Biblioteka dataWEB ASP

Instrukcja programisty ©Wojciech Pawłowski 2019

Wersja
Biblioteka `dataWEB.dll`, wersja: `1.0.85`, `32 bit`, data kompilacji: `2019-11-21`

Wstęp
Biblioteka ***dataWEB*** powstała dla uproszczenia tworzenia bazodanowych aplikacji WEB z wykorzystaniem danych *XML* produkowanych przez *MS SQL Server* oraz transformacji *XSLT* tych danych po stronie serwera. Jako silnik transformacji *XSLT* wykorzystywana jest biblioteka *MSXML2 6.0*.

Biblioteka pracuje w środowisku *IIS ASP*. Stanowi warstwę pośrednią pomiędzy skryptami biblioteki a jądrem *ASP*. Jest interpreterem skryptów ***dataWEB***. Programista nie musi znać *ASP* ani żadnego innego języka programowania po stronie serwera.

Dla efektywnego tworzenia aplikacji ***dataWEB*** wymagana jest znajomość języka *T-SQL* wraz z przetwarzaniem *XML* (dyrektywa *FOR XML*) oraz znajomość transformacji *XSLT 1.0* z wykorzystaniem *X-Path*. Biblioteka pozwala także na tworzenie wynikowego *HTML* bez użycia *XML* i transformacji *XSLT* a nawet bez połączenia z bazą danych.

Dyrektywa *FOR XML* umożliwia tworzenie dowolnych hierarchicznych struktur danych, które po transformacji *XSLT* tworzą strukturalny kod *HTML*. Część logiki aplikacji można z łatwością przenieść do arkusza *XSLT* stosując operacje *xsl:if* oraz *xsl:choose* a także definiować wewnątrz arkusza *XSLT* skrypty *Java Script* i przy pomocy nich budować dodatkowe funkcje niedostępne w *X-Path*.

Biblioteka umożliwia łatwe dwukierunkowe przekazywanie parametrów wywołań *SQL*. Dzięki temu można na przykład w jednym kroku zaprezentować wynik *HTML* i przekazać zwrotne dane do dalszego przetwarzania. Jako parametry wejściowe i wyjściowe mogą być stosowane także dane typu *XML*.

Biblioteka pozwala na rekurencyjne przetwarzanie kodu skryptów. Wynik *HTML* skryptu może zostać przeanalizowany powtórnie w celu wykonania zawartych w nim komend. Pozwala to na łatwą implementację hierarchii master-detail *HTML* o dowolnej liczbie poziomów zagnieżdżeń w wynikowym *HTML*.

Biblioteka upraszcza dostęp do zmiennych formularza *HTML*, parametrów wywołań *URL*, zmiennych sesji, plików *cookies* oraz parametrów transformacji XSLT. Dostęp do tych danych jest zwykłym odwołaniem do odpowiednio skonstruowanej nazwy zmiennej. Zmienne mogą także odczytywać i zapisywać dane typu *XML* także w *cookies* i zmiennych sesji lub plikach.

Biblioteka pozwala na tworzenie wersji językowych. Implementowane jest to poprzez dostęp do zmiennych o zawartości zależnej od aktualnego języka oraz wygodnej implementacji słownika w pliku słownika *.ini*. Biblioteka wewnętrznie pracuje w *UNICODE* a pliki z nią współpracujące zapisywane są jako *UNICODE litle endian* lub *UTF-8*. Wynik *HTML* może zostać zwracany w dowolnej wersji kodowej.

Biblioteka wyposażona jest w programowalny *cache*, który pozwala na selektywny zapis na lokalny dysk i późniejszy odczyt tylko wygenerowanych bloków HTML lub XML, bez konieczności ponownego kosztownego czasowo ich przetwarzania.

Biblioteka pozwala na osadzanie w niej i przetwarzanie skryptów *JScript* oraz *VbScript* także z możliwością tworzenia w nich procedur i funkcji. Biblioteka pozwala także na dołączanie specjalizowanych bibliotek *COM*, oraz posiada wbudowane mechanizmy obsługi poczty elektronicznej poprzez bibliotekę *CDO*.

Dzięki osadzeniu skryptów aplikacji w plikach tekstowych, możliwa jest późniejsza dowolna modyfikacja aplikacji oraz jej rozbudowa przez webmastera znającego niniejszą dokumentację. Nie jest wymagane żadne narzędzie *IDE* do projektowania aplikacji. Nie jest wymagany także kompilator. Do edycji w wystarczającym stopniu nadaje się *Notepad++*

Biblioteka pozwala na upload plików. Można opcjonalnie ustalić maksymalny rozmiar pliku. W przypadku plików graficznych: jpg, png, bmp oraz gif można opcjonalnie przeskalować obraz do maksymalnych rozmiarów: podanej szerokości i wysokości.

Można śledzić czas wykonania poszczególnych komend środowiska oraz przechwytywać kody błędów. Informacje te są zapisywane w plikach *.log*. Śledzenie czasu wykonania ma szczególne znaczenie przy optymalizacji stosowanych zapytań *T-SQL* i w konsekwencji pozwala na poprawę wydajności aplikacji.

Aplikacja dataWEB

Aplikacja w środowisku ***dataWEB*** składa się z plików:

- roboczego - `.asp`
- konfiguracyjno/skryptowego - `.ini`
- skryptów html - `.shtml`
- słownika - `.ini`
- oraz plików: `.sql, .xsl, .xml`

Aplikacja może zostać podzielona na wiele plików `.asp`, każdy z nich może posiadać własny plik konfiguracyjny `.ini`, lub mogą współużytkować ten sam plik konfiguracyjny.

Plik .asp *(kodowanie utf-8)*

Plik `.asp` o dowolnej nazwie musi znajdować się w katalogu roboczym ***IIS***.
Powinien zawierać poprawne wywołanie metody `Query` obiektu klasy `dataWEB.Page`

Przykład pliku .asp

```
<%
With Response
    .Buffer=True
    .Expires=-1
    .AddHeader "Content-Type", "text/html; charset=utf-8"
    .AddHeader "Content-Language", "en, pl"
    .Charset = "utf-8"
    .CodePage = 65001
End With
Dim e
Set e = Server.CreateObject("dataWEB.Page")
e.Query Request, Response, Server, Session, "config", "main"
%>
```

Parametr `config` (obowiązkowy) definiuje nazwę pliku konfiguracyjnego `.ini` przeznaczonego dla wykonywanego pliku `.asp`
Parametr `main` (opcjonalny) definiuje nazwę głównego pliku skryptu html `.shtml` aplikacji.
Plik główny zostanie załadowany i przetworzony jako pierwszy.
Nazwy plików powinny zostać podane bez rozszerzenia.

Jeśli parametr `main` zostanie opuszczony:
`e.Query Request, Response, Server, Session, "config"`
jako pierwszy zostanie uruchomiony skrypt zawarty w sekcji `[main]` pliku konfiguracyjnego.

Ścieżki do plików podaje się wyłącznie jako ścieżki względne do położenia pliku `.asp`

Plik kofiguracyjny .ini *(kodowanie Unicode litle endian)*

```
;średnik na początku linii stanowi komentarz i nie jest przetwarzany
;zapisy stanowią parę: klucz=wyrażenie

[db]
connectionString=łańcuch_połączenia_adodb
dictionary=nazwa_pliku_słownika
defaultLang=domyślny_symbol_języka
;klucze connectionString, dictionary oraz defaultLang są opcjonalne

errorLog=1
;errorLog=1 - do pliku errorLog.log będą zapisywane informacje o błędach
;errorLog=0 - nie będą zapisywane informacje o błędach
;W pliku logu zapisywane są w kolejnych kolumnach:
;data i czas, QueryString, nazwa pliku skryptu, nazwa procedury, numer błędu, opis błędu

timeLog=1
;timeLog=1 - do pliku timeLog.log będą zapisywane informacje o czasie wykonania przetwarzanych
;komend oraz o czasie przetwarzania całego żądania,timeLog=0 - nie będą zapisywane informacje

cache=folder_chache
;klucz cache wymagany jest tylko, jeśli będzie używana komenda ##cache…## lub ##cacheXML…##

cookiesPath=sklep/
;klucz cookiesPath wymagany, kiedy aplikacja osadzona jest w podkatalogu domeny,
;zapewnia zgodność dla cookies Firefox tworzonych po stronie przeglądarki

[xslt]
;sekcja [xslt] jest opcjonalna, może zawierać nazwy i wartości parametrów XSLT
;parametry zostaną wstawione do wszystkich transformacji XSLT
;wielkość liter nazw parametrów ma znaczenie
NazwaParametru1=WarośćParametru1
```

```
NazwaParametru2=WarośćParametru2

[license]
key1=klucz1
key2=klucz3
…
keyN=kluczN
;klucze licencyjne biblioteki dataweb.dll przypisane do adresu url aplikacji

[main]
;lista par: klucz=wyrażenie
klucz1=wyrażenie1
klucz2=wyrażenie2

[email]
;definicja serwera SMTP do wysyłania poczty przez aplikację
from=adres@email
smtp=serweremail.com
port=25
ssl=0
login=LOGIN_DO_SERWERA_POCZTY_SMTP
haslo=HASŁO_DO_SERWERA_POCZTY_SMTP
```

Plik słownika .ini *(kodowanie Unicode litle endian)*

Użycie pliku słownika jest opcjonalne. Umożliwia on tworzenie wersji językowych aplikacji.
Plik słownika zawiera nazwy sekcji oznaczające język oraz pary kluczy i wartości osobne dla każdego języka.

Nazwy sekcji muszą być dowolnymi łańcuchami tekstowymi bez białych znaków. Wygodnie aby oznaczały kod języka (niekoniecznie tak musi być).

Klucze muszą być dowolnymi łańcuchami tekstowymi bez spacji i znaków nowej linii.

Wartości mogą być dowolnymi tekstami zawierające tłumaczenie określonych fraz na język wskazany w nazwie sekcji, bez znaków nowego wiersza, o długości do 1024 znaków.

Przykład pliku słownika:

```
[pl]
zapisz=Zapisz
08=Witaj mój przyjacielu!
[en]
zapisz=Save
08=Hello, my friend!
```

Nazwy plików:

Nazwy plików nie mogą zawierać znaków: `@`, `#` oraz `$`.
Nazwy plików w skryptach podaje się bez rozszerzenia.
Rzeczywiste pliki muszą być zlokalizowane w katalogu roboczym wykonywanego pliku .asp, lub katalogach podrzędnych do niego. Muszą posiadać w nazwach rozszerzenia: `.ini` `.shtml` `.sql` `.xml` oraz `.xsl`.
Wszystkie pliki `.shtml` `.ini` oraz `.sql` muszą posiadać kodowanie *UNICODE Litle Endian (UNICODE)*.
Pliki `.xsl oraz` `.xml` powinny posiadać kodowanie *UTF-8*.

Pliki skryptów aplikacji .shtml *(kodowanie Unicode litle endian)*

Pliki mogą zawierać dowolny kod html wraz z komendami środowiska dataWEB.

Formaty komend:

Format zapisu komend:
`##nazwaKomendy~argument1~argument2 ... ~argumentN##`
Nazwy komend składają się wyłącznie z liter, ich wielkość nie ma znaczenia.
Argumenty mogą być zmiennymi: `@nazwa_zmiennej`, lub stałymi tekstowymi

Zmienne:

Nazwy zmiennych zawsze rozpoczynają się znakiem `@`.
Zmienne, których nazwa jest postaci `@*nazwa_zmiennej` są zmiennymi wskaźnikowymi. Są referencją do obiektów XMLDocument.
Pozostałe zmienne przechowują dane typu tekstowego lub zawierają definicję zmiennej lub komendy albo przepis skryptu.
Zmiennych nie trzeba deklarować, wystarczy przypisać lub odczytać jej wartość.
Zmienna *read/write* wcześniej nie ustawiona przyjmuje wartość pustą.
Nazwy zmiennych nie mogą zawierać znaku nowej linii, znaku `~` oraz sekwencji `##`
Wielkość liter w nazwach zmiennych nie ma znaczenia z wyjątkiem parametrów transformacji XSLT @T_

Typy zmiennych:

Typ zmiennych zależy od formatu zapisu jej nazwy:

Zmienne specjalne: *read only*
`@sessionID` – zwraca bieżący identyfikator sesji
`@isMobile` - zwraca `1` jeśli żądanie pochodzi z urządzenia mobilnego, lub `0` jeśli nie.
`@guid` - zwraca nowe `GUID`
`@asp` - zwraca nazwę bieżącego pliku `.asp`, np: `start.asp`
`@url` – zwraca url bieżącego pliku `.asp`, np. `http://mojadomena.pl/start.asp`
`@buffer` - przechowuje strumień wyjściowy w trybie `S`. (*read/write wyjątkowo)*
`@emailError` - zwraca komunikat o błędzie po wysłaniu e-maila
`@body` – zwraca zawartość sekcji BODY żądania http
`@date` – zwraca bieżącą datę w formacie *`yyyy-mm-dd`*
`@now` – zwraca bieżący czas i datę formacie *`yyyy-mm-dd hh:nn:Ss`*
`@version` – zwraca nazwę i wersję biblioteki ***dataWEB***
`@fXML` – zwraca wszystkie parametry formularza HTML w formacie XML o postaci:
`<form name1="value1" name2="value2"/>`
gdzie `name1, name2` to nazwy pola formularza html (małe litery)
`@qXML` – zwraca wszystkie parametry żądania query url w formacie XML o postaci:
`<query name1="value1" name2="value2"/>`
gdzie `name1, name2` to nazwy parametrów żądania asp (małe litery)
`@sXML` – zwraca wszystkie parametry sesji asp w formacie XML o postaci:
`<session name1="value1" name2="value2"/>`
gdzie `name1, name2` to nazwy parametrów sesji asp (małe litery)
`@cXML` – zwraca wartości wszystkich plików cookies w formacie XML o postaci:
`<cookies name1="value1" name2="value2"/>`
gdzie `name1, name2` to nazwy plików cookies (małe litery)

`@zmienna` **- zmienna lokalna** *read/write*
Zawartość zmiennej lokalnej przechowywana jest tylko podczas przetwarzania pojedynczego żądania.
Jako zmiennych lokalnych nie można używać nazw zmiennych specjalnych podanych powyżej.

`@C_nazwaCookies[#daysExpires]` **- zmienna cookies** *read/write*
Zmienna cookies zachowywana jest w plikach cookies w przeglądarce klienta
do zdefiniowanego czasu wygasania, domyślnie w zakresie sesji przeglądarki.

Opcjonalnie, w operacjach zapisu do zmiennej `@C_` można do nazwy dodać znak `#` a po nim liczbę dni wygasania zawartości, np: wykonanie komendy `##set~@C_user#30~kowalski##`
wyśle do przeglądarki *cookies* o nazwie `user` z zawartością *`kowalski`* wygasające za `30` dni.

`@Q_queryVariable` - **zmienna query url** *read only*
np: przy wywołaniu `dataWEB.asp?zmienna1=aaa&zmienna2=bbb`
zmienna `@Q_zmienna1` zwróci wartość: *`aaa`*, a `@Q_zmienna2` wartość *`bbb`*

Natomiast zmienna `@Q_` zwróci tekst: *`zmienna1=aaa&zmienna2=bbb`*
Można dzięki temu badać, czy istnieją jakiekolwiek zmienne query url:
`##if~@Q_~~wyrażenie1~wyrażenie2##`
Jeśli brak zmiennych query url to wykonywane jest `wyrażenie1`, w przeciwnym wypadku `wyrażenie2`.

`@S_SessionVariable` - **zmienna sesji** *read/write*
Zmienna zachowuje się podobnie jak *cookies*, lecz w sposób niejawny.
Nie da się odczytać jej wartości w przeglądarce klienta.
Zmienna zachowuje ważność tylko podczas sesji.

`@F_FieldName` - **wartość pola formularza html** *read only*
np: `@F_ID` zwróci wartość pola `<input name="ID">` formularza html.

`@P_Section_Parameter` - **wartość parametru z sekcji pliku konfiguracyjnego** *read/write**
Np: `@P_parameters_maxrow` zwróci wartość `30` jeśli w pliku konfiguracyjnym będzie zapis `maxrow=30` w sekcji `[parameters]`

*zapis do zmiennych `@P_Section_Parameter` podlega następującym ograniczeniom:
- znaki nowego wiersza zamieniane są automatycznie na pojedyncze spacje,
- długość tekstu ograniczana jest do 1024 znaków.

W parametrach mogą być wpisywane fragmenty html wraz z komendami. Ograniczeniem jest jednak brak możliwości stosowania znaków nowej linii.

`@P_Section` – **tablica wartości wszystkich parametrów sekcji** *read only*

Np: jeśli w pliku konfiguracyjnym będzie zapis :
```
[i]
X=ala
Y=ma
Z=kota
```

`@P_i` zwróci tekst:
ala
ma
kota

`@T_nazwaParametruXSLT` – **parametr transformacji XSLT** *read/write, wielkość liter ma znaczenie*
Ustawienie zmiennej skutkuje dodaniem parametru do każdej następującej po tym fakcie transformacji XSLT w obrębie pojedynczego żądania http.

Np.: `##set~@T_Tytuł~Moja strona##` przekaże wartość: `Moja strona` dla parametru `$Tytuł` w szablonie transformacji XSLT. Aby parametr mógł być wykorzystywany w szablonie XSLT, konieczna jest jego deklaracja w pliku XSL: `<xsl:param name="Tytuł"/>`

`@D_klucz` – **wartość klucza w pliku słownika (*dictionary*) dla bieżącego języka** *read only*
Np.: `@D_zapisz` zwróci wartość `save` jeśli bieżącym językiem jest `en` a w pliku słownika istnieje zapis:

```
[en]
…
zapisz=save
```

W celu zmiany bieżącego języka należy użyć komendy `##setLang~…##`

Jeśli w pliku słownika nie będzie wskazanego klucza, zmienna zwróci nazwę klucza, np.:
`<input type="button" value="##read~@d_OK##" />`
Utworzy przycisk z tekstem `OK` jeśli nie będzie klucza o nazwie `OK` w pliku słownika w sekcji bieżącego języka.

`@V_ServerVariable` - **wartość zmiennej serwera serverVariables** *read only*
Np: `@V_REMOTE_ADDR` zwróci adres przeglądarki klienta.
Przykładowe nazwy zmiennych:
`@V_HTTP_HOST`, `@V_URL`, `@V_REMOTE_ADDR`, `@V_HTTP_USER_AGENT`, `@V_HTTP_COOKIES`

`@X_SqlFileName` - **zawartość pliku SQL** *read only*
Zmienna zwraca zawartość istniejącego pliku `.sql`
Np: `@X_pliki_sql\kontrahenci` zwróci zawartość pliku: `pliki_sql\kontrahenci.sql`

`@M_XmlFileName` - **zawartość pliku XML** *read/write*
Zmienna zwraca lub zapisuje zawartość pliku `.xml`
Np: `@M_parametry` zwróci zawartość pliku: `parametry.xml`

Jeśli zmienna zostanie ustawiona na wartość pustą, plik `.xml` jest usuwany.
`##set~@M_parametry~##`
Usunie plik o nazwie `parametry.xml`, jeśli istnieje.

`@H_shtmlFileName` - **zawartość pliku SHTML** *read only*
Zmienna zwraca zawartość istniejącego pliku `.shtml`
Np: `@H_skrypty\logowanie` zwróci zawartość pliku: `skrypty\logowanie.shtml`

`@R_adres` – **redirect** *read only*
Zmienna niczego nie zwraca, lecz wywołuje metodę `Redirect` obiektu `Response` do wskazanego `adresu`.
Z uwagi na zabezpieczenia IIS adres musi być w domenie żądania.
Odczyt zmiennej przerywa wykonywanie skryptu.

Np: `##jumpif~@ismobile~1~@r_mobile/strona.html~##`
Jeśli żądanie pochodzi z urządzenia mobilnego, bieżący skrypt jest przerywany
i następuje przekierowanie do adresu: `mobile/strona.html`

Zmienne z wartością domyślną:

@Q_nazwa#wartośćDomyślna
@C_nazwa#wartośćDomyślna
@F_nazwa#wartośćDomyślna
@S_nazwa#wartośćDomyślna
@V_nazwa#wartośćDomyślna

Jeśli wartość zmiennej odczytana standardowo daje pustą wartość, zwracana jest wartość domyślna.
Człon `wartośćDomyślna` musi być zwykłym tekstem.

`##read~@c_kolor#czerwony##`
zwróci `czerwony` jeśli cookies o nazwie `kolor` będzie puste.

Zmienne alternatywne: **@FQ_ @FS_ @FC_ @QS_ @QC_**

Zmienne alternatywne zwracają pierwszą wartość niepustą z dwóch wartości:
`@FS_nazwa` zwraca zawartość pola formularza o nazwie `nazwa`, jeśli pole nie istnieje - zwraca wartość `@S_nazwa`
`@FQ_nazwa` zwraca zawartość pola formularza o nazwie `nazwa`, jeśli pole nie istnieje - zwraca wartość `@Q_nazwa`
`@FC_nazwa` zwraca zawartość pola formularza o nazwie `nazwa`, jeśli pole nie istnieje - zwraca wartość `@C_nazwa`
`@QS_nazwa` zwraca wartość parametru query url `nazwa`, jeśli parametr nie istnieje - zwraca wartość `@S_nazwa`
`@QC_nazwa` zwraca wartość parametru query url `nazwa`, jeśli parametr nie istnieje - zwraca wartość `@C_nazwa`

Uogólnieniem zmiennych alternatywnych jest komenda `##isEmpty…##`

Zmienne automatyczne: **@FS_nazwa! @FC_nazwa! @QS_nazwa! @QC_nazwa!**

Zmienne automatyczne zachowują się tak samo jak zmienne alternatywne lecz dodatkowo zapisują wartość do zmiennej nr 2 jeśli zmienna nr 1 jest niepusta.

Przykład: `@FS_producent!`
Odczytywana jest zmienna pola formularza HTML `<input name="producent" />`
Jeśli pole formularza nie istnieje, zwracana jest wartość zmiennej sesji o nazwie `producent`.
Jeśli pole formularza istnieje, zwracana jest zawartość tego pola i dodatkowo ustawiana jest wartość zmiennej sesji o nazwie `producent` na zawartość tego pola.

Zmienne **@FC_nazwa#ile_dni!** oraz **@QC_nazwa#ile_dni!** Pozwalają dodatkowo określić czas ważności plików *cookies* np.: `@FC_kategoria#5!` wybrana kategoria będzie pamiętana przez 5 dni.

Zmienne kopiujące: **@FS_nazwa% @FC_nazwa% @QS_nazwa% @QC_nazwa%**

Zmienne kopiujące zwracają wartość zmiennej nr 1 jednocześnie ustawiając zawartość zmiennej nr 2 na wartość odczytaną z pierwszej, także wartość pustą.

Przykład: `@FS_producent%`
Odczytywana jest zmienna pola formularza HTML `<input name="producent" />`
Jeśli pola formularza nie ma, zwracana jest wartość pusta i jednocześnie kasowana jest wartość zmiennej sesji `@S_producent`
Jeśli pole formularza istnieje, zwracana jest zawartość tego pola i jednocześnie zmienna sesji `@S_producent` ustawiana jest na tę samą wartość
W zmiennych kopiujących wartość drugiej zmiennej nie jest nigdy odczytywana.

Zmienne **@FC_nazwa#ile_dni%** oraz **@QC_nazwa#ile_dni%** pozwalają dodatkowo określić czas ważności plików cookies

Zmienne definicyjne: **@@zmienna**

Zmienne definicyjne przechowują definicje dowolnych zmiennych:
`##set~@zmienna~definicja##`

Odczyt zmiennej wykonuje się poprzedzając jej nazwę dodatkowym znakiem `@`, np.:
`##set~@zmienna~q_user##`
`##read~@@zmienna##`
zwróci do wyjścia zawartość zmiennej query url o nazwie `user`
tak samo jak: `##read~@q_user##`

Definicja zmiennej może zostać wstawiona z innej zmiennej:
`##setText~@a~q_user##`
`##set~@zmienna~@a##`
`##read~@@zmienna##`

Teksty:

Stałe tekstowe nie mogą zaczynać się znakiem `@` i nie mogą zawierać sekwencji `##`. Jeśli istnieje potrzeba wyprowadzenia tych znaków, należy użyć ekwiwalentów html. Teksty mogą zawierać znaki nowej linii.

Wyrażenia:
Wyrażenia mogą być nazwami zmiennych lub tekstem,
nie mogą zawierać znaków ~ oraz sekwencji ##.

Komentarze:
wstawienie znaku ; (średnik) na początku linii wyłącza całą linię skryptu z dalszego przetwarzania.

```
;to jest komentarz
 ;a to nie jest komentarz
```

Łamanie linii:

```
##sqlFORXML
~@X_motoshop\userEdit$
~##
```

W komendach środowiska można stosować znaki nowej linii przed znakiem ~ oraz przed znakami ##
Znaki CRLF zostaną wówczas pominięte.

W każdym innym przypadku znak nowej linii CRLF zostanie odczytany i w zależności od kontekstu zwrócony do wyjścia:

```
##read~przykładowy
tekst##
```

zwróci do wyjścia:
przykładowy
tekst

Procedury:

```
;begin @zmienna
…
kod html zawierający komendy
…
;end
```

Wstawia do zmiennej @zmienna blok kodu html zawierający dowolne komendy ##...##. Zmienne i komendy zawarte wewnątrz bloku nie zostaną przetworzone i nie zostaną zwrócone do wyjścia. Średnik musi być pierwszym znakiem wiersza, po nazwie zmiennej musi nastąpić znak nowej linii. Po słowie end musi nastąpić znak nowej linii. Zmienna musi być *read/write*, nie może być zmienną typu @P_ Najlepiej aby była zmienną lokalną @zmienna. W celu przetworzenia ciała procedury zawartej w zmiennej należy użyć komend:
execute, if, jumpIf, choise, jumpChoise, case, jumpCase, regexp.

Komenda: set (lub S)

```
##set~@zmienna1~wyrażenie1~@zmienna2~wyrażenie2... ~@zmiennaN~wyrażenieN##
```

Ustawia wartości zmiennych na wartości wyrażeń
gdzie @zmiennaX - nazwa dowolnej zmiennej *write*
Zmienne i wyrażenia muszą wystąpić parami.

Wyrażenia mogą być zmiennymi:

```
##set~@zmienna~@zmiennaABC##
```

Przypisze @zmiennej wartość odczytaną ze @zmiennejABC.

Komenda: setIf

```
##setIf~wyrażenie1~wyrażenie2~@zmienna~wyrażenie3~wyrażenie4##
```

Jeśli wartość wyrażenie1 jest równa wartości wyrażenie2 to @zmienna zostanie ustawiona na wartość wyrażenie3, w przeciwnym wypadku na wartość wyrażenie4, gdzie @zmienna - nazwa dowolnej zmiennej *write*
Każde z wyrażeń może być zmienną lub stałą tekstową, także pustą.

Komenda: setCommand

```
##setCommand~@zmienna~komenda~Arg1~Arg2...~ArgN##
```

gdzie @zmienna - nazwa dowolnej zmiennej *read/write*
Wstawia do zmiennej definicję komendy: ##komenda~Arg1~Arg2...~ArgN##

Komenda: setText

`##setText~@zmienna~tekst##`
gdzie `@zmienna` - nazwa dowolnej zmiennej *write.*
Wstawia do zmiennej tekst.
Tekst może zawierać znaki nowej linii, znaki `@` oraz znaki `~`.
Nie może zawierać sekwencji `##`

Komenda: setLang

`##setLang~wyrażenie##`
Ustawia bieżący język na wartość wyrażenia. Nie zwraca niczego do wyjścia.

`##setLang~@C_lang##`
Ustawia bieżący język słownika na wartość odczytaną z pliku cookies o nazwie `lang`

Komenda: setXML

`##setXML~@xml~@Var1~@Var2~...~@VarN##`
Ustawia zmienną `@xml` na tekst:
`<root><item name="Var1" value="Value1"/><item name="Var1" value="Value1"/> … </root>`

Jeśli wartość zmiennej `@VarN` będzie pusta, atrybut `value` nie zostanie wstawiony:
`<item name="Var1" />`

Wszystkie parametry komendy muszą być zmiennymi. Zmienna `@xml` musi być *write.*

Komenda: append

`##append~@zmienna~wyrażenie1~wyrażenie2~wyrażenie3...~wyrażenieX##`
wstawia do zmiennej *write* połączenie wartości wyrażeń.
Wyrażenia mogą być zmiennymi.

Komenda: replace

`##replace~source_text~find~replace~@zmienna##`
Przypisuje do zmiennej `@zmienna` tekst po zastąpieniu wyrażeń `find` wyrażeniami `replace`.

`##replace~@zmienna~@find~@replace##`
Zastąpienie następuje bezpośrednio w zmiennej `@zmienna`

Wszystkie parametry mogą być zmiennymi.

Komenda: htmlEncode

`##htmlEncode~@zmienna##`
Koduje wartość `@zmienna` zamieniając znaki kluczowe html: < > ” na odpowiadające im ekwiwalenty:
`< > "`

`##htmlEncode~@zmienna~@zmienna2##`
Koduje wartość zmiennej jak wyżej, zapisując wynik w `@zmienna2`

Komenda: href

`##href~wyrażenie~tekst1~wyrażenie1...tekstN~wyrażenie##`
Tworzy i wyprowadza do wyjścia adres url utworzony z `tekstówX` i `wyrażeńX`, kodując argumenty odczytane z `wyrażeńX` jako *url encode*.

Jeśli wartość parametru `wyrażenie` będzie pusta, wstawiana jest nazwa pliku `.asp` bieżącego skryptu.
Pozostałe parametry są opcjonalne ale muszą występować parami: `tekst1~wyrażenie1`
Wyrażenia mogą być nazwami zmiennych.

`##href~~user~@C_user~dokument~@typ_dokumentu##`
wyprowadzi do wyjścia tekst: *`start.asp?user=kowalski&dokument=FA`*

`##href~##`

wyprowadzi do wyjścia tekst: `start.asp` *(jeśli bieżący skrypt wywołany jest z pliku start.asp)*

Komenda: read (lub R)

`##read~wyrażenie##`
zwraca do wyjścia wartość wyrażenia.
Jeśli wyrażenie jest zmienną zawierającą komendy, nie zostaną one przetworzone.

Komenda: xmlItem

`##xmlItem~@var~@xml~queryString##`
Przeszukuje i odczytuje ze zmiennej zawierającej xml tekst określony tekstem `queryString` *(xPath)* i zapisuje wynik w zmiennej `@var`.

Jeśli zmienna `@var` nie zostanie podana:
`##xmlItem~~@xml~queryString##`
odczytany tekst zostanie zwrócony do wyjścia (obiekt response).

Jako `queryString` należy podać tekst.

Na przykład:
```
##xmlItem~~<root><item name="i" value="x"/><item name="Test" value="abcd"/></root>
~root/item[2]/@value##
```
zwróci do przeglądarki tekst: `abcd`

Komenda: xmlNode

`##xmlNode~@var~@xml~queryString##`
Przeszukuje i odczytuje ze zmiennej zawierającej xml gałąź xml określoną tekstem `queryString` *(xPath)* i zapisuje wynik jako dokument XML w zmiennej `@var`.
Zmienna `@var` może być także zmienną wskaźnikową `@*var`

Jeśli zmienna `@var` nie zostanie podana:
`##xmlNode~~@xml~queryString##`
Utworzony dokument XML tekst zostanie zwrócony do wyjścia (obiekt response).

Jako `queryString` należy podać tekst.

Na przykład:
```
##xmlNode~~<root><item name="i" value="x"/><item name="Test" value="abcd"/></root>~root/item[2]##
```
zwróci do przeglądarki tekst: `<item name="Test" value="abcd"/>`

Komenda: xmlNodes

`##xmlNodes~@var~@xml~queryString~root##`
Tworzy wynikowy dokument xml z głównym węzłem o nazwie podanej w parametrze `root` i wstawia do niego elementy wynikające z wyszukania z dokumentu `@xml` za pomocą `queryString` *(xPath),* po czym zapisuje wynik jako dokument XML w zmiennej `@var`. Jeśli wyszukanie nie zwróci żadnych elementów, zmienna `@var` nie zostanie ustawiona.

Zmienna `@var` jak również `@xml` mogą być zmiennymi wskaźnikowymi `@*var`, `@*xml`

Jeśli zmienna `@var` nie zostanie podana:
`##xmlNode~~@xml~queryString~root##`
utworzony dokument XML w postaci tekstowej, zostanie zwrócony do wyjścia (obiekt Response).

Parametr `queryString` jest obowiązkowy, należy podać zwykły tekst (nie można użyć zmiennej).

Jeśli parametr `root` pozostanie pusty, jako nazwę elementu głównego zostanie zastosowana nazwa: *root*.

Na przykład:
```
##xmlNodes~@a~<root><item name="i" value="x"/><item name="Test" value="abcd"/><item
name="Test2" value="egc"/></root>~root/item[position()<3]~wynik##
```

Ustawi zmienną `@a` na wartość:
```
<wynik><item name="i" value="x"/><item name="Test" value="abcd"/></wynik>
```

Komenda: xmlAppend

`##xmlAppend~@xml~queryString~@xml2##`

Do węzła będącego wynikiem przeszukania dokumentu zawartego w parametrze `@xml` wstawia drugi dokument odczytany z parametru `@xml2`
Parametr `queryString (xPath)` określa wyszukiwany węzeł, do którego nastąpi wstawienie. Jeśli parametr ten będzie pusty, drugi xml zostanie wstawiony do węzła głównego (`root)`.
Jeśli parametr `queryString` nie wskaże istniejącego węzła, wstawienie nie nastąpi.
Parametr `@xml` musi wskazywać na istniejącą zmienną, `@xml2` może być zmienną lub tekstem.

Przykład 1 – bez parametru `queryString`:

```
##set~@xml~<root><el1></el1><el2></el2></root>##
##xmlAppend~@xml~~<dodane><el3></el3></dodane>##
```

Wynik w zmiennej **`@xml`**:
`<root><el1></el1><el2></el2><dodane><el3></el3></dodane></root>`

Przykład 2 – z parametrem `queryString`:

```
##set~@xml~<root><el1></el1><el2></el2></root>##
##xmlAppend~@xml~/root/el2~<dodane><el3></el3></dodane>##
```

Wynik w zmiennej **`@xml`**:
`<root><el1></el1><el2><dodane><el3></el3></dodane></el2></root>`

Komenda: xmlRemove

`##xmlRemove~@xml~queryString ##`

Z dokumentu `XMLDOcument` zawartego w zmiennej `@xml` zostaną usunięte wszystkie węzły, znalezione za pomocą parametru `queryString`. Parametr `queryString` jest obowiązkowy.

Przykład:

```
##set~@xml~<root><el1></el1><el2></el2></root>##
##xmlRemove~@xml~/root/el2~##
```

Wynik w zmiennej **`@xml`**:
`<root><el1></el1></root>`

Komenda: isEmpty

`##isEmpty~wyrażenie1~wyrażenie2~@zmienna##`
Zapisuje do `@zmienna` wartość z `wyrażenie1` jeśli jest ona niepusta. W przeciwnym wypadku zapisuje wartość z `wyrażenie2`.

`##isEmpty~wyrażenie1~wyrażenie2##`
Zwraca do wyjścia wartość z `wyrażenie1` jeśli jest ona niepusta. W przeciwnym wypadku zwraca wartość z `wyrażenie2`

Wyrażenia mogą być dowolnymi zmiennymi:
`##isEmpty~@f_user~@c_user~@user##`
Sprawdza login z pola formularza html. Jeśli nie ma pola formularza lub pole jest puste, wpisuje do zmiennej @user zawartość pliku *cookies* o nazwie `user`

Komenda: execute

`##execute~wyrażenie##`
Wykonuje komendę lub skrypt zawarty w `wyrażeniu` i zwraca wynik do wyjścia.

`##execute~@nazwaZmiennej##`
przetwarza komendę lub skrypt odczytany ze zmiennej `@nazwaZmiennej`.

`##execute~@nazwaZmiennej$##`
zwraca wartość zmiennej do wyjścia.

`##execute~@h_nazwaPliku##`
przetwarza i zwraca do wyjścia zawartość pliku `.shtml`

##execute~@h_nazwaPliku$##
zwraca do wyjścia zawartość pliku .shtml bez przetwarzania zawartości

##execute~tekst##
zwraca tekst – to samo co ##read~tekst##

Komenda: break
##break~wyrażenie1~wyrażenie2##
przerywa wykonywanie skryptu jeśli wartości wyrażeń są równe

Komenda: breakNot
##breakNot~wyrażenie1~wyrażenie2##
przerywa wykonywanie skryptu jeśli wartości wyrażeń są różne

Komendy: if, jumpIf
##if~wyrażenie1~wyrażenie2~wyrażenie3~wyrażenie4##

Jeżeli wartości wyrażenie1 i wyrażenie2 są równe, przetwarzane jest wyrażenie3.
W przeciwnym wypadku przetwarzane jest wyrażenie4.

Jeśli wyrażenie3 lub wyrażenie4 jest zmienną typu @H_nazwaPliku to można dodać na końcu zmiennej znak $.
Wówczas zawartość pliku zostanie zwrócona do wyjścia bez przetwarzania.

Przykład:
##if~@C_user~~~@H_nazwaPliku2##
Jeśli użytkownik jest zalogowany (cookies user jest niepuste)
to zostanie przetworzona zawartość pliku nazwaPliku2.shtml

Dla komendy: jumpIF - po wykonaniu lub odczycie niepustego wyrażenia 3 lub 4, dalsza część skryptu jest przerywana (break).

Przykład:
##jumpIF~@C_user~~~##
Jeśli cookies user jest puste, przetwarzanie skryptu jest przerywane.

Komendy: choise, jumpChoise
##choise~wyrażenie~wyrażenie1~wyrażenie2~…~wyrażenieN...~wyrażenieElse##
jeśli wartość parametru: wyrażenie będzie poprawną wartością liczbową,
przetwarza zawartość ze wskazanego wyrażeniaX (adresowanie od 1 do N).

Jeśli wyrażenie nie będzie poprawną wartością liczbową, lub jego wartość przekroczy możliwy adres, przetwarzane jest wyrażenieElse

Każdy z parametrów może być zmienną lub stałą tekstową:
##choise~@Adres~@N1~@N2~@N3...~@NN~@else##

Przykład:
##set~@imie~2##
piesek ma na imię ##choise~@imie~reksiu~burek~azor~##
zwróci tekst: *piesek ma na imię burek*

Jeśli wyrażenieX będzie zmienną zawierającą skrypt lub komendę i będzie ona zakończona znakiem $, nie nastąpi przetworzenie.

Dla komendy: jumpChoise - po wykonaniu wyrażenia dalsza część skryptu jest przerywana (break).

Komendy: case, jumpCase
##case~wyrażenie~wartość1~wyrażenie1~…~wartośćN~wyrażenieN...~wyrażenieElse##
poszukiwana jest pierwsza wartość zgodna z wartością wyrażenia i przetwarzane jest wyrażenie związane z tą wartością.

Jeśli wartość parametru wyrażenie nie zostanie odnaleziona, przetwarzane jest wyrażenieElse

Każdy z parametrów może być zmienną lub stałą tekstową:
`##case~@zmienna~@N1~@H_skrypt1~@N2~@H_skrypt2...~@H_skryptElse##`

Przykład:
```
##set~@imie~burek##
<p>piesek jest ##case~@imie~reksiu~ładny~burek~wściekły~##</p>
```
zwróci tekst: *piesek jest wściekły*
Jeśli `wyrażenieX` będzie zmienną zawierającą skrypt lub komendę i będzie ona zakończona znakiem **$**, nie nastąpi przetworzenie.

Dla komendy: `jumpCase` - po wykonaniu wyrażenia wynikowego dalsza część skryptu jest przerywana (break).

Komendy: select, jumpSelect

`##select~parametr~sekcja~wyrażenieElse##`
Jeżeli odnaleziony zostanie `parametr` w sekcji `sekcja` pliku konfiguracyjnego, to przetwarzana jest wartość tego parametru, w przeciwnym wypadku przetwarzana jest wartość wyrażenia `wyrażenieElse`.

Każdy z parametrów komendy może być zmienną:
`##select~@p~@s~@e##`

Jeśli jako `parametr` podana zostanie zmienna zakończona znakiem `$` to odnaleziona w pliku konfiguracyjnym wartość tego parametru nie zostanie przetworzona:
`##select~@p$~@s~@e##`

Jeśli `wyrażenieElse` jest zmienną zakończoną znakiem `$`, to w przypadku nie odnalezienia parametru wartość odczytana z `wyrażenieElse` nie zostanie przetworzona:
`##select~@p~@s~@e$##`

Parametr komendy: `wyrażenieElse` jest opcjonalny:
`##select~parametr~sekcja##`
lub
`##select~parametr~sekcja~##`

Dla komendy: `jumpSelect` - po wykonaniu wyrażenia wynikowego dalsza część skryptu jest przerywana (break).

Komenda: redirect

`##redirect~wyrażenie##`
Przekierowuje wywołanie do adresu odczytanego z wartości `wyrażenia`.

Na przykład:
`##redirect~mobile/index.html##`
albo:
`##redirect~@adres##`

Z uwagi na zabezpieczenia IIS, adres musi być w domenie żądania.
Wykonanie komendy `redirect` przerywa dalsze wykonywanie bieżącego skryptu.

Komenda: equal

`##equal~wyrażenie1~wyrażenie2~wyrażenie3~wyrażenie4##`
Sprawdza, czy wartość `wyrażenie1` jest równa wartości `wyrażenie2`.
Jeśli tak, do wyjścia zwracana jest wartość `wyrażenie3`.
Jeśli nie, do wyjścia zwracana jest wartość `wyrażenie4`.
W odróżnieniu od komendy `if`, wyrażenia nie są nigdy przetwarzane.

Komenda: email

`##email~adresOdbiorcy~tematWiadomości~treśćWiadomości~czyFormatHTML##`
Wysyła e-mail, wymaga zapisów w sekcji `[email]` w pliku konfiguracyjnym, definiujących parametry serwera poczty SMTP.

Jako parametry wywołania można użyć dowolnych zmiennych lub stałych tekstowych
`##email~@N1~@N2~@N3~@N4##`

Parametr `czyFormatHTML`: definiuje, czy e-mail ma być wysyłany w formacie HTML (=`1`) lub zwykłego tekstu (=`0`).

Przykład:
```
##email~adres@poczta.pl~Ważna informacja~Pozdrawiam serdecznie~0##
```

Komenda: stream

`##stream~R##`
ustawia strumień wyjściowy na obiekt `Response` (domyślnie)

`##stream~B##`
kasuje wewnętrzną zmienną `@buffer` i kieruje strumień wyjściowy do tej zmiennej.
Przydatne jeśli istnieje potrzeba zapamiętania wyniku HTML przetwarzanego bloku celem na przykład umieszczenia go w treści wiadomości e-mail.
Wynik przechowywany jest w zmiennej specjalnej `@buffer`

Przykład:
```
##stream~b##
##set~@buffer~<html><head></head><body>##
##sqlth~@x_query1$~##
##read~<p>Pozdrawiam serdecznie</p></body></html>##
##email~email@poczta.pl~Tabela wyników~@buffer~1##
##stream~r##
```

Komenda: cache

`##cache~ile_dni~par1~par2~…~parN##`
Zapisuje strumień wyjściowy do pliku `.shtml` w folderze zdefiniowanym w parametrze `cache` pliku konfiguracyjnego w sekcji `[db]`.

Nazwa pliku jest utworzona z wartości parametrów `parN` rozdzielonych znakami ~
Zasięgiem komendy jest tylko aktualnie wykonywany skrypt.
Parametr `ile_dni` określa, ważność pliku.

Jeśli istnieje plik, który nie jest przeterminowany, do wyjścia zostanie zwrócona zawartość pliku a dalsza część skryptu jest przerywana (`break`).

Jeśli plik jest już przeterminowany lub go nie ma, komendy w obrębie skryptu zostaną wykonane, zapisane w pliku `.shtml` oraz zwrócone do wyjścia. Jeśli którakolwiek komenda w obrębie skryptu zakończy się błędem, nie nastąpi zapis wyniku do pliku `.shtml`

Wszystkie parametry mogą być zmiennymi lub stałymi tekstowymi.

Komenda `##cache…##` nie może być używana wraz z komendą `##stream…##`, bowiem wykorzystują one obydwie zmienną `@buffer`. Przed wywołaniem komendy `##cache..##`, obiekt ***stream*** musi być ustawiony domyślnie, czyli na wartość `R`

Komenda `##cache…##` w znakomity sposób zwiększa wydajność i zmniejsza obciążenie SQL Serwera.
Należy pamiętać, że stosowanie komendy `##cache..##` powinno być stosowane tylko tam, gdzie nie jest istotny kontekst odpowiedzi lub gdzie odpowiedź jest powtarzalna dla podanych parametrów.

Mechanizm ***cache*** można wyłączyć globalnie usuwając parametr `cache` w sekcji `[db]` pliku konfiguracyjnego.

Komenda: cacheXML

`##cacheXML~@ile_dni~@zmienna~@nazwa_pliku~@par1~@par2~…~@parN##`
Do zmiennej `@nazwa_pliku` (parametr wyjściowy komendy) wpisana zostaje nazwa zmiennej pliku XML o postaci: `M_folderCache\wartośćPar1~wartośćPar2~…~wartośćParN`

Jeśli istnieje plik XML, który nie jest przeterminowany, zmienna `@zmienna` zostanie ustawiona na zawartość tego pliku a dalsza część skryptu zostanie przerwana (`break`).

Jeśli plik XML nie istnieje lub jest przeterminowany, dalsza część skryptu zostanie normalnie wykonana.
Na końcu skryptu należy zadbać o zapisanie wyniku XML do pliku o nazwie przechowywanej z zmiennej `@nazwa_pliku`, np.:
```
##cacheXML~0~@z~@plik~tekst~@par1~@par2##
…
…
##set~@@plik~@z##
```

Wszystkie parametry mogą być zmiennymi lub stałymi tekstowymi.
Mechanizm ***cacheXML*** można wyłączyć globalnie usuwając parametr `cache` w sekcji `[db]` pliku konfiguracyjnego.

Komenda: clearCache

`##clearCache~par1~par2~…~parN##`
Usuwa plik shtml `cache` o nazwie złożonej z wartości podanych parametrów (patrz opis komendy `##cache…##`).

`##clearCache##`
Usuwa bezwarunkowo wszystkie pliki shtml `cache`.

Komenda: clearCacheXML

`##clearCacheXML~par1~par2~…~parN##`
Usuwa plik xml `cache` o nazwie złożonej z wartości podanych parametrów (patrz opis komendy `##cache…##`).

`##clearCacheXML##`
Usuwa bezwarunkowo wszystkie pliki xml `cache`.

Komenda: oldCache

`##clearCache~dni##`
Usuwa wszystkie pliki SHTML `cache` utworzone przed podaną liczbą dni.
Parametr dni jest opcjonalny. Jeśli zostanie pominięty lub podana liczba dni = 0, usunięte zostaną wszystkie pliki SHTML cache.

Komenda: oldCacheXML

`##clearCacheXML~dni##`
Usuwa wszystkie pliki xml `cache` utworzone przed podaną liczbą dni.
Parametr dni jest opcjonalny. Jeśli zostanie pominięty lub podana liczba dni = 0, usunięte zostaną wszystkie pliki XML cache.

Komendy: SQL

sqlForXML (lub **X**) – zwraca wynik XML polecenia SQL Server z dyrektywą FOR XML poddany transformacji XSLT
`##sqlForXML~skryptSqlForXML~root~parametr1~parametr2...~parametrN~nazwaPlikuXSL##`

sqlString - zwraca tabelę w formacie tekstowym bez nagłówków kolumn oraz separatorów kolumn i wierszy
`##sqlString~skryptSQL~@count~parametr1~parametr2...~parametrN##`

sqlTab - zwraca tabelę w formacie tekstowym bez nagłówków kolumn, jako separatory kolumn zostaną użyte znaki TAB a jako separatory wierszy CRLF
`##sqlTab~skryptSQL~@count~parametr1~parametr2...~parametrN##`

sqlAdoXML - zwraca tabelę w formacie ADO XML poddaną transformacji XSLT
`##sqlAdoXML~skryptSQL~@count~parametr1~parametr2...~parametrN~nazwaPlikuXSL##`

sqlTD - zwraca tabelę w postaci html <table> bez nagłówków
`##sqlTD~skryptSQL~@count~parametr1~parametr2...~parametrN##`

sqlTH - zwraca tabelę w postaci html <table> z nagłówkami kolumn <th> o nazwach zgodnych z nazwami kolumn tabeli
`##sqlTH~skryptSQL~@count~parametr1~parametr2...~parametrN##`

sqlArray - zapisuje zawartość tabeli w zmiennej tabelarycznej `@zmienna`
`##sqlArray~skryptSQL~@count~parametr1~parametr2...~parametrN~@zmienna##`

sqlData - wykonuje polecenie SQL i zwraca do wyjścia wartość parametru o podanym numerze (od 1)
`##sqlData~skryptSQL~nr_parametru~parametr1~parametr2...~parametrN##`

sqlExecute (lub **E**)- wykonuje polecenie SQL i nie zwraca niczego do wyjścia
`##sqlExecute~skryptSQL~@count~parametr1~parametr2...~parametrN##`

sqlStream (lub **Q**) wykonuje polecenie SQL zawierające dyrektywy FOR JSON lub FOR XML i zwraca wynik do wyjścia
`##sqlStream~skryptSQL~@zmienna~parametr1~parametr2...~parametrN##`

Wszystkie komendy SQL zwracają do wyjścia wynik polecenia `SELECT` z wyjątkiem komendy `sqlExecute`, która przeznaczona jest dla poleceń `INSERT`, `DELETE`, `UPDATE` oraz `sqlArray`, która zwraca wynik do zmiennej tabelarycznej.

Jeśli wyrażenie `skryptSQL` jest zmienną, to zakończenie jej znakiem **$** powoduje, że wynikowy html nie będzie przetwarzany, `np.: ##sqlTH~@X_indeksy$~@count~@kodtowaru##`

W przeciwnym wypadku wynikowy html zostanie przetworzony, czyli zawarte w nim komendy zostaną wykonane, np.: `##sqlTH~@X_indeksy~@count~@kodtowaru##`

Zamiast podawania zmiennej @X_ określającą nazwę pliku SQL, można wpisać definicję komendy SQL wprost:
`##sqlTH~select top(10) * from produkty--$~##`
Podwójny myślnik `--` stanowi komentarz języka *T-SQL*, zatem znak **$** nie zostanie przetworzony przez SQL Server. Znak **$** jest jednak wskazówką aby nie przetwarzać przez aparat skryptu wynikowego html.

Do zmiennej `@count` *`(write)`* zostanie wpisana ilość zwróconych (lub przetworzonych) wierszy.
Zastosowanie zmiennej jest opcjonalne:
`##sqlString~nazwaPlikuSQL~~parametr1~parametr2 ... ~parametrN##`

Komenda `##sqlForXML…##` nie zwraca ilości wierszy do zmiennej `@count`. W miejsce zmiennej `@count` należy wpisać nazwę węzła głównego (korzeń XML):
`##sqlForXML~SELECT … FOR XML AUTO~root~plikiXSL/plik1##`

Jeśli parametr `root` będzie pusty, należy zadbać aby w poleceniu T-SQL została dodana dyrektywa `ROOT('root_name')` lub `ROOT` bez argumentu do polecenia T-SQL, np.:
`##sqlForXML~SELECT … FOR XML AUTO, ROOT~~plikiXSL/plik1##`

Parametr `nazwaPlikuXSL` można przekazać jako pusty. Wówczas będzie poszukiwany plik o nazwie zgodnej z nazwą pliku SQL, np.:
`##sqlForXML~skryptSqlForXML~root~parametr1~parametr2...~parametrN~##`
Do przetworzenia wynikowego XML zostanie zastosowany plik `skryptSqlForXML.xsl`, o ile istnieje.

Komenda `##sqlStream…##` zwraca wynik do zmiennej. Jeśli nazwa zmiennej nie zostanie podana, wynik zwracany jest do wyjścia:
`##sqlStream~skryptSQL~~parametr1~parametr2...~parametrN##`
lub bez parametrów:
`##sqlStream~skryptSQL~##`

Dyrektywa `FOR JSON` możliwa jest tylko w SQL wykonujących się na instancjach SQL Server począwszy od wersji 2016.

Parametry wywołań SQL:

Jako parametr można użyć stałych tekstowych albo nazw zmiennych.
Zarówno nazwa parametru, wartość tekstowa, jak i nazwa pliku nie może zawierać znaku `~` *tylda*.
Wartość pusta zamieniana jest na `NULL`.

Parametry są przekazywane jako typ `SQL: nvarchar(k), ADO: adVarWChar(k)`
W zależności od długości tekstu `k`:
`k < 20` to `SQL: nvarchar(20), ADO: adVarWChar(20)`
`k` w zakresie `20` do `4000` to `SQL: nvarchar(k), ADO: adVarWChar(k)`
`k > 4000` to `SQL: nvarchar(max), ADO: adVarWChar(-1)`

Jeśli nazwa zmiennej zostanie poprzedzona podkreślnikiem _ to parametr taki stanie się parametrem wyjściowym polecenia SQL: `_@N` to parametr wyjściowy, natomiast: `@N` to parametr wejściowy

Jako parametry wyjściowe mogą być stosowane tylko zmienne *read/write.*
Typem parametru wyjściowego jest zawsze `SQL: nvarchar(max), ADO: adVarWChar(300000)`
Parametry wejściowe jak i wyjściowe mogą także przekazywać dane typu XML.

W wyniku wykonania polecenia SQL, zmienna otrzymuje wartość przypisaną z parametru wyjściowego:
`##sqltd~$X_nazwaPlikuSQL$~~parametr1~_@C_user ... ~parametrN##`
ustawi *cookies* o nazwie `user` wartością drugiego parametru polecenia SQL.

`##sqltd~@X_nazwaPlikuSQL$~~parametr1~_@P_parameters_lastUser ... ~parametrN##`
ustawi wartość parametru `lastUser` sekcji `[parameters]` w pliku konfiguracyjnym.

Parametr wyjściowy o wartości *`NULL`* zostanie zamieniony na pusty ciąg tekstowy przed przypisaniem do zmiennej.

Dla komendy `##sqlData…##` zamiast wpisywania nazwy parametru wyjściowego zwracanego do wyjścia, można zastosować sam podkreślnik. Wówczas wynik nie zostanie zapisany w zmiennej, lecz trafi tylko do wyjścia:
`<div>liczba produktów: ##sqlData~select ? = count(*) from Magazyn--$~1~_##</div>`

Parametry wejściowe i wyjściowe należy zastąpić znakami pytajnika `?`.
Kolejność pytajników odpowiada kolejności parametrów.

Przykład 1 (jeden parametr wejściowy)
```
##sqlth~SELECT KodKontrahenta, Adres, FROM dbo.Kontrahenci
WHERE KodKontrahenta = isnull(?, '')--$
~~@c_user##
```

Przykład 2 (nazwa pliku: `kontrahenci.sql`, dwa parametry wejściowe)
```
SELECT KodKontrahenta, Adres, Telefon FROM dbo.Kontrahenci WHERE Miejscowość = ? AND Grupa = ?
```
Komenda: `##sqlth~@X_kontrahenci$~~Warszawa~ODBIORCY##`

Przykład 3 (nazwa pliku: `kontrahenci.sql`, dwa parametry wejściowe oraz jeden wyjściowy)
```
SELECT KodKontrahenta, Adres, Telefon
FROM dbo.Kontrahenci WHERE Miejscowość = ? AND Grupa = ?; SET ? = Getdate();
```

Komenda: `##sqlth~@X_kontrahenci$~~Warszawa~ODBIORCY~_@P_parameters_lastTime##`
Ustawia parametr lastTime w pliku konfiguracyjnym `.ini` (sekcja `[parameters]`)

Przykład 4 (jeden parametr wejściowy)
```
##sqlAdoXML~SELECT top(20) KodTowaru, Nazwa, Producent, Cena
FROM Magazyn WHERE KodTowaru LIKE ? + '%'
ORDER BY KodTowaru--$~~@f_kod~xsl\towary##
```
Utworzy wynik transformacji XSLT na podstawie pliku: `xsl\towary.xsl` z podanego wprost polecenia SQL, pobierając parametr z formularza html z pola `<input name="kod">`.

Dla komend **`sqlForXML`** oraz **`sqlAdoXML`** zamiast nazwy pliku XSL można zastosować parametr wyjściowy:
`##sqlForXML~root~@par2~@par2~_@parOut##`
W wyniku wykonania komendy do zmiennej `@parOut` zostanie wstawiony wynikowy XML a do wyjścia nic nie zostanie zwrócone (transformacja XSLT nie zostanie wykonana). XML przechowany w zmiennej można potem poddać transformacji XSLT za pomocą komendy **`##transform~…##`**, która daje możliwość ustawienia parametrów w arkuszu XSL.

Dla komend **`sqlForXML`** oraz **`sqlAdoXML`** można przypisać parametr transformacji zamiast parametru SQL.
W tym celu na końcu nazwy zmiennej należy dodać znak **^** , na przykład:
`##sqlForXML~@x_widok1~~@a~_@b~@Tytuł^~##`
Wartość zmiennej `@Tytuł` nie zostanie przekazana do plecenia SQL. Polecenie SQL otrzyma jeden parametr wejściowy o wartości odczytanej ze zmiennej `@a` oraz jeden parametr wyjściowy zwrócony do zmiennej `@b`.
Natomiast transformacja XSLT otrzyma parametr o nazwie `Tytuł` o wartości odczytanej ze zmiennej `@Tytuł`

Komenda: transform (lub T) (transformacja XSLT)
`##transform~@XML~nazwa_pliku_XSL~par1~@value1~par2~@value2~...~parN~@valueN##`
Wykonuje transformację XSLT na zawartości zmiennej `@XML`
Parametry `parX` oraz ich wartości `@valueX` są opcjonalne, ale muszą występować parami.

Aby parametry mogły zostać wykorzystane, muszą być zdefiniowane w pliku XSL:
`<xsl:param name="user"/>`

Nazwy parametrów `parX` muszą być tekstami (nie można użyć zmiennych). Wartości mogą być dowolnymi wyrażeniami.
Parametr komendy: `nazwa_pliku_XSL` musi być tekstem. Można na końcu nazwy pliku dodać znak $. Dzięki temu wynikowy HTML nie zostanie poddany przetwarzaniu przez aparat skryptu, tylko od razu zwrócony do wyjścia.

Komenda: ftSplit (fulltext search split)
`##ftSplit~wyrażenie~czy_początki~@zmienna1~@zmienna2~...~@zmiennaN##`
Rozkłada wartość argumentu `wyrażenie` na poszczególne słowa pomijając nadmiarowe spacje i wstawia kolejne słowa do zmiennych. Minimalna liczba zmiennych to 1.
Jeśli atrybut `czy_początki = 1` to do zmiennych trafia zapis w formacie `"słowo*"`
Zarówno `wyrażenie` jak i atrybut `czy_początki` mogą być zmiennymi.

Komenda: com (lub C) (ActiveX COM)
`##com~nazwaSkładnika.nazwaKlasy~nazwaMetody~argument0~argument1...~argumentN##`

Tworzy instancję klasy `nazwaKlasy` składnika ActiveX COM i wywołuje metodę publiczną (`sub`) tej klasy.
Metoda przyjmuje tylko jeden parametr - referencję do tablicy (`string`) wypełnioną wartościami poszczególnych argumentów. Jako argumenty można podać nazwy dowolnych zmiennych.
Jeśli nazwa zmiennej zostanie poprzedzona znakiem podkreślnika _ to parametr staje się parametrem wyjściowym komendy (podobnie jak w komendach SQL). Wszystkie parametry komendy mogą być zmiennymi.

```
##com~mojDodatek.klasa1~metoda1~_@a~burek~@b##
```

1.Zostanie utworzona tablica `string` o rozmiarze `0-2 i wypełniona następująco:`
`Tablica(1)` = *burek*
`Tablica(2)` = *wartość odczytana ze zmiennej @b*
2. Zostanie wywołana `metoda1` klasy `klasa1` z argumentem `Tablica()`
3. Do zmiennej `@a` zostanie przypisana zawartość `Tablica(0)`

Komenda: copyQC

```
##copyQC~q1~q2~…~qn##
```

Kopiuje wartości listy wybranych parametrów *query url* do plików *cookies* o tych samych nazwach.
Jeśli brak wybranego parametru *query url* w przetwarzanym żądaniu lub parametr ten jest pusty, plik *cookies* zostanie skasowany.
Jako nazwy parametrów q muszą zostać podane teksty.
Utworzone pliki *cookies* zachowują ważność tylko w obrębie sesji przeglądarki.

Komenda: copyQS

```
##copyQC~q1~q2~…~qn##
```

Kopiuje wartości listy wybranych parametrów *query url* do *zmiennych sesji* o tych samych nazwach.
Jeśli brak wybranego parametru *query url* w przetwarzanym żądaniu lub parametr ten jest pusty, *zmienna sesji* zostanie skasowana. Jako nazwy parametrów q muszą zostać podane teksty.

Komenda: clearC

```
##clearC##
```

kasuje wszystkie *cookies*

```
##clearC~c1~c2~c3~…~cn##
```

kasuje wybrane cookies

Na przykład:
`##clearC~user~productID##`
skasuje *cookies* o nazwach: `user` oraz `productID`
Jako nazwy cookies należy podać tekst (nie można użyć zmiennych).

Komenda: clearS

```
##clearS##
```

kasuje wszystkie zmienne obiektu `Session`.

```
##clearS~s1~s2~s3~…~sn##
```

kasuje wybrane zmienne obiektu `Session`.
Na przykład:
`##clearS~user~productID##`
skasuje zmienne obiektu *session* o nazwach: `user` oraz `productID`
Jako nazwy zmiennych obiektu session należy podać tekst (nie można użyć zmiennych).

Komenda: varSet

```
##varSet~@type~@name~@value~@days##
```

Tworzy zmienną o podanym typie oraz nazwie i ustawia wartość zmiennej na wartość wyrażenia:
`##varSet~C~user~reksio~30##`
jest ekwiwalentem dla:
`##set~@C_user#30~reksio##`

Parametr `@days` jest opcjonalny i powinien być używany tylko dla zmiennych typu `C` (*cookiess*):
`##varSet~C~user~reksio~##`
lub:
`##varSet~C~user~reksio##`

Wszystkie parametry mogą być zmiennymi bądź tekstami.
Komendą `varSet` można tworzyć tylko zmienne *read/write* takie jak: `@C_ @S_ @P_ @M_` oraz zmienne lokalne, z wyjątkiem zmiennych specjalnych.

Dla zmiennych lokalnych parametr `@type` musi być pusty:
`##varSet~~user~burek##`
jest ekwiwalentem:
`##set~@user~burek##`

Komenda: varVal

`##varVal~@type~@name~@N##`
Wstawia do zmiennej `@N` wartość zmiennej typu `@type` o nazwie `@name`:
`##set~@c_user~burek##`
`##varVal~C~user~@N##`
`##read~@N##`
zwróci słowo *burek*, tak samo jak:
`##set~@c_user~burek##`
`##read~@c_user##`

Wszystkie parametry mogą być zmiennymi bądź tekstami.

Dla zmiennych lokalnych parametr `@type` musi mieć pustą wartość:
`##varVal~~user~@N##`
wstawi do zmiennej `@N` wartość zmiennej lokalnej `@user`

Komendy: scriptVB, scriptJS

`##scriptVB~tekst_skryptu~par0~par1…~parN-1##`
`##scriptJS~tekst_skryptu~par0~par1…~parN-1##`

Uruchamia skrypt w języku **VBScript** lub **JScript**.

Parametry parX przekazywane są jako referencja w tablicy t() indeksowanej od 0.

Na przykład:
```
##set~@b~2##
##scriptvb~
t(0)=sqr(t(1))
~@a~@b##
##read~@a##
```
zwróci do wyjścia pierwiastek z dwóch *1,4142135623731*

Aparat skryptu ma dostęp do obiektów: Server, Response, Request, Session oraz Connection (ADODB.Connection).

Na przykład:
```
##scriptvb~
t(0) = Request.ServerVariables("SCRIPT_NAME")
~@a##
##read~@a##
```

Komendy: procVB, procJS

`##procVB~tekst_definicji_procedury##`
`##procJS~tekst_definicji_procedury##`
Tworzy procedurę `Sub` lub `Function` w aparacie skryptu.

Aparat skryptu jest tworzony na czas przetwarzania pojedynczego żądania.
Utworzone procedury i funkcje są dostępne do zakończenia żądania.

Procedury i funkcje utworzone komendą `##procVB~…##` oraz `##procJS~…##` nie są uruchamiane.
Można je wywoływać za ze skryptów *VBScript* oraz *JScript* oraz z innych procedur i funkcji aparatu skryptu.

Jeśli pierwsza z procedur jest utworzona na przykład w języku VBScript, pozostałe też muszą być w tym języku tworzone. Po wykonaniu komendy `##scriptReset##` można kolejne procedury pisać w innym języku.

Przykład:

```
##procVB~
Function Silnia(i)
If i < 2 THEN
  Silnia = 1
Else
  Silnia = i * Silnia(i-1)
End If
End Function##
##set~@a~5##
##scriptVB~t(0) = Silnia(cint(t(0)))~@a##
##read~@a##
```

Komenda: scriptReset

`##scriptReset##`
Usuwa wszystkie procedury ze skryptów VBScript oraz JScript.

Komenda: connect

##connect##
Tworzy połączenie z bazą danych SQL. Stosowanie jej jest konieczne tylko jeśli skrypt VB lub JS będzie korzystał z obiektu Connection.

Połączenie z bazą danych wywoływane jest automatycznie wraz z komendami, które takiego połączenia niezbędnie wymagają (np. `##sqlForXml…##`). Nie jest konieczne zatem konieczność jawne wywoływanie komendy `##connect##`. Połączenie z bazą danych jest zamykane dopiero na koniec przetwarzania całego wywołania `.asp`.

Jednakże komenda `scriptVB` oraz `scriptJS` ma także dostęp do obiektu `Connection`, chociaż nie musi z niego korzystać. Dlatego jeśli skrypt VB lub JS korzysta z obiektu `Connection` a przed komendą `##scriptVB…##` lub `##scriptJS…##` w zakresie żądania .asp nie była używana jakakolwiek inna komenda odwołująca się do bazy danych, należy użyć komendy `##connect##`.

Komenda: function

`##function~@zmienna~argument~funkcja1~funkcja2~...~funkcjaN##`
Do zmiennej `@zmienna` zapisywany jest wynik wykonania kolejnych `funkcji` z `argumentu`.
Wszystkie parametry (argument i funkcje) mogą być zmiennymi.

```
##function~@zmienna~ala ma kota~ucase~alfanum##
##read~@zmienna##
```
zwróci wynik: `ALAMAKOTA`

Jeśli funkcja wymaga własnych parametrów, przekazywane są wówczas jako kolejne parametry komendy:
`##function~@zmienna~ala ma kota~mid~8~3~ucase## ##read~@zmienna##`
zwróci wynik: `KOT`

Dostępne funkcje:

- `len`
- `left~ile_znaków`
- `right~ile_znaków`
- `mid~od_znaku~ile_znaków`
- `ltrim`
- `rtrim`
- `trim`
- `ucase`
- `lcase`
- `normalize` – usuwa wszystkie spacje: nadmiarowe, początkowe i końcowe oraz białe znaki (CR, LF, TAB, char160)
- `alfanum` – usuwa wszystkie znaki oprócz liter (także polskich) i cyfr
- `append~wyrażenie` – dołącza wartość wyrażenia
- `split~separator~indeks` – zwraca element o adresie `indeks` z tablicy uzyskanej funkcją split, np.: `##function~@a~ala ma kota~split~ ~1## ##read~@a##` zwróci: `ma`
- `minus` – jeśli argument może zostać zinterpretowany jako liczba, zwraca wartość pomnożoną razy -1
- `+~składnik` – dodaje składnik (argument i składnik muszą być liczbami)
- `-~odjemnik` – odejmuje odjemnik (argument i odjemnik muszą być liczbami)
- `*~mnożnik` – mnoży przez mnożnik (argument i mnożnik muszą być liczbami)
- `/~dzielnik` – dzieli przez dzielnik (argument i dzielnik muszą być liczbami)
- `int` – jeśli argument jest datą lub liczbą zwraca wartość całkowitą z argumentu
- `bool` – zwraca `1` jeśli argumentem będzie: `1, -1, true, yes, on, tak`, jeśli nie to: `0`
- `isnumeric` – zwraca `1` jeśli argument może zostać zinterpretowany jako liczba, lub `0` jeśli nie
- `format~łańcuch_formatujący` – zwraca argument tekstowy sformatowany
  np. `##setText~@f~@@@\-@@@\-@@@####function~@w~123456789~format~@f####read~@w##` zwróci `123-456-789`
- `formatDate~łańcuch_formatujący` - konwertuje argument na datę i formatuje
- `formatNumeric~łańcuch_formatujący` - konwertuje argument na liczbę i formatuje
- `dateAdd~symbol_okresu~ilość_okresów` – konwertuje argument na datę i dodaje określoną ilość okresów, wynik jest w postaci *`yyyy-mm-dd hh:Nn:Ss`*
- `replace~szukany_tekst~zastępujący_tekst`
- `MD5` – zwraca skrót MD5 z argumentu
- `MD5UTF8` – zwraca skrót MD5 z argumentu zakodowanego jako UTF-8
- `URL` – koduje jako url encode
- `HTML` – koduje jako HTML encode

- save~@zmienna2 – * zapisuje wynik pośredni w @zmienna2
- read – zwraca do wyjścia wynik pośredni

* Możliwe jest odwołanie się do wyniku pośredniego przechowywanego w @zmienna oraz zapisanie do innej zmiennej wyniku pośredniego:
##function~@a~6~*~@a~+~4~save~@b~*~@a~read## ##read~@b##
zwróci: *1600 40*

Komenda: regexp

##regexp~sourceString~pattern~wyrażenie1~wyrażenie2##
Wykonywana jest metoda Test obiektu RegExp.
Jeśli Test zwróci True, przetwarzane jest wyrażenie1, w przeciwnym wypadku wyrażenie2.
##regexp~@f_email~^\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*$~~zły adres##

Jeśli wyrażenie1 lub wyrażenie2 jest zmienną i nazwa jej zakończona jest znakiem **$**, zawartość zmiennej nie zostanie przetwarzana tylko wprost zwrócona do wyjścia:
##regexp~@f_email~^\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*$~~@H_komunikaty/zlyEmail$##

Komenda: md5

##md5~@wyrażenie~@zmienna##
Ustawia zmienną na skrót MD5 z wartości wyrażenia.

Komenda: upload

##upload##
Pobiera dane z formularza: method="post" encType="multipart/form-data"
Przy tak podanym jak powyżej typie kodowania formularza, komenda musi zostać wywołana przed komendami fileSave oraz przed odwołaniem do zmiennych @f_. W jednym żądaniu ASP nie można użyć jednocześnie komendy upload oraz odwołania do zmiennej @body.

Ważne!
Należy skonfigurować IIS na maksymalną ilość otrzymywanych danych:
ASP -> Właściwości limitów -> Maksymalny limit treści jednostki żądania

Komenda: fileSave

##fileSave~fieldName~contentFilter~maxLen~path~maxX~maxY
~@contentType~@fileName~@oryginalFileName~@errorInt~@errorText##

Zapisuje plik pobrany z formularza: method="post" encType="multipart/form-data"
Przed wywołaniem komendy należy obowiązkowo wywołać komendę upload
Plik bmp zostanie automatycznie skonwertowany na plik JPG.

Przykładowo:
##fileSave~plik1~image/jpeg~200000~pobrane~600~400~~@fileName##
przeskaluje pobrany plik JPG do wymiarów maksymalnych: 600x400 pixeli i zapisze w podkatalogu pobrane, zmienna @fileName zwróci unikatową nazwę pliku (GUID), pod jaką plik zostanie zapisany.
Jeśli rozmiar pobieranego pliku przekroczy 200000 bajtów, zapis nie nastąpi.
Jeśli użytkownik pobierze inny typ pliku niż JPG, zapis pliku również nie nastąpi.
Jeśli zapis pliku nie nastąpi zmienna @fileName nie zostanie ustawiona.

<u>Parametry wejściowe obowiązkowe – stałe tekstowe:</u>

fieldName – definiuje nazwę pola w formularzu html, z którego nastąpił zapis pliku

contentFilter – może opcjonalnie zawierać listę akceptowalnych typów przesyłanej informacji,
np.: image/jpeg image/gif - jako separator typów można wpisać spację

maxLen – maksymalny rozmiar w bajtach, 0 lub pusty tekst – bez ograniczenia rozmiaru pliku

path – określa ścieżkę do folderu podrzędnego, w którym ma zostać zapisany plik, jeśli zostanie podany pusty tekst, zapis nastąpi w folderze żądania asp

maxX, maxY – maksymalne wymiary dla przeskalowania JPG, proporcje obrazu zostaną zachowane niezależnie od podanych wymiarów, podanie 0 lub pustego tekstu – przeskalowanie nie nastąpi

Parametry wyjściowe opcjonalne - zmienne

`@contentType` – zwraca typ zawartości przesyłanego pliku, np. `application/octet-stream`, `text/plan`, `image/png` itd.

`@fileName` – zwraca nazwę GUID, pod jaką plik zostanie zapisanych na dysku, jeśli zostanie zwrócona pusta wartość to plik nie został zapisany.

`@oryginalFileName` – zwraca oryginalną nazwę pliku

`@errorInt` – przy błędzie zwraca numer błędu, jeśli błąd nie wystąpi, zwraca pustą zawartość

`@errorText` – przy błędzie zwraca opis błędu, jeśli błąd nie wystąpi, zwraca pustą zawartość

Komenda: array

`##array~@N~`**`OPERACJA`**`~arg1~arg2~...argN##`
Operacje na tablicach, gdzie **`OPERACJA`** to litery: `C`, `W`, `R`, `V`, `D`, `L`, `J`, `S` lub `T`

Tablice (wyłącznie prostokątne) mogą być zapisywane w zmiennych read/write,
z wyjątkiem zmiennych typu @P_
Każdy wiersz tablicy musi zawierać tę samą liczbę kolumn.
Adresowanie wiersza odbywa się za pomocą adresu tekstowego lub indeksu numerycznego (adresowanie od `0`).
Adresowanie kolumny odbywa się zawsze za pomocą indeksu numerycznego (adresowanie od `0`).
Jako wyszukiwany adres tekstowy traktowana jest zawartość pierwszej kolumny (Indeks = `0`)
Jeśli jako adres wiersza poda się wartość numeryczną (liczba całkowita), adres będzie dotyczył wiersza o podanym numerze.
Jeśli jako adres wiersza poda się wartość tekstową, adres będzie dotyczył wiersza pierwszej zgodnej wartości w kolumnie `0`.
Komenda nie zwraca zawartości do wyniku html z wyjątkiem operacji `C` *read Cell* w sytuacji, jeśli nie ma ostatniego parametru.

Operacja: C (read Cell)
`##array~@N~C~adresWiersza~numerKolumny~@N2##`
Odczytuje wartość komórki ze zmiennej `@N` i wstawia ją do zmiennej `@N2`
Jeśli poda się nieistniejący `adresWiersza` lub `numerKolumny` zmienna `@N2` nie zostanie ustawiona.
Zamiast parametrów `adresWiersza` oraz `numerKolumny` można użyć zmiennych:
`##array~@N~C~@W~@K~@N2##`

Jeśli nie będzie ostatniego parametru:
`##array~@N~C~adresWiersza~numerKolumny##`
zawartość komórki zostanie zwrócona do wyjścia

Operacja: W (Write cell)
`##array~@N~W~adresWiersza~numerKolumny~wyrażenie##`
wstawia wartośc `wyrażenia` do wybranej komórki zmiennej `@N`.
Parametr `adresWiersza` musi wskazywać na istniejący wiersz tablicy.
Parametr `numerKolumny` musi wskazywać na istniejącą kolumnę tablicy.
Zamiast parametrów `adresWiersza` oraz `numerKolumny` można użyć zmiennych:
`##array~@N~W~@W~@K~@N2##`

Operacja: R (write Row)
`##array~@N~R~adresWiersza~wartość1~wartość2...~wartośćN##`
Jeśli `addresWiersza` będzie pusty, nowy wiersz zostanie dodany przed istniejące wiersze.
Jeśli `addresWiersza` będzie równy `-1`, nowy wiersz zostanie dodany za istniejące wiersze.
Jeśli `addresWiersza` będzie istniejącym adresem wiersza, wiersz zostanie zastąpiony nowym.
Nowy wiersz tworzony jest z podanej listy wartości.
Zamiast wartości tekstowych można używać zmiennych:
`##U~@N~R~@adresWiersza~@N1~@N2...~@N3##`

Operacja: V (Variables set)
`##array~@N~V~adresWiersza~@zmienna0~@zmienna1...~@zmiennaN##`
Odczytuje pojedynczy wiersz ze zmiennej tabelarycznej i przypisuje zawartość kolejnych komórek wiersza do poszczególnych zmiennych. Jeżeli jako `addresWiersza` zostanie podane -1 odczytany zostanie wiersz ostatni.

Nazwy zmiennych mogą zostać opuszczone:
`##array~@N~V~7~@a~@b~~@e##`

odczyta ze zmiennej tabelarycznej `@n` ósmy wiersz i wstawi zawartość komórek z kolumn: `0, 1, 3` do zmiennych: `@a, @b, @e`

Operacja: D (Delete row)
`##array~@N~D~adresWiersza##`
Usuwa wybrany wiersz.
Jeśli `adresWiersza = -1` to usuwany jest wiersz ostatni.

Operacja: L (Length)
`##array~@N~L~@N1##`
Wstawia do zmiennej `@N1` liczbę wierszy tablicy `@N`

Operacja może być także stosowana do zwykłych zmiennych.
Jeśli wartość zmiennej jest pusta to wynikiem będzie `0`.
Jeśli zmienna posiada niepustą wartość, wynikiem będzie `1`.

`##array~@N~L~##`
Jeśli nie zostanie podana nazwa zmiennej, wynik zostanie zwrócony do wyjścia, dla pustej zmiennej `@N` zostanie zwrócone `0` (zero).

`##array~@N~L##`
Jeśli operacja będzie posiadała tylko 2 argumenty, wynik także zostanie zwrócony do wyjścia, lecz dla pustej zmiennej `@N`, zwrócony zostanie pusty łańcuch tekstowy.

Operacja: J (Join)
`##array~@N~J~row_separator~col_separator~@zmienna##`
Zapisuje w zmiennej `@zmienna` zawartość zmiennej tabelarycznej `@N` przy użyciu wskazanych `separatorów`.
Separatory nie mogą być zmiennymi.

Operacja: S (Split)
`##array~@N~S~row_separator~col_separator~wyrażenie##`
Wstawia do zmiennej tabelarycznej tabelę pobraną z wyrażenia.
Jeśli `row_separator` będzie pusty, jako separator wiersza zostanie przyjęta sekwencja `CRLF`
Jeśli `col_separator` będzie pusty, jako separator kolumn zostanie przyjęty znak `TAB`
Separatory nie mogą być zmiennymi.

Operacja: T (Table SQL)
`##array~@N~T~nazwaTabeliTymczasowej~@count##`
Pobiera tablicę ze zmiennej `@N` i tworzy z niej tabelę tymczasową SQL o nazwie `#nazwaTabeliTymczasowej` w sesji SQL Server dla pojedynczego żądania HTTP.
Jeżeli tabela tymczasowa o podanej nazwie już istnieje, zostanie najpierw usunięta.
Po zakończeniu obsługi żądania, tabela tymczasowa jest automatycznie usuwana przez SQL Server.

Liczbę wierszy tabeli tymczasowej można odczytać ze zmiennej `@count`:
`##array~@N~T~koszyk~@ilePozycjiMaKoszyk##`
Zmienna `@count` jest opcjonalna: `##array~@N~T~nazwaTabeli~##`

Kolumny tabeli tymczasowej posiadają nazwy `col0`, `col1` ..., `colN`.
Ilość kolumn tabeli będzie ilością kolumn w wierszach tablicy.

Zamiast parametru `nazwaTabeli` można użyć zmiennej:
`##array~@N~T~@N2~##`

Komenda: forEach

`##forEach~@N~col~@zmienna~wyrażenie##`
Wykonuję pętlę po wszystkich wierszach zmiennej tabelarycznej `@N` wstawiając do `@zmienna` zawartości komórki bieżącego wiersza z kolumny `col` i przetwarzając za każdym razem `wyrażenie`.
Wszystkie parametry mogą być zmiennymi.

Jeśli `col = -1` to do `@zmiennej` zostanie wstawiony zamiast zawartości kolumny, kolejny adres wiersza:
`##forEach~@N~-1~@zmienna~wyrażenie##`
`@zmienna` będzie przybierać wartości od `0` do `ilości wierszy - 1`

Jeśli zmienna tabelaryczna `@N` będzie pusta lub nie zostanie podana jej nazwa, do `@zmiennej` zostaną wstawione kolejne liczby od `0` do wartości parametru `col - 1`
`##forEach~~3~@zmienna~wyrażenie##`
`@zmienna` będzie przybierać wartości od `0` do `2`

Parametr `@zmienna` jest także opcjonalny:
`##forEach~~3~~wyrażenie##`
`wyrażenie` zostanie przetworzone `3` razy

Komenda: split

`##split~wyrażenie~separator~@zmienna~@count##`
Tworzy zmienną tabelaryczną z wartości wyrażenia.

Jako separator musi być zastosowany tekst.
Jeśli separatorem będzie pusty łańcuch tekstowy, wartość wyrażenia zostanie rozłożona na znaki.
Do zmiennej `@count` zostanie wpisana ilość wierszy zmiennej tabelarycznej.

Zastosowanie zmiennej @count jest opcjonalne:
`##split~wyrażenie~separator~@zmienna~##`

Komenda: log

`##log~nazwa_pliku~wyrażenie1~wyrażenie2~…~wyrażenieN##`
Dopisuje do pliku `nazwa_pliku.log` (*kodowanie UTF-8*) bieżącą datę i wartości kolejnych wyrażeń, rozdzielając je znakiem tabulacji. Jeśli plik nie istniał wcześniej, zostanie utworzony.

Komenda: debug (lub **D**)

`##debug~@zmienna1~@zmienna2~...~@zmiennaN##`
Zapisuje do pliku `errorLog.log` (*kodowanie UTF-8*) bieżącą wartość zmiennych.

Czas wykonania

Jeśli w pliku konfiguracyjnym w sekcji `[db]` zostanie ustawiony parametr `timeLog=1`, będzie można śledzić czas wykonania (wydajność) poszczególnych wykonywanych komend oraz łączny czas przetwarzania żądania. Kolejne zapisy tworzone są dopiero po wykonaniu komendy lub żądania. W związku z tym taka właśnie chronologia zostanie utrwalona w pliku `timeLog.log` (*kodowanie ANSI*)

Jeśli komenda wywołuje inne komendy, to w pierwszej kolejności pojawią się zapisy dotyczące wykonania komend wywoływanych a dopiero po nich zapis dotyczący komendy wywołującej. Na samym końcu przetwarzania zostanie wstawiony zapis dotyczący czasu przetwarzania całego żądania oraz dodatkowe znaki nowej linii w celu wizualnego oddzielenia kolejnych żądań w pliku log.
Kolumny pliku `timeLog.log` zawierają kolejno: *datę i czas*, *czas wykonania*, *znacznik czasowy początku*, *znacznik czasowy końca* oraz *przepis komendy* lub *adres url żądania* i rozdzielane są znakiem `TAB`.
Czas wykonania oraz znaczniki początku i końca podawane są w milisekundach.