Tworzenie własnych komend

Plik o nazwie kończącej się na Command (np. MyfirstCommand.inc) nalezy wgrać do katalogu apps/edokumenty/commands/

Należy zmienić nazwę klasy i zaimplementować metodę execute.

```
<?php
require_once(COMMANDS_DIR.'AbsCommand.inc');

/**
* ExampleCommand
* Szablon komendy dla Workflow.
* Należy zaimplementować następujące metody z interfejsu ICommand:
*      - getDescription
*      - getExpandedDescription
*      - getCommandApi
*      - execute
* Opis metod poniżej w definicji.
*
* Dodatkowo z klasy AbsCommand mamy do dyspozycji:
*  - $this->action - bean PSAction tabela proc_actions
*  - $this->stageBean - bean etapu StageOfProc tabela stages
*
*  - $stage = $this->getStageDefObj() - zwraca obiekt (klasy Bean) definicji etapu jaki jest wykonywany czyli dane z stage
*  - $procedure = $this->getProcedure() - zwraca obiekt (klasy Bean) procedury (tabela procedures) *
*
* @uses AbsCommand
* @uses ICommand
* @final
* @author Tomasz Świenty
* @version 0.1
* @copyright Copyright (c) BetaSoft
*/
final class ExampleCommand extends AbsCommand implements ICommand {



   /**
    * getDescription
    * Metoda zwracająca nazwę komendy (krótki opis). Nazwa ta pojawia się na liście wyboru komend.
    *
    * @static
    * @access public
    * @return string
    */
   public static function getDescription() {

       return Translator::translate('Moja nowa komenda');

   }



   /**
    * getExpandedDescription
    * Metoda zwracająca dłuższy opis komendy. Może zawierać znaki HTML.
    *
    * @param string $params - lista parametrów w formacie json (na razie nie jest obsługiwane)
    * @static
    * @access public
    * @return string
```

```
     */
    public static function getExpandedDescription($params = NULL) {

        return Translator::translate('Moja nowa komenda - rozszerzony opis');

    }



    /**
     * getCommandApi
     * Metoda zwracająca API komendy domyślnie jest implementowana przez AbsCommand i zwraca pustą tablicę.
     * Format api to tablica asocjacyjna, której kluczami głównymi są nazwy parametrów (najczęściej 6 znakowe) a każdy z ty
     * posiada definicję parametru również w postaci tablicy.
     *
     * Definicja parametru określana jest przez 3 atrybuty (klucze)
     *  - (strint)label - krótka nazwa parametru
     *  - (strint)dscrpt - dokładniejszy opis parametru
     *  - (bool)required - oznaczenie czy parametr jest wymagany
     *
     * Dodatkowo w tej tablicy (głównej) może się pojawić klucz example, które podaje przykład listy parametrów - jednak od
     * 3.7 jest to zbędne gdyż parametry są definiowane w dedykowanym formularzu i przechowywane są w formacie json a nie c
     *
     * @param string $params - lista parametrów w formacie json (na razie nie jest obsługiwane)
     * @static
     * @access public
     * @return array
     */
    public static function getCommandApi($params = NULL) {

        $api = array(
            'contid' => array(
                'label'     => Translator::translate('Kontrahent'),
                'dscrpt'    => Translator::translate('identyfikator kontrahenta (contacts.contid, <b>Lista kontrahentów ->
                'required'  => TRUE,
            ),
            'example' => 'contid="1"'
        );

        return $api;

    }



    /**
     * execute
     * Metoda odpowiedzialna za wykonanie komendy.
     * Poniżej znajdują się też najważniejsze rzeczy jakie trzeba zrobić!
     *
     * @param Bean $bean - obiekt formularza dokumentu albo sprawy do dyspozycji w tym obiekcie mamy wartości z pól
     * danego formularza (tabele documents (wraz z dodatkowymi tabelami), processes).
     * Wartości pobieramy metodą $bean->get('dscrpt');
     * @param string $params - lista parametrów jaka została zdefiniowana dla tej komendy w konkretnej procedurze.
     * @access public
     * @return bool|CommandException
     */
    public function execute(Bean $bean, $params) {

        // parsowanie parametrów oraz przekazanie kontekstu beana
        // po wykonaniu tego mamy do dyspozycji atrybut $this->params zawierający tablicę sparsowanych parametrów
```

```
        // (jeśli parametr jest w formacie SQL to w tablicy będzie dostępny wynik zapytania, jeśli np featid::89 to wartość
        parent::parseParams($params, $bean);

        // jeśli do działania komendy wymagane są jakieś parametry a użytkownik ich nie podał lub nie udało się
        // ich sparsować to należ wykonać poniższe sprawdzenie
        // Wyjątek CommandException przerywa działanie komedny!
        if ((empty($params)) OR (!is_array($this->params))) {
            $this->setMessage(Translator::translate('Komenda nie może zostać wykonana ze względu na brak parametrów lub nie
            $this->setMessage(Translator::translate('Komenda nie została wykonana.'), 'ERROR');
            throw new CommandException($this);
        }

        // api komendy do weryfikacji parametrów
        $api = self::getCommandApi();

        // walidajca parametrów
        if ((!array_key_exists('contid', $this->params)) OR (!is_numeric($this->params['contid']))) {
            $this->setMessage(sprintf(Translator::translate('Komenda nie może zostać wykonana ze względu na brak parametru
            $this->setMessage(Translator::translate('Komenda nie została wykonana.'), 'ERROR');
            throw new CommandException($this);
        }

        // klucz główny tabeli doc_id lub prc_id (kontekst procedury)
        $keyval = $bean->getPkeyValue();

        // $this->action - bean PSAction tabela proc_actions
        // $this->stageBean - bean etapu StageOfProc tabela stages

        // Zwraca obiekt (klasy Bean) definicji etapu jaki jest wykonywany czyli dane z stages_def - definicja etapu
        $stage = $this->getStageDefObj();

        // Zwraca obiekt (klasy Bean) procedury (tabela procedures)
        $procedure = $this->getProcedure();

        // Informacja o tym co zostało zrobione
        $this->setMessage(sprintf(Translator::translate('Kontakt o identyfikatorze %d został dodany do dokumentu.'), $this-

        return TRUE;

    }

} // class ExampleCommand

?>
```