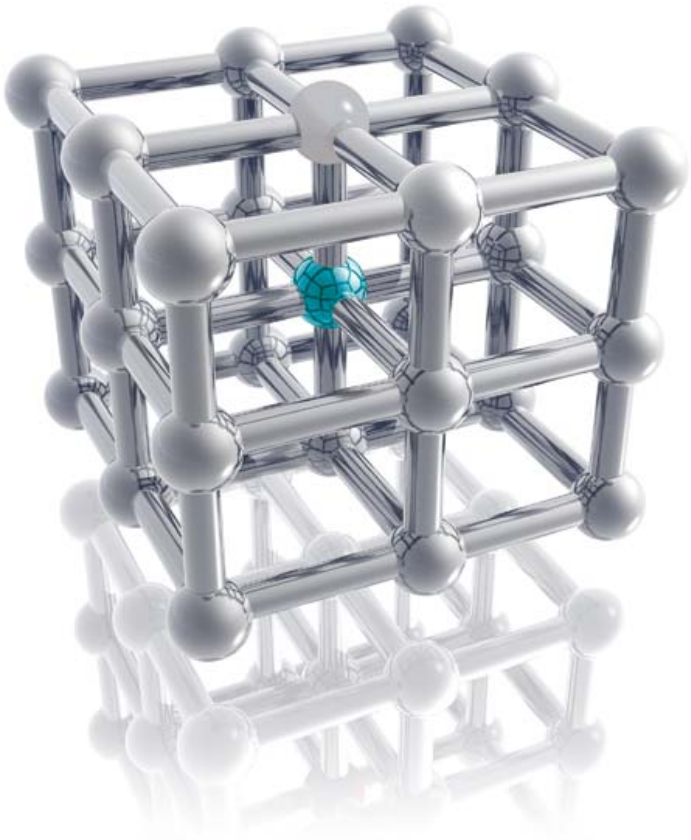

Skrypty
Podręcznik użytkownika

Dok. Nr PLP5068
Wersja: 29-07-2007

ASKOM Sp. z o. o., ul. Józefa Sowińskiego 13, 44-121 Gliwice,
tel. +48 (0) 32 3018100, fax +48 (0) 32 3018101,
http://www.askom.com.pl, e-mail: office@askom.com.pl

1. Uwagi ogólne

Celem modułu skryptów jest rozszerzenie funkcji systemu **asix** o:

- wykonywanie nietypowych obliczeń na wartościach zmiennych procesowych;
- zaprogramowanie nietypowych reakcji na zdarzenia wewnątrz modułu **asix**;
- przekazywanie na zewnątrz systemu **asix** wartości zmiennych i innych informacji o działaniu systemu;
- pozyskiwanie danych z nietypowych źródeł danych.

W celu realizacji powyższych zadań moduł skryptów umożliwia:

- automatyczne uruchamianie wielu współbieżnych skryptów stworzonych w różnych językach z możliwością określenia priorytetu/ważności skryptu;
- wymianę danych pomiędzy skryptami;
- wywoływanie przez jeden skrypt funkcji znajdujących się w innych skryptach stworzonych w różnych językach (skrypty biblioteczne);
- przekazywanie sterowania do systemu **asix** w wyniku zdarzeń powstałych w czasie wykonywania skryptu (akcje operatorskie);
- kontrolę czasu wykonywania skryptów;
- śledzenia zmian w pliku zawierającym program skryptu i automatyczne ponowne uruchomienie skryptu po wykryciu takich zmian.

2. Wymagania programowe

Moduł skryptów zrealizowany jest w oparciu o technologię ActiveX ® Scripting i może wykonywać skrypty stworzone w językach, których interpretery realizują tę technologię.

Moduł skryptów testowany był z wykorzystaniem Microsoft Windows Script 5.5, który zawiera interpretery JScript 5.5 oraz VBScript 5.5.

Moduł skryptów może być wykorzystywany w systemach Microsoft Windows NT 4.0, Microsoft Windows 2000, Microsoft Windows Me oraz Microsoft Windows 98.

Jeżeli Windows Script nie jest zainstalowany, to można go znaleźć pod adresem http://www.microsoft.com/msdownload/vbscript/scripting.asp.

Pod adresem http://msdn.microsoft.com/scripting znajduje się Microsoft Script Debugger, który można wykorzystać do uruchamiania skryptów.

Moduł skryptów pracuje w systemie **asix** w wersji 3 lub wyższej.

3. Uruchamianie i wykonywanie skryptów

Skrypty mogą być uruchamiane automatycznie przez moduł skryptów w wyniku odpowiednich deklaracji w pliku konfiguracyjnym aplikacji lub poprzez akcję operatorską SKRYPT. Niniejszy rozdział opisuje uruchamianie skryptów w wyniku odpowiedniego sparametryzowania pliku konfiguracyjnego aplikacji systemu **asix**.

Deklarowanie skryptów rozszerzających możliwości funkcjonalne systemu **asix** zapewnia program Architekt:

Architekt > *Obszary i komputery* > moduł *Skrypty i akcje*

Zakładka ***Skrypty*** umożliwia zadeklarowanie skryptów rozszerzających możliwości funkcjonalne systemu **asix**. Skrypty należy zadeklarować w kolejnych pozycjach, podając nazwę skryptu oraz deklarując plik i dodatkowe opcje skryptu zgodnie z opisaną poniżej składnią.

Nazwa skryptu musi być różna od nazw innych parametrów deklarujących użycie skryptu. Nazwa skryptu nie może być nazwą parametru modułu skryptów oraz nie może być słowem „skrypt" i „script".

Deklaracja pliku skryptu wraz z dodatkowymi opcjami wymaga zapisu zgodnie z następującą składnią:

plik_skryptu parametry_skryptu parametry_wykonawcze

gdzie:

plik_skryptu	- ścieżka oraz nazwa pliku zawierającego program skryptu; rozszerzenie nazwy pliku określa domyślny język skryptu;
parametry_skryptu	- parametry przekazywane do skryptu; parametry skryptu są dostępne w skrypcie za pomocą obiektu Asix.Script.Parameters (lub Asix.Script.Arguments);
parametry_wykonawcze	- parametry przekazywane do modułu skryptów (a nie do skryptu) i określające parametry wykonywania skryptu różne od domyślnych;

Do parametrów wykonawczych należą: ograniczenia czasu wykonania, aktywacja debuggera, deklaracja wątku, w którym zostanie wykonany skrypt i jego priorytet, typ skryptu, język itp. Niektóre parametry (np. ograniczenia) odpowiadają parametrom modułu skryptów. Umieszcza się je jako parametry wykonawcze tylko wtedy, gdy określony skrypt ma specjalne wymagania, różne od tych określonych za pomocą parametrów definiujących sposób wykonywania skryptów – patrz: Architekt > *Obszary i komputery* > moduł *Skrypty i akcje* / zakładka *Parametry skryptów*.

Parametry wykonawcze podaje się w konwencji programu cscript, tj. są one poprzedzone podwójnym ukośnikiem.

Parametry wykonawcze:

//S	- w danej chwili może być uruchomiony tylko jeden skrypt z parametrem //S, zawarty w podanym pliku; parametr ma znaczenie tylko w przypadku uruchamiania skryptu jako akcji operatorskiej systemu **asix**; moduł skryptów odrzuci próbę ponownego uruchomienia skryptu w sytuacji, gdy poprzednie uruchomienie

skryptu z parametrem //S, zawartego w tym samym pliku, jeszcze się nie zakończyło; brak tego parametru powoduje, że można równocześnie uruchamiać wiele skryptów w oparciu o program zawarty w tym samym pliku.;

//X - po uruchomieniu skryptu jest również uruchamiany debugger (zainstalowany w systemie jako domyślny); debugger zatrzymuje wykonanie skryptu na pierwszej instrukcji skryptu;

//D - wystąpienie błędu w skrypcie powoduje uruchomienie debuggera;

//IT:nnn - ograniczenie czasu wykonania części inicjalizacyjnej wyrażone w milisekundach;

//T:nnn - ograniczenie czasu wykonania części zdarzeniowej wyrażone w milisekundach;

//E:nazwa - podaje nazwę języka skryptu (np. //E:JScript);

//Watek:nazwa[,pri] - podaje nazwę wątku, w którym wykonuje się skrypt oraz określa priorytet pri wątku; nazwa wątku może być dowolna i służy grupowaniu skryptów wykonywanych w jednym wątku;

priorytety wątku skryptu:

0	IDLE – bezczynny,
1	LOWEST – najniższy,
2	BELOW_NORMAL - poniżej normalnego,
3	NORMAL - normalny (domyślny),
4	ABOVE_NORMAL - powyżej normalnego,
5	HIGHEST – najwyższy,
6	TIME_CRITICAL - krytyczny czasowo;

domyślnym priorytetem wątku jest priorytet 3 (normalny).

//U - podanie tego parametru spowoduje, że nie będą generowane zdarzenia „OnRead" oraz funkcje zdarzeń czasowych zdefiniowane metodami SetInterval i ExecuteAt, jeśli okna aplikacji **asix** są ukryte (np. za pomocą akcji operatorskiej UKRYJ).

3.1. Wątki

Ponieważ skrypty mogą realizować zadania o różnych wymaganiach czasowych począwszy od przeliczania wartości często odczytywanych zmiennych procesowych po okresową generację raportów, moduł skryptów pozwala grupować skrypty o podobnych wymaganiach czasowych i podobnym stopniu ważności.

Każdy skrypt wykonuje się w kontekście jakiegoś wątku. Każdy wątek może zawierać wiele skryptów. Skrypty znajdujące się w oddzielnych wątkach wykonywane są współbieżnie. Skrypty znajdujące się w tym samym wątku wykonują się tylko wtedy, gdy nie wykonują się inne skrypty należące do tego samego wątku. Każdy wątek posiada określony priorytet, który decyduje o stopniu ważności wykonujących się w nim skryptów, tzn. skrypt znajdujący się w wątku o wyższym priorytecie może przerwać działanie każdego skryptu wykonującego się w wątku o niższym priorytecie. Takie przerwanie wystąpi zazwyczaj wtedy, gdy skrypt znajdujący się w wątku o wyższym priorytecie musi zareagować na wystąpienie jakiegoś zdarzenia. Takim zdarzeniem może być na przykład odczyt nowej wartości zmiennej procesowej.

Priorytet wątku jest określony wartością liczbową od 0 do 6. Wartości priorytetu odpowiadają wartościom priorytetów wątków systemu operacyjnego Microsoft Windows i mają następujące znaczenie:

0 – bezczynny (ang. IDLE)
1 – najniższy (ang. LOWEST)
2 – poniżej normalnego (ang. BELOW NORMAL)
3 – normalny (ang. NORMAL) - domyślny
4 – powyżej normalnego (ang. ABOVE NORMAL)
5 – najwyższy (ang. HIGHEST)
6 – krytyczny czasowo (ang. TIME_CRITICAL)

Priorytet 3 (normalny) jest domyślnym priorytetem wątku, tj. wątek ma priorytet 3, jeśli w pliku inicjalizacyjnym nie został określony inny priorytet.

Moduł skryptów tworzy wątki w chwili inicjalizacji. Wątki są usuwane w czasie kończenia aplikacji systemu **asix**. Każdy wątek (poza wątkiem domyślnym) ma swoją nazwę określoną w pliku inicjalizacyjnym. Niezależnie od zapisów w pliku inicjalizacyjnym, moduł skryptów zawsze tworzy wątek domyślny. Wątek domyślny nie posiada nazwy i ma priorytet 3 (normalny). W wątku domyślnym wykonywane są wszystkie skrypty nie posiadające przypisania do określonego wątku.

Przypisanie skryptu do określonego wątku następuje poprzez podanie odpowiedniego parametru wykonawczego skryptu. Parametr ma postać:

//wątek:*nazwa_wątku[:priorytet]*

gdzie:

nazwa_wątku	- dowolnie wybrana nazwa wątku
priorytet	- liczba z zakresu od 0 do 6 określająca priorytet wątku

Ta sama nazwa wątku może wystąpić w deklaracjach wielu skryptów. Wszystkie takie skrypty będą wykonywane w tym samym wątku. Priorytet tego samego wątku można określać w jednej lub kilku deklaracjach. Jeśli występuje on w kilku deklaracjach, to musi mieć tą samą wartość.

Na podstawie analizy pliku inicjalizacyjnego moduł skryptów tworzy odpowiednie wątki oraz przypisuje do nich zadeklarowane skrypty.

PRZYKŁAD

raporter1 = raporty.vbs //wątek=raporty:2
raporter2 = raporty.vbs //wątek=raporty
przeliczniki = przelicznik.vbs //wątek=przeliczniki:4
monitor = monitor.vbs

W powyższym przykładzie zostaną utworzone trzy wątki: wątek domyślny o priorytecie 3 (normalny), wątek „raporty" o priorytecie 2 (poniżej normalnego) oraz wątek „przeliczniki" o priorytecie 4 (powyżej normalnego). Na podstawie programu skryptu zawartego w pliku o nazwie *raporty.vbs* zostaną utworzone dwa skrypty o nazwach „raporter1" oraz „raporter2". Oba skrypty będą wykonywane w wątku o nazwie „raporty". Na podstawie programu zawartego w pliku „przelicznik.vbs" zostanie utworzony skrypt o nazwie „przeliczniki". Skrypt będzie się wykonywał w wątku nazwie „przeliczniki" o

priorytecie 4 (powyżej normalnego). W deklaracji skryptu „monitor" pominięto deklarację wątku - skrypt zostanie wykonany w wątku domyślnym, który ma priorytet 3 (normalny).

Priorytety skryptów należy dobierać w taki sposób, aby wykonywanie skryptów nie zakłócało pracy systemu **asix**. Główne zadanie systemu **asix** realizujące funkcje graficznego interfejsu użytkownika wykonuje się na priorytecie 3 (normalny). Skrypty pracujące w wątkach o tym samym lub wyższym priorytecie mogą spowodować spowolnienie odświeżania masek, jeśli realizacja zadań realizowanych przez te skrypty wymaga zbyt dużo czasu procesora. Szczególną ostrożność należy zachować w przypadku skryptów pracujących w wątkach o priorytetach wyższych niż normalny, ponieważ ich działanie może zakłócić funkcjonowanie innych komponentów systemu **asix**, takich jak moduł zbierania danych pomiarowych i moduł archiwizacji danych pomiarowych.

Skrypty należące do tego samego wątku są inicjalizowane w kolejności występowania deklaracji tych skryptów w pliku inicjalizacyjnym. Kończenie skryptów wykonywane jest w odwrotnej kolejności. Kolejność inicjalizacji i kończenia skryptów może mieć istotne znaczenie dla skryptów wywołujących funkcje znajdujące się w innych skryptach i wymieniających dane z innymi skryptami.

Skrypty działające w różnych wątkach mogą wymieniać pomiędzy sobą dane. Jest również możliwe wywoływanie przez jeden skrypt funkcji, które znajdują się w innym skrypcie i w innym wątku.

3.2. Kontrola czasu wykonania skryptu

Kontrola czasu wykonania funkcji skryptów umożliwia detekcję wadliwie działających skryptów i jest realizowana poprzez określenie maksymalnego czasu wykonywania się skryptu (timeout).

Można określić dwa ograniczenia czasowe dla każdego skryptu. Pierwsze z nich ma zastosowanie w czasie inicjalizacji skryptu, a drugie jest wykorzystywane do ograniczenia czasu działania skryptu w wyniku reakcji na zdarzenie zewnętrzne. Przekroczenie czasu działania skryptu powoduje jego przerwanie. Skrypt nie będzie się już dalej wykonywał. Wprowadzenie dwóch różnych ograniczeń umożliwia określenie maksymalnego czasu wykonania skryptu w czasie jego inicjalizacji, która może wymagać wykonania czynności długotrwałych (takich jak uruchomienie programu Excel, czy inicjalizacja dostępu do bazy danych) oraz maksymalnego czasu w czasie normalnej pracy skryptu (reakcja na zdarzenia), który jest zazwyczaj krótszy.

Maksymalne czasy wykonania skryptów deklarowane są w programie Architekt:

Architekt > *Obszary i komputery* > moduł *Skrypty i akcje* / zakładka *Parametry skryptów*

- ***Maksymalny czas obsługi zdarzenia przez skrypt*** - parametr określa maksymalny czas obsługi zdarzenia przez skrypt. Jeśli wartością tego parametru jest 0, to czas obsługi zdarzenia nie jest ograniczony. Jednostką jest czas wyrażony w milisekundach.

  Wartość domyślna: 5000 milisekumd
- ***Maksymalny czas wykonania części inicjalizacyjnej skryptu*** - parametr określa maksymalny czas wykonywania części inicjalizacyjnej skryptu. Jeśli wartością tego parametru jest 0, to czas wykonywania części inicjalizacyjnej skryptu nie jest ograniczony. Jednostką jest czas wyrażony w milisekundach.

  Wartość domyślna: 15000 milisekumd

Pominięcie powyższych parametrów spowoduje przyjęcie wartości domyślnych.

Powyższe parametry określają ograniczenia dla każdego skryptu, który nie posiada specyficznych ustawień. Dla każdego skryptu można określić odrębne wartości ograniczeń. Służą do tego parametry wykonawcze „//IT:*nnn*" dla maksymalnego czasu inicjalizacji i „//I:*nnn*" dla maksymalnego czasu reakcji na zdarzenie. „*nnn*" określa czas w milisekundach.

Kontrola czasu wykonania skryptu jest wyłączana w czasie, gdy skrypt jest zatrzymany w wyniku pracy krokowej lub ustawienia „breakpoint'u" w debuggerze skryptów.

3.3. Śledzenie zmian w pliku skryptu

Moduł skryptów systemu **asix** automatycznie śledzi zmiany czasu zapisu pliku, w którym znajduje się program skryptu. W przypadku zmiany tego czasu, co następuje w wyniku wprowadzania zmian do pliku lub przekopiowania na jego miejsce innego pliku, moduł skryptów kończy działanie aktualnie wykonywanego skryptu, odczytuje ponownie zawartość pliku skryptu i uruchamia skrypt w oparciu o nowy program. Powyższa funkcja umożliwia wprowadzanie poprawek do skryptu w trakcie działania systemu **asix** bez konieczności kończenia całej aplikacji.

3.4. Współpraca z debuggerem skryptów

Moduł skryptów systemu **asix** umożliwia uruchamianie skryptów pod kontrolą debuggera skryptów. Praca z debuggerem skryptów jest możliwa tylko wtedy, gdy w systemie został zainstalowany debugger skryptów. Opisane poniżej funkcje modułu skryptów w zakresie współpracy z debugger został przetestowane z wykorzystaniem Microsoft Script Debugger. Można również wykorzystywać debuggery zawarte w pakietach Excel, Word, InterDev i FrontPage. Można wykorzystywać debuggery zgodne z technologią ActiveX ® Scripting.

Głównym parametrem odpowiedzialnym za współpracę jest parametr ***Ustawienia debuggera*** w pliku konfiguracyjnym aplikacji:

- ***Ustawienia debuggera*** - parametr określa zakres współpracy modułu skryptów z debuggerem skryptów
  - ***Uruchamiaj skrypty bez*** debuggera – wybranie tej opcji oznacza, że debugger nie zostanie nigdy uruchomiony; w przypadku timeoutu lub błędu skryptu do panelu programu AS zostanie wyprowadzona informacja o błędzie z podaniem numeru linii i znak; parametry wykonawcze //X i //D podawane w deklaracji skryptu nie mają znaczenia, jeśli zadeklarowana zostanie opcja *Uruchamiaj skrypty bez debuggera*;
  - ***Uruchamiaj skrypty pod kontrolą debuggera –*** opcja umożliwia uruchamianie skryptów pod kontrolą debuggera;
  - ***Uruchamiaj skrypty i włączaj debugger w chwili inicjalizacji modułu skryptów;***

  Wartość domyślna: *Uruchamiaj skrypty bez debuggera*

Parametr ustawiany w module Architekt:

Architekt > *Obszary i komputery* > moduł *Skrypty i akcje* / zakładka *Parametry skryptów*

Jeśli w deklaracji skryptu w pliku inicjalizacyjnym podano parametr wykonawczy „//X", to nastąpi zatrzymanie skryptu na pierwszej wykonanej w nim instrukcji. Dalsza praca skryptu jest kontrolowana za pośrednictwem debuggera. Parametr wykonawczy „//X" ma znaczenie tylko wtedy, gdy parameter ***Ustawienia debuggera*** ma wartość ***Uruchamiaj skrypty pod kontrolą debuggera*** lub ***Uruchamiaj skrypty i włączaj debugger w chwili inicjalizacji modułu skryptów***.

Kontrola czasu wykonania skryptu jest wyłączana w czasie, gdy skrypt jest zatrzymany w wyniku pracy krokowej lub ustawiania „breakpoint'u" w debuggerze skryptów.

4. Ogólna postać skryptu

4.1. Części skryptu

Skrypt składa się zazwyczaj z części inicjalizacyjnej i części reagującej na zdarzenie. Skrypt może posiadać tylko część inicjalizacyjną. Dotyczy to zwłaszcza skryptów uruchamianych w wyniku akcji operatorskich. Wszystkie czynności w skryptach tego typu są realizowane w czasie uruchamiania skryptu. Pozostałe skrypty wykonują czynności inicjalizacyjne, w tym przyłączanie się do źródeł zdarzeń, a następnie przechodzą do fazy oczekiwania na zdarzenia.

Poniżej przedstawiono przykład skryptu posiadającego odrębną część inicjalizacyjną oraz część reagującą na zdarzenie.

Listing 1. Fragment skryptu posiadającego odrębną część inicjalizacyjną oraz część reagującą na zdarzenie polegające na pozyskaniu przez moduł ASMEN nowej wartości zmiennej procesowej.

VBScript

```
'Cześć inicjalizacyjna
Set Panel = Asix.Panel
Panel.Message("Test OnRead")

Set Var = Asix.Variables("Emisja_CO2")

Var.OnRead = GetRef("OnRead")

'Cześć reagująca na zdarzenia
Function OnRead( val, stat, tim )
      Panel.Message("Nowa wartość")
End Function
```

JScript

```
//Cześć inicjalizacyjna
var Panel = Asix.Panel;
Panel.Message("Test OnRead");

var Var= Asix.Variables( "Emisja_CO2" );

Var.OnRead = OnRead;

//Cześć reagująca na zdarzenia
function OnRead( val, stat, time )
{
  Panel.Message("Nowa wartość");
}
```

4.2. Deinicjalizacja skryptu

Skrypt nie posiada wyróżnionej części deinicjalizacyjnej. Aby umożliwić zaprogramowanie czynności wykonywanych w czasie zakończenia pracy przez skrypt, takich jak zwolnienie zajętych zasobów, wprowadzone zostało zdarzenia *OnTerminate* obiektu *Asix* oraz *OnTerminate* obiektu *Script*.

Listing 2. Fragment skryptu zawierającego zdarzenie 'OnTerminate' obiektu Asix oraz 'OnTerminate' obiektu Script

VBScript

```
'Cześć inicjalizacyjna
Asix.OnTerminate=GetRef("OnASIXTerminate")
Asix.Script.OnTerminate  = GetRef("OnTerminate")
'Cześć reagująca na zdarzenia
Function OnASIXTerminate()
    '...
End Function

Function OnTerminate(a)
    '...
End Function
```

JScript

```
//Cześć inicjalizacyjna
Asix.OnTerminate = OnASIXTerminate;
Asix.Script.OnTerminate = OnTerminate;
//Cześć reagująca na zdarzenia
function OnASIXTerminate()
{
  //...
}
function OnTerminate()
{
  //...
}
```

Zdarzenie *OnASIXTerminate* i *OnTerminate* wykonują się w chwili zamykania aplikacji systemu **asix**. Są one wykonywane tylko wtedy, gdy skrypt jest aktywny w czasie zamykania aplikacji (był uruchomiony poprzez wpis w sekcji [SKRYPTY]). Zdarzenie *OnASIXTerminate* jest wykonywane przed zdarzeniem *OnTerminate*.

4.3. Dostęp do zasobów systemu asix

Dostęp do zasobów systemu **asix** odbywa się za pośrednictwem zestawu obiektów udostępnianych przez moduł skryptów. Każdy skrypt uruchamiany przez moduł skryptów ma bezpośredni dostęp do obiektu *Asix* - głównego obiektu udostępnianego przez moduł skryptów. Dostęp do innych obiektów odbywa się za pomocą obiektu *Asix* i innych obiektów pozyskanych za pomocą obiektu *Asix*.

Listing 3. Fragment skryptu przedstawiającego dostęp do obiektu Panel oraz wyprowadzenie za jego pomocą komunikatu o treści „komunikat".

VBScript

```
Dim Panel
Set Panel = Asix.Panel
Panel.Message( "komunikat")
```

JScript

```
var Panel = Asix.Panel;
Panel.Message( "komunikat");
```

W powyższym przykładzie pokazano dostęp do obiektu *Panel* oraz wyprowadzenie za jego pomocą komunikatu o treści „komunikat". Przyporządkowanie obiektu (tutaj *Panel*) do zmiennej języka (w tym przypadku jest to zmienna „Panel") nie jest konieczne, ale może mieć wpływ na czas wykonania skryptu, jeśli następują częste odwołania do danego obiektu. Jeśli skrypt często wyprowadza komunikaty, to korzystniej jest przechowywać obiekt *Panel* w oddzielnej zmiennej.

Listing 4. Fragment skryptu wyprowadzającego komunikat bez użycia dodatkowej zmiennej.

VBScript	JScript
Asix.Panel.Message("komuniakat")	Asix.Panel.Message("komunikat");

Powyżej przedstawiono przykład wyprowadzania komunikatu bez użycia dodatkowej zmiennej.

Jednym z najważniejszych obiektów udostępnianych przez moduł skryptów jest obiekt *Variable*, który reprezentuje zmienną procesową. Obiekt *Variable* jest zwracany przez obiekt *Asix* w wyniku wykonania funkcji *Variables*.

W poniższym przykładzie pokazano dostęp do obiektu reprezentującego zmienną.

Listing 5. Fragment skryptu realizującego dostęp do obiektu reprezentującego zmienną o nazwie „Emisja_CO2".

VBScript	JScript
Set Var = Asix.Variables("Emisja_CO2")	var Var= Asix.Variables("Emisja_CO2");

4.4. Reakcja na zdarzenia

Przypisanie określonej funkcji do zdarzenia ma postać:

Listing 6. Przypisanie określonej funkcji do zdarzenia.

VBScript	JScript
obiekt.nazwa_zdarzenia= GetRef(„nazwa_funkcji")	obiekt.nazwa_zdarzenia= nazwa_funkcji;

Gdzie „obiekt" jest wyrażeniem zwracającym określony obiekt (np. nazwa zmiennej, która odnosi się do określonego obiektu), „nazwa_zdarzenia" jest nazwą zdarzenia generowanego przez obiekt „obiekt" a „nazwa_funkcji" nazwą funkcji (lub podprogramu **Sub** w przypadku języka VBScript), która będzie wykonana w chwili wystąpienia zdarzenia.

Należy tutaj zaznaczyć, że aby została wykonana funkcja reakcji na zdarzenie, określony obiekt musi istnieć w chwili wystąpienia tego zdarzenia.

Listing 7. Fragment skryptu z błędnym przypisaniem funkcji do zdarzenia.

VBScript

```
F()
Function F()
    Dim Var
    Set Var = Asix.Variables("Emisja_CO2")
    Var.OnRead = GetRef("OnRead")
End Function

Function OnRead( val, stat, tim )
End Function
```

JScript

```
F();
function F()
{
   var Var= Asix.Variables( " Emisja_CO2" );
   Var.OnRead = OnRead;
}

function OnRead( val, stat, time )
{
}
```

Powyżej przedstawiono przykład błędnego przypisania funkcji do zdarzenia. Po zakończeniu się funkcji „F" zmienna „Var" przechowująca obiekt typu *Variable*, jako zmienna lokalna funkcji „F", przestaje istnieć, a wraz z nią przestaje istnieć obiekt typu *Variable*. Aby powiązanie funkcji ze zdarzeniem było skuteczne, obiekt generujący musi być przechowany w innym miejscu (np. w zmiennej globalnej).

W powyższych przykładach funkcja reakcji na zdarzenie miała taką samą nazwę jak zdarzenie. W ogólnym przypadku nazwa funkcji nie ma związku z nazwą zdarzenia i może być dowolna.

W funkcji reakcji na zdarzenie jest dostępny obiekt, który wygenerował zdarzenie. Obiekt generujący zdarzenie jest dostępny za pomocą słowa **this** w języku JScript i **Me** w języku VBScript.

Listing 8. Fragment skryptu wyprowadzającego do panelu kontrolnego systemu asix nazwę zmiennej przy użyciu funkcji OnRead.

VBScript

```
Dim Var
Set Var = Asix.Variables("Emisja_CO2")
Var.OnRead = GetRef("OnRead")
Function OnRead( val, stat, tim )
call Asix.Panel.Message(Me.Name)
End Function
```

JScript

```
var Var= Asix.Variables( "Emisja_CO2" );
Var.OnRead = OnRead;

function OnRead( val, stat, time )
{
Asix.Panel.Message(this.Name);
}
```

W powyższym przykładzie funkcja „OnRead" wyprowadzi do panelu kontrolnego systemu **asix** nazwę zmiennej, której wartość pomiarowa została odczytana.

Aby usunąć powiązanie pomiędzy źródłem zdarzenia i funkcją reakcji na zdarzenie należy jako funkcję reakcji na zdarzenie podać wartość pustą.

Listing 9. Sposób podania wartości pustej jako funkcji reakcji na zdarzenie.

VBScript

```
Var.OnRead = Empty
```

JScript

```
Var.OnRead = null;
```

Funkcje obsługi zdarzeń są wykonywane po zakończeniu wykonywania bieżącego fragmentu skryptu, tj. części inicjalizacyjnej skryptu, lub funkcji obsługi zdarzenia. To znaczy, bieżący ciąg instrukcji skryptu nie jest przerywany przez funkcję obsługi zdarzenia. Wyjątkiem od tej zasady może być wywołanie metody obiektu zewnętrznego, tj. obiektu spoza zestawu obiektów udostępnianych przez moduł skryptów. Obecna wersja modułu skryptów nie zawiera metod dopuszczających wykonanie funkcji obsługi zdarzenia w trakcie realizacji metody.

5. Funkcje modelu obiektowego modułu skryptów

5.1. Dostęp do danych pomiarowych

Metoda *Variables* obiektu *Asix* zapewnia dostęp do zmiennych procesowych systemu **asix**. W wyniku jej wykonania zostaje zwrócona wartość będąca obiektem reprezentującym zmienną. Argumentem jest nazwa oraz typ archiwum. W przypadku niepowodzenia, tj. np. w przypadku braku danej wartości w bazie danych ASMEN'a, metoda zwraca wartość pustą, co można wykorzystać w skrypcie do detekcji błędnych nazw zmiennych.

Listing 10. Przykład skryptu realizującego dostęp do zmiennej procesowej systemu asix wraz z detekcją błędnych nazw zmiennych.

VBScript

```
Set Var = Asix.Variables("Emisja_CO2")
If TypeName(Var) = "Nothing" Then
    Asix.Panel.Message("Błędna nazwa" )
End If
```

JScript

```
var Var= Asix.Variables( "Emisja_CO2" );
if ( Var == null )
    Asix.Panel.Message("Błędna nazwa" );
```

Każdorazowe wywołanie funkcji z tym samym argumentem powoduje zwrócenie nowego obiektu odwołującego się do tej samej zmiennej **asix**'a. Czas życia takiego obiektu to czas istnienia odwołania do tego obiektu w skrypcie.

Obiekt *Variable* posiada składowe *Value*, *Status* i *Time* określające odpowiednio wartość, status i czas ostatnio odczytanej danej pomiarowej. Odczyt danych pomiarowych jest automatycznie realizowany przez moduł ASMEN systemu ASIX w cyklu odświeżania wynikającym z definicji zmiennej procesowej. Do odczytu aktualnej wartości pomiaru można również wykorzystać metodę *Read* obiektu *Variable* – wymusza to odświeżenie wartości zmiennej procesowej w chwili wykonania metody. Wartości pól *Value*, *Status* i *Time* nie ulegają zmianie w trakcie wykonywania bieżącego fragmentu skryptu tj. części inicjalizacyjnej skryptu, lub funkcji obsługi zdarzenia. Wyjątkiem od tej zasady może być wywołanie metody obiektu zewnętrznego, tj. obiektu spoza zestawu obiektów udostępnianych przez moduł skryptów lub tych metod modułu skryptów, których celem jest zmiana wartości tych pól (np.: *Read*, *ArcRead*, zapis do pola *Current* itd.).

5.2. Dostęp do danych archiwalnych

Jeśli argumentem funkcji *Variables* obiektu *Asix* jest nazwa zmiennej wraz z jednoliterowym kodem typu archiwum, to moduł skryptów będzie próbował otworzyć bazę pomiarów archiwalnych. Jeśli próba zakończy się pomyślnie, to wartością logiczną pola *WithArchive* obiektu *Info* będzie **true** (dostęp do obiektu *Info* następuje poprzez pole *Info* obiektu *Variable*). Można wtedy wykorzystać metody *ArcRead*, *ArcReadAt*, *ArcSeek* i *ArcTell* do przeglądu danych archiwalnych. Należy zwrócić uwagę, że wartości danych archiwalnych są zwracane przez obiekt *Variable* za pomocą składowych *Value*, *Status* i *Time*, które wykorzystywane są również do zwracania wartości bieżących. Aby uniknąć

niejednoznaczności, obiekt *Variable* posiada pole *Current*, która ma wartość **true**, jeśli zwracane są wartości bieżące i **false** w przeciwnym wypadku. Wykonanie jakiejkolwiek funkcji dostępu do danych archiwalnych powoduje ustawienie tego pola na wartość **false**. Gdy pole *Current* ma wartość **false**, funkcja reakcji na zdarzenie *OnRead* nie jest wywoływana. Aby przywrócić dostęp do bieżących pomiarów, należ przypisać polu *Current* wartość logiczną **true**. Spowoduje to aktualizację pól *Value*, *Status* i *Time* wartościami bieżącymi.

Poniżej przedstawiono przykład skryptu odczytującego wartość zmiennej o nazwie „Emisja" z archiwum typu D. Odczytywana jest wartość z 20 stycznia 2002 z godziny 17:50:31. Przykładowy skrypt zawiera wywołanie metody *ArcParam* obiektu *Variable* z parametrem INTR_NEAREST. Wywołanie tej metody spowoduje zwrócenie najbliższej wartości, jeśli archiwum nie zawiera danej pomiarowej z określonej chwili. Jeśli odczyt wartości archiwalnej zakończy się pomyślnie, to do panelu kontrolnego zostanie wyprowadzona odczytana wartość.

Listing 11. Przykład skryptu odczytującego wartość zmiennej o nazwie „Emisja" z archiwum typu D.

VBScript

```
Dim Stat
Dim Panel
Dim Var
Set Panel = Asix.Panel

Set Var = Asix.Variables("Emisja,D")

'Określenie sposobu interpolacji danych - interpolacja
'najbliższą daną.
Var.ArcParam( INTR_NEAREST )

Stat = Var.ArcReadAt(#1/20/2002 17:50:31#)

If Stat = STAT_OK Then
    If Var.Status = OPC_QUALITY_GOOD Then
        Panel.Message("Emisja= " & CStr(Var.Value)_
        & " mg/h" )
    End If
End If
```

JScript

```
var Stat;
var Panel = Asix.Panel;
var Var = Asix.Variables("Emisja,D");
var Time = new Date(2002, 0, 20, 17, 50, 31);

//Określenie sposobu interpolacji danych - interpolacja
//najbliższą daną.
Var.ArcParam( INTR_NEAREST );

Stat = Var.ArcReadAt(Time.getVarDate());

if ( Stat == STAT_OK )
    if ( Var.Status == OPC_QUALITY_GOOD )
        Panel.Message("Emisja= " + Var.Value + " mg/h" );
```

5.3. Status danych

Statusy danych pomiarowych są wyrażone jako statusy OPC (ang. OLE for Process Control). Dotyczy to pola *Status* zmiennej *Variable*, funkcji obsługi zdarzenia *OnRead*, funkcji *Write* obiektu *Variable* i in.. Wykorzystując statusy OPC można używać wartości symbolicznych prezentowanych w poniższej tabeli.

Tabela 1. Wartości statusów danych pomiarowych.

Wartość symboliczna	Wartość numeryczna (kod heksadecymalny)
OPC_QUALITY_MASK	C0
OPC_QUALITY_BAD	00
OPC_QUALITY_CONFIG_ERROR	04
OPC_QUALITY_NOT_CONNECTED	08
OPC_QUALITY_DEVICE_FAILURE	0c
OPC_QUALITY_SENSOR_FAILURE	10
OPC_QUALITY_LAST_KNOWN	14
OPC_QUALITY_COMM_FAILURE	18
OPC_QUALITY_OUT_OF_SERVICE	1C
OPC_QUALITY_UNCERTAIN	40
OPC_QUALITY_SENSOR_CAL	50
OPC_QUALITY_EGU_EXCEEDED	54
OPC_QUALITY_SUB_NORMAL	58
OPC_QUALITY_LAST_USABLE	44
OPC_QUALITY_GOOD	C0
OPC_QUALITY_LOCAL_OVERRIDE	D8
OPC_LIMIT_MASK	03
OPC_LIMIT_OK	00
OPC_LIMIT_LOW	01
OPC_LIMIT_HIGH	02
OPC_LIMIT_CONST	03

Wykorzystanie statusów OPC wymaga odpowiedniej parametryzacji modułu ASMEN systemu **asix**. W sekcji [ASMEN] powinien zostać umieszczony zapis:

STATUS_OPC = Tak

Jeśli moduł ASMEN nie używa statusów OPC, to moduł skryptów wyprowadzi komunikat ostrzegawczy do panelu kontrolnego systemu **asix**. Nie będzie wtedy możliwe utworzenie obiektu *Variable*.

Oprócz wartości zdefiniowanych w standardzie OPC, w systemie **asix** zdefiniowane są specyficzne dla tego systemu statusy. Zajmują one bity 8-16 statusu danej pomiarowej.

Tabela 2. Statusy danych charakterystyczne dla systemu asix.

Wartość symboliczna	Wartość numeryczna (kod heksadecymalny)	Znaczenie
AS_STAT_MASK_SOURCE	0300	Maska bitowa pola określającego źródło danej.
AS_STAT_NORMAL	0000	Dana pozyskana w trybie podstawowym.
AS_STAT_HIST	0100	Dana historyczna.
AS_STAT_MANU	0200	Dana wprowadzona ręcznie.
AS_STAT_SIMUL	0300	Dana symulowana.
AS_STAT_MASK_ORIGIN	0400	Maska bitowa pola określającego źródło danej (ASPAD).
AS_STAT_LOCAL	0000	Dana pozyskana lokalnie.
AS_STAT_COPY	0400	Dana skopiowana z innego archiwum.
AS_STAT_SECOND_LIMIT	0800	Wraz z OPC_LIMIT_LOW lub OPC_LIMIT_LOW oznacza przekroczenie limitu HIGHHIGH lub LOWLOW.
AS_STAT_UTC_TIME	4000	Czas danej wyrażony jest w postaci UTC. Ten bit statusu normalnie nie pojawia się w module skryptów, ponieważ czas zawsze wyrażony jest w postaci lokalnej (VT_DATE). Wyjątkiem jest ustawienie pola *UTCMode* obiektu *Variable*. Bit ma również znaczenie w przypadku metody *Write* obiektu *Variable* (patrz: *Uwagi na temat czasu.*)
AS_STAT_MASK_OPC_STAT	00FF	Maska bitowa określająca najmłodszy bajt statusu, gdzie jest zawarty 8-bitowy status wg standardu OPC.

5.4. Dynamiczne składowe obiektów

Większość obiektów modelu obiektowego realizowanego przez moduł skryptów zezwala na dodawanie do obiektu nowych pól i metod. Pozwala to na powiązanie informacji z obiektem, którego one dotyczą. Pozwala to także na wzajemne powiązanie ze sobą

obiektów. Przykładem zastosowania dynamicznych składowych i wzajemnego powiązania obiektów może być zadanie polegające na przeliczenie wartości każdej z określonego zestawu zmiennych (zmienne źródłowe) i zapisanie wyniku w innej zmiennej (zmienne docelowe). Z każdą zmienną źródłową powiązana jest określona zmienna docelowa, zaś przeliczenie wartości i zapis wyniku następuje w funkcji reakcji na zdarzenie *OnRead*. Zdarzenie *OnRead* jest generowane przez obiekt *Variable* po pozyskaniu przez moduł ASMEN nowych wartości pomiarowych.

Listing 12. Przykład skryptu zawierającego dynamiczne składowe obiektów.

VBScript

```
Set VarSrc1 = Asix.Variables("Src1")
VarSrc1.Set(„Dst") = Asix.Variables("Dst1")
VarSrc1.OnRead = GetRef("OnRead")

Set VarSrc2 = Asix.Variables("Src2")
VarSrc2.Set(„Dst") = Asix.Variables("Dst2")
VarSrc2.OnRead = GetRef("OnRead")
'itd.
Function OnRead( val, stat, tim )
call Me.Dst.Write( -val, stat, time )
End Function
```

Jscript

```
var VarSrc1= Asix.Variables( "Src1" );
VarSrc1.Dst = Asix.Variables( "Dst1" );
VarSrc1.OnRead = OnRead;

var VarSrc2= Asix.Variables( "Src2" );
VarSrc2.Dst = Asix.Variables( "Dst2" );
VarSrc2.OnRead = OnRead;
//itd.
function OnRead( val, stat, time )
{
this.Dst.Write( -val, stat, time );
}
```

W powyższym przykładzie zmienne źródłowe mają nazwy typu „Srcn", a zmienne docelowe „Dstn". Każdy obiekt reprezentujący zmienną źródłową posiada dynamicznie dodaną składową o nazwie „Dst". Składowa ta będzie przechowywać odsyłacz do obiektu reprezentującego zmienną docelową. W funkcji reakcji na zdarzenie *OnRead* (wspólnej dla wszystkich zmiennych źródłowych) następuje wykorzystanie składowej dynamicznej „Dst" do lokalizacji odpowiedniej zmiennej docelowej i zapisu do niej zanegowanej wartości zmiennej źródłowej. W przykładzie pokazano również sposób korzystania ze słów **Me** (VBScript) i **this** (JScript). Słowa te pozwalają na dostęp do obiektu, na rzecz którego wykonywana jest funkcja obsługi zdarzenia, a to z kolei pozwala napisać jedną uniwersalną funkcję obsługi zdarzenia dla wielu zmiennych. Oczywiście, w przypadku większej ilości zmiennych, zamiast szeregu deklaracji **var** VarSrcn, należałoby zmienne źródłowe umieścić w jakiejś strukturze (np. tablicy) gromadzącej wszystkie zmienne źródłowe. Dla zmiennych docelowych nie jest wymagana żadna struktura, ponieważ są one przechowywane w składowych dynamicznych „Dst" zmiennych źródłowych.

UWAGA Obiekty zezwalające na tworzenie dynamicznych składowych mogą rezerwować pewien zestaw nazw składowych, które zostaną wykorzystane w przyszłości. Zarezerwowanych nazw składowych nie wolno wykorzystywać w skrypcie. Każdy obiekt rezerwuje nazwy *Diagnostics* i *Explicite*.

UWAGA Należy tutaj zauważyć, że język VBScript nie rozróżnia wielkości liter w czasie dostępu do składowych obiektu. Jeśli w języku JScript utworzymy dwie składowe dynamiczne o nazwach „aa" i „AA":

obiekt.aa = 5;
obiekt.AA = 6;

to w języku VBScript będzie możliwy dostęp tylko do składowej o nazwie „aa", niezależnie od tego, czy użyje się konstrukcji „obiekt.aa" czy też „obiekt.AA".

W języku JScript składowe dynamiczne tworzone są poprzez użycie operatora dostępu do składowej („.") w instrukcji przypisania (jak w przykładzie powyżej). Jeśli składowa nie istnieje, to zostanie utworzona oraz zostanie do niej przypisana nowa wartość. W języku VBScript nie można użyć tej samej metody. Instrukcja przypisania do nieistniejącej składowej kończy się błędem. W tym celu, każdy obiekt zezwalający na dynamiczne dodawanie składowych udostępnia metodę Set, której wywołanie ma postać:

obiekt.Set(*nazwa_składowej*) = *wartość*

gdzie: *nazwa_składowej* to łańcuch znaków ujęty w cudzysłowy określający nazwę nowej składowej.

Jeśli w chwili wykonania metody *Set*, składowa o podanej nazwie nie istnieje, to zostanie ona utworzona.

Analogicznie do metody *Set*, każdy obiekt zezwalający na dodawanie składowych dynamicznych posiada metodę *Get*, która zwraca wartość składowej.

obiekt.Get(*nazwa_składowej*)

Korzystanie z metody *Get* zazwyczaj nie jest konieczne. Wyjątkiem jest operacja pobrania dynamicznej metody obiektu w języku VBScript, tj. wtedy gdy należy potraktować metodę jako pole i pobrać jego zawartość jako odsyłacz do metody.

```
Asix.GlobalData.Set(„Przelicznik") = GetRef(„Przelicznik") ‘metoda dynamiczna
Set Przelicznik = Asix.GlobalData.Get(„Przelicznik") ‘pobranie zawartości pola o nazwie Przelicznik
Przelicznik() ‘wywołanie funkcji
```

Ze względu na składnię języka VBScript, instrukcja:

```
Set Przelicznik = Asix.GlobalData.Przelicznik
```

nie pozwala na jednoznaczną interpretację jako pobranie pola, a nie jego wywołanie jako metody, ponieważ VBSvript pozwala pomijać nawiasy funkcji, które nie mają parametrów. W takiej sytuacji metoda *Get* pozwala uniknąć niejednoznaczności.

Z obu metod można korzystać w każdym języku skryptu. Metody można również wykorzystać do tworzenia oraz dostępu do składowych, których nazwy są sprzeczne z regułami języka skryptu.

Wartością składowej dynamicznej obiektu może być nie tylko dana prosta (liczba, tekst), ale także obiekt i metoda (funkcja).

W przypadku, gdy wartością składowej dynamicznej jest funkcja, to można ją wykorzystywać tak jak metodę obiektu, którego jest składową. W szczególności, jeśli słowa kluczowe **Me** języka VBScript i **this** języka Jscript są wykorzystane wewnątrz takiej funkcji, to odnoszą się do obiektu, którego funkcja jest składową.

PRZYKŁAD

Listing 13. Przykład skryptu zawierającego składową dynamiczną w formie metody o nazwie „Fun" dodaną do obiektu 'Variable'.

VBScript

```
Set Var = Asix.Variables("Emisja_CO2")
Var.Set("Fun") = GetRef("Fun")
Var.Fun()
Function Fun()
  Asix.Panel.Message( "Zmienna " + Me.Name )
End Function
```

JScript

```
var Var= Asix.Variables( "Emisja_CO2" );
Var.Fun = Fun;
Var.Fun(); //wywołanie dynamicznej metody
function Fun()
{
Asix.Panel.Message( "Zmienna " + this.Name );
}
```

W powyższym przykładzie do obiektu *Variable* dodano składową dynamiczną w formie metody o nazwie „Fun". Wykonanie obu skryptów spowoduje wyprowadzenie do panelu kontrolnego systemu ASIX komunikatu o treści „Zmienna Emisja_CO2". Wywołanie tej funkcji jako funkcji samodzielnej (instrukcja „Fun()" a nie „Var.Fun()") zakończy się błędem, ponieważ w takim przypadku słowa kluczowe **Me** oraz **this** nie odnoszą się do obiektu *Variable*.

UWAGA Należy unikać tworzenia powiązań pomiędzy obiektami tworzących zamknięty cykl.

5.5. Przeliczanie i zapis danych pomiarowych

Skrypty można wykorzystać do realizacji funkcji przeliczających wykonujących nietypowe przeliczenia wartości danych pomiarowych. Za pomocą skryptu nie można jednak zmodyfikować wartości odczytanej z urządzenia zewnętrznego, tak jak robi to moduł ASMEN przy pomocy funkcji przeliczających. Można natomiast wykorzystać funkcję *Write* obiektu *Variable* do zapisu wyniku przeliczeń wartości jednej zmiennej do zmiennej, która jest zadeklarowana w kanale NONE i pełni jedynie funkcję odbiornika przeliczonych wartości. Wynik przeliczania powinien być zazwyczaj związany z czasem oraz statusem. Aby można było przyporządkować nową wartość zmiennej kanału NONE razem ze statusem, kanał NONE musi być odpowiednio sparametryzowany. W sekcji tego kanału należy zadeklarować parametr ***Zapis czasu i statusu***.

Brak tego parametru spowoduje zapis jedynie wartości zmiennej.

Metoda *Write* obiektu *Variable* ma postać:

Write(*wartość*[, *status*[, *czas*]])

Pominięcie parametru *status* spowoduje przyjęcie, że jest on równy OPC_QUALITY_GOOD, tzn. dana jest poprawna, zaś pominięcie parametru *czas* spowoduje przyjęcie czasu bieżącego. Jeśli status i czas danej pomiarowej są nieistotne, to można wykorzystać pole *Value* do przypisania zmiennej nowej wartości tzn. instrukcje:

obiekt.Write(5)

i

obiekt.Value = 5

są równoważne. (*obiekt* oznacza obiekt *Variable*)

Jeśli kanał NONE nie posiada parametru ***Zapis czasu i statusu***, to status i czas podawany w operacji zapisu są ignorowane. W przypadku fizycznych kanałów transmisji, możliwość zapisu czasu i statusu zależy od danego kanału.

Zapis do zmiennej można również wykorzystać do tworzenia przebiegów symulowanych, oraz do zapisu wartości do urządzeń zewnętrznych.

Dodatkowe informacje na temat realizacji funkcji przeliczających znajdują się w punkcie *5.15. Realizacja bibliotek funkcji* oraz *5.16. Uwagi na temat czasu*.

UWAGA Zapis do zmiennej za pomocą operacji przypisania do pola *Value* lub metody *Write* powoduje wykonanie fizycznego transferu nowej wartości do urządzenia zewnętrznego (lub kanału NONE). Nie oznacza to automatycznej aktualizacji pól *Value*, *Status* i *Time*. Dopiero wykonanie fizycznego odczytu z urządzenia zewnętrznego może zaktualizować te pola. Po wykonaniu zapisu, moduł ASMEN systemu **asix** dokonuje automatycznego fizycznego odczytu nowych wartości, co powoduje aktualizację powyższych pól oraz ewentualną generację zdarzenia ***OnRead***. Aktualizacja tych pól nastąpi jednak dopiero po zakończeniu wykonywania bieżącego zdarzenia lub części inicjalizacyjnej skryptu. Aby wymusić aktualizację tych pól zaraz po zapisie, należy wykonać metodę ***Read***.

5.6. Zmienne tablicowe

Moduł ASMEN systemu **asix** pozwala na odczyt i zapis zmiennych tablicowych.

Zmienna tablicowa to zestaw danych pomiarowych składający się z wielu danych tego samego typu. Ze wszystkimi danymi wchodzącymi w skład zmiennej tablicowej związany jest jeden status i jeden czas. Jeśli obiekt *Variable* odnosi się do zmiennej tablicowej, to pole *Value* oraz parametr „wartość" funkcji reakcji na zdarzenie *OnRead* również jest tablicą. Jest to tablica w znaczeniu języka VBScript. Dana zapisywana do pola *Value* oraz parametr „wartość" metody *Write* również musi być tego typu tablicą. Dodatkowo, w języku JScript dana ta może być obiektem *Array* języka JScript. Zapisywana dana powinna zawierać przynajmniej tyle elementów ile zawiera ich zmienna, której dotyczy obiekt *Variable*. Ilość elementów, z których składa się zmienna tablicowa można odczytać z pola *Count* obiektu *Info*. Należy zwrócić uwagę, że w przypadku zmiennych tablicowych, zmienia się sposób dostępu do konkretnych wartości zmiennej. Należy posługiwać się numerem konkretnej danej w tablicy, czyli indeksem. Pierwszy element tablicy ma numer 0. Język JScript nie zezwala na odczyt poszczególnych wartości tablicy znajdującej się w polu *Value* lub parametrze „wartość" funkcji *OnRead*. Taką tablicę należy przekształcić do obiektu *VBArray* i dopiero z tego obiektu można odczytać poszczególne wartości składające się na zmienną tablicową. Zawartość obiektu *VBArray* nie może być jednak modyfikowana. Modyfikacja elementów tablicy obiektu *VBArray* jest możliwa po przekształceniu tego obiektu do obiektu *Array* za pomocą metody *toArray* obiektu *VBArray*.

Poniżej przedstawiono kilka przykładów obsługi zmiennych tablicowych w skryptach.

PRZYKŁAD 1

Listing 14. Przepisanie wartości z jednej zmiennej tablicowej do drugiej.

VBScript

```
Set Var1 = Asix.Variables("Emisja_CO")
Set Var2 = Asix.Variables("Emisja_CO2")

Var1.Value = Var2.Value
'lub
Var1.Write(Var2.Value) 'można również podać status i czas
```

Jscript

```
var Var1= Asix.Variables( "Emisja_CO" );
var Var2= Asix.Variables( "Emisja_CO2" );

Var1.Value = Var2.Value;
//lub
Var1.Write(Var2.Value); //można również podać status i czas
```

W powyższym przykładzie, w języku JScript nie była wymagana żadna konwersja tablicy na obiekty pośrednie ponieważ metoda *Write* oraz pole *Value* akceptują tablice w znaczeniu języka VBScript.

PRZYKŁAD 2

Listing 15. Odczyt dwuelementowej tablicy z jednej zmiennej, modyfikacja jej elementów i zapis do drugiej zmiennej.

VBScript

```
Set Var1 = Asix.Variables("Emisja_CO")
Set Var2 = Asix.Variables("Emisja_CO2")

Dim  arr

arr = Var1.Value

arr(0) = cint(arr(0)) +1
arr(1) = cint(arr(1)) +1

Var2.Value = arr
'lub
Var2.Write(arr) 'można również podać status i czas
```

JScript

```
var Var1= Asix.Variables( "Emisja_CO" );
var Var2= Asix.Variables( "Emisja_CO2" );
//konwersja do obiektu VBArray
var arr1 = new VBArray(Var1.Value);
//konwersja do obiektu Array
var arr2 = arr1.toArray();

arr2[0] += 1;
arr2[1] += 1;

Var2.Value = arr2;
//lub
Var2.Write(arr2); //można również podać status i czas
```

W powyższym przykładzie do każdego elementu tablicy zmiennej „Var1" dodano liczbę 1. Wynikowa tablica została zapisana do zmiennej „Var2". W przypadku języka JScript należało dokonać konwersji do obiektu *VBArray*, a następnie do obiektu *Array*.

PRZYKŁAD 3

Listing 16. Przepisanie nowych wartości do dwuelementowej zmiennej tablicowej.

VBScript

```
Set Var = Asix.Variables("Emisja_CO")

Dim arr(1)
arr(0) = 1
arr(1) = 2

Var.Value = arr
'lub
Var.Write(arr)  'można również podać status i
czas
'Zapis z użyciem funkcji Array
Set Var = Asix.Variables("Emisja_CO")

Var.Value = Array(1,2)
'lub
Var.Write(Array(1,2))  'można również podać
status i czas
```

Jscript

```
var Var= Asix.Variables( "Emisja_CO" );

var arr = new Array(1,2);

Var.Value = arr;
//lub
Var.Write(arr); //można również podać status i
czas
```

PRZYKŁAD 4

Listing 17. Dostęp do poszczególnych elementów tablicy w funkcji obsługi zdarzenia OnRead.

VBScript

```
Set Var = Asix.Variables("Emisja_CO")

Var.OnRead = GetRef("OnRead")

Function OnRead( val, stat, tim )
Asix.Panel.Message "Elementy:" & CStr(val(0))
& CStr(val(1))
End Function
```

Jscript

```
var Var= Asix.Variables( "Emisja_CO" );

Var.OnRead = OnRead;

function OnRead( val, stat, time )
{
  var arr = new VBArray(val);
  Panel.Message("Elementy:" +arr.getItem(0) +","
+arr.getItem(1);
}
```

Funkcja *OnRead* wyprowadza komunikat zawierający wartości poszczególnych elementów tablicy do panelu kontrolnego systemu **asix**. W języku JScript wymagana była konwersja parametru „val" do obiektu *VBArray*.

5.7. Zmienne tablicowe i operacje na tekstach

Moduł skryptów pozwala przypisać tekst do zmiennej tablicowej, której elementy mają długość jednego bajta. Zmienne te są zazwyczaj skojarzone z funkcją przeliczającą NIC_TEXT. Przypisanie tekstu polega na tym, że do kolejnych elementów zmiennej tablicowej podstawiane są kolejne znaki tekstu. Przypisywany tekst może być dowolnej długości. Jeśli ilość znaków tekstu jest mniejsza niż ilość elementów zmiennej tablicowej,

to pozostałe elementy zmiennej otrzymują wartość 0. Jeśli przypisywany tekst jest dłuższy niż zmienna tablicowa, to nadmiarowe znaki są ignorowane.

PRZYKŁAD

Listing 18. Przykład skryptu przedstawiającego sposób operowania na tekstach.

VBScript

```
Set Var = Asix.Variables("Tekst")
Var.Value = "To jest tekst"
Set Dzisiaj = Now()
Var.Value = " Dzisiaj jest " & Dzisiaj
```

JScript

```
var Var= Asix.Variables( "Tekst" );
Var.Value = "To jest tekst";
var Dzisiaj = new Date();
Var.Value = "Dzisiaj jest " + Dzisiaj;
Var.Value = new String("To jest tekst");
```

Opisany wyżej sposób operowania na tekstach jest obsługiwany przez moduł skryptów w wersji 1.06.000 lub wyższej.

5.8. Dostęp do systemu alarmów

Model obiektowy modułu skryptowego umożliwia generację i sprawdzanie stanu alarmów systemu **asix**. Alarmy są identyfikowane za pomocą numerów i są zadeklarowane w odpowiednich plikach konfiguracyjnych. Informacje na ten temat można znaleźć w dokumentacji systemu **asix**.

Metody *Raise*, *Cancel* i *State* obiektu *Asix* służą odpowiednio do ustawiania, zerowania i sprawdzania stanu alarmu. Parametry funkcji *Raise* i *Cancel* pozwalają podać dwa parametry alarmu oraz czas *(patrz: 5.16. Uwagi na temat czasu)*.

5.9. Wykonywanie akcji operatorskich

Za pomocą metody *ExecuteAction* można wykonać akcję operatorską. Parametrem metody jest tekst określający akcję, która ma być wykonana. Szczegółowe informacje na temat akcji operatorskich znajdują się w dokumentacji systemu **asix**.

PRZYKŁAD

```
Asix.ExecuteAction( "Otworz Emisja.msk" )
```

Powyższa instrukcja spowoduje otwarcie maski „emisja.msk".

5.10. Dostęp do zawartości pliku inicjalizacyjnego

Model obiektowy umożliwia dostęp do zapisów w sekcjach pliku inicjalizacyjnego (pliki INI aplikacji systemu **asix** w wersjach poprzedzających pakiet **asix5**). Umożliwia to metoda *Parameters* obiektu *Asix*. Metoda zwraca obiekt-kolekcję wszystkich parametrów umieszczonych w sekcji o nazwie określonej parametrem funkcji *Parameters*. Każdy element tej kolekcji jest obiektem zawierającym pola *Name* oraz *Value*, które zwracają nazwę parametru oraz jego wartość. Ponieważ pole *Value* jest polem domyślnym, to można jego nazwę pominąć. Obiekt zwracany przez funkcję *Parameters*, jako kolekcja, posiada pole *Count* określające ilość parametrów w sekcji (czyli ilość elementów kolekcji) oraz metodę *Item*, która pozwala na dostęp do parametru poprzez podanie jego numeru kolejnego (począwszy od wartości 1). Nazwy parametrów w sekcji pliku inicjalizacyjnego mogą się powtarzać.

Listing 19. Przykład skryptu służącego do przeglądania sekcji o nazwie xxx pliku inicjalizacyjnego.

VBScript

```
Set Parameters = Asix.Parameters(„xxx")
For Each Parameter in Parameters
    Dim keyName
    keyName  = UCase(Parameter.Name)
    Select Case keyName
      Case "ZMIENNA"
          '.......
      Case "DIAGNOSTYKA"
          '.....
      Case Else
          Call Asix.Panel.Error( "Błędny parameter " + _
                    Parameter.Name + "=" + Parameter)
    End Select
Next
```

JScript

```
var Parameters = Asix.Parameters(„xxx");
for ( Parameter in Parameters )
   {
   var Name  = new String(Parameters[Parameter].Name)
   Name = Name.toUpperCase();
   switch (Name.valueOf() )
        {
        case "ZMIENNA":
            break;
        case "DIAGNOSTYKA":
            break;
        default:
            Panel.Error( "Błędny parameter " +
                    Parameters[Parameter].Name + "=" +
                    Parameters[Parameter]);
        break;
        }
}
```

Listing 20. Przykład skryptu umożliwiającego dostęp do trzeciego parametru poprzez jego numer.

VBScript

```
Set Parameters = Asix.Parameters(„xxx")
If Parametrs.Count >= 3 Then
Set xx = Parameters (3) 'trzeci parametr
End If
```

JScript

```
var Parameters = Asix.Parameters(„xxx");
if ( Parameters.Count >= 3 )
   xx = Parameters.Item(3) //trzeci parametr
```

5.11. Dostęp do parametrów umieszczonych w deklaracji skryptu

W deklaracji skryptu w pliku inicjalizacyjnym i w treści akcji SKRYPT można umieścić parametry przekazywane do skryptu.

Wewnątrz skryptu można odczytać te parametry wykorzystując pole *Arguments* obiektu *Script* (obiekt *Script* jest wartością pola *Script* obiektu *Asix*). Wartością pola *Argument* jest obiekt-kolekcja parametrów. Jako kolekcja, obiekt ten posiada pole *Count* oraz metodę *Item*.

Listing 21. Przykład skryptu wyprowadzającego wartości parametrów do panelu kontrolnego.

VBScript

```
Set Parameters = Asix.Script.Arguments

For each Parameter in Parameters
    Asix.Panel.Message( "Parameter " + Parameter)
```

Jscript

```
Parameters = Asix.Script.Arguments;

for ( Parameter in Parameters )
   Asix.Panel.Message( "Parametr=" +
                                   Parameters[Parameter]);
```

Parametry są również dostępne poprzez użycie numeru parametru (pierwszy ma numer 1).

Listing 22. Fragment skryptu deklarującego numer parametru.

VBScript

```
Xx = Asix.Script.Arguments (1)
'pierwszy parametr
```

JScript

```
xx = Asix.Script.Arguments.Item(1) //pierwszy
parametr
```

Kolekcja zawarta w polu *Asix.Script.Argument* ma takie same własności jak opisana w poprzednim punkcie kolekcja parametrów w sekcji pliku inicjalizacyjnego.

5.12. Zdarzenia związane z upływem czasu

- Metoda ***SetInterval***
- Metoda ***ExecuteAt***

Funkcje *SetInterval* i *ExecuteAt* obiektu *Asix* pozwalają powiązać wykonanie określonej funkcji z upływem czasu. Metoda *SetInterval* powoduje cykliczne wywoływanie funkcji określonej parametrem, a metoda *ExecuteAt* powoduje jednorazowe wykonanie funkcji o podanym czasie. Funkcje zwracają identyfikator powiązania funkcji z czasem, który można wykorzystać do anulowania takiego powiązania. Metoda *SetInterval* posiada dodatkowo parametr pozwalający na określenie wyrównania czasu w stosunku do pełnej wielokrotności interwału.

Metoda ***SetInterval*** ma postać:

SetInterval(*funkcja, interwał*[,*przesunięcie*])

Parametry czasowe są wyrażone w milisekundach.

Wywołanie w postaci *SetInterval(funkcja,5000)* spowoduje cykliczne wywoływanie funkcji co pięć sekund, natomiast wywołanie *SetInterval(funkcja,5000,0)* spowoduje, że funkcja będzie wywoływana o 00:00:00, 00:00:05, 00:00:10 itd. Wywołanie w postaci

SetInterval(funkcja,5000,1000) spowoduje wywoływanie funkcji o 00:00:01, 00:00:06, 00:00:11 itd. Jeżeli *Interval* jest liczbą ujemną to *funkcja* wykona się tylko jeden raz.

Metoda ***ExecuteAt*** ma postać:

ExecuteAt(*funkcja*, *czas*)

Czas określony jest jako czas absolutny *(patrz: 5.16. Uwagi na temat czasu).*

Listing 23. Przykład skryptu cyklicznego wywoływania funkcji.

VBScript

```
Call Asix.SetInterval( GetRef("Timer"), 1000 )

'Funkcja wykonuje się co 1 sekundę
Function Timer()
'.....
End Function
```

JScript

```
Asix.SetInterval( Timer, 1000 );

//Funkcja wykonuje się co 1 sekundę
function Timer()
{
//.....
}
```

Powiązanie funkcji z czasem usuwa się za pomocą funkcji *ClearInterval*.

5.13. Dostęp do zasobów zewnętrznych

Skrypty wykonywane przez moduł skryptów mogą mieć dostęp do zasobów zewnętrznych takich jak arkusze kalkulacyjne, poczta elektroniczna itp. Dostęp do zasobów zewnętrznych odbywa się za pomocą obiektów tworzonych przez funkcje wbudowane w język skryptów.

Poniżej pokazano uruchamianie programu Excel.

Listing 24. Skrypt uruchamiający program Excel.

VBScript

```
Dim Excel
Set Excel = CreateObject("Excel.Application")
```

Jscript

```
var Excel = new ActiveXObject("Excel.Application");
```

Oprócz dostępu do obiektów zewnętrznych nie związanych bezpośrednio ze skryptami, istnieją obiekty, których głównym celem jest rozszerzenia zakresu funkcji realizowanych przez skrypt. Podstawowym przykładem takich obiektów jest *FileSystemObject*, który umożliwia dostęp do plików i katalogów.

Obiekt *FileSystemObject* tworzony jest w poniższy sposób.

Listing 25. Przykład skryptu tworzącego obiekt 'FileSystemObject'.

VBScript

```
Dim fso
Set fso = CreateObject("Scripting.FileSystemObject")
```

Jscript

```
var fso = new ActiveXObject("Scripting.FileSystemObject ");
```

Innym przykładem są obiekty udostępniane przez Windows Script Host, które umożliwiają dostęp do zasobów sieciowych, rejestru systemu, uruchamianie programów i inne funkcje. Poniżej podano listę niektórych obiektów.

WshShell	- pozwala tworzyć procesy oraz skróty. Zapewnia dostęp do rejestru systemu operacyjnego. Pozwala zapisywać informacje do systemowego pliku log.
WshEnvironment	- zapewnia dostęp do zmiennych środowiskowych.
WshSpecialFolders	- zapewnia dostęp do katalogów takich jak „Pulpit", „Moje dokumenty" itp.

Poniżej zamieszczony został przykład dostępu do zmiennych środowiskowych systemu.

Listing 26. Przykład skryptu powodującego wyprowadzenie komunikatu o ilości procesorów systemu komputerowego.

VBScript

```
Set WshShell = CreateObject("WScript.Shell")
Set Env = WshShell.Environment("SYSTEM")
Asix.Panel.Message( "Ilość procesorów="_
      &Env("NUMBER_OF_PROCESSORS"))
```

Jscript

```
WshShell = new ActiveXObject("WScript.Shell");
Env = WshShell.Environment("SYSTEM");
Asix.Panel.Message( "Ilość procesorów=" +
      Env("NUMBER_OF_PROCESSORS"));
```

5.14. Wymiana danych pomiędzy skryptami

Obiekt *Asix* posiada dwa pola *GlobalData* i *GlobalThreadData*, które umożliwiają wymianę danych pomiędzy skryptami. Oba pola zwracają obiekty, które posiadają tylko takie składowe, jakie zostały im przypisane przez skrypty. Składowe obiektu *GlobalData* są dostępne wszystkim skryptom obsługiwanym przez moduł skryptów. Ponieważ czas dostępu do obiektu jest długi, został wprowadzony obiekt *ThreadGlobalData*. Każdy wątek utworzony przez moduł skryptów posiada jeden własny obiekt *ThreadGlobalData*. Za pośrednictwem tego obiektu mogą być wymieniane dane pomiędzy skryptami należącymi do tego samego wątku. Czas dostępu do obiektu *GlobalData* jest około 30 razy dłuższy niż do obiektu *GlobalThreadData*.

Utworzenie nowej składowej obiektu globalnego następuje poprzez utworzenie nowej składowej dynamicznej w sposób opisany w punkcie *5.4. Dynamiczne składowe obiektów*. Dane mogą być wymieniane pomiędzy skryptami napisanymi w różnych językach.

5.15. Realizacja bibliotek funkcji

Ponieważ dynamiczne dodawanie składowych dotyczy zarówno pól jak i metod, to może zostać ono wykorzystane do tworzenia bibliotek funkcji używanych przez inne skrypty. Umożliwia to również korzystanie z funkcji napisanych w innych językach programowania. Aby utworzyć bibliotekę należy zgromadzić funkcje biblioteczne w oddzielnym pliku i przypisać je do obiektu globalnego *GlobalData* lub *ThreadGlobalData*. Przykładowo skrypt o nazwie „Sum.js" może zawierać funkcję o nazwie „Sum", która

sumuje argumenty i zwraca wynik. Funkcja „Sum" może być następnie wykorzystana w skrypcie VBScript.

Zawartość pliku „Sum.js":

Listing 27. Przykład skryptu "Sum.js" z funkcją sumującą argumenty i zwracającą wynik.

```
Asix.GlobalData.Sum = Sum;  //Pola obiektu „GlobalData” są dostępne dla wszystjich skryptów

function Sum(a,b)
{
  return a+b;
}
```

Powyższą funkcję możemy wykorzystać w skrypcie zawartym w pliku „Licz.vbs" w języku VBScript:

```
Asix.Panel.Message( "Sum=" + CStr(Asix.GlobalData.Sum(5,2)))
```

Uruchomienie skryptu zawartego w pliku „Licz.vbs" spowoduje wyprowadzenie do panelu kontrolnego systemu **asix** komunikatu o treści „sum=7".

Aby powyższy przykład zadziałał poprawnie, to część inicjalizacyjna skryptu „Sum.js" musi wykonać się przed częścią inicjalizacyjną skryptu „Licz.vbs". W związku z tym deklaracje skryptów w parametrze ***Skrypty*** pliku konfiguracyjnego aplikacji muszą zostać umieszczone we właściwej kolejności:

```
Sum = Sum.js
Licz = Licz.vbs
```

Powyższy przykład zadziała poprawnie tylko wtedy, gdy zarówno skrypt biblioteczny jak i skrypt korzystający ze skryptu bibliotecznego znajdują się w tym samym wątku, ponieważ tylko wtedy kolejność deklaracji skryptów odpowiada kolejności inicjalizacji skryptów. Jeśli skrypt biblioteczny zostałby umieszczony w innym wątku, to jego inicjalizacja przebiegałaby współbieżnie z inicjalizacją skryptu używającego funkcji bibliotecznych, co z kolei mogłoby doprowadzić do błędu dostępu do niezainicjowanych składowych obiektu *GlobalData*.

W przypadku, gdy skrypt biblioteczny i skrypt użytkowy znajdują się w tym samym wątku, należy korzystać z obiektu „ThreadGlobalData" zamiast „GlobalData". Czas dostępu do obiektu „GlobalData" jest około 30 razy dłuższy niż do obiektu GlobalThreadData.

Wykorzystując fakt, że wewnątrz metody jest dostępny za pomocą słów **Me** i **this** obiekt, którego jest ona składową, w module bibliotecznym można umieścić metody, które będą działały w kontekście obiektu *Variable*.

Listing 28. Przykład skryptu modułu bibliotecznego.

VBScript

```
Asix.GlobalData.Set(„Przelicznik") = _
                              GetRef(„Przelicznik")
Function Przelicznik()
'dostęp do wartości zmiennej poprzez Me
End Function
```

JScript

```
Asix.GlobalData.Przelicznik = Przelicznik;
function Przelicznik()
{
 //dostęp do wartości zmiennej poprzez this
}
```

Listing 29. Przykład skryptu modułu wykorzystującego bibliotekę.

VBScript

```
Set Var = Asix.Variables("Emisja_CO2")
Var.Set("Przelicznik") = GlobalData.Przelicznik
Var.OnRead = GetRef(„OnRead")
Function OnRead()
    Me.Przelicznik( ) 'wywołanie biblioteki
End Function
```

JScript

```
var Var= Asix.Variables( "Emisja_CO2" );
Var.Przelicznik = GlobalData.Przlicznik;
Var.OnRead= OnRead; //wywołanie dynamicznej metody
function OnRead()
{
    this.Przelicznik(); //wywołanie biblioteki
}
```

To samo można uzyskać, gdy funkcja przelicznik ma parametr odnoszący się do obiektu *Variable*.

Listing 30. Przykład skryptu modułu bibliotecznego (gdy funkcja przelicznik ma parametr odnoszący się do obiektu Variable).

VBScript

```
Asix.GlobalData.Set(„Przelicznik") = _
                              GetRef(„Przelicznik")
Function Przelicznik( Var )
'dostęp do wartości zmiennej poprzez parametr Var
End Function
```

JScript

```
Asix.GlobalData.Przelicznik = Przelicznik;
function Przelicznik( Var )
{
 //dostęp do wartości zmiennej poprzez parametr Var
}
```

Moduł wykorzystujący bibliotekę:

Listing 31. Przykład skryptu modułu wykorzystującego bibliotekę (gdy funkcja przelicznik ma parametr odnoszący się do obiektu Variable).

VBScript

```
Set Var = Asix.Variables("Emisja_CO2")
Var.OnRead = GetRef(„OnRead")
Function OnRead(val, stat, time)
    call Asix.GlobalData.Przelicznik(Me) 'wywołanie
                                         'biblioteki
End Function
```

JScript

```
var Var= Asix.Variables( "Emisja_CO2" );
Var.OnRead= OnRead; //wywołanie dynamicznej metody
function OnRead(val, stat, time)
{
   Asix.GlobalData.Przelicznik(this);//wywołanie
    //biblioteki
}
```

W powyższym przykładzie zastosowano niekorzystne ze względu na czas wykonania rozwiązanie polegające na każdorazowym odwoływaniu się do obiektu *GlobalData* wewnątrz funkcji obsługi zdarzenia. Bardziej efektywne rozwiązanie polega na zapamiętaniu funkcji *Przelicznik* w zmiennej lokalnej skryptu w celu późniejszego wywołania:

Listing 32. Przykład skryptu modułu wykorzystującego bibliotekę (funkcja przelicznik zapamiętywana jest w zmiennej lokalnej skryptu w celu późniejszego wywołania).

VBScript

```
Set Var = Asix.Variables("Emisja_CO2")
Set Przelicznik = Asix.GlobalData.Get(„Przelicznik”)
Var.OnRead = GetRef(„OnRead”)
Function OnRead(val, stat, time)
    call Przelicznik(Me) ‘wywołanie biblioteki
End Function
```

Jscript

```
var Var= Asix.Variables( "Emisja_CO2" );
var Przelicznik= Asix.GlobalData.Przelicznik;
Var.OnRead= OnRead; //wywołanie dynamicznej metody
function OnRead(val, stat, time)
{
   Przelicznik(this);//wywołanie biblioteki
}
```

Oczywiście, w przypadku, gdy funkcja *Przelicznik* dokonuje prostego przeliczenia wartości pomiarowej i nie wymaga dostępu do innych pól obiektu *Variable*, to zamiast przesyłać obiekt *Variable*, można po prostu przesłać wartość zmiennej (i ew. parametry):

Listing 33. Przykład skryptu z zastosowaniem funkcji 'Przelicznik' (przesyłanie wartości zmiennej zamiast obiektu 'Variable').

VBScript

```
....
Function OnRead(val, stat, time)
   call Przelicznik(val) ‘wywołanie biblioteki
End Function
```

JScript

```
...
function OnRead(val, stat, time)
{
   Przelicznik(val);//wywołanie biblioteki
}
```

5.16. Uwagi na temat czasu

Niektóre metody i pola zwracają czas danej pomiarowej, alarmu itp. Czas jest wyrażony w formacie VT_DATE. Format ten jest wykorzystywany przez wiele obiektów ActiveX oraz język VBScript. W języku JScript można go wykorzystać przekazując bezpośrednio do innych funkcji modelu obiektowego wymagających podania czasu. Aby można było dokonywać obliczeń na czasie w tym formacie w języku JScript, należy go przekształcić na format czasu wykorzystywany przez obiekt *Date* języka JScript.

Jeśli, na przykład, chcemy wykonać w języku JScript obliczenia arytmetyczne na czasie zmiennej „Var" to możemy przekonwertować go w następujący sposób:

```
var when = new Date(Var.Time);
```

Możemy teraz wykorzystać wszystkie metody obiektu *Date* do wykonania obliczeń (metody obiektu *Date* są opisane w dokumentacji języka JScript). Jeśli chcemy wyznaczyć chwilę czasu o 20 sekund poprzedzającą czas określony zmienną „when" z powyższego przykładu, to możemy to zrobić w następujący sposób:

```
when.setTime( when.getTime() -20000 );
```

Można dowolnie wykorzystywać funkcje operujące na czasie wbudowane w dany język. Należy tutaj zwrócić uwagę na przekazywanie czasu **do** modułu skryptów. Powinien on być przekazywany w formacie VT_DATE. Dotyczy to zwłaszcza języka JScript, w którym użycie zmiennej (w znaczeniu języka JScript) zawierającej czas lub funkcji zwracającej

czas (np. czas jako obiekt *Date*), może powodować przekazanie tekstu reprezentującego czas zamiast wartości czasu.

Na przykład metoda *Write* obiektu *Variable* wywołana w poniższy sposób:

```
var when = new Date();
obiekt.Write( 5, OPC_QUALITY_GOOD,when );
```

spowoduje przekazanie tekstu zamiast wartości czasu. Należy posłużyć się metodą „getVarDate()" do przekazania czasu w poprawnym formacie:

```
obiekt.Write( 5,OPC_QUALITY_GOOD,when.getVarDate() );
```

Dotyczy to wszystkich metod i pól wymagających przekazania czasu do modułu skryptów w języku JScript. W poniższym przykładzie w języku JScript użycie metody „getVarDate" jest zbędne ponieważ metoda *Message* obiektu *Panel* wymaga łańcucha znaków, a nie czasu jako parametru:

```
var when = new Date();
Asix.Panel.Message( „Teraz jest " + when.getVarDate() );
```

Wyprowadzony przez metodę *Message* komunikat będzie miał poprawną postać, ale taki sam efekt można uzyskać wykonując instrukcję:

```
Asix.Panel.Message( „Teraz jest " + when );
```

W obydwu przypadkach język JScript dokonuje konwersji czasu do postaci łańcucha znaków i wynik konwersji oraz sumowania łańcuchów przekazuje do metody *Message*.

W języku VBScript nie jest wymagana dodatkowa konwersja czasu przed przekazaniem go do modułu skryptów.

Czas wykorzystywany w module skryptów jest czasem lokalnym. Może to prowadzić do nieprawidłowości spowodowanej niejednoznacznością czasu w przypadku cofania czasu o jedną godzinę przy zmianie na czas zimowy. Dotyczy to szczególnie odczytu danych historycznych. Pole ***UTCMode*** obiektu ***Variable*** pozwala zmienić sposób reprezentacji czasu we wszystkich polach i parametrach metod obiektu ***Variable*** dotyczących czasu. Po ustawianiu tego pola na wartość **true**, czas będzie wyrażony jako czas UTC (ang. Coordinated Universal Time, poprzednio nazywany Greenwich Mean Time). Pozwala to uniknąć niejednoznaczności w okresie zmiany czasu na czas zimowy, ale tylko wtedy, gdy czas danych pomiarowych bieżących oraz archiwalnych jest również wyrażony jako czas UTC. Należy zwrócić uwagę, że metody „getUTC...." języka JScript nie będą działać poprawnie, jeśli pole to ma wartość **true**. Jeśli pole to ma wartość **true**, to pole *Status* obiektu ***Variable*** będzie miało ustawiony bit AS_STAT_UTC_TIME. Jeśli parametr *Status* metody ***Write*** obiektu ***Variable*** ma ustawiony bit AS_STAT_UTC_TIME, to parametr *Time* powinien zawierać czas UTC.

6. Model obiektowy modułu skryptów

W kolejnych rozdziałach przedstawiono wszystkie obiekty udostępniane przez moduł skryptów.

6.1. Obiekt Application

Dostęp do obiektu *Application* uzyskuje się poprzez pole *Application* obiektu *Asix*.

6.1.1. Pola

Pole ***Direktory***

Pole zwraca nazwę katalogu startowego aplikacji.

Pole ***Name***

Pole zwraca nazwę aplikacji. Jest to główna część nazwy pliku inicjalizacyjnego.

Pole ***Language***

Pole zwraca język aplikacji (np. „pl" dla języka polskiego).

Pole ***IniFileName***

Pole zwraca pełną nazwę pliku inicjalizacyjnego aplikacji.

6.1.2. Metody

Obiekt nie posiada metod.

6.1.3. Zdarzenia

Obiekt nie generuje zdarzeń.

6.1.4. Składowe zarezerwowane

Składowe zarezerwowane mają nazwy: Extra, Explicite, Diagnostics, Close, Catalogue, Catalog.

6.2. Obiekt Alarms

Dostęp do obiektu *Alarms* uzyskuje się poprzez pole *Alarms* obiektu *Asix*.

6.2.1. Pola

Obiekt nie posiada pól.

6.2.2. Metody

Metoda ***Cancel***

Postać wywołania:

obiekt.Cancel(*ident* [, *wart1*[,*wart2*[,*czas*[,*global*]]]])

gdzie:

obiekt	- obiekt „Alarms";
identyfikator	- identyfikator alarmu;
wart1	- parametr alarmu; jeśli parametr jest pominięty, to przyjmowana jest wartość 0;
wart2	- parametr alarmu; jeśli parametr jest pominięty, to przyjmowana jest wartość 0;
czas	- czas zaniku alarmu; jeśli parametr jest pominięty, to przyjmowany jest czas bieżący;
global	- wartość logiczna określająca czy alarm ma zakres globalny (**true**) czy lokalny (**false**); jeśli parametr jest pominięty, to przyjmowana jest wartość **false**.

Metoda powoduje skasowanie alarmu o podanym identyfikatorze.

Jeśli metoda została pomyślnie wykonana, to zwracana jest wartość **true**. W przeciwnym wypadku zwracana jest wartość **false**.

Parametry *wart1* i *wart2* są parametrami liczbowymi, które przekazywane są do systemu alarmów **asix**'a. Wartości te mogą być wstawione do wyświetlanego tekstu alarmu. Wymaga to odpowiedniej definicji alarmu (szczegółowe informacje znajdują się w opisie systemu alarmów **asix**'a).

Parametr *global* ma znaczenie tylko dla sieciowych, redundancyjnych instalacji systemu alarmów. Alarmów globalnych należy używać wtedy, gdy skrypt wykrywający zdarzenia instalowany jest tylko na jednym komputerze i jego zgłoszenia mają być akceptowane niezależnie od aktualnego stanu systemu. Alarmy lokalne używane są wtedy, gdy skrypt jest uruchamiany na wszystkich stanowiskach systemu, a jego zgłoszenia będą akceptowane tylko na stanowisku, które w danej chwili pełni rolę stanowiska aktywnego.

Metoda ***Raise***

Postać wywołania:

obiekt.Raise(ident [, wart1[,wart2[,czas[,global]]]])

gdzie:

obiekt	- obiekt „Alarms";
identyfikator	- identyfikator alarmu;
wart1	- parametr alarmu; jeśli parametr jest pominięty, to przyjmowana jest wartość 0;
wart2	- parametr alarmu; jeśli parametr jest pominięty, to przyjmowana jest wartość 0;
czas	- czas zgłoszenia alarmu; jeśli parametr jest pominięty, to przyjmowany jest czas bieżący;
global	- wartość logiczna określająca, czy alarm ma zakres globalny (**true**) czy lokalny (**false**); jeśli parametr jest pominięty, to przyjmowana jest wartość **false**.

Metoda powoduje ustawienie alarmu.

Jeśli funkcja została pomyślnie wykonana, to zwracana jest wartość **true**. W przeciwnym wypadku zwracana jest wartość **false**.

Metoda ***State***

Postać wywołania:

obiekt.State(ident)

gdzie:

obiekt	- obiekt „Alarms";
identyfikator	- identyfikator alarmu.

Metoda zwraca wartość **true**, jeśli alarm jest ustawiony, oraz **false** w przeciwnym wypadku.

6.2.3. Zdarzenia

Obiekt nie generuje zdarzeń.

6.2.4. Składowe zarezerwowane

Składowe zarezerwowane mają nazwy: Extra, Explicite, Diagnostics, OnNew.

6.3. Obiekt Asix

Obiekt *Asix* jest bezpośrednio dostępny przez każdy skrypt wykonywany pod kontrolą modułu skryptów systemu **asix**.

6.3.1. Pola

Pole ***Alarms***

Pole zwraca obiekt typu ***Alarms***, który umożliwia dostęp do systemu alarmów systemu **asix**.

Pole ***Application***

Pole zwraca obiekt typu ***Application,*** który umożliwia dostęp do informacji na temat wykonywanej aplikacji.

Pole ***GlobalData***

Pole zwraca obiekt typu ***GlobalData***, który umożliwia wymianę danych pomiędzy skryptami. Skrypty mogą znajdować się w różnych wątkach.

Pole ***Mode***

Pole zwraca liczbę określającą bieżący tryb pracy systemu **asix**. Pole może mieć następujące wartości symboliczne:

Tabela 3. Wartości symboliczne pola 'Mode' obiektu 'Asix'.

Wartość symboliczna	Wartość numeryczna	Znaczenie
Application	2	**asix** pracuje w trybie normalnym (wykonywanie aplikacji)
Initialize	3	**asix** jest w trakcie inicjalizacji
Terminate	4	**asix** w trakcie kończenia pracy

Pole ***Panel***

Pole zwraca obiekt typu ***Panel***, który umożliwia wyprowadzanie informacji do panelu kontrolnego systemu **asix**.

Pole ***Script***

Pole zwraca obiekt typu ***Script***, który umożliwia dostęp do informacji o aktualnie wykonywanym skrypcie.

Pole ***Scripter***

Pole zwraca obiekt typu ***Scripter***, który umożliwia dostęp do informacji o module skryptów AsScripter.

Pole ***ThreadGlobalData***

Pole zwraca obiekt typu ***ThreadGlobalData***, który umożliwia wymianę informacji pomiędzy skryptami znajdującymi się w tym samym wątku.

Pole ***Version***

Pole *Version* zwraca numer wersji systemu **asix** w postaci numer_wersji*256 + numer_podwersji.

6.3.2. Metody

Metoda ***ClearInterval***

Postać wywołania:

obiekt.ClearInterval(*identyfikator*)

gdzie:

obiekt	- obiekt *Asix*;
identyfikator	- liczba zwrócona przez metodę *SetInterval* lub *ExecuteAt*.

Metoda anuluje skutek wykonania funkcji *SetInterval* lub *ExecuteAt*.

Metoda ***CreateEmptyObject***

Metoda jest przeznaczona dla skryptów w języku VBScript. Zwraca pusty obiekt, do którego skrypt możne dodawać nowe składowe. Obiekt pozwala przesłać grupę logicznie powiązanych składowych (np. zmiennych) do innego skryptu za pośrednictwem obiektu *GlobalData*. Obiekt jest kolekcją swoich składowych.

Metoda ***ExecuteAction***

Postać wywołania:

obiekt.ExecuteAction(*akcja*)

gdzie:

obiekt	- obiekt *Asix*;
akcja	- łańcuch znaków przekazywany do systemu **asix** i określający akcję, jaka ma zostać wykonana.

Zestaw dostępnych akcji jakie mogą być wykonane za pomocą metody *ExecuteAction* znajduje się w dokumentacji systemu **asix**.

Metoda ***ExecuteAt***

Postać wywołania:

obiekt.ExecuteAt(*funkcja, czas*)

gdzie:

obiekt	- obiekt *Asix*;
funkcja	- funkcja, jaka ma zostać wykonana;
czas	- czas, w jakim ma zostać wykonana funkcja.

Metoda zwraca identyfikator, który może służyć do anulowania skutku wykonania tej funkcji (*ClearInterval*). Po wywołaniu funkcji określonej parametrem *funkcja* identyfikator przestaje być ważny. Jeśli *czas* odnosi się do przeszłości, to funkcja *funkcja* zostanie

wykonana natychmiast (a dokładniej, po zakończeniu wykonywanego w chwili bieżącej fragmentu skryptu).

Metoda ***Parameters***

Postać wywołania:

obiekt.Parameters(*nazwa_sekcji*)

gdzie:

obiekt	- obiekt *Asix*;
nazwa_sekcji	- łańcuch znaków określający nazwę sekcji.

Metoda *Parameters* pozwala na przetwarzanie pliku inicjalizacyjnego aplikacji. Jako parametr podaje się nazwę sekcji pliku inicjalizacyjnego. W wyniku wywołania zostanie zwrócona kolekcja obiektów typu *Parameter*.

Metoda ***SetInterval***

Postać wywołania:

obiekt.SetInterval(*funkcja, interwał*[,*przesunięcie*])

gdzie:

obiekt	- obiekt *Asix*;
funkcja	- funkcja wywoływana cyklicznie;
interwał	- odstęp czasu w milisekundach pomiędzy kolejnymi wywołaniami funkcji *funkcja*; jeśli i*nterwał* jest liczbą ujemną, to *funkcja* wykona się tylko jeden raz;
przesunięcie	- czas przesunięcia w milisekundach. Parametr określa czas przesunięcia w stosunku do pełnej wielokrotności interwału (np. jeśli *interwał* wynosi 1 godz., a *przesunięcie* 15 minut, to funkcja będzie uruchamiana o 0:15, 1:15, 2:15 itd.) Jeśli parametr jest pominięty, to funkcja będzie wywoływana bez wyrównania czasu.

Metoda zwraca identyfikator, który może zostać wykorzystany w funkcji *ClearInterval* w celu anulowania skutku wywołania funkcji *SetInterval*. Jeśli parametr *interwał* jest liczbą ujemną to identyfikator zwrócony przez funkcję *SetInterval* zostanie unieważniony po pierwszym wywołaniu funkcji *funkcja*.

Metoda ***Sleep***

Postać wywołania:

obiekt.Sleep(*milisekundy*)

gdzie:

obiekt	- obiekt *Asix*;
milisekundy	- czas zawieszenia wykonywania skryptu w milisekundach.

Metoda powoduje zawieszenie wykonania skryptu na czas określony parametrem. W czasie trwania funkcji ograniczenia czasowe **są** naliczane. W czasie wykonania funkcji skrypt **nie** reaguje na zdarzenia. W czasie wykonywania tej funkcji nie wykonują się również inne skrypty pracujące w tym samym wątku. Metoda *Sleep* nie ma wpływu na

działanie skryptów w innych wątkach. Parametrem funkcji jest czas wyrażony w milisekundach.

Metoda ***Variables***

Postać wywołania:

obiekt.Variables(*nazwa_zmiennej[, bieżąca]*)

gdzie:

obiekt	- obiekt *Asix*;
nazwa_zmiennej	- łańcuch znaków określający nazwę zmiennej; na końcu nazwy zmiennej po przecinku można podać jednoliterowy kod archiwum, jeśli zmienna będzie wykorzystywana do przeglądu archiwum;
bieżąca	- wartość logiczna określająca, czy wymagany jest dostęp do bieżących wartości zmiennej. Parametr definiuje działanie metody *Variables* w sytuacji, gdy brak jest zmiennej określonej parametrem *nazwa_zmiennej* w bazie zmiennych modułu ASMEN. Jeśli podano **true,** to zmienna musi znajdować się w bazie danych modułu ASMEN. Jej brak spowoduje, że metoda *Variables* zakończy się niepomyślnie – obiekt *Variable* nie zostanie utworzony. Jeśli podano **false**, to zmienna może być nieobecna w bazie danych modułu ASMEN, musi być jednak obecna w archiwum modułu ASPAD. W takim przypadku *nazwa_zmiennej* musi zawierać kod typu archiwum. Jeśli parametr jest pominięty, to przyjmowana jest wartość **true**, tzn. wymagany jest dostęp do wartości bieżących.

Metoda zwraca obiekt *Variable*, który pozwala na dostęp do danych pomiarowych bieżących i/lub archiwalnych. Każdorazowe wywołanie metody z tymi samymi argumentami powoduje zwrócenie nowego obiektu odwołującego się do tej samej zmiennej systemu **asix**. W przypadku, gdy wartością parametru *bieżąca* jest **true** oraz brak jest zmiennej w bazie danych modułu ASMEN, to metoda zwraca obiekt pusty. Metoda zawsze zwraca obiekt pusty, jeśli zmienna nie występuje w bazie danych modułu ASMEN ani nie jest archiwizowana (nie występuje w bazie danych modułu ASPAD). W przypadku pomyślnego utworzenia obiektu *Variable* pole ***WithArchive*** obiektu ***Info*** pozwala określić, czy jest możliwy dostęp do wartości archiwalnych zmiennej.

Jeśli zmienna znajduje się w bazie zmiennych modułu ASMEN, to pole *Current* nowo utworzonego obiektu *Variable* będzie miało wartość **true** oraz pola *Value*, *Status* i *Time* zostaną ustawione odpowiednio do ostatnio wykonanego odczytu wartości zmiennej przez moduł ASMEN. Pole ***WithArchive*** obiektu ***Info*** będzie miało wartość **true**, jeśli parametr *nazwa_zmiennej* zawiera kod typu archiwum oraz zmienna jest archiwizowana w archiwum tego typu.

W przypadku potrzeby dostępu do wartości archiwalnych zmiennych, które nie posiadają wartości bieżących, tj. nie znajdują się w bazie danych ASMENA, należy podać **false** jako wartość parametru *bieżąca*. Pole *Current* tak utworzonego obiektu *Variable* będzie stale zawierać wartość **false**, a pole *Status* będzie początkowo zawierać wartość OPC_QUALITY_BAD. Dopiero wykonanie metod *ArcRead* lub *ArcReadAt* spowoduje odpowiednie ustawienie pól *Value*, *Status* i *Time*. Takie zachowanie obiektu *Variable* dotyczy jedynie sytuacji, gdy zmienna nie posiada wartości bieżących. Jeśli zmienna znajduje się w bazie zmiennych modułu ASMEN, to wartość parametru *bieżąca* nie ma znaczenia. Obecnie, tylko zmienne w archiwum przebiegów wzorcowych mogą nie być zawarte w bazie zmiennych modułu ASMEN.

Jeśli utworzenie obiektu *Variable* nie jest możliwe, to metoda zwraca obiekt pusty. Poniżej przedstawiono sposób w jaki można zweryfikować rezultat wykonania metody.

Listing 34. Przykład skryptu przedstawiającego sposób weryfikowania metody 'Variables'.

VBScript

```
Set Var = Asix.Variables("Emisja_CO2")
If TypeName(Var) = "Nothing" Then
    Asix.Panel.Message("Błędna nazwa zmiennej" )
End If
```

JScript

```
var Var= Asix.Variables( "Emisja_CO2" );
if ( Var == null )
    Asix.Panel.Message("Błędna nazwa zmiennej" );
```

6.3.3. Zdarzenia

Zdarzenie ***OnTerminate***

Zdarzenie jest generowane w chwili zakończenia aplikacji. Zdarzenie wystąpi przed zdarzeniem *OnTerminate* obiektu *Script*.

6.3.4. Składowe zarezerwowane

Składowe zarezerwowane mają nazwy: *Test, Freeze, Await, BreakAwait, IsAwaiting, Quit, Extra, Explicite, OnASIXPhase, Diagnostics, Modules, Drivers, ConnectObject, CreateObject, Masks, Pictures, Dump* i *Utils*.

6.4. Obiekt GlobalData

Dostęp do obiektu *GlobalData* uzyskuje się poprzez pole *GlobalData* obiektu *Asix*.

Obiekt służy do przekazywania danych pomiędzy skryptami. W celu wymiany danych pomiędzy skryptami należy wybrać nazwę składowej, a następnie przypisać jej wartość. Składowe obiektu *GlobalData* są dostępne dla wszystkich skryptów także tych, które znajdują się w różnych wątkach. Obiekt jest kolekcją swoich składowych.

Składowymi obiektu mogą być dowolne dane, także inne obiekty (w tym funkcje skryptu).

6.4.1. Pola

Obiekt posiada tylko takie pola, jakie zostały mu nadane przez skrypty.

6.4.2. Metody

Obiekt posiada tylko takie metody, jakie zostały mu nadane przez skrypty.

6.4.3. Zdarzenia

Obiekt posiada tylko takie zdarzenia, jakie zostały mu nadane przez skrypty.

6.4.4. Składowe zarezerwowane

Składowe zarezerwowane mają nazwy: *Extra, Explicite* i *Diagnostics*.

6.5. Obiekt Info

Dostęp do obiektu *Info* uzyskuje się poprzez pole *Info* obiektu *Variable*.

6.5.1. Pola

Pole ***ArcAttr***

Pole zwiera atrybuty archiwizacji zmiennej. Obecnie, pole może zawierać sumę logiczną następujących wartości:

VARATTR_NOPACK	- powtarzające się wartości zmiennej nie są pakowane;
VARATTR_SIM	- wartości zmiennej są symulowane;
VARATTR_RESTORE	- ostatnia wartość archiwalna jest początkową wartością zmiennej po uruchomieniu systemu;
VARATTR_ALL	- zapisywane są wszystkie wartości.

Pole ***ArcCycle***

Pole zwiera cykl archiwizacji zmiennej wyrażony w sekundach.

Pole ***Arcdt***

Pole zwiera dokładność czasową rejestracji zmiennej wyrażoną w sekundach.

Pole ***Arcdx***

Pole zwiera minimalną zmianę wartości zmiennej, która powoduje jej zapis w archiwum.

Pole ***ArchiveType***

Pole zwraca łańcuch określający typ archiwum (np. „M"), w którym jest archiwizowana zmienna.

Pole ***ArcHorizon***

Pole zwiera horyzont archiwizacji zmiennej.

Pole ***Count***

Pole zwiera ilość elementów z jakich składa się zmienna. Dla zwykłych zmiennych pole to ma wartość 1. Dla zmiennych tablicowych pole określa ilość elementów w tablicy.

Pole ***RefreshPeriod***

Pole zwiera cykl odświeżania zmiennej wyrażony w sekundach.

Pole ***WithArchive***

Pole zwraca wartość logiczną **true**, jeśli jest możliwy dostęp do archiwum oraz **false** w przeciwnym wypadku.

6.5.2. Metody

Obiekt nie posiada metod.

6.5.3. Zdarzenia

Obiekt nie generuje zdarzeń.

6.6. Obiekt Panel

Dostęp do obiektu *Panel* uzyskuje się poprzez pole *Panel* obiektu *Asix*.

Obiekt *Panel* umożliwia wysyłanie informacji do Panelu Kontrolnego systemu **asix**.

6.6.1. Pola

Pole ***Ident***

Pole *Ident* to łańcuch znaków identyfikujący źródło komunikatów wyprowadzanych przez obiekt *Panel*. Jeśli polu nie nadano innej wartości, to moduł skryptów nadaje jej wartość taką samą jak nazwa skryptu określona w deklaracji skryptu.

6.6.2. Metody

Metoda ***Error***

Postać wywołania:

obiekt.Error(*tekst[,diag]*)

gdzie:

obiekt - obiekt *Panel*;
tekst - łańcuch znaków określający wyświetlany komunikat;
diag - jeśli parametr ma wartość logiczną **true**, to komunikat zostanie zapisany tylko w pliku logu aplikacji systemu **asix**. Jeśli parametr jest pominięty, to jest przyjmowana wartość **false**, tzn. komunikat zostanie wyprowadzony do panelu kontrolnego.

Metoda powoduje wyprowadzenie komunikatu do panelu kontrolnego systemu **asix**. Komunikat ma status „BŁĄD".

Metoda ***Message***

Postać wywołania:

obiekt.Message(*tekst[,diag]*)

gdzie:
obiekt - obiekt *Panel*;
tekst - łańcuch znaków określający wyświetlany komunikat;
diag - jeśli parametr ma wartość logiczną **true**, to komunikat zostanie zapisany tylko w pliku logu aplikacji systemu **asix**. Jeśli parametr jest pominięty, to jest przyjmowana wartość **false**, tzn. komunikat zostanie wyprowadzony do panelu kontrolnego.

Metoda powoduje wyprowadzenie komunikatu do panelu kontrolnego systemu **asix**. Komunikat ma status „ZWYKŁY".

Metoda ***Popup***

Postać wywołania:

obiekt.Popup(*tekst*)

gdzie:
obiekt - obiekt *Panel*;
tekst - łańcuch znaków określający wyświetlany komunikat.

Metoda powoduje otwarcie okienka informacyjnego zawierającego tekst określony parametrem *tekst*.

Metoda ***Warning***

Postać wywołania:

obiekt.Warning(*tekst[,diag]*)

gdzie:
obiekt - obiekt *Panel*;
tekst - łańcuch znaków określający wyświetlany komunikat;
diag - jeśli parametr ma wartość logiczną **true**, to komunikat zostanie zapisany tylko w pliku logu aplikacji systemu **asix**. Jeśli parametr jest pominięty, to przyjmowana jest wartość **false**, tzn. komunikat zostanie wyprowadzony do panelu kontrolnego.

Metoda powoduje wyprowadzenie komunikatu do panelu kontrolnego systemu **asix**. Komunikat ma status „OSTRZEŻENIE".

6.6.3. Zdarzenia

Obiekt nie generuje zdarzeń.

6.6.4. Składowe zarezerwowane

Składowe zarezerwowane mają nazwy: *Extra, Explicite* i *Diagnostics*.

6.7. Obiekt Parameter

Obiekt *Parameter* reprezentuje parametr umieszczony w sekcji pliku inicjalizacyjnego lub parametr przekazany do skryptu w deklaracji skryptu. W ostatnim przypadku pole *Name* może być puste. Obiekt *Parameter* jest elementem kolekcji jaką jest obiekt *Parameters*.

6.7.1. Pola

Pole **domyślne**

Polem domyślnym jest pole *Value*.

Pole ***Name***

Pole jest łańcuchem znaków określającym nazwę parametru.

Pole ***Value***

Pole jest łańcuchem znaków określającym wartość parametru.

6.7.2. Metody

Obiekt nie posiada metod.

6.7.3. Zdarzenia

Obiekt nie generuje zdarzeń.

6.7.4. Składowe zarezerwowane

Obiekt nie zezwala na dynamiczne dodawanie nowych składowych.

6.8. Obiekt Parameters

Obiekt jest kolekcją obiektów *Parameter* reprezentującą parametry sekcji pliku inicjalizacyjnego lub parametry przekazane do skryptu w deklaracji skryptu. Parametry są umieszczone w kolejności ich występowania w pliku inicjalizacyjnym lub deklaracji skryptu. Obiekt *Parameters* jest zwracany przez metodę *Parameters* obiektu *Asix* oraz przez pole *Arguments* obiektu Script.

6.8.1. Pola

Obiekt posiada pola, które posiada kolekcja tj. pole *Count* określające ilość parametrów w kolekcji.

6.8.2. Metody

Obiekt posiada metody, które posiada każda kolekcja tj. metoda *Item* pozwalająca na dostęp do poszczególnych elementów kolekcji poprzez użycie numeru elementu. Elementy są numerowane począwszy od wartości 1.

6.8.3. Zdarzenia

Obiekt nie generuje zdarzeń.

6.8.4. Składowe zarezerwowane

Składowe zarezerwowane mają nazwy: *Extra, Explicite* i *Diagnostics*.

6.9. Obiekt Script

Dostęp do obiektu *Script* uzyskuje się poprzez pole *Script* obiektu *Asix*.

6.9.1. Pola

Pole ***Arguments***

Pole zwraca obiekt-kolekcję *Parameters* zawierającą parametry umieszczone w deklaracji skryptu.

Pole ***File***

Pole zwraca nazwę pliku zawierającego program skryptu.

Pole ***Ident***

Pole zwraca identyfikator liczbowy skryptu.

Pole ***Name***

Pole zwraca nazwę skryptu.

Pole ***Parameters***

Pole jest synonimem pola *Arguments*.

Pole ***Timeout***

Pole zwraca bieżącą wartość ograniczenia czasu wykonania skryptu w milisekundach. Zapis do tego pola powoduje ustawienie nowej wartości ograniczenia. Zapis powoduje jednocześnie ponowne rozpoczęcie odmierzania czasu wykonania skryptu. Zapisywana wartość jest czasem wyrażonym w milisekundach. Zapis do pola może zostać wykorzystany w przypadku konieczności wykonywania czynności wymagających dłuższego czasu. Zapisanie wartości 0 spowoduje wyłączenie kontroli czasu wykonania skryptu.

6.9.2. Metody

Obiekt nie posiada metod.

6.9.3. Zdarzenia

Zdarzenie ***OnTerminate***

Zdarzenie jest generowane w chwili kończenia pracy skryptu. Zdarzenie wystąpi po zdarzeniu *OnTerminate* obiektu *Asix*, o ile skrypt będzie kończył działanie razem z aplikacją systemu **asix**.

6.9.4. Składowe zarezerwowane

Składowe zarezerwowane mają nazwy: *Extra*, *Explicite*, *Dump* i *Diagnostics*.

6.10. Obiekt Scripter

Dostęp do obiektu *Scripter* uzyskuje się poprzez pole *Scripter* obiektu *Asix*.

6.10.1. Pola

Pole ***Version***

Pole zwraca numer wersji modułu skryptów w postaci:

Numer_główny*16777216+ numer_dodatkowy *65536+ numer_wydania
(lub (Numer_główny<<24)+ (numer_dodatkowy<<16)+ numer_wydania).

6.10.2. Metody

Obiekt nie posiada metod.

6.10.3. Zdarzenia

Obiekt nie generuje zdarzeń.

6.10.4. Składowe zarezerwowane

Składowe zarezerwowane mają nazwy: *Extra, Explicite*, *Diagnostics* i *Manager*.

6.11. Obiekt ThreadGlobalData

Dostęp do obiektu *ThreadGlobalData* uzyskuje się poprzez pole *ThreadGlobalData* obiektu *Asix*.

Podobnie jak obiekt *GlobalData* obiekt służy do przekazywania danych pomiędzy skryptami. W przypadku obiektu *ThreadGlobalData* wymiana danych dotyczy tylko skryptów znajdujących się w tym samym wątku. Obiekt jest kolekcją swoich składowych.

6.11.1. Pola

Obiekt posiada tylko takie pola, jakie zostały mu nadane przez skrypty.

6.11.2. Metody

Obiekt posiada tylko takie metody, jakie zostały mu nadane przez skrypty.

6.11.3. Zdarzenia

Obiekt posiada tylko takie zdarzenia, jakie zostały mu nadane przez skrypty.

6.11.4. Składowe zarezerwowane

Składowe zarezerwowane mają nazwy: *Extra, Explicite* i *Diagnostics*.

6.12. Obiekt Variable

Obiekt pozwala na dostęp do wartości zmiennych procesowych. Obiekt jest zwracany przez metodę *Variables* obiektu *Asix*.

6.12.1. Pola

Pole **domyślne**

Polem domyślnym obiektu *Variable* jest wartość zmiennej, tj. wartość zwracana przez pole *Value*.

Pole ***Current***

Pole zwraca wartość logiczną **true**, jeśli pola *Value*, *Status* i *Time* odnoszą się do bieżących danych pomiarowych i **false** jeśli odnoszą się one do wartości archiwalnych. Zapis wartości **true** do tego pola powoduje aktualizację pól *Value*, *Status* i *Time* bieżącymi wartościami pomiarowymi pozyskanymi z modułu ASMEN oraz wznowienie generacji zdarzenia *OnRead*. Wykonanie funkcji *ArcRead* i *ArcReadAt* powoduje ustawienie wartości tego pola na **false**. Zdarzenie *OnRead* nie jest generowane, jeśli pole ma wartość **false**. Zapis wartości **false** do pola *Current* nie ma żadnego znaczenia.

UWAGA Jeśli zmienna nie posiada wartości bieżących a jedynie archiwalne, tzn. nie ma jej w bazie zmiennych modułu ASMEN, to zapis wartości **true** do pola *Current* jest traktowany jako błąd.

Pole ***Info***

Pole zwraca obiekt ***Info*** zawierający informacje o zmiennej.

Pole ***Name***

Pole zwraca łańcuch określający nazwę zmiennej taką, jaka została zadeklarowana w bazie danych modułu ASMEN.

Pole ***Overruns***

Pole zwraca liczbę określającą ile razy nie doszło do aktualizacji pól *Value*, *Status* i *Time* z powodu zajętości skryptu. Odczyt tego pola powoduje jednocześnie jego wyzerowanie.

Niezerowa wartość tego pola świadczy o zbyt wolnym działaniu skryptu w stosunku do częstości odświeżania i ilości obsługiwanych zmiennych.

Pole ***Status***

Pole zwraca liczbę określającą status wartości zmiennej. Wykorzystywane są statusy OPC (patrz: *5.3. Status danych*).

Pole ***Time***

Pole zwraca stempel czasu związany z wartością zmiennej. Czas jest czasem lokalnym.

Pole ***UTCMode***

Pole określa sposób reprezentacji czasu we wszystkich polach i parametrach metod obiektu *Variable* dotyczących czasu. Jeśli wartością tego pola jest **false**, to jest to czas lokalny. Jeśli polu *UTCMode* nadano wartość **true**, to czas będzie wyrażony jak czas UTC (ang. Coordinated Universal Time, poprzednio nazywany Greenwich Mean Time). Domyślną wartością tego pola jest **false**. Należy zwrócić uwagę, że metody „getUTC...." języka JScript nie będą działać poprawnie jeśli pole to ma wartość **true** (patrz: Uwagi na temat czau). Z pola *UTCMode* można korzystać począwszy od wersji 1.02.000 modułu skryptów.

Pole ***Value***

Pole zwraca wartość zmiennej. Może to być bieżąca wartość zmiennej lub wartość archiwalna w zależności od wartości zwracanej przez pole *Current*.

6.12.2. Metody

Metoda ***ArcParam***

Postać wywołania:

obiekt.ArcParam([*param*])

gdzie:

obiekt	- obiekt *Variable*;
param	- określa sposób interpolacji wartości dla funkcji ***ArcReadAt***:

INTR_LINEAR	interpolacja liniowa
INTR_PREV	interpolacja poprzednią wartością
INTR_NEXT	interpolacja następną wartością
INTR_NEAREST	interpolacja najbliższą wartością

Metoda zwraca wartość 0, jeśli zakończyła się pomyślnie i wartość ujemną w przypadku błędu.

Metoda ***ArcRead***

Postać wywołania:

obiekt.ArcRead([*typ*])

gdzie:

obiekt - obiekt *Variable*;
typ - typ operacji odczytu w stosunku do wskaźnika archiwum:

READ_CURR - będzie czytana bieżąca dana;
READ_NEXT - będzie czytana następna dana.

Jeśli parametr *typ* został pominięty, to przyjmowana jest wartość READ_NEXT.

Metoda zwraca wartość 0, jeśli zakończyła się pomyślnie i wartość ujemną w przypadku błędu. Metoda może również zwracać wartości dodatnie, które mają następujące znaczenie:

STAT_NoData - brak danych;
STAT_EarlierTime - próba odczytu przed początkiem archiwum;
STAT_LaterTime - próba odczytu poza końcem archiwum;
STAT_Break - dziura w archiwum;
STAT_NotReady - danych jeszcze nie sprowadzono; wartość może zostać zwrócona w przypadku odczytu archiwum sieciowego.

Metoda powoduje ustawienie pól *Value*, *Status* i *Time* obiektu *Variable* wartościami pomiaru odczytanymi z archiwum.

Metoda ***ArcReadAt***

Postać wywołania:

obiekt.ArcReadAt(*czas*)

gdzie:
obiekt - obiekt *Variable*;
czas - określa czas danej (czas absolutny).

Metoda zwraca wartości takie jak funkcja ***ArcRead***. Metoda powoduje ustawienie pól *Value*, *Status* i *Time* obiektu *Variable* wartościami odczytanymi z archiwum. Jeśli archiwum nie zawiera danej z chwili określonej parametrem *czas*, to pole *Value* zostanie ustawione na podstawie interpolacji innych wartości zapamiętanych w archiwum. Sposób tej interpolacji jest określony metodą ***ArcParam***. Jeśli metoda ***ArcParam*** nie została wcześniej wywołana, to rezultat wywołania metody *ArcReadAt* jest niezdefiniowany. Po pomyślnym wykonaniu metody *ArcReadAt*, pole *Time* zawsze zawiera czas równy czasowi określonemu parametrem *czas*.

Metoda ***ArcSeek***

Postać wywołania:

obiekt.ArcSeek(*czas*[, *punkt_startowy*])

gdzie:
obiekt - obiekt *Variable*;
czas - określa nową pozycję wskaźnika archiwum i jest czasem absolutnym w przypadku ORIGIN_SET. Czas absolutny podawany jest w formacie VT_DATE. W pozostałych przypadkach czas jest wartością numeryczną określającą

przesunięcie w sekundach odniesione do bieżącej pozycji wskaźnika archiwum (dla ORIGIN_CUR) lub do końca archiwum (dla ORIGIN_END).

punkt_startowy - parametr określa punkt czasu, do którego odnosi się parametr *czas*, i może przyjmować następujące wartości:
ORIGIN_SET
ORIGIN_CUR
ORIGIN_END
Jeśli parametr jest pominięty, to przyjmuje się, że parametr ma wartość ORIGIN_SET, zaś parametr *czas* określa czas absolutny.

Metoda zwraca wartość 0, jeśli zakończyła się pomyślnie i wartość ujemną w przypadku błędu. Dodatkowe informacje na temat czasu znajdują się w *5.16. Uwagi na temat czasu*.

Metoda nie powoduje aktualizacji pól *Value*, *Time* i *Status*. Aby zaktualizować te pola, należy wykonać metodę *ArcRead*.

Metoda ***ArcTell***

Postać wywołania:

obiekt.ArcTell

gdzie:
obiekt - obiekt *Variable*.

Metoda zwraca czas, na jaki jest ustawiony wskaźnik archiwum.

Metoda ***Read***

Postać wywołania:

obiekt.Read( )

gdzie:
obiekt - obiekt *Variable*;

Metoda powoduje odczyt wartości z urządzenia zewnętrznego lub zmiennej umieszczonej w kanale NONE. Po wykonaniu tej metody, pola *Value*, *Status* i *Time* obiektu *Variable* zostaną odpowiednio zaktualizowane. Jeśli pola te służyły uprzednio do przeglądu archiwum zmiennej, to po wykonaniu funkcji *Read* będą się one odnosić do wartości bieżących, tzn. wartością pola *Current* będzie **true**.

Metoda zwraca 0, jeśli operacja zapisu zakończyła się pomyślnie.

Metoda ***Write***

Postać wywołania:

obiekt.Write(*wartość*[, *status*[,*czas*]])

gdzie:
obiekt - obiekt *Variable*;
wartość - wartość do zapisu;

status	- status zapisywanej wartości. Jeśli parametr jest pominięty, to przyjmuje się status „dana poprawna". Wykorzystywane są statusy OPC (patrz: *5.3. Status danych*);
czas	- stempel czasu zapisywanej wartości. Jeśli parametr jest pominięty, to przyjmowany jest czas bieżący.

Metoda pozwala na zapis wartości do urządzenia zewnętrznego lub zmiennej umieszczonej w kanale NONE. Parametr *status* i *czas* mają znaczenie tylko dla zmiennych znajdujących się w kanale NONE, który dodatkowo musi mieć określony parametr:

ZAPIS_CZASU_I_STATUSU = Tak

Metoda zwraca 0, jeśli operacja zapisu zakończyła się pomyślnie.

Metoda *Write* nie powoduje aktualizacji pól *Value*, *Status* i *Time*. Pole te mogą zostać zaktualizowane w wyniku wykonania metody *Read* lub w wyniku automatycznego odczytu nowej wartości przez moduł ASMEN (o ile pola te odnoszą się do wartości bieżących a nie archiwalnych, tzn. jeśli pole *Current* ma wartość **true**).

6.12.3. Zdarzenia

Zdarzenie ***OnRead***

Zdarzenie jest generowane w chwili, gdy moduł ASMEN odczyta nową daną pomiarową. Funkcja reakcji na zdarzenie powinna mieć postać:

Listing 35. Przykład skryptu z funkcją reakcji na zdarzenie obiektu 'Variable'.

VBScript

```
Function nazwa_funkcji( wartość, status, czas )
....
End Function
```

JScript

```
function nazwa_funkcji( wartość, status, czas )
{
....
}
```

Parametry funkcji określają odpowiednio wartość, status oraz czas danej pomiarowej.

Wykorzystywane są statusy OPC (patrz: *5.3. Status danych*).

Zdarzenie jest generowane tylko wtedy, gdy pole *Current* ma wartość **true**.

6.12.4. Składowe zarezerwowane

Składowe zarezerwowane mają nazwy: *Extra, Explicite, Diagnostics, First, Last, Next, Previous, WithBreaks, SkipBreaks, NextBreak, PrevBreak, FirstBreak, LastBreak, Break, BreakEnd, Find, Search, Info, AtBreak, Move, AtData, Archive, MoveTo, Refresh, Open, ArcParam.*

7. Parametryzacja modułu skryptów

Parametryzacja modułu skryptów dla pakietu **asix5** odbywa się przy użyciu programu Architekt.

Patrz: *Architekt – podręcznik użytkownika*, rozdz. *3.14. Konfiguracja parametrów skryptów i akcji*

8. Akcja operatorska Skrypt

Skrypt może być uruchomiony także poprzez wykorzystanie akcji operatorskiej *Skrypt*.

Składnia akcji jest następująca:

SKRYPT *nazwa_pliku*, *parametry_skryptu parametry_wykonawcze*

gdzie:

nazwa_pliku - określa nazwę pliku zawierającego skrypt. Jeżeli ścieżka dostępu nie zostanie podana, to przeszukiwane są katalogi ścieżki masek. Jeżeli nie podane zostanie rozszerzenie pliku, to dodawane jest vbs.

Użycie *parametry_skryptu* i *parametry_wykonawcze* jest identyczne jak dla skryptów uruchamianych poprzez parametr Skrypty (Architekt > *Obszary i komputery* > moduł *Skrypty i akcje* > zakładka *Skrypty*).

9. Agregacja

9.1. Wstęp

AsScripter w wersji 1.7 rozszerzono o kilka funkcji pozwalających na pobieranie serii danych archiwalnych, a także na pobieranie atrybutów zmiennych z bazy zmiennych i dostęp do informacji o bieżącym użytkowniku systemu **asix**.

9.2. Odczyt serii danych z archiwum Aspada

Dostęp do serii danych archiwalnych realizowany jest za pośrednictwem modułu **asix** Connect. Do obiektu Variable dodano 2 funkcje analogiczne do dostępnych w Asix Connect funkcji Automation dostępu do danych archiwalnych.

9.2.1. ReadRaw - Odczyt danych surowych

Odczyt serii surowych danych archiwalnych jest realizowany za pomocą metody ReadRaw obiektu Variable:

obiekt.ReadRaw (*czas_początku*, *czas_końca*, *dane*)

ReadRaw czyta wartości, statusy i stemple czasu zmiennej z archiwum z podanego okresu. Czyta rzeczywiste wartości zapisane w archiwum, zwracając również wartości obejmujące końce przedziału.

Parametry *czas_początku* i *czas_końca* muszą być typu Date lub String i są czasami lokalnymi lub UTC, w zależności od ustawienia pola UTCMode obiektu. Dla typu String przyjmuje się, że czas podany jest zgodnie z regułami określonymi dla czasu względnego OPC (**Błąd! Nie można odnaleźć źródła odwołania.**).

Po zakończeniu operacji czytania parametr *dane* zawiera tablicę przeczytanych próbek. Tablica zawiera tyle wierszy ile jest przeczytanych próbek. Każdy wiersz zawiera 3 elementy:

0. wartość – typu Double; zawiera dane źródłowe skonwertowane do typu Double,
1. stempel czasu – typu Date, w konwencji zgodnej z ustawieniem pola UTCMode.
2. jakość – typu Integer; zawiera 32-bitowy status OPC danych historycznych (**Błąd! Nie można odnaleźć źródła odwołania.**).

W tablicy zwracane są również wartości graniczne dla podanego okresu czasu. Jeżeli w archiwum nie zarejestrowano próbki dokładnie w chwili czasowej *czas_początku*, to zwracana jest ostania próbka sprzed tej chwili czasowej. Jeżeli w archiwum nie zarejestrowano próbki dokładnie w chwili czasowej *czas_końca*, to zwracana jest pierwsza próbka zarejestrowana za tą chwilą czasową. Jeżeli nie można pobrać z archiwum wartości granicznych, to zwracana jest próbka, której pole jakość ma wartość OPCHDA_NOBOUND.

Przykład użycia

W poniższym przykładzie czytane są dane surowe archiwum D zmiennej Var z okresu od StartTime do EndTime.

```
On Error Resume Next
Err.Number = 0

Set VarArc = Asix.Variables( "Var,D" )
VarArc.ReadRaw StartTime, EndTime, RawData
Panel.Message "ReadRaw " & StartTime & ", " & EndTime & ", RawData"
If Err.Number = 0 Then
    Panel.Message "RawData(" & LBound(RawData, 1) & "-" & UBound(RawData, 1) & ", " _
        & LBound(RawData, 2) & "-" & UBound(RawData, 2) & ")"
    For i = LBound(RawData, 1) To UBound(RawData, 1)
        Panel.Message "   Value=" & RawData(i, 0) & ", Time=" & RawData(i, 1) & ",
Quality=" _
           & Hex(RawData(i, 2))
    Next
Else
    Panel.Message "Error # " & Hex(Err.Number) & " " & Err.Description
End If
```

9.2.2. ReadProcessed - Odczyt danych zagregowanych

Odczyt serii zagregowanych danych archiwalnych jest realizowany za pomocą metody ReadProcessed obiektu Variable:

obiekt.ReadProcessed(*czas_początku*, *czas_końca*, *nazwa_agregatu*, *interwał_agregacji*, *dane*)

ReadProcessed oblicza agregaty danej zmiennej w podanym okresie.

Znaczenie parametrów *czas_początku*, *czas_końca* i *dane* jest takie samo jak dla metody ReadRaw.

Parametr *interwał_agregacji* powinien zawierać długość interwału agregacji podaną jako typ Date lub String (czas względny OPC). Interwał agregacji będzie liczony zawsze od początku dnia w czasie UTC i nie powinien zawierać składników D, MO, Y.

Parametr *nazwa_agregatu* powinien zawierać jedną z poniższych nazw agregatu.

Nazwa angielska	**Nazwa polska**	**Sposób wyliczania agregatu**
Start	*Początek*	Wartość na początku interwału. Stempel czasu jest stemplem czasu początku interwału.
End	*Koniec*	Wartość na końcu interwału. Stempel czasu jest stemplem czasu końca interwału.
Delta	*Przyrost*	Różnica wartości na końcu i na początku interwału.
Min	*Min*	Minimalna wartość w interwale.
Max	*Max*	Maksymalna wartość w interwale
Range	*Zakres*	Różnica między maksymalną a minimalną wartością w interwale.
Total	*Suma*	Suma wartości ważonych czasowo w interwale (całka po czasie).
Average	*Średnia*	Średnia wartości ważonych czasowo w interwale.
Average0	*Średnia0*	Średnia wartości ważonych czasowo w interwale. W okresach, w których wartość jest niedostępna do obliczeń używana jest wartość 0.
Quality_Good	*Jakość_Dobra*	Procent próbek o jakości dobrej w interwale (1 = 100%).
Quality_Uncertain	*Jakość_Niepewna*	Procent próbek o jakości niepewnej w interwale (1 = 100%).
Quality_Bad	*Jakość_Zła*	Procent próbek o jakości złej w interwale (1 = 100%).

Przykład użycia

W poniższym przykładzie czytane są średnie 5-minutowe archiwum D zmiennej Var z okresu od StartTime do EndTime.

```
On Error Resume Next
Err.Number = 0

Set VarArc = Asix.Variables( "Var,D" )
VarArc.ReadProcessed StartTime, EndTime, "Average", "5M", ProcData
Panel.Message "ReadProcessed " & StartTime & ", " & EndTime & ", Average, 5M,
ProcData"
If Err.Number = 0 Then
    Panel.Message "ProcData(" & LBound(ProcData, 1) & "-" & UBound(ProcData, 1) & ",
" _
        & LBound(ProcData, 2) & "-" & UBound(ProcData, 2) & ")"
    For i = LBound(ProcData, 1) To UBound(ProcData, 1)
        Panel.Message "   Value=" & ProcData(i, 0) & ", Time=" & ProcData(i, 1) _
            & ", Quality=" & Hex(ProcData(i, 2))
    Next
Else
    Panel.Message "Error # " & Hex(Err.Number) & " " & Err.Description
End If
```

9.2.3. Format czasu OPC

Składnia formatu czasu względnego OPC jest następująca:
keyword +/- offset +/- offset …

Możliwe wartości keyword i offset podane są w tabeli poniżej. Spacje i znaki tabulacji są ignorowane. Każdy parametr offset musi być poprzedzony liczbą całkowitą specyfikującą jego krotność i kierunek.

Keyword	**Opis**
NOW	Czas bieżący
SECOND	Początek bieżącej sekundy
MINUTE	Początek bieżącej minuty
HOUR	Początek bieżącej godziny
DAY	Początek bieżącego dnia
WEEK	Początek bieżącego tygodnia
MONTH	Początek bieżącego niesiąca
YEAR	Początek bieżącego roku

Offset	**Opis**
S	Przesunięcie czasu w sekundach
M	Przesunięcie czasu w minutach
H	Przesunięcie czasu w godzinach
D	Przesunięcie czasu w dniach
W	Przesunięcie czasu w tygodniach
MO	Przesunięcie czasu w miesiącach
Y	Przesunięcie czasu w latach

Na przykład, napis DAY -1D+7H30M mógłby reprezentować czas początkowy danych do raportu dziennego generowanego w dniu bieżącym (DAY = pierwszy stempel czasu dnia dzisiejszego). Zapis -1D daje pierwszy stempel czasu dnia wczorajszego, +7H daje godzinę 7:00 wczoraj, +30M daje godzinę 7:30 wczoraj; znak + w ostatnim offsecie jest przenoszony z poprzedniego offsetu).

Podobnie, MONTH-1D+5h oznacza godzinę 5:00 ostatniego dnia poprzedniego miesiąca, NOW-1H15M oznacza 1 godzinę i 15 minut temu, a YEAR+3MO oznacza datę 1 kwietnia bieżącego roku.

W formacie tym można wyrazić również długość okresu czasu. Należy wtedy w opisanym formacie pominąć pierwszy człon keyword.

9.2.4. Statusy OPC dla danych historycznych

Statusy danych historycznych powstają przez złożenie podstawowego 16-bitowego statusu na 16 młodszych bitach i 16-bitowego pola na 16 starszych bitach. W poniższej tabeli podano znaczenie 16 starszych bitów.

Wartość symboliczna	**Kod szesnastkowy**
OPCHDA_EXTRADATA	0x00010000
OPCHDA_INTERPOLATED	0x00020000
OPCHDA_RAW	0x00040000
OPCHDA_CALCULATED	0x00080000
OPCHDA_NOBOUND	0x00100000
OPCHDA_NODATA	0x00200000
OPCHDA_DATALOST	0x00400000
OPCHDA_CONVERSION	0x00800000
ASKOM_HDA_ARCHIVE_END	0x01000000

Znaczenie pierwszych ośmiu bitów jest zgodne ze specyfikacją OPC, a bit ASKOM_HDA_ARCHIVE_END oznacza, że próbka jest ostatnią aktualnie dostępną w archiwum.

9.3. ReadAttribute - Odczyt atrybutów zmiennych z bazy zmiennych

Obiekt Variable zapewnia dostęp do atrybutów zmiennej zapisanych w bazie zmiennych za pośrednictwem metody RaedAttribute.

obiekt.ReadAttribute(*nazwa_atrybutu*)

Dla zmiennej *obiekt* metoda zwraca wartość określonego atrybutu.

Jako parametr *nazwa_atrybutu* należy przekazać nazwę atrybutu lub jedną ze stałych podanych w tabeli.

Nazwa stałej	**Znaczenie**
atrDescription	Opis zmiennej
atrEU	Jednostka pomiarowa
atrRangeLo	Dolny zakres pomiarowy
atrRangeHi	Górny zakres pomiarowy
atrSampleRate	Okres próbkowania
atrArchiveRate	Okres archiwizacji

Przykład użycia

```
Set VarObj = Asix.Variables("Var,D")
VarName = VarObj.ReadAttribute("Nazwa")
VarChan = VarObj.ReadAttribute("Kanał")
VarFun = VarObj.ReadAttribute("Funkcja przeliczająca")
VarDescr = VarObj.ReadAttribute(atrDescription)
```

9.4. Dostęp do informacji o bieżącym użytkowniku systemu asix

Informacje o bieżącym użytkowniku systemu Asix są udostępnione jako nowe pola obiektu Asix:

obiekt.CurrentUserId – identyfikator bieżącego użytkownika
obiekt.CurrentUserName – nazwa bieżącego użytkownika
obiekt.CurrentUserLevel – poziom uprawnień bieżącego użytkownika

10. Index

11. Spis wydruków skryptów

12. Spis tabel

13. Spis treści