząć od założenia tabeli. Tabelę tworzymy za pomocą komendy *create table*.

Przykład założenia tabeli

```
-- Table: cregisters.creg_r_imi
-- DROP TABLE cregisters.creg_r_imi;

CREATE TABLE cregisters.creg_r_imi
(
  nazwa character varying(250),
  ulica character varying(80),
  budnr character varying(10),
  kwota double precision
)
INHERITS (cregisters.register_entry)
WITH (
  OIDS=FALSE
);
ALTER TABLE cregisters.creg_r_imi OWNER TO edokumenty;
GRANT ALL ON TABLE cregisters.creg_r_imi TO edokumenty;
GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE cregisters.creg_r_imi TO http;
```

Uwaga ! Pola używane przez eDokumenty są dziedziczone - nie trzeba ich zakładać. Są to pola:

```
id____, cregid, uid____, is_del, adduid, adddat, lm_uid, lm_dat, doc_id, prc_id, evtid, fkelid, cre_id, tpstid, stcuid, st
```

Następnie w module rejestry zakładamy nowy rejestr, a w polu nazwa tabeli wprowadzamy nazwę założonej tabeli. System będzie od nas żądać, aby nazwa tabeli rozpoczynała się od "creg_". Jeśli system wykryje, że tabela rejestru jest założona w bazie, zapyta, czy dodać pozycje na podstawie znalezionej tabeli.

Tworzenie raportu

Po utworzeniu tabeli w schema cregisters rozpoczynającej się od ciągu creg_ należy utworzyć raport np:

```
SELECT ('CREGISTER_ENTRY') AS clsnam, cd.id____ AS keyval, cd.*
FROM cregisters.creg_ddm_dokumenty cd
INNER JOIN cregisters.creg_archiv_formularz af ON cd.menuid = af.formularz
WHERE {FILTER_STRING} AND cd.is_del IS NOT true
{ORDER_BY}
{LIMIT}
```

Raport należy podlinkować do rejestru ustawiając w tabeli registers pole rep_id. W tabeli reports.reports dla rep_id = raportowi dla rejestru należy ustawić is_sys = TRUE

Sumowanie oraz grupowanie

Lista pozycji rejestru obsługuje grupowanie oraz sumowanie zdefiniowane w raporcie. **Do działania niezbędne jest użycie w zapytaniu SQL, znacznika {ORDER_BY}.**

Raport może zawierać grupowanie po kilku polach/kolumnach - należy je wpisać (rozdzielone przecinkiem) do sekcji "Grupowanie" na zakładce "Definicja" edytora raportów.

Możliwe jest sumowanie wartości poszczególnych kolumn - należy je wpisać (rozdzielone przecinkiem) do sekcji "Sumowanie" na zakładce "Definicja" edytora raportów.

Ważne tabele

```
register (klucz główny: id____)
register_entry (klucz główny: id____ , klucz obcy: )
register_fields (klucz główny: id____, klucz obcy: )
register_links (klucz główny: id____, klucz obcy: )
```

Tabele:

- cregisters.register - lista rejestrów.
- cregisters.register_entry -
- cregisters.register_fields
- cregisters.register_links

Uwaga! Kluczem obcym w registers_entry referującym do rejestru jest XXXX

Rejestr jako lista - zakładka

w dokumencie

Definiujemy powiązanie rejestru z typem dokumentu (jeżeli ma to być lista a nie formularz to ustawiamy parametr *collection* na *true*):

```
INSERT INTO cregisters.register_links (cregid, keyval, clsnam, params)
VALUES ({cregisters.register.id____}, {types_of_documents.dctpid}, 'DOCUMENT', '{"collection":true}')
```

w zadaniu

Definiujemy powiązanie rejestru z typem zdarzenia:

```
INSERT INTO cregisters.register_links (cregid, keyval, clsnam, params) VALUES ({cregisters.register.id____}, -1, 'EVENT',
(można zastępować EVENT innym typem zdarzenia: TODO, MEETING, PHONECALL).
```

w sprawie

Definiujemy powiązanie rejestru z kategorią spraw:

```
INSERT INTO cregisters.register_links (cregid, keyval, clsnam, params) VALUES ({cregisters.register.id____}, {dossiers.dos
```

Jeżeli w miejsce {dossiers.dos_id} wstawimy wartość -1, wówczas zakładka z listą pojawi się na każdej sprawie.

dodatkowe opcje

Jeżeli chcemy aby lista/zakładka pokazała się dopiero po dodaniu pierwszego wpisu to parametry należy dodatkowo ustawić parametr *always_visible* na *false*.

```
{"collection":true,"always_visible":false}
```

Podrejestr w rejestrze

Aby zbudować strukturę hierarchiczną rejestru wystarczy zlinkować odpowiednio 2 wcześniej utworzone rejestry. Pierwszy ze wskazanych zacznie się pojawiać jako lista rekordów w formacie rejestru nadrzędnego.

```
INSERT INTO cregisters.register_links (cregid, keyval, clsnam, params)
VALUES ({cregisters.register.id____}, {cregisters.register.id____}, 'REGISTER', '{"collection":true}')
```

Uwaga! Id podrejestru jest wprowadzany w insercie jako pierwsze, następny jest id rejestru do którego będzie należeć podrejestr.

W raporcie w podrejestrze za filtrowanie rekordów odpowiada makro {FILTER_STRING}, które dokleja do zapytania warunek po atrybucie *cre_id* (*cre_id* wskazuje na rekord rejestru nadrzędnego).

Definicja rejestru - parametry

Walidacja wpisu w rejestrze

Walidacja odbywa się po zapisaniu formularza (rekord jest już w bazie ale transakcja nie jest jeszcze zatwierdzona). Dane zostaną zapisane jeżeli zapytanie SQL *query* zwróci TRUE. W przeciwnym wypadku zmiany nie zostaną zapisane (ROLLBACK) i pokaże się komunikat o treści zdefiniowanej w parametrze *message*.

przykład:

```
{
  "validator": [
    {
      "message": "Nieprawidłowe dane!",
      "query": "SELECT (data_urodzenia < now()) AND (strlen(pesel) = 11) FROM cregisters.creg_usc WHERE id_____ = {PKE"
    }
  ]
}
```

Konfiguracja wyglądu okna dialogowego

Kod należy wkleić do parametrów. Można łączyć go z innymi parametrami.

```
{
  "dialog": {
    "width": "600px",
    "height": "600px"
  }
}
```

Definicje pól dla rejestru

Formatowanie pól i formatki

Pola rejestrów można formatować za pomocą następujących znaczników: Przykładowe pole typu HTML, z określoną wielkością i lokalizacją. Parametry można łączyć z innymi.

Duże pole edycyjne:

```
{
  "type": "html",
  "widget": {
    "width": "574px",
    "height": "540px",
    "top": "105px",
    "left": "10px"
  }
}
```

Małe pole z wyborem z drzewka wartości słownikowej:

```
{
  "widget": {
    "width": "270px",
    "top": "62px",
    "left": "10px"
  },
  "type": "dbtreeselector",
  "class": "MyCustomTree",
  "path": "./scripts/MyCustomTree.inc",
  "params": {
    "prc_id": "{prc_id}",
  }
}
```

```

    "doc_id": "{doc_id}"
  }
}

```

Małe pole z wyborem z dowolnego drzewka:

```

{
  "widget": {
    "width": "270px",
    "top": "62px",
    "left": "10px"
  },
  "type": "dbtreeselector",
  "sql": "SELECT c.strcid AS keyval, c.prn_id, c.strnam AS name__, CASE WHEN (SELECT count(*) FROM cregisters.creg_struct
}

```

Akcje (javascript)

Możliwe jest przypisanie skryptu (javascript) dla akcji na polach formularza. np.:

```

{
  "onChange": "$({'DIALOG_NAME'}_v51').value='123';"
}

```

```

{
  "onChange": "var dat = new Date(); dat.setMonth(dat.getMonth() + parseInt({'DIALOG_NAME'}_v480').value); CalendarInpu
}

```

Walidacja wartości w polach

przykłady:

```

-- liczba dowolnej długości
{"validator": "/^\\d+$/"}

-- kwota
"validator": "/^([0-9]*)[\\.,][0-9][0-9]$/"}

-- godzina
{"validator": "/^([01]?[0-9]|2[0-3]):[0-5][0-9]$/"}

-- Liczba całkowita i zmiennoprzecinkowa
{"validator": "/^([0-9]*)[\\.,]{0,1}([0-9]{0,3})$/"}

```

ustalenie wymagalności dla pola:

```

{"required": true}

```

Ustawianie wartości domyślnych

Jeżeli chcemy aby pole było listą wyboru, to definiujemy w parametrach (register_fields.params) domyślną wartość (defaultValue):

```

-- Id tworzącego dokument
{"defaultValue": "{SQL::SELECT adduid FROM documents WHERE doc_id = {doc_id}}"}

-- domyślne dane zalogowanego użytkownika
{"defaultValue": "{SQL::SELECT o.firnam || ' ' || o.lasnam || ' (' || COALESCE(o.orunsm, '') || ' - ' || o.ndenam || ') ' AS

-- przepisanie opisu sprawy do pola w rejestrze

```

```

{"defaultValue":"{SQL::SELECT dscrpt FROM processes WHERE prc_id::text = nullif('{prc_id}','')}"}

-- domyślna data
{"defaultValue": "{SQL::SELECT CURRENT_DATE}"}

-- numeracja wg schematu (wymagana funkcja get_counter)
{"value":"{SQL::select case when {cregisters.creg_usterki.numer}::text='' then 'U/' || get_counter(1,'U') else {cregisters

```

Możliwe jest też ustawienie wartości wyliczanej za każdym razem gdy dokonujemy zapisu rejestru (dla pól ukrytych):

```

-- Imię i nazwisko dokonującego zmian w rejestrze
{"value":"{SQL::select firnam || ' ' || lasnam from users where usr_id={LOGGED_USR_ID}"}

```

1. **defaultValue** jest parsowane tylko dla formularza nowego wpisu w rejestrze (na akcji Open oraz Save).

value jest parsowane zawsze na akcji Save niezależnie od trybu (edycja, nowy) wyłącznie dla pól:

1. ukrytych poprzez definicję pola (register_fields.hidden = TRUE)
2. ukrytych poprzez parametr visible (register_fields.params = {"visible":false})
3. nieaktywnych (register_fields.params = {"enabled":false})

Pole jako lista wyboru

Jeżeli chcemy aby pole było listą wyboru, to definiujemy w parametrach (register_fields.params) zapytanie zwracające rekordy typu (klucz,wartość), dodatkowo ustawiamy domyślną wartość (defaultValue):

```

{"sql":"SELECT usr_id, lasfir FROM orgtree_view WHERE is_del IS NOT TRUE ORDER BY lasfir", "defaultValue":"{SQL::SELECT ad

{"sql":"SELECT nazwa AS value, nazwa AS caption FROM cregisters.creg_slownik WHERE typ = 'PRZYCZYNA' ORDER BY nazwa"}

```

Parametry: sql, defaultValue, są objęte standardowym mechanizmem parsowania [parametrów](#) (tak jak np. w przypisaniach w [workflow](#)).

Listy połączone

Użycie znacznika pola, które jest listą wyboru, SQLu innej listy spowoduje jej automatyczne odświeżanie/filtrowanie.

przykład:

w rejestrze cregisters.creg_moj_rejestr, pole "grupa" jest zdefiniowana jako select z listą grup

```

{"sql":"SELECT grp_id,grpnam FROM groups"}

```

"pracownik" jest listą pracowników/użytkowników

```

{"sql":"SELECT usr_id,usrnam FROM users WHERE is_del IS NOT TRUE AND (CASE WHEN {cregisters.creg_moj_rejestr.grupa} = '' T

```

Taka konfiguracja spowoduje przeładowanie listy pracowników przy każdej zmianie grupy.

Pole tekstowe statyczne (tylko dla danych typu text). Pole nie musi mieć odwzorowania w bazie danych.

```

{"type":"static", "widget":{"style":"font-weight:bold;"}}

```

Pole tekstowe typu HTML

```

{"type":"html"}

```

Pole tekstowe typu ComboBox

```

{"type": "combobox", "autoSearch": 2, "sql": "SELECT usr_id,usrnam FROM users WHERE is_del IS NOT TRUE AND (firnam ~* E'^{SEARCH_
{"type": "combobox", "autoSearch": 2,
"sql": "SELECT lasnam || ' ' || firnam AS value, lasnam || ' ' || firnam AS caption FROM users WHERE is_del IS NOT TRUE AND

```

Znacznik {SEARCH_TEXT} zostanie zastąpiony wpisanym w pole tekstem

1. autoSearch - ilość znaków po których wpisaniu zostanie uruchomione wyszukiwanie / podpowiadanie (wartość -1 spowoduje wyłączenie automatycznego wyszukiwania i pokazanie ikony lupki)

Pole typu Lookup

Pole to wygląda jak ComboBox. Różnica polega na tym, że wyszukiwanie odbywa się tylko za pomocą "lupki", a wartością pola będzie dana pobrana z bazy pod kluczem {valueField}. Wartość prezentowaną w polu określamy w parametrze {labelField}.

Kolumna **clsnam** w zapytaniu spowoduje pokazanie ikony "i" umożliwiającej otwarcie formularza obiektu powiązanego z danym clsnam (np. 'CONTACT' as clsnam, contid AS keyval - da możliwość otwarcia panelu klienta). Klucz użyty do otwarcia obiektu pobrany zostanie z zapytania z kolumny **keyval**, lub jeśli nie podano to z kolumny wskazanej w {valueField}.

Znacznik {SEARCH_TEXT} zostanie zastąpiony wpisanym w pole tekstem

Znacznik {FILTER_STRING} zostanie zastąpiony wartością z parametru "sql_filter"

```

{"sql": "select usr_id, usrnam, 'USER' as clsnam FROM users WHERE {FILTER_STRING}", "sql_filter": "firnam ~* E'^{SEARCH_TEXT}"
{"sql": "SELECT devcid, name__ || ' - ' || COALESCE(sernum) AS device FROM cregisters.creg_devices WHERE {FILTER_STRING}",
{"sql": "SELECT contid, COALESCE(name_2, name_1) || ' ' || f_addr AS caption, 'CONTACT' as clsnam FROM contacts_view WHERE
  "manager": {
    "class": "MyCustomSListBoxManager",
    "path": "./scripts/MyCustomSListBoxManager.inc"
  }}

```

Przykład dla wyszukiwania po kilku frazach rozdzielonych spacjami. "sql_filter" zostanie powtórzony dla każdej frazy.

```

{"sql": "select usr_id, usrnam, 'USER' as clsnam FROM users WHERE ({FILTER_STRING[AND]})", "sql_filter": "(firnam ~* E'^{SEARCH_

```

Wynikowy SQL dla powyższego przykładu i wyszukiwanego tekstu "jan kow":

```

select usr_id, usrnam, 'USER' as clsnam FROM users WHERE ((firnam ~* E'jan' OR lasnam ~* E'jan') AND (firnam ~* E'kow' OR

```

[Dodatkowe informacje](#)

Pole typu wybór z drzewka

Wybór wartości następuje poprzez zaznaczenia węzła z drzewka a następnie zapamiętanie opisu węzła w polu. Drzewko definiujemy za pomocą SQL.

Dla przykładu niech posłuży drzewko magazynów dostępne poprzez słownik. Istotne jest aby kwerenda SQL zwracała odpowiednio nazwane pola (poprzez aliasy).

| Nazwa pola | Opis |
|------------|---|
| keyval | Klucz główny |
| prn_id | Klucz referujący do klucza głównego wskazujący na element nadrzędny |
| name | Nazwa węzła |
| icon | Wyświetlana ikona. Można użyć słowa kluczowe jak FOLDER oraz ITEM, które wskazują odpowiednio na węzeł grupujący (FOLDER) oraz np. element końcowy struktury drzewa (ITEM). Przykład: foldery a w folderach dokumenty |

| | |
|--------|---|
| enable | Czy dany element jest aktywny. Jeśli nie ma tego atrybutu to domyślnie każdy jest aktywny |
| | |

```
{"type":"dbtreeselector", "sql":"SELECT wahaaid AS keyval, prn_id, name__, 'FOLDER' AS icon__ FROM warehouses"}
```

Pole jako status

W definicji pola, w polu Alias wpisujemy "tpstid"

Disablowanie pola

Jeśli pole ma być tylko do odczytu to należy dla niego określić atrybut enabled:

```
{"enabled":false}
```

ToolBar

```
{"type":"toolbutton", "icon":"new.gif", "visible":1, "doRefresh":true, "onclick":["moj_skrypt.inc", "MojaKlasa1", "mojaFunkcja",
```

1. icon: plik ikony bez ścieżki która wskazuje domyślnie na ./public_html/framework/img/toolbarIcons/24x24/

Skrypt "app/edokumenty/scripts/moj_skrypt.inc"

1. doRefresh: wartość true spowoduje przeładowanie formularza wpisu w rejestrze

```
<?php
class MojaKlasa1 {

    public function __construct() {
    }

    public function mojaFunkcja($params) {
        $params = json_decode($params, TRUE);

        jscript::alert(json_encode($params));
    }
}
?>
```

Dodanie wpisu w rejestrze:

```
{
  "type": "toolbutton",
  "icon": "new.png",
  "enabled": 1,
  "doRefresh": false,
  "onclick": [
    "",
    "Application",
    "openDialogByCls",
    {
      "cregid": {TUTAJ_WPISUJEMY_ID_REJESTRU_Z_TABELI_CREGISYSYSTER.REGISTERS},
      "mode": "new",
      "AFTER_SUBMIT": "{AFTER_SUBMIT}"
    },
    "CREGISTER_ENTRY",
    0
  ]
}
```

```
}

```

Wywołanie / otwarcie formularza poprzez clsnam i keyval (np. otwarcie tego samego wpisu w nowym oknie czyli edycja):

```
{"type":"toolbutton","icon":"edit.gif","enabled":1,"doRefresh":true,"onclick":["","Application","openDialogByCls","","CREG

```

Usuń wpis z rejestru:

```
{"type":"toolbutton","icon":"del.gif","enabled":1,"doRefresh":true,"onclick":["","Application","openDialogByCls","mode":

```

Otwarcie dialoga z parametrami (można przekazać dowolne dane z rekordu rejestru). Poniższa definicja to przykład już samej wartości parametrów JSON.

```
{
  "type": "toolbutton",
  "icon": "edit.gif",
  "visible": 1,
  "doRefresh": false,
  "onclick": [
    "nChangeElementsNumber2.inc",
    "nChangeElementsNumberInitializer",
    "init",
    {
      "parametr_1": "{DIALOG_NAME}",
      "parametr_2": "{cregisters.creg_n_elements.adddat}",
      "afterSubmit": "{AFTER_SUBMIT}"
    }
  ]
}
```

Filtrowanie listy

Dla rejestru można ustawić stały filtr w parametrach (cregisters.register.params)

```
{"FILTER_STRING":"is_del IS TRUE"}
```

Liczniki

Wykorzystując tabelkę i funkcję get_counter można generować własne numery dla pozycji rejestrów.

```
{"value":"{SQL::select case when {cregisters.creg_umowy_handlowe.nr_umowy}::text='' then user_workspace.get_nr_umowy('HO'}
```

Modyfikacje JSON bezpośrednio w bazie danych

Sposób na zmianę wartości jednego pola w obiekcie typu JSON (dla PostgreSQL v9.3+):

```
CREATE OR REPLACE FUNCTION "json_set_value"(
  "json"          json,
  "key_to_set"    TEXT,
  "value_to_set"  anyelement
)
RETURNS json
LANGUAGE sql
IMMUTABLE
STRICT
AS $function$
SELECT COALESCE(
  (SELECT ('{' || string_agg(to_json("key") || ':' || "value", ',') || '}')
```

```

FROM (SELECT *
      FROM json_each("json")
      WHERE "key" <> "key_to_set"
      UNION ALL
      SELECT "key_to_set", to_json("value_to_set")) AS "fields"),
'{}'
)::json
$function$;

UPDATE cregisters.register_field SET params = json_set_value(params, 'doRefresh', true) WHERE id_____ = 1;

UPDATE cregisters.register_field SET params = json_set_value(params, 'value', 'SQL::SELECT ''tekst "koło''') WHERE id_____

```

Uprawnienia do rejestrów

Prawo systemowe bswfms.cregisters daje dostęp do modułu Rejestrów oraz okna uruchamianego poprzez Shift+R. Prawo to nie daje możliwości przeglądania (czytania wpisów) rejestrów (należy udostępnić enumeratywnie każdy z rejestrów). Prawo bswfms.system.cregisters_manage daje możliwość zarządzania (tworzenie, usuwanie i modyfikacja definicji rejestrów) oraz przeglądania i modyfikacji wpisów we wszystkich rejestrach. Aby udostępnić rejestr dla wybranej grupy osób należy w konfiguracji uprawnień wybrać osoby lub grupy i wskazać poziom udostępnienia. Rodzaje uprawnień do rejestru:

1. Odczyt wszystkich (READ) – umożliwia przeglądanie/odczyt wszystkich wpisów w danym rejestrze
2. Odczyt powiązanych (READ_LINKED) - umożliwia przeglądanie/odczyt wpisów z kontekstu innego obiektu np. dokumentu, sprawy, itp.). Jeżeli użytkownik posiada prawo do odczytu w obiekcie nadrzędnym, będzie miał również dostęp do danych z rejestru powiązanych z tym obiektem.

Kolejne uprawnienia działają w kontekście pierwszych dwóch. Np.: Aby zapisać/zmodyfikować wpis w rejestrze z poziomu dokumentu, do którego mamy prawo zapisu, potrzebne są prawa: odczyt powiązanych oraz zapis.

1. Zapis (UPDATE) – umożliwia zapis/modyfikację wpisów w rejestrze
2. Usuwanie (DELETE) - umożliwia usuwanie wpisów z rejestru
3. Zmiana statusu (CHANGE_STATUS) - umożliwia modyfikację samego statusu dla wpisów w rejestrze (Uwaga! Wymaga również prawa Zapis)

Rekordy rejestrów podlegają blokadzie do zapisu w przypadku kiedy status rekordu jest FINAL lub ACCEPTED. Blokowane są również te rekordy których element nadrzędny (skojarzony dokument lub nadrejestr) jest w statusie FINAL lub ACCEPTED. Dodatkowo można sterować własnością {enabled} dla składowych rejestru:

```

PASEK ZADAŃ NAD LISTĄ
REKORD
PRZYCISK

```

Migracja rejestrów z innej bazy

[Import rejestrów](#)

Indywidualna zakładka w Rejestrach

System eDokumenty umożliwia tworzenie dla danego rejestru własny szablon z przyciskami w module Rejestry. W pierwszej kolejności tworzymy plik xml o nazwie conf_ID.xml, gdzie ID to klucz z tabeli cregisters.register (id) w następującej lokalizacji :

/apps/edokumenty/var/cfg/cregisters/

i strukturze:

```

<?xml version="1.0" encoding="UTF-8"?>
<tabs>
  <tab label="{register.label_}" rep_id="ID">
    <buttons>
      <button>
        <id>new</id>
        <label>Nowy</label>

```

```

        <dscrpt>Nowy wpis</dscrpt>
        <onclick>
            App.openDialogByCls('REGISTER_ENTRY', null,
                ({afterSubmit:'AFTER_SUBMIT', mode:'new',cregid:ID}).toJSONString())
        </onclick>
        <icon>new.gif</icon>
        </button>
        <button>
        <id>edit</id>
        <label>Edycja</label>
        <dscrpt>Edytuj wpis</dscrpt>
        <onclick>
            App.openDialogByCls('REGISTER_ENTRY', {KEYVAL},
                ({afterSubmit:'AFTER_SUBMIT', mode:'edit',cregid:ID}).toJSONString())
        </onclick>
        <icon>edit.gif</icon>
        </button>
        <button>
        <id>delete</id>
        <label>Usuń</label>
        <dscrpt>Usuń</dscrpt>
        <onclick>
            App.openDialogByCls('REGISTER_ENTRY', {KEYVAL},
                ({afterSubmit:'AFTER_SUBMIT', mode:'del',cregid:ID}).toJSONString())
        </onclick>
        <icon>delete.gif</icon>
        </button>

    </buttons>

</tab>
</tabs>

```

Dostosowanie:

W tabs: **rep_id** : ID raportu przypisanego do rejestru

W button Wartość dla **cregid:ID** ID rejestru dla, którego mają być wywołane dialogi.

(Opcjonalnie) W tabs ustawić **label** statycznie (domyślnie wartość pobierana z nazwy rejestru).

Po wykonaniu tych zmian można wstawić własne przyciski (np Custom Widget).

Auto uzupełnianie pól przy dodawaniu nowego rejestru

Jeżeli chce aby pewne pole w dialogu uzupełniło się, to do onclicka tego przycisku musimy dodać ten kod:

```
asyncLibrary.postMessage('REGISTER_ENTRY_1dlg_v559=103');
```

Gdzie wartość **REGISTER_ENTRY_1dlg_v559** odpowiada id pola z dialoga, którego chce, uzupełnić

Edycja rejestru przez Excel

Parametry do CustomWidget:

```
{
  "script": "webdav/WebDAVOpenURL.inc",
  "schema": "excel",
  "fileName": ".xlsx",
```

```

"params": {
  "cregid": "{CREGID}",
  "filter_string": "cregid = {CREGID}"
},
"image": "24x24/exampleUnit.gif"
}

```

Podstawowa obsługa edycji rejestru jest wbudowana w system i do działania wystarczy sam CustomWidget. Jeżeli chcemy dodać własną obsługę zapisu/odczytu danych, to należy:

1. Dodać skrypt do katalogu apps/edokumenty/scripts/webdav/schema/{schema} Przykład: [Handler.inc](#)

Przydatne konstrukcje i zapytania

```

-- użycie w parametrach do przycisków i pól wartości {DOC_ID} powoduje błąd po wejściu na rekord rejestru jeśli jest pusty
select pprosm from documents where doc_id = COALESCE(NULLIF('{DOC_ID}', ''), '0')::int

```

Import z CSV

javascript na onClick do użycia czy to w definicji .xml czy w CustomWidget:

```

App.createDialog('CRegisterImportWizard', 'CRegisterImportWizard', './modules/CRegisters/forms/CRegisterImportWizard.inc',

```