CLARION 5

Language Reference

Dane o oryginale:

This publication supports Clarion 5. It is possible that it may contain technical or typographical errors. TopSpeed Corporation provides this publication „as is", without warranty of any kind, either expressed or implied.

TopSpeed Corporation
150 East Sample Road
Pompano Beach, Florida 33064
(954) 785-4555

Authorized translation from English Language Edition.

Translation © Speed
Poznań, 2000

Speed
ul. Ratajczaka 18
61-815 Poznań
tel/fax (0-61) 851-77-62, 855-73-73

SPIS TREŚCI

WSTĘP – GENEZA JĘZYKA CLARION

Bruce D. Barrington, CEO, TopSpeed Corporation

Jak to się często zdarza, sam sobie narobiłem roboty. Kupiłem pierwszy PC, jaki zobaczyłem i zapragnąłem go zaprogramować. I oto co zrobiłem. Pascal był zbyt prymitywny, C w ogóle jeszcze nie był dostępny. Tak więc spróbowałem BASIC-a. Wszystko, czego on wymagał to kawałek ekranu i klawiatury. Zgadza się? Być może jeszcze indeksowanego dostępu. Zgadza się?

Błąd! Można było na tym pracować, ale nie można było tego robić w sposób przejrzysty. Właśnie spędziłem 10 lat na pracy przy wytwarzaniu oprogramowania za pomocą narzędzi, które sam opracowałem. Lubiłem je. Może nadszedł czas, by podzielić się tym, czego się nauczyłem. Może świat potrzebował jeszcze jednego języka programowania, a w szczególności – uniwersalnego języka programowania aplikacji biznesowych. Opracowanego specjalnie dla PC-tów.

Trochę to brzmi sprzecznie – nazywać język programowania aplikacji biznesowych językiem uniwersalnym. Jednak w świecie PC istnieje wiele biznesowych „języków", które nadają się „do wszystkiego" oprócz zastosowań uniwersalnych. Pisanie makr dla arkuszy kalkulacyjnych jest programowaniem, jak przypuszczam, ale makra z trudem można nazwać uniwersalnym językiem programowania. Z tego powodu, większość języków programowania baz danych nie jest językami uniwersalnymi. Są to w rzeczywistości skrypty przeznaczone do uruchamiania przez program sterujący (motor) bazy danych. Skrypty definiują rolę, jaką odgrywa program sterujący bazy danych pełniąc obowiązki Twojej aplikacji. Nawet język dBase, którego programy mogą być kompilowane i działać jako samodzielne, nie jest w rzeczywistości językiem uniwersalnym.

Zgodnie z moją definicją, język uniwersalny powinien posiadać zdolność wykorzystania całego repertuaru możliwości dawanych przez platformę systemową. Oznacza to, że program powinien mieć zdolność do czytania dowolnej sekcji dowolnego pliku rozpoznawanego przez system operacyjny. Powinien wykorzystywać wszystkie możliwości interfejsu użytkownika. Powinien potrafić się porozumieć, za pomocą standardowych mechanizmów, z innymi uniwersalnymi językami programowania i komponentami software'owymi. Język uniwersalny nie narzuca programowi swojego interfejsu i filozofii działania. Nie wznosi barier, które trzeba przełamywać. Przeciwnie, gwarantuje szeroki wachlarz, w ramach swojej platformy, możliwości rozwiązywania problemów programistycznych.

Dlaczego jednak ograniczać nowy język tylko do platformy PC? Inne wiodące języki programowania są przenośne. Zdecydowałem jednak, że PC-ty zasługują na specjalne traktowanie. Już w roku 1984, gdy zacząłem projektować Clarion na poważnie, PC-ty stanowiły znaczący procent wszystkich zainstalowanych na świecie komputerów. Przy tym różniły się one znacznie od innych komputerów. Były urządzeniami dedykowanymi dla pojedynczego użytkownika, z własną klawiaturą i monitorem. Klawiatura i monitor były dostępne bezpośrednio, bez żadnych modemów i łączy komunikacyjnych. To były idealne maszyny dla interaktywnych aplikacji. Postanowiłem wykorzystać te cechy wbudowując do mojego nowego języka obsługę wyświetlania opartą na mapowaniu pamięci. Jeśli program napisany w Clarionie mógł działać na „tylko" 40, czy 50 milionach komputerów, było to dla mnie wystarczające.

Kierowało mną mocne przekonanie, że programowanie powinno być prostsze, że języki programowania powinny być łatwiejsze w czytaniu i pisaniu, że kiepska produktywność związana z tworzeniem oprogramowania wynika ze źle zaprojektowanych narzędzi programistycznych.

To przekonanie wynikło między innymi z poniższych przesłanek. Dlaczego ktoś ma tworzyć pętlę IF następująco: IF...THEN BEGIN;instrukcje;END ELSE... (Pascal). Czemu służą słowa kluczowe THEN, BEGIN i END w tej strukturze? Dlaczego trzeba używać „:=" zamiast "=" w instrukcjach przypisania (Pascal, Modula-2, Ada). Czy twórca języka nie wiedział, że przypisania są najczęściej występującymi instrukcjami w programie i że wpisanie „:=" wymaga plątania palców na klawiaturze (konieczność wciśnięcia Shift)? Co powiedzieć o klauzuli READ...AT END języka COBOL ustawiającej zmienną końca pliku, która jest testowana w celu zakończenia pętli odczytu? Dlaczego pętla nie może sprawdzić końca pliku? Deklarując zmienną, dlaczego muszę pamiętać, by kompilator przekonwertował ją w łączonych wyrażeniach? Czy kompilator nie może zapamiętać tego za mnie?

Tworzenie stylu

Tak więc zasiadłem do tworzenia nowego języka programowania, który miał być zwarty (łatwy w pisaniu) i wyrazisty (łatwy w czytaniu). Zacząłem pracę niejako od końca. Najpierw napisałem wiele programów, tak długo eksperymentując ze składnią i semantyką, aż programy te zaczęły wyglądać przyzwoicie. Następnie napisałem mały podręcznik języka. Gdy był gotowy, mój zespół zabrał się za pisanie kompilatora. Język zmieniał się z dnia na dzień. Nasze stare zapiski oddają cały ten energiczny i interaktywny proces. Pojawiało się wiele pomysłów, wiele z nich było odrzucanych ze względu na sztukę programowania. Inne z powodu kiepskiej technologii. Niektóre były po prostu obłąkane. Podobnie, jak gatunki w teorii Darwina, przetrwały tylko niektóre z nich.

Osobiście klasyfikuję języki programowania na trzy style: zorientowany na symbole, zorientowany na sentencje i zorientowany na instrukcje. Języki zorientowane na symbole, takie jak Pascal, czy C są zwarte, ale nie wyraziste. Traktują one program jak zbiór symboli (słów kluczowych, nazw danych, stałych, znaków interpunkcyjnych, etc.) oddzielonych od siebie „białymi spacjami" (spacje, CR/LF, komentarze, czasami przecinki). Kompilator wyłapuje symbole ignorując spacje. Języki zorientowane na symbole są jednowymiarowe, dlatego programista stosuje „białe spacje" w celu dodania kolejnego wymiaru do swojego programu.

```
typedef    struct  {
  unsigned    char    Type;          /* typ struktury */
  unsigned    Vlen;                  /* zmienna długość */
  unsigned    char    Dplac;         /* pozycje dziesiętne w liczbie dziesiętnej */
  void        *Use;                  /* wskaźnik na zmienną */
} Usedef
```

Ten programista C zrobił wszystko, co tylko możliwe, by uzyskać czytelną definicję typu. Ale lewy nawias klamrowy wygląda jakby „wisiał" za słowem kluczowym **struct**. A Usedef „wisi" sobie za prawym nawiasem klamrowym. Przy tym wszystkim nawiasy klamrowe nie prezentują się najlepiej w roli symboli rozgraniczających.

Języki zorientowana na sentencje, takie jak COBOL i wiele innych języków programowania baz danych, są czytelne, ale nie są zwarte. Czasami instrukcje zawarte w sentencjach czyta się jak perfekcyjny tekst angielski. Ta instrukcja języka COBOL jest wyjątkowo czytelna:

MULTIPLY PRINCIPAL BY RATE GIVING PAYMENT ROUNDED.

Ale niewiele bardziej, niż:

```
Payment = Principal * Rate
```

Tłumaczyłem sobie, że w kontekście całego programu, druga instrukcja jest łatwiejsza do odczytania, niż pierwsza, co skłania do rozpisywania się w całym akapicie. Inne formaty sentencji już nie przypominają tak bardzo języka angielskiego. Znalazłem ten „kwiatek" w podręczniku języka xBase:

```
EDIT   [FIELDS <field list>] [<scope>][FOR <expL1>]
       [WHILE <expL2>][FREEZE <field>]
       [KEY<expr1> [,<expr2>]] [LAST] [LEDIT] [REDIT]
       [LPARTITION] [NOAPPEND] [NOCLEAR] [NODELETE]
       [NOEDIT | NOMODIFY] [NOLINK] [NOMENU] [NOOPTIMIZE]
       [NORMAL][NOWAIT][PARTITION <expN1>][PREFERENCE <expC1>]
       [SAVE][TIMEOUT <expN2>] [TITLE <expC2>]
       [VALID [:F] <expL3> [ERROR <expC3>]] [WHEN <expL4>]
       [WIDTH <expN3>] [[WINDOW <window name1>]
       [IN [WINDOW] <window name2> | IN SCREEN]]
       [COLOR SCHEME <expN4>] | COLOR <color pair list>]
```

Wow! To są oczywiście angielskie wyrazy, ale czy są one czytelne? Czy jakikolwiek programista jest w stanie rozpoznać sens powyższej konstrukcji bez wertowania podręcznika? Pomijając wiele innych kwestii, chciałbym zapytać: kto wymyślił klauzulę **WHILE** i klauzulę **WHEN** w tej samej instrukcji? Chce mi się wrzeszczeć przez okno!

Moje eksperymentalne programy były zorientowane na instrukcje, tak jak stare fasony stylów stosowanych w FORTRAN i BASIC. Języki zorientowane na instrukcje korzystają z faktu, że programy źródłowe są zapisane w plikach ASCII; każda linia programu jest rekordem pliku. Tak więc można zastosować granice rekordu, eliminując tym samym konieczność stosowania znaków interpunkcyjnych. Oparłem się na formacie instrukcji zapewniającym zwięzłość, czytelność i wszechstronność:

```
label STATEMENT[(parameters)] [,ATTRIBUTE[(parameters)]] ...
```

Atrybuty są stosowane tylko przy deklarowaniu danych. Instrukcje wykonywalne wykorzystują format standardowego wywoływania procedur. Oczywiście zdefiniowałem różne formaty dla instrukcji przypisania (A = B) oraz (IF, CASE, etc.).

Etykieta instrukcji zaczyna się w pierwszej kolumnie (pierwsza pozycja rekordu). Instrukcja bez etykiety nie może się zaczynać w pierwszej kolumnie. Instrukcja kończy się wraz z końcem wiersza, chyba, że oznaczono znakiem pionowej kreski (|), że jest kontynuowana w następnym. Zaadaptowałem z Moduli-2 średniki do pełnienia roli opcjonalnych separatorów instrukcji leżących w jednym wierszu. Adoptując z Moduli-2 koncepcję ignorowania pustych instrukcji, wyeliminowałem różnice pomiędzy separatorami a terminatorami, które doprowadzały do frustracji rzesze programistów Pascala.

Takie podejście wyeliminowało znaki interpunkcyjne konieczne do identyfikowania etykiet i oddzielnych instrukcji. Bloki instrukcji są inicjowane przez instrukcję złożoną, taką jak na przykład IF, a kończone instrukcją separującą, np. ELSE (inicjującą kolejny blok) lub przez instrukcję END (bądź znak kropki). Nie występują tu słowa kluczowe pełniące rolę tylko i wyłączenie ozdobników.

Deklarowanie danych

W swoich początkach COBOL był nazywany językiem samo-dokumentującym się. Wynikało to z jasnego oddzielenia danych i instrukcji. Każdy element przetwarzany przez program w COBOL musiał być zadeklarowany w sekcji danych: zmienne, stałe, pliki, rekordy, indeksy, nawet sekwencje sortowania i formaty raportów. Wiedziałem, że te deklaracje mają zasadnicze znaczenie przy dokumentowaniu programu biznesowego. Czułem też, że w naszym nowym formacie instrukcji powinniśmy jeszcze zwiększyć ich czytelność.

W końcu lat 60-tych IBM promował język PL/I jako sukcesora języka COBOL. Był on jednak dla wielu rozczarowaniem, chociaż wprowadził kilka interesujących, nowych idei. Poprzez skondensowanie słów kluczowych dla deklarowania danych i wprowadzenie zagnieżdżonych komentarzy (/*comment*/), PL/I udostępniał wystarczająco wiele miejsca na komentowanie każdej deklaracji. COBOL był zaprojektowany do stosowania długich, opisowych nazw danych, ale programiści nie lubią ich stosować. Wiadomo, że programiści lubią stosować wcięcia i kolumny w celu zwiększenia czytelności napisanego przez siebie kodu. Aranżując sekcję danych w układzie kolumnowym ograniczamy wtedy ilość znaków do określonej liczby. Poza tym programiści stosują krótkie nazwy dla danych w sekcjach procedur. Długie nazwy powodują wydłużenie wyrażeń, w których są używane, co jest po prostu mało wygodne. Z tego powodu wielu programistów COBOL używało krótkich etykiet i tym samym pisało programy, które wcale nie były tak dobrze udokumentowane, jak by się można było spodziewać.

Programiści PL/I obchodzili ten problem komentując swoje deklaracje i instrukcje. Jeśli pojawiało się pytanie, co znaczy nazwa określonej danej, można było sobie na nie odpowiedzieć sprawdzając jej deklarację. Zarządzałem wielkimi projektami w PL/I w latach 60-tych i utwierdziłem się w przekonaniu, że instrukcje deklaracyjne wymagają trzech elementów: etykiety instrukcji, typu danych oraz komentarza.

Nowy format instrukcji jest doskonały. Etykieta instrukcji pojawia się po lewej stronie, gdzie jest najlepiej widoczna. Słowa kluczowe dla typów danych są krótkie (BYTE, REAL, DIM, etc.) w celu pozostawienia jak najwięcej miejsca na komentarze. Miejsce oszczędza również wykrzyknik występujący w roli znacznika komentarza. COBOL i PL/I korzystają z „poziomów” przy deklarowaniu struktur danych. Każda zmienna ma przypisany numer poziomu. Zmienna z najwyższym numerem poziomu jest częścią poprzedniej zmiennej, o niższym numerze poziomu. Jeśli zmienna nie jest częścią struktury danych, jej poziom jest określany jako „01” lub „77”. Nigdy nie lubiłem stosowania poziomów i bardzo się zdziwiłem, gdy zostały przeniesione również do PL/I. Uważam, że są marnotrawieniem miejsca. Przy tym wszystkim niech mi ktoś jeszcze odpowie co oznacza „77” i w jakim celu niestrukturalna zmienna ma w ogóle posiadać określony poziom? Wybrałem GROUP (nazwa pochodzi od „group item” z COBOL-a) jako złożoną strukturę danych, którą będziemy nazywali grupą. Jest to mechanizm zbliżony do **record...end** stosowanego w Pascal-u, Moduli-2, czy ADA; bądź **struct{..}** stosowanego w C.

Wcinając zagnieżdżone instrukcje GROUP otrzymujemy bardzo czytelną deklarację:

```
Error     GROUP,PRE(Err)          ! Informacja o błędzie
Date        DATE                  ! Data błędu
Time        TIME                  ! Czas błędu
Device      STRING(12)            ! Aktywne urządzenie
Message     GROUP                 ! Komunikat błędu
MsgCode       STRING(@P###P)      ! Kod błędu
              STRING(' - ')
MsgText       STRING(32)          ! Tekst komunikatu
            END
          END
```

COBOL i PL/I dopuszczają stosowanie tych samych nazw danych w różnych strukturach danych. Odesłania do takich nazw wymagają zastosowania kwalifikatora nazwy, którym jest nazwa struktury. Jest to użyteczna konstrukcja, z tego względu, że te same pola cyklicznie pojawiają się w więcej niż jednej strukturze danych (np. ACCT-NO IN OLD-VENDOR, ACCT-NO IN CURRENT-PAYEE, itp.). Wielu programistów wzbrania się przed stosowaniem takiej nomenklatury, gdyż powoduje ona tworzenie długich referencji. Zamiast tego kodują prefiksy mnemoniczne dla każdego pola (np. VND-ACCT-NO). Zajmuje to trochę czasu i redukuje dostępny obszar nazw. By uzyskać zadowalający rezultat, włączyłem opcjonalny atrybut prefiksu, który może być dołączany do dowolnej struktury danych (np. **PRE(VND)**). Elementy struktury są kwalifikowane poprzez umieszczenie przed ich nazwą prefiksu I znaku dwukropka (np. VND:AcctNo, PAY:AcctNo).

W celu zachowania funkcjonalności przypisań „MOVE CORRESPONDING" COBOL-a i "BY NAME" PL/I, została dodana instrukcja „głębokiego" przypisania przenosząca korespondujące elementy z jednej grupy do drugiej:

```
DestinationGroup :=: SourceGroup
```

Jako język programowania aplikacji biznesowych, Clarion potrzebował bogatego zestawu bazowych typów danych. Zawarto w nim wszystkie rozmiary liczb całkowitych i rzeczywistych w celu zachowania kompatybilności z zewnętrznymi formatami rekordów i list parametrów. Upakowane liczby dziesiętne zostały włączone w celu rozwiązania problemów z zaokrągleniami i zmniejszenia zajętości pamięci. Mogą one być deklarowane w całym zakresie rozmiarów. Włączono również różnorodne formaty łańcuchowe (stałe, Pascal, C), wraz z funkcjami na nich operującymi. Wreszcie, włączono także formaty dla daty i czasu, w takiej postaci, by były w odniesieniu do nich możliwe bezpośrednie operacje arytmetyczne:

```
Jutro = Today() + 1
```

Co jednak ze złożonymi strukturami danych? W językach ALGOLo-podobnych, takich jak Pascal, Modula-2, Ada, czy C, grupy i tablice są deklarowane jako typy. Deklarujesz typ, następnie deklarujesz grupę lub tablice jako egzemplarz tego typu. Nigdy nie lubiłem tej składni. W programach biznesowych większość grup i tablic deklaruje się tylko raz. Myślenie o nazwie typu, kodowanie instrukcji **TYPE** jest zupełnie niepotrzebnym obciążeniem. Nigdy nie zakładałem, że grupa lub tablica ma być deklarowana jako typ. Opisują one zależności przechowywanych danych, a nie ich typ. Dlatego też deklarację typu zachowałem tylko jako opcjonalną. Deklaracja w Clarionie z atrybutem **TYPE** powoduje zadeklarowanie typu danych, który może być używany w występujących później strukturach lub dla struktur występujących w postaci parametru. Deklaracja pozbawiona atrybutu **TYPE** powoduje zarówno zadeklarowanie typu danych, jak i zmiennej tego typu. Zaadoptowałem tutaj instrukcję

LIKE języka PL/I deklarującą zmienną predefiniowanego typu. Czułem, że takie podejście zachowuje najlepsze cechy obu rozwiązań:

```
Totals      GROUP,PRE(QTR)
GrossPay        DECIMAL(12,2)
Deductions      DECIMAL(12,2)
NetPay          DECIMAL(12,2)
            END

YTD:Totals  LIKE(Totals),PRE(YTD)
```

Typy danych bez bólu

Język komputerowy charakteryzuje się silnymi typami danych, jeśli każdy element danych posiada pojedynczy typ danych i składnia języka czyni możliwym podgląd tego elementu w postaci innego typu. Wielu ekspertów uważa, że silne typy danych czynią program bardziej niezawodnym. Być może. Ale przecież programy z silnymi typami danych są trudniejsze w pisaniu, ograniczają zakres stosowania procedur uniwersalnych, wymagają niepotrzebnego czuwania nad typami danych. Co więcej, nigdy nie słyszałem o programiście COBOL oskarżającego **REDEFINES** (stosowanego powszechnie do wyłączenia silnych typów danych) o powodowanie problemów z niezawodnością. Przy okazji, programiści COBOL nie są wcale bezkrytyczni względem swojego języka. Instrukcja **ALTER** popadła w niełaskę wiele lat temu, gdyż efektem jej stosowania była niestabilność programów.

Nie chciałem, by nasz nowy język posiadał silne typy danych. Przede wszystkim chciałem zapewnić obsługę redeklaracji podobnych do **REDEFINES** lub **union type** w C. Redeklaracje są szczególnie przydatne do implementacji rekordów (rekordy wariantowe w Pascal-u) i do obsługi szczególnych przypadków programistycznych. Przygotowałem atrybut **OVER** w celu osiągnięcia tego zamierzenia:

```
MonthNames  STRING('JanFebMarAprMayJunJulAugSepOctNovDec')
Month       STRING(3),DIM(12),OVER(MonthNames)
```

Poza tym oczekiwałem, że struktury typu grupa będą mogły być traktowane jak łańcuchy. Jest to osłabienie typowania danych, gdyż grupy mogą zawierać typy danych inne niż łańcuchy. Ale z drugiej strony grupy potrzebują określonej funkcjonalności. Muszą być przemieszczane, przekazywane w postaci parametru, czy (ostrożnie) porównywane. To jest przeszkoda, oczywiście. Większość numerycznych typów danych nie może być przekształcana w łańcuchy, tak więc grupy zawierające elementy numeryczne nie były na ogół przekształcane poprawnie. Ujemne liczby całkowite robiły się większe od dodatnich, a liczby zmiennoprzecinkowe były przekształcane zupełnie losowo. Projektowanie wymaga kompromisów, tak więc postawiłem na funkcjonalność, godząc się przy tym na pewne ryzyko.

Było bardzo ważne dla typów danych Clariona zachowanie prostej konstrukcji procedur ogólnego przeznaczenia. Jeśli procedura oczekiwała parametru numerycznego, taki powinien być jej dostarczony. Uważałem, że to śmieszne, dostarczać różnych funkcji numerycznych do obsługi danych różnych typów numerycznych, jak to jest w językach będących sukcesorami ALGOL-u. Wybiegając nieco w przyszłość, myślałem o polimorfizmie, jak zaimplementowano to w C++, co wymagało oddzielnych funkcji dla każdego typu danych, ale umożliwiało wywoływanie ich za pomocą wspólnej nazwy.

W oryginalnej wersji Clariona, parametry nie były właśnie prototypowane. Cokolwiek pojawiło się w liście argumentów, było wykorzystywane przez procedurę. Clarion wymaga obecnie określenia typów parametrów, ale dopuszcza stosowanie

nieokreślonych typów danych. Procedury Clariona zawsze były prawdziwie polimorficzne dla danych niestrukturalnych.

Parametry w Clarionie są prototypowane do przekazania poprzez wartość bądź adres. Clarion nie obsługuje wskaźników. Jest tak z dwóch powodów. Po pierwsze, wskaźniki nie troszczą się o informację o typie danych i mogą stać się łatwo bezużyteczne. Po drugie, dereferencje wskaźników (składnia odróżniająca wskaźnik od jego celu) niepotrzebnie komplikują program. Z mojego doświadczenia wynika, że większość błędów w programach C wynika z pomyłek w stosowaniu wskaźników.

Wybraliśmy zmienne referencyjne, tak, jak to zaimplementowano w C, do obsługi przekierowań. Zmienna referencyjna zawiera informacje o celu, na który wskazuje, jak i informacje o jego typie. Poza tym zmienne referencyjna podlega automatycznej dereferencji w momencie, gdy zostanie użyta. Dzięki temu nie ma możliwości pomylenia jej z celem. Weźmy pod uwagę poniższy przykład:

```
CompanyA    FILE
              :
            END
CompanyB    FILE
              :
            END
Company     &FILE                ! Dane przetwarzanej firmy
  CODE
    CASE CompanyLetter           ! Która firma jest przetwarzana?
    OF 'A'
      Company &= CompanyA        ! Wskaźnik na firmę A
    OF 'B'
      Company &= CompanyB        ! Wskaźnik na firmę B
    END
    OPEN(Company)                ! Otwarcie wybranej firmy
```

Zmienna referencyjna *Company* jest ustawiana przez instrukcję przypisania referencyjnego (&=). Kompilator powinien zaprotestować, jeśli typy danych nie są zgodne. Następnie zmienna referencyjna może zostać użyta w dowolnym kontekście właściwym dla jej celu.

Wartości pośrednie

Kolejnym ważnym elementem jest automatyczna konwersja typów. Jestem przekonany, że deklarujesz typ danych po to, by wiedział o tym kompilator. Porządny kompilator powinien generować konwersje typów danych wtedy, gdy jest to konieczne. Bardzo dobry kompilator powinien dokonać próby rozpoznania wyrażeń i zapewnić logiczną ich konwersję. Na przykład, jeśli chcę dodać łańcuch do liczby całkowitej, jest podstawne, by kompilator przyjął, że łańcuch zawiera liczby ASCII i wygenerował odpowiednią konwersję. Analogicznie, jeśli dokonuje operacji konkatenacji liczby całkowitej i łańcucha, oczekuję, że kompilator najpierw odpowiednio przekonwertuje liczbę całkowitą. Wybierając właściwe typy danych dla wartości pośrednich, kompilator może bezpiecznie konwertować typy danych w wyrażeniu bez utraty informacji w nich zawartych. Jeśli dzielimy dwie liczby całkowite, dobry kompilator przechowuje rezultat w wartości pośredniej wraz z częścią ułamkową. Jeśli dodajemy liczbę całkowitą do łańcucha, kompilator także powinien używać ułamkowej wartości pośredniej, gdyż łańcuch może reprezentować wartości ułamkowe.

Oczywiście informacja może zostać utracona, na przykład przy przenoszeniu wartości, poprzez instrukcję przypisania, czy przekazywanie jej w postaci parametru procedury. Przeniesienie wartości rzeczywistej do całkowitej powoduje utratę części ułamkowej.

Przeniesienie liczby rzeczywistej na upakowana dziesiętną, powoduje zaokrąglenie do najmniejszej znaczącej cyfry dziesiętnej. Niektóre języki, jak np. Pascal, wymagają, by konwersje danych były jawnie wywoływane. Wydawało mi się, że określając typ danych programista oczekuje od kompilatora absolutnego przestrzegania, by element danych leżał w odpowiednim zakresie wartości.

Wcześniejsze wersje Clarion używały tylko dwóch typów danych dla numerycznych wartości pośrednich: 32-bit liczby całkowitej ze znakiem (LONG) oraz 64-bitowje liczby zmiennoprzecinkowej (REAL). Operacja dzielenia i każda operacja, w której przynajmniej jeden operand był typu REAL, dawała w rezultacie wartość pośrednią REAL. Ta strategia zapewniała wystarczającą dokładność, gdyż REAL mógł zajmować maksymalną dla Clariona liczbę cyfr (15). Okazało się, że nie wszystko jest tak, jak należy. Dwa równoważne wyrażenia: 1 / 2 oraz 2 / 4 mogą dać w rezultacie dwie wartości zmiennoprzecinkowe różniące się najmniej znaczącym bitem. Jest to różnica zazwyczaj pomijalna w obliczeniach. Ale nie podczas porównywania! Programista ma prawo oczekiwać, że jedna druga równa się dwóm czwartym. Mając dobre chęci, mogę unikać stosowania porównań dwóch liczb REAL ale oczekuję, że poniższe wyrażenie zawsze będzie działało:

```
IF Hours > Normal * 1.5
```

Stosowanie REAL do przyjęcia wyrażenia po prawej stronie nie pozwala na zaufanie wynikowi porównania. Rozwiązaliśmy ten problem w Clarion for Windows poprzez zaimplementowanie stało przecinkowej wartości pośredniej z 31 cyframi dziesiętnymi po każdej stronie kropki dziesiętnej. Ta zmiana zwiększyła również maksymalne znaczenie do 31 cyfr.

Struktury sterujące

Tak jak języki biznesowe COBOL i PL/I oferują preferowany model dla deklarowania danych, tak Modula-2 oferuje najlepsze struktury sterujące. Zmodyfikowałem instrukcje **IF** Moduli-2 wprowadzając możliwość zamiany słowa kluczowego **THEN** przez separator instrukcji. Dało to taki efekt, że wyeliminowane zostały zbędne instrukcje **THEN** z wielowierszowych struktur **IF**. Adoptując **ELSIF** Moduli-2**,** wyeliminowałem konieczność stosowania rozbudowanych struktur **IF** i licznych ich terminatorów

```
IF Number < 0
  Sign = -1
ELSIF Number > 0
  Sign = +1
ELSE
  Sign = 0
END
```

Używałem również Moduli-2 w roli protoplasty dla instrukcji **CASE** Clariona. W Moduli-2 obsługa **CASE** opiera się na wymienieniu etykiet przypadków I ich zakresów – jest to bardzo użyteczne. Nie podobała mi się jedna zastosowana tam interpunkcja. Słowo kluczowe **OF** wprowadza etykietę pierwszego przypadku, zaś kolejne etykiety przypadków są inicjowane przez kreski pionowe (“|”).

Uznałem, że taka interpunkcja nie jest najlepsza i jest przy tym mało intuicyjna. Zamiast niej zastosowałem słowo kluczowe **OF** do wprowadzania etykiet przypadków. Przewidziałem także słowo kluczowe **OROF** przy wyliczaniu etykiet przypadków oraz słowo kluczowe **TO** dla ich zakresów. Te zmiany spowodowały, że składnia stała się bardzo przyjazna:

```
CASE SUB(Name,1,1)
OF 'A' TO 'M' OROF 'a' TO 'm'
  DO FirstHalf
OF 'N' TO 'Z' OROF 'n' TO 'z'
  DO SecondHalf
ELSE
  DO FirstHalf
END
```

W Moduli-2 po raz pierwszy zobaczyłem we właściwym kontekście zastosowaną pętlę **LOOP**. W Moduli-2, **LOOP...END** wykonuje bezwarunkową pętlę, którą kończy instrukcja **EXIT**. Rozszerzyłem tę koncepcje o instrukcję **CYCLE** powodującą wznowienie wykonywania pętli. Zmieniłem również **EXIT** na **BREAK** ponieważ już wcześniej zastosowałem **EXIT** w innym przypadku. Zaimplementowałem pętle warunkowe poprzez dodanie czterech opcjonalnych klauzul dla instrukcji **LOOP**:

```
LOOP I = 1 TO 100 BY 2
LOOP 10000 TIMES
LOOP WHILE Count > 0
LOOP UNTIL EndOfFile
```

Zawsze byłem zdanie, że dobrze zorganizowany program wymaga zastosowania podprogramów (routines). Lokalny podprogram jest blokiem instrukcji, które mogą być usunięte z głównego bloku programu i zastąpione instrukcją wywołania podprogramu. Trafne dobranie nazw podprogramów powoduje, że główny program staje się krótszy, nie tracąc przy tym swej czytelności. COBOL i BASIC wykorzystują do tego **PERFORM** oraz **GOSUB**. Procedury lokalne w Pascal-u i Moduli-2 również przypominają podprogramy jednak wymagają prototypowania w celu określenia typów parametrów. Nie chciałem wprowadzać w podprogramach możliwości przekazywania parametrów, gdyż założyłem, że będą w nich widoczne wszystkie dane procedury wywołującej podprogram. Stworzyłem instrukcję **ROUTINE** inicjującą lokalny podprogram. Podprogramy **ROUTINE** są umieszczane na końcu procedury lub funkcji i wywołuje się je za pomocą instrukcji **DO**.

Liczne języki pozwalają na wykonanie pojedynczej instrukcji z listy instrukcji w oparciu o wybierającą ją liczbę całkowitą. W FORTRAN wykorzystuje się do tego wyliczone **GOTO**. COBOL korzysta z **GOTO...DEPENDING ON**. BASIC stosuje **ON...GOTO** oraz **ON...GOSUB**. Zaimplementowałem podobną funkcjonalność, dzięki której możemy wykonać instrukcję dowolnego typu z listy instrukcji w zależności od wyrażenia całkowitego. Nazwałem tę strukturę **EXECUTE** "na cześć" powszechnej instrukcji **XEQ** języka maszynowego wykonującej pojedynczą instrukcję adresowaną przez jej operand. Ta nowa struktura jest, jestem o tym głęboko przekonany, unikalna i stosowana tylko w języku Clarion, ale za to bardzo przydatna:

```
EXECUTE UpdateAction
  ADD(Master)
  PUT(Master)
  DELETE(Master)
END
```

Oswajanie interfejsu użytkownika

W roku 1970 pracowałem dla McDonnell Douglas Automation Company, gdzie uczestniczyłem w opracowaniu pierwszego komputera IV/70 budowanego dla Four Phase Systems, Inc. To była cudowna maszyna - 96K stałej pamięci, rozmiary niewiele większe od PC-ta. To, co czyniło to pudełko interesującym, to obsługa

wyświetlania: 32 terminale podłączone do 8 portów video odświeżanych bezpośrednio z pamięci. Przed IV/70, każdy terminal, którego używałem był urządzeniem komunikacyjnym. Można było śledzić pojedyncze znaki pojawiające się na ekranie zaraz po ich odebraniu przez terminal. W IV/70 cały nowy ekran pojawiał się co trzydziestą sekundy. Była to doskonała platforma dla programów interaktywnych. Jakoś nikt tego jednak nie zauważył. Four Phase sprzedawało system głównie jako zamiennik klastrowanych terminali IBM. Myślałem o dużo szerszym zastosowaniu. W roku 1973 założyłem firmę, która miała za zadanie opracować system informacyjny zarządzający szpitalem., bazujący na komputerze IV/70. Napisałem wielodostępny system operacyjny i język makropoleceń w nim działający. Następnie opracowałem pre-procesor makropoleceń i mały system informacyjny dla szpitala. Cały proces zajął mi około 9 miesięcy. Język makropoleceń odwoływał się do terminali tak, jakby posiadały własną pamięć (naprawdę tak było!). Aplikacja obsługi szpitala „malowała" ekran przenosząc znaki z pamięci terminala, następnie umieszczała opisy pól wprowadzania w tablicy pól użytkownika i przekazywała sterowanie systemowi operacyjnemu, by mógł przetworzyć dane. Gdy dane do pola zostały wprowadzone lub został wciśnięty specjalny klawisz, sterowanie było zwracane do aplikacji. Centralnym punktem tej strategii w sposób wyraźny był system operacyjny. Klawisze funkcyjne były podłączone do procedur ekranowych. Procedury ekranowe tworzyły tablice pól, które były podłączone do procedur edycji pól. Program nie działał w sensie konwencjonalnym. W rzeczywistości nie było czegoś takiego, jak program, a jedynie zestaw procedur reagujących na zdarzenia systemu operacyjnego. Wszystkim sterował system operacyjny. Był przygotowany na spełnianie potrzeb programisty. Nasi programiści stali się ostatecznie biegłymi specjalistami, gdyż większa część systemu szpitalnego musiała zostać zaprojektowana, zaimplementowana I w pełni przetestowana zanim sprzęt został połączony kabelkami. To nigdy nie było jednak intuicyjne. Każdy z naszych programistów przeszedł nieprawdopodobne przeszkolenie. Programowanie sterowane zdarzeniami jest trudne do opanowania. Później, jeden z najmocniejszych przebłysków intuicji, jakie kiedykolwiek miałem, uświadomił mi, że system sterowany zdarzeniami może być kontrolowany przez konwencjonalny program. Interfejs użytkownika mógł być wywołany przez pojedynczą instrukcję. W Clarionie nazwałem ją ACCEPT. Wyjście z pętli ACCEPT mogła zwrócić sterowanie do systemu operacyjnego. Wewnątrz tej pętli można było umieszczać wpisy odpowiadające za obsługę zdarzeń. Niewielki zestaw funkcji mógł zręcznie zidentyfikować, jakie zdarzenie zaszło i którego pola dotyczy.

Systemy sterowane zdarzeniami zawsze wydawały mi się "wewnętrznie-zewnętrzne". Byłem wewnątrz, przywiązany do wiosła i wysłuchiwałem rytmu podawanego prze „dobosza" przetwarzając zdarzenia, które mi przekazywał. Uświadomiłem sobie, że ACCEPT może ponownie uczynić mnie szefem. Teraz bęben należy do mnie! Mogłem wywoływać system operacyjny, a nie poruszać się okrężną drogą. Ale jak powinien Clarion odwzorowywać ekran? Cóż, jeśli literały ekranowe są danymi I pola ekranowe są danymi, układ ekranu możemy potraktować jak strukturę danych, nieprawdaż? Bez zbędnego udziwniania, nazwałem tę strukturę **SCREEN**. Instrukcja **OPEN**(MyScreen) wyświetla ekran. Pętla **ACCEPT** uaktywnia klawiaturę I obsługuje dane wprowadzane przez jej operatora. Gdy operator zakończy wprowadzania danych do pola lub naciśnie klawisz skrótu, instrukcja **ACCEPT** jest przerywana i zwraca sterowanie do programu. Instrukcja **CLOSE**(MyScreen) przywraca stan ekranu sprzed otwarcia "MyScreen".

Deklarowanie ekranów nie tylko ułatwia ich przetwarzanie, ale również samo projektowanie. Zespół pracujący nad Clarionem opracował specjalny moduł rysowania ekranów i zintegrował go ze środowiskiem; w efekcie jego działania są generowane odpowiednie struktury **SCREEN**. Moduł rysowania ekranów może oczywiście także i czytać struktury **SCREEN**. Potrzebny dowód? Wystarczy umieścić

w edytorze kodu źródłowego kursor wewnątrz struktury **SCREEN** i wywołać moduł rysowania ekranu. Zinterpretuje on kod źródłowy i wyświetli odpowiedni ekran. Teraz wystarczy nanieść nasze poprawki i powrócić do edycji kodu źródłowego. Stara struktura **SCREEN** jest zastępowana jej zmodyfikowaną wersją.

Interaktywne, wizualne tworzenie ekranów nie jest możliwe bez istnienia deklarowalnej struktury. Podobną utworzyłem dla raportów. Struktury **REPORT** zawierają układ drukowania linii raportu, nagłówki stron, stopki stron. Instrukcja **PRINT** odpowiada za odpowiednie zarządzanie drukowanymi danymi i rozmieszczanie ich na stronie. Edytor raportów jest także zintegrowany z edytorem kodu źródłowego, tak że struktury **REPORT** mogą być przetwarzane podobnie, jak struktury **SCREEN**.

Otwieranie okien

Tak się szczęśliwie złożyło, że układ naszego interfejsu doskonale pasował do Microsoft Windows. Programiści Windows przeżywali trudne chwile, kto był temu winien? Przykładowy programik wyświetlający „Hello World" napisany w popularnym C++ miał objętość 8 stron! Windows był potrzebny prosty model obsługi komunikatów, taki, jaki zastosowano w pętli ACCEPT Clariona. Zdecydowaliśmy się to zrobić. Zamieniliśmy strukturę SCREEN strukturą WINDOW, wprowadzając gramatykę niezbędną do deklarowania obiektów i właściwości systemu Windows. Wprowadziliśmy wielowątkowość w celu umożliwienia stosowania okien wewnętrznych MDI (multiple document interface). Zmieniliśmy gramatykę struktury REPORT, by było możliwe stosowanie trybu WYSIWYG, podkładów, zagnieżdżonych grup i ich nagłówków oraz stopek, czy wyliczeń.

Instrukcja ACCEPT stała się strukturą, w ramach której zamknięte zostało przetwarzanie zdarzeń. Przygotowaliśmy kompilator do kooperacji z bibliotekę uruchomieniową w celu ukrycia kierunku wywołań procedur wykorzystywanych do obsługi zdarzeń okna. Wywołanie do procesora działającego okna jest generowane powyżej pętli ACCEPT. Pętla sama w sobie jest generowana jako wbudowana procedura przetwarzania zdarzeń. Procesor okna tworzy niezbędne obiekty, wskazuje wspólną procedurę przetwarzania zdarzeń dla wszystkich zdarzeń przez nie generowanych. Ten procesor zdarzeń obsługuje „domowe" zdarzenia, takie jak na przykład odświeżanie oraz wywołuje procedurę wbudowaną przetwarzania zdarzeń do obsługi pozostałych. Gdy okno zostaje zamknięte, procesor okna zwraca sterowanie do instrukcji następującej po pętli ACCEPT.

Dla programisty Clarion to wszystko jest zupełnie proste. Otwiera okno, wpada w pętlę ACCEPT. Pętla ACCEPT powtarza się dla każdego zdarzenia, które może przechwycić program. Zamyka okna i wypada poza pętlę. Zdefiniowaliśmy konwencjonalny zestaw funkcji do identyfikacji zdarzeń I obiektów, których one dotyczą. Kod niezbędny do przetworzenia typowego okna dialogowego wygląda następująco:

```
OPEN(Window)                          ! Otwarcie okna
ACCEPT                                ! Uaktywnienie okna
  CASE FIELD()                        ! Które pole  wymaga uwagi?
  OF ?OK                              ! 'OK' wymaga uwagi
    CASE EVENT()                      ! Jakie zdarzenie zaszło?
    OF EVENT:Accepted                 ! 'OK' zostało wciśnięte
      :                               ! Przetwarzanie przycisku OK
      CLOSE(Window)                   ! Zamknięcie okna
    END                               ! Koniec CASE EVENT()
  OF ?Cancel                          ! 'Cancel' wymaga uwagi
    CASE EVENT()                      ! Jakie zdarzenie zaszło?
    OF EVENT:Accepted                 ! 'Cancel' zostało wciśnięte
      :                               ! Przetwarzanie przycisku 'Cancel'
      CLOSE(Window)                   ! Zamknięcie okna
    END                               ! Koniec CASE EVENT()
  ELSE                                ! To musi być zdarzenie niezależne od pola
    CASE EVENT()                      ! Jakie zdarzenie zaszło?
    OF EVENT:CloseWindow              ! Okno powinno zostać zamknięte
      :                               ! Przetwarzanie zamykanego okna
    END                               ! Koniec CASE EVENT()
  END                                 ! Koniec CASE FIELD()
END                                   ! Koniec ACCEPT
RETURN                                ! Powrót do miejsca wywołania
```

Produktem ubocznym naszej biblioteki uruchomieniowej, która została zorientowana obiektowo, stała się korekta poważnej niedoskonałości języka Clarion – braku wariantowości kompilatora. Deklarowanie ekranów, raportów, czy plików jest bardzo przejrzyste. Z drugiej jednak strony powoduje ograniczenia. Dlatego, że po skompilowaniu, nie można zmieniać większości deklaracji już w czasie działania programu. Większość rozszerzeń języka wnioskowanych przez programistów Clarion było związanych z udostępnieniem możliwości zmiany zadeklarowanych atrybutów przez program. W naszej bibliotece uruchomieniowej dla Windows te struktury są obiektami. Obiekty posiadają właściwości, a te oczywiście można zmieniać. W dowolnym czasie. Ponieważ już stosowaliśmy znak kropki zarówno w roli terminatora struktury, jak i znaku kropki dziesiętnej, nie mogliśmy zaimplementować standardowej notacji obiektowej w postaci *obiekt.właściwość*. Z tego względu zastosowaliśmy nawiasy klamrowe do wyróżnienia właściwości. W tej notacji, dowolny zadeklarowany atrybut , np. tekst wyświetlany w przycisku, może być modyfikowany poprzez instrukcję w postaci:

```
?Button{PROP:Text} = 'Mój przycisk'
```

Projektowanie bazy danych

Dążyłem do zaimplementowania prostej składni bazodanowej, która zapewniałaby obsługę trzech podstawowych metod dostępu do pliku: bezpośrednią, sekwencyjną i indeksowaną. Odpowiednia organizacja pliku również powinna być prosta: plik powinien zawierać nagłówek, po którym następowałyby rekordy danych stałej długości. Nagłówek powinien opisywać układ rekordu i powiązane z nim klucze oraz pola memo rezydujące w oddzielnych plikach.

Taka aranżacja jest zbliżona do używanej przez dBase – rekord może być dostępny sekwencyjnie lub bezpośrednio poprzez klucz lub jego względny numer. Opracowałem strukturę **FILE**, podobną do **FD** języka COBOL**,** do deklarowania plików i ich komponentów:

```
Detail      FILE,PRE(DTL),NAME('C:\LEDGER\DETAIL.DAT')
AcctKey       KEY(DTL:AcctNo,DTL:Period,DTL:Date)
BatchKey      KEY(DTL:Batch,DTL:Period),DUP
Comment       MEMO(4096)
              RECORD                                  ! Rekord danych
AcctNo          SHORT                                 ! Numer konta
Period          BYTE                                  ! Okres rozliczenia
Date            DATE                                  ! Data transakcji
Batch           STRING(12)                            ! Identyfikator
Amount          DECIMAL(12,2)                         ! Kwota (+/- = debet/kredyt)
              END
            END
```

Sekwencyjne przetwarzanie oparłem na instrukcjach **SET**, **NEXT**, **PREVIOUS** oraz **SKIP**. Instrukcja **SET** ustala sekwencję (wg klucza lub względnej pozycji rekordu) oraz punkt startowy dla pozostałych instrukcji pobierających rekordy. Te instrukcje można połączyć wraz z funkcją określania końca pliku (**EOF**) w pętlę przetwarzania rekordów:

```
SET(Dtl:AcctKey)            ! Ustawienie sekwencji numerów kont
LOOP UNTIL EOF(Detail)      ! Przeglądanie każdego rekordu
  NEXT(Detail)              ! Odczytanie następnego rekordu
  :
END
```

Instrukcja **GET** odczytuje record w sposób bezpośredni poprzez klucz lub jego względny numer. Co ważne, **GET** przeszkadza w sekwencyjnym przetwarzaniu poprzez resetowanie następnego rekordu. Instrukcje **PUT** i **DELETE** przetwarzają rekordy pobrane przez **NEXT**, **PREVIOUS** lub **GET**. Instrukcja **ADD** dopisuje nowy rekord do bazy danych.

Taka gramatyka dostępu do bazy danych zapewnia efektywność, niezawodność i wszechstronność – zasadniczy i popularny składnik naszego produktu. Ponieważ język Clarion cały czas się rozwijał, musiał spełniać nowe oczekiwania. Twórcy oprogramowania chcieli mieć dostęp do plików dBase. Tak więc dodaliśmy bibliotekę procedur dla dBase (nazwaliśmy biblioteki procedur modułami rozszerzeń języka - LEM - Language Extension Modules). Następnie pojawił się Novell z wbudowana obsługą środowiska klient-serwer w postaci Btrieve (indeksowanie bazujące na serwerze). Niektóre aplikacje Clarion potrzebowały Btrieve w celu zapewnienia lepszej obsługi transakcyjności. Tak więc dwie współpracujące z nami firmy opracowały LEM dla Btrieve. Później pojawiło się DB2. I RDB. I Oracle. I SQL Server. I wiele innych baz danych działających na PC-tach lub dostępnych z nich. Planowaliśmy obsługę bezpośrednich wywołań funkcji C w następnych wersjach języka, tak by dowolna baza z API opartym na C była dostępna z poziomu programu w Clarionie. Stało się jednak dla mnie jasne, że nie stanowi to dobrego rozwiązania. Język biznesowy ogólnego przeznaczenia nie powinien stosować różnych gramatyk dla różnych formatów baz danych. Migracja do innego formatu mogłaby spowodować wiele problemów z przeniesieniem do niego naszej aplikacji.

Język Clarion potrzebował standaryzacji, wbudowanej obsługi dla wszystkich popularnych baz danych. Sugerowano, że zaadaptowaliśmy SQL jako naszą gramatykę bazy danych. Wziąłem ją poważnie pod uwagę i spróbowałem przepisać kilka typowych programów w Clarionie posługując się wbudowanym SQL. Nie potrwało długo, a doszedłem do wniosku, że jest to zupełna utopia. Gdy stosuje się go jako język programowania, SQL staje się krańcowo „przegadany” i nieelegancki.

```
DECLARE X CURSOR
  FOR SELECT *
    FROM Detail
    ORDER BY Dtl:AcctNo,Dtl:Period,Dtl:Date
  END
END
OPEN X
LOOP
  FETCH X
  IF ReturnCode = 100 THEN BREAK.
  :
END
CLOSE X
```

Kursor SQL są nie tylko nieeleganckie, są wręcz niemal bezużyteczne. Nie można na przykład pominąć kursora, w celu odświeżenia na ekranie poprzedniej strony rekordów. Nie można ich także przemieszczać, na przykład skoczyć do „Jones" podczas przeglądania wg alfabetu. Doszedłem do wniosku, że po zamianie składni Clariona dostępu do bazy danych na składnię SQL musiałbym „w pośpiechu opuścić miasto".

Tak więc zdecydowaliśmy się na zaimplementowanie podmienianych sterowników baz danych. Programiści Clariona są przywiązani do swojej gramatyki, ale potrzebują dostępu do innych formatów baz danych. Budując na istniejącej strukturze języka nie chcieliśmy dać im możliwość wykorzystania posiadanej wiedzy i jednocześnie rozszerzenia tworzonych przez nich aplikacji. W naszej nowej technologii driver'ów baz danych, postaraliśmy się, by wszystkie bazy danych wyglądały jednakowo– jest to niebagatelna korzyść.

Nowy widok

By opracować sterowniki baz SQL, zmapowaliśmy składnię SQL na naszą własną gramatykę bazy danych. Nasza instrukcja **SET** konstruuje odpowiednią instrukcję SQL **SELECT**, która jest umieszczana przy pierwszym wystąpieniu operacji **NEXT** lub **PREVIOUS**. Jeśli zmienimy kierunek, np. **NEXT**...**PREVIOUS**, sterownik umieszcza inny **SELECT** z odmienną klauzulą **ORDER BY**. Nasza instrukcja **GET** umieszcza **SELECT**...**FETCH**. **ADD** wstawia **INSERT**. Instrukcja **GET**...**DELETE** wywołuje **DELETE**, a **GET**...**PUT** - **UPDATE**. Kilka funkcji, takich jak, relatywny dostęp do rekordu, nie jest obsługiwanych w bazach SQL, poza tym implementacja jest całkowicie kompletna.

Jednakże nasza gramatyka bazy danych nie jest w stanie zapewnić obsługi niektórych istotnych funkcji SQL. Programy w Clarionie implementują filtry rekordów podczas ich czytania, opuszczając po prostu te, które nie spełniają warunków:

```
LOOP UNTIL EOF(Part)
  NEXT(Part)
  IF Prt:OnHand > 0 THEN CYCLE
  :
END
```

Baza SQL może filtrować rekordy na serwerze, co oszczędza sporo czasu. Programy w Clarionie łączą pliki odczytując rekord nadrzędny w celu odczytania wartości klucza, który posłuży do pobrania rekordów podrzędnych. Baza SQL zwraca w rezultacie rekord nadrzędny i podrzędne przy jednej operacji dostępu. Programy w Clarionie odczytują wszystkie pola wszystkich rekordów, SQL daje w rezultacie tylko te pola, których wymagaliśmy. Oczywiście baza SQL nie zgaduje naszych życzeń.

Musimy jej ściśle określić, czego oczekujemy. Tak więc opracowaliśmy strukturę VIEW do tego właśnie celu:

```
View    VIEW(Part),FILTER('PRT:OnHand = 0')
            PROJECT(PRT:Number,PRT:Name,PRT:OnHand,PRT:Usage)
            JOIN(Vendor,PRT:Vendor,VND:Number)
              PROJECT(VND:Name,VND:Address,VND:CityStateZip)
            END
        END
```

Ta struktura VIEW określa zamierzenia programu Clariona, tak że driver bazy danych może wykorzystywać dowolne usługi oferowane przez motor bazy danych. Driver bazy danych wykonuje operacje filtrowania (wybór rekordów), łączenia (pobieranie powiązanych rekordów), wybierania (wybór pól) lub informuje serwer bazy danych, by je wykonał. W obu przypadkach wydajność jest optymalizowana.

Istnieje także problem z implementacją optymistycznej współbieżności pod SQL. By zaktualizować współdzielony plik, program w Clarionie odczytuje i zapamiętuje rekord. Następnie, przed aktualizacją, rekord jest blokowany, odczytywany ponownie, porównywana jest aktualna z zawartość z zapamiętaną. Jeśli są takie same, zmiany są zapamiętywane w bazie danych. Jeśli aktualna i zapamiętana zawartość są różne, oznacza to, że rekord w międzyczasie został zmieniony przez innego operatora. O tym fakcie jest informowany operator stacji roboczej, który chce zaktualizować rekord. Ten proces jest właśnie nazywany "optymistyczną współbieżnością", bazuje on na oczekiwaniu, że rekordy zazwyczaj pozostają niezmienione. SQL implementuje optymistyczną współbieżność za pomocą klauzuli WHERE, która wymaga, by wszystkie aktualizowane pola nie zmieniły w międzyczasie swej wartości. Jeśli tak się stanie w przypadku choć jednego z nich, zwracany jest stosowny błąd. Ponieważ Clarion nie posiadał składni spełniającej takie wymagania, dodaliśmy instrukcję WATCH. Jest ona wywoływana przed GET, NEXT, czy PREVIOUS w celu zainicjowania optymistycznej współbieżności. Gdy rekord jest pobierany, driver zapamiętuje jego kopię. W odpowiedzi na instrukcję PUT, sterownik albo odczytuje rekord ponownie w celu dokonania porównania, albo przekazuje instrukcję UPDATE...WHERE do bazy SQL. Jeśli rekord zmienił się, PUT daje w rezultacie błąd.

Nasz pierwszy kompilator

Wersja 1.0 Clariona pojawiła się na rynku w maju 1986 roku wraz z kompilatorem i interpreterem. Kompilator Clarion produkował kod pośredni, który był następnie interpretowany przez Clarion Processor. Kod pośredni był na tyle zwarty, że duże aplikacje mogły działać przy niewielkiej ilości dostępnej pamięci operacyjnej (256K), która charakteryzowała w owym czasie komputery PC. Udawało się to dzięki generowaniu przez kompilator binarnego opisu każdej instrukcji deklaracji. Następnie dane były adresowane przez dwubajtowy wskaźnik do opisu binarnego. Tak więc operacja dodania liczby całkowitej do łańcucha i sformatowania rezultatu zgodnie ze wzorcem zabierała pięć bajtów (jeden bajt dla operacji dodania, cztery bajty dla wskaźników na liczbę całkowitą i opisów wzorca). Dla każdej operacji Processor sprawdzał typy danych elementów, które brały w niej udział i przeprowadzał niezbędne konwersje.

Jednakże zwarty kod pośredni nie był podstawowym celem, który przyświecał nam przy projektowaniu. Interpretując wyjście z kompilatora, Processor mógł wykonywać aplikacje Clariona z pominięciem etapu linkowania. W roku 1985 i jeszcze długo później, linkowanie było procesem pochłaniającym bardzo dużo czasu. Nasi klienci

doceniali możliwość szybkiego testowania, jednak dawali nam również do zrozumienia, że "prawdziwe" języki programowania tworzą pliki wykonywalne .EXE! Już w następnym roku wypuściliśmy Clarion Translator konwertujący kod pośredni Clarion do plików **.**OBJ poprzez zastąpienie kodów operacji przez wywołania procedur. Wskaźniki były przekazywane jako parametry. Ta strategia dobrze nam służyła przez sześć lat, ale przysparzała też pewnych problemów:

- Mieliśmy problemy z bibliotekami zewnętrznymi **.**OBJ. Mogły być dołączane do pliku **.**EXE programu w Clarionie, jednak nie mogły być wykonywane bezpośrednio przez Processor. Stworzyliśmy proces, który konwertował odpowiednie pliki **.**OBJ na specjalny format binarny (LEM), który mógł być wykonywany przez procesor i zmieniany wstecz do **.**OBJ przez Translator. Proces ten jednak był skomplikowany i przydatny do stosowania tylko dla wyrafinowanych twórców oprogramowania.
- Proste programy w Clarionie produkowały duże pliki **.**EXE. Do pliku EXE były dołączane procedury, które nigdy nie były w nim wywoływane. Z tego powodu program „Hello World" miał 141KB.
- Aplikacje Clariona działały wolniej niż aplikacje napisane w C, Pascal-u, czy Moduli-2, ponieważ programy Clariona sprawdzały typy danych w czasie swego działania, a inne języki – w czasie kompilacji.
- Nie było potrzeby unikania linkowania w fazie testowania. Nowe linkery obsługujące biblioteki uruchomieniowe mogły zlinkować program do testowania tak szybko, że dłużej trwałoby ładowanie Processora.

Najważniejsze było to, że potrzebowaliśmy technologii, która pozwoliłaby nam przenieść się do Windows, trybu chronionego, OS/2, UNIX, 32-bitów, czy architektur innych niż Intel.

Nowy partner

W maju 1990 roku rozwiązaliśmy te i inne problemy nabywając licencję na technologię TopSpeed od Jensen & Partners International (JPI), firmy brytyjskiej. JPI została założona w roku 1988 przez Nielsa Jensena, założyciela Borland International, który odszedł z tej firmy wraz z grupą zajmującą się tworzeniem języka programowania. Opracowali oni linię produktów TopSpeed charakteryzujących się doskonałymi kompilatorami. Obejmowała ona kompilatory C, C++, Pascal oraz Modula-2 współdzielące ten sam generator kodu i system utrzymywania projektów. JPI określało kompilatory jako „front-end", a generator kodu – jako „back-end".

Zaczęliśmy niezwłocznie tworzyć front-end Clariona. Jak zwykle okazało się to trudniejsze, niż początkowo sądziliśmy. Język wymagał większych zmian, niż oczekiwaliśmy. Cały projekt trwał dłużej i pochłonął więcej środków, niż było to zakładane. Ale rezultaty okazały się wstrząsające. Wiedzieliśmy, że back-end TopSpeed był dobry, ale zdumieliśmy się, gdy "Sito Eratostenesa" (algorytm znajdujący liczby pierwsze) napisane w Clarionie działało dwa razy szybciej, niż ten sam program napisany w Turbo C++ Borlanda. Kupiliśmy również licencję na technologię linkowania TopSpeed nie wyobrażając sobie nawet jak bardzo jest dobra. Unikalna technologia „Smart Linking" TopSpeed produkuje doskonały kod eliminując z pliku wykonywalnego .EXE wszelkie procedury i elementy danych statycznych, do których nie ma odwołań. Co więcej, podczas gdy pracowaliśmy nad własnym front-end, JPI opracowało automatyczny ładowacz nakładek, biblioteki DLL dla DOS, DOS extender i zaanonsowało wsparcie dla 32-bitów. Dzięki tej technologii pozbyliśmy się

ostatecznie problemów z wydajnością, które charakteryzują wszystkie języki wysokiego poziomu.

We wrześniu 1991 przedstawiliśmy nasz nowy produkt na pierwszej konferencji Clarion Developers. Zaprezentowane zostały nowe funkcje oraz przesłanki współpracy Clarion z TopSpeed. Na gorąco, korzystając z okazji, ja i Niels Jensen, zaczęliśmy rozmawiać o połączeniu obu firm. Łatwo uzyskaliśmy porozumienie. Produkty TopSpeed mogły dzięki temu zaistnieć na rynku amerykańskim I dotrzeć do szerokiej rzeszy programistów. Produkty Clarion pozyskały ich technologię. Mogliśmy być pierwszymi, którzy połączyli wiodącą technologię kompilatorów z narzędziami wytwarzania aplikacji biznesowych. Po trochę przydługich negocjacjach, połączenie firm zostało przypieczętowane w kwietniu 1992 roku. Dwa i pół roku później, po kompletnym zjednoczeniu działań i linii produktów obu firm, doszło do zmiany nazwy na TopSpeed Corporation. W październiku 1994, TopSpeed Corporation wypuściła na rynek Clarion for Windows, pierwszy wspólny produkt połączonych firm.

Teraźniejszość

Te zapiski tworzyły początkowo wstęp do *Programmer's Guide*, dostarczanego wraz z Clarion Database Developer Version 3.0, od kwietnia 1993 r. Uzupełnienia i poprawki są konieczne za względu na pojawienie się kolejnych wersji Clariona dla Windows. Są one cały czas wprowadzane. Wytwarzanie oprogramowania to dla mnie proces mozolnego układania konstrukcji z klocków, dopóki nie osiągnie ona spójnej całości. Wierzę, że zamierzonym finałem będzie prawidłowy projekt. W odniesieniu do Clarion for Windows czułem, że jeszcze daleka droga przed nami. Teraz już nie jestem tego taki pewien. pozostało do ułożenia już tylko kilka klocków…

1 - WPROWADZENIE

Language Reference Manual

Clarion jest zintegrowanym środowiskiem przeznaczonym do pisania aplikacji bazodanowych i zarządzających informacją działających pod kontrolą systemu operacyjnego Windows. Fundamentem tego środowiska jest język programowania Clarion. W tym podręczniku można znaleźć dokumentacje tego języka ujętą w nowoczesną formę. Chociaż nie jest to książka drukowana, w niej pierwszej powinniśmy sprawdzać składnię, której chcemy użyć w deklaracjach, instrukcjach, czy funkcjach. Wszędzie, gdzie było to możliwe zamieściliśmy przykłady wzięte "z życia" w celu zilustrowania przykładów zastosowania opisywanych elementów języka.

Organizacja rozdziałów

ROZDZIAŁ 1 - Wprowadzenie stanowi wprowadzenie do podręcznika Clarion Language Reference. Zawiera krótki opis każdego rozdziału, przewodnik ułatwiający posługiwanie się konwencjami przyjętymi w podręczniku.

ROZDZIAŁ 2 - Format kodu źródłowego programu zawiera ogólny opis szkieletu programu w języku Clarion (dla Windows). Znajdziemy tu opisy zasad stosowania znaków interpunkcyjnych, znaków specjalnych, słów zastrzeżonych, zasad budowania bloków kodu źródłowego języka Clarion.

ROZDZIAŁ 3 – Deklaracje zmiennych zawiera opisy prostych typów danych stosowanych do deklarowania zmiennych. Zawarte są tu też informacje o wzorcach formatowania, za pomocą których są wyświetlane wartości zmiennych.

ROZDZIAŁ 4 – Deklaracje egzemplarzy opisuje wszystkie złożone struktury danych, takie jak GROUP, CLASS, FILE, VIEW i QUEUE.

ROZDZIAŁ 5 – Atrybuty deklaracji opisuje wszystkie atrybuty mogące modyfikować deklaracje zmiennych i egzemplarzy.

ROZDZIAŁ 6 - Okna opisuje struktury APPLICATION i WINDOW oraz ich elementy składowe.

ROZDZIAŁ 7 - Raporty opisuje strukturę REPORT i jej elementy składowe.

ROZDZIAŁ 8 - Kontrolki zawiera opisy wszystkich kontrolek, które możemy umieszczać w oknach APPLICATION i WINDOW oraz w raportach REPORT.

ROZDZIAŁ 9 Atrybuty okien i raportów zawiera opisy wszystkich atrybutów, które mogą modyfikować deklaracje struktur APPLICATION, WINDOW i REPORT oraz kontrolek w nich zawartych.

ROZDZIAŁ 10 - Wyrażenia opisuje składnię umożliwiającą łączenie zmiennych, procedur i stałych w wyrażenia numeryczne, łańcuchowe i logiczne.

ROZDZIAŁ 11 - Przypisania definiuje wszystkie metody pozwalające na przypisanie wartości lub wyrażenia do zmiennej. W tym rozdziale są również opisane

operacje BCD (na liczbach dziesiętnych kodowanych binarnie) oraz zasady konwersji typów danych.

ROZDZIAŁ 12 – Sterowanie kodem opisuje złożone instrukcje wykonywalne umożliwiające sterownie przepływem kodu.

ROZDZIAŁ 13 – Procedury wbudowane dokumentuje wszystkie wbudowane procedury Clariona.

DODATEK A - DDE, OLE i OCX dokumentuje procedury umożliwiające dynamiczną wymianę danych (Dynamic Data Exchange - DDE), łączenie i osadzanie obiektów (Object Linking and Embedding - OLE) oraz obsługę kontrolek OCX (OLE Custom Controls).

DODATEK B - Zdarzenia dokumentuje instrukcje EQUATE dla zdarzeń generowanych i obsługiwanych przez aplikację napisaną w Clarionie.

DODATEK C – Właściwości runtime dokumentuje właściwości runtime.

DODATEK D – Kody błędów dokumentuje błędy kompilacji i działania programu.

DODATEK E – Instrukcje wcześniejszych wersji zawiera opisy instrukcji języka Clarion zachowane w celu zapewnienia zgodności z poprzednimi jego wersjami.

Konwencje i symbole dokumentacji

Symbole są stosowane w diagramach składni w sposób nastepujący:

Symbol	Znaczenie
[ ]	Nawiasy kwadratowe oznaczają opcjonalny (nie wymagany) atrybut lub parametr.
( )	Nawiasy okrągłe wyznaczają listę parametrów.
| |	Kreski pionowe wyznaczają listy parametrów, z których tylko jedna jest możliwa do zastosowania.

Konwencja zapisu przykładów kodu źródłowego stosowana w podręczniku jest następująca:

```
IF NOT SomeDate          ! IF oraz NOT to słowa kluczowe
  JakasData = TODAY()    ! JakasData to nazwa zmiennej zawierającej datę
END                      ! TODAY i END są słowami kluczowymi
```

SŁOWA KLUCZOWE CLARION	Dowolne słowo pisane samymi wielkimi literami jest słowem kluczowym języka Clarion.
NazwyDanych	Stosowane są wielkie i małe litery przy definiowaniu nazw zmiennych, obiektów itp.
Komentarze	Są pisane zazwyczaj małymi literami

Przyjęcie powyższych konwencji miało na celu zapewnienie większej przejrzystości i czytelności przykładów kodu źródłowego.

Format definicji w podręczniku

Każdy element języka programowania Clarion użyty w podręczniku jest pisany WIELKIMI LITERAMI.. Elementy języka są opisane w specjalnych diagramach składni, z dokładnymi opisami i przykładami kodu źródłowego.

Elementy te są dokumentowane w grupach zestawionych logicznie, z uwzględnieniem ich hierarchicznych zależności. Dlatego też spis treści tego podręcznika nie jest ułożony alfabetycznie. Generalnie, typy danych i struktury są dokumentowane na początku rozdziału, następnie są opisywane ich atrybuty, później instrukcje, a na końcu funkcje.

Format dokumentacji stosowany w podręczniku został przedstawiony poniżej.

SŁOWO_KLUCZOWE (krótki opis)

[etykieta] **SŁOWO_KLUCZOWE**(| *parametr1*| [*parametr2*]) [**ATRYBUT1**( )] [**ATRYBUTE**( )]
| *alternatywa* |
| *parametr* |
| *lista* |

SŁOWO_KLUCZOWE	Skrócony opis instrukcji.
parametr1	Kompletny opis parametru, włącznie z jego relacjami z innymi parametrami.
alternatywna lista parametrów	Kompletny opis alternatywnych parametrów, włącznie z relacjami z innymi parametrami.
parametr2	Kompletny opis parametru, włącznie z jego relacjami z innymi parametrami. Ujęcie go w nawiasy kwadratowe oznacza, że jest to parametr opcjonalny – możemy go użyć lub pominąć.
ATRYBUT1	Opis atrybutu i jego wpływu na SŁOWO_KLUCZOWE.
ATRYBUT2	Opis atrybutu i jego wpływu na SŁOWO_KLUCZOWE.

Zwięzły opis działania **SŁOWO_KLUCZOWE**. W wielu przypadkach SŁOWO_KLUCZOWE będzie atrybutem wcześniej opisanego elementu języka. Czasami SŁOWO_KLUCZOWE nie posiada ani parametrów, ani atrybutów.

Generowane zdarzenia: Jeśli SŁOWO_KLUCZOWE powoduje wygenerowanie jakichś zdarzeń, są one wymieniane w tym miejscu.

Typ rezultatu: Typ zwracanej danej, jeśli SŁOWO_KLUCZOWE jest funkcją

Raportowane błędy: Jeśli SŁOWO_KLUCZOWE generuje błędy, które są przechwytywane przez funkcje ERROR i ERRORCODE, są one wyszczególnione w tym miejscu.

Powiązane procedury: Jeśli SŁOWO_KLUCZOWE definiuje strukturę danych, procedury, które operują na tej strukturze są wymienione w tym miejscu.

Przykład:

```
PoleJeden = PoleDwa + PoleTrzy                          ! To jest przykład kodu źródłowego
PoleTrzy = SŁOWO_KLUCZOWE(PoleJeden, PoleDwa)  ! Komentarze umieszczamy po wykrzykniku
```

Porównaj: Inne powiązane z danym opisem słowa kluczowe i opisy

Konwencje Clariona

Standardowa data

Standardowa data Clariona jest liczbą dni, które upłynęły od 28 grudnia 1800 r. Zakres dostępnych dat rozciąga się od 1 stycznia 1801 r. (wartość 4) do 31 grudnia 9999 r. (wartość 2.994.626). Procedury operujące na datach nie dadzą prawidłowego rezultatu dla dat leżących poza wymienionym zakresem. Kalendarz Clariona uwzględnia oczywiście wszystkie lata przestępne leżące w zdefiniowanym zakresie.

Dzieląc datę standardową modulo 7 otrzymujemy dzień tygodnia, gdzie: 0 = Niedziela, 1 = Poniedziałek itd.

Typ LONG ze wzorcem formatowania daty czasu (@D) jest standardowo używany jako typ służący do przechowywania daty. Wprowadzenie do pola daty z dwoma tylko cyframi określającymi rok, powoduje domyślne przyjęcie stulecia dla następnych 20 lub poprzednich 80 lat. Na przykład, jeśli wprowadzimy datę 01/01/01 rezultatem będzie data 01/01/2001 jeśli bieżący rok (wg wskazania zegara systemowego) jest późniejszy niż 1980, a 01/01/1901 jeśli bieżącym rokiem jest 1980 lub wcześniejszy.

Typ DATE jest zaś formatem stosowanym przez Btrieve Record Manager i niektóre inne systemy plików. Pole DATE jest wewnętrznie konwertowane do typu LONG zawierającego standardową datę Clariona, przed wykonaniem jakiejkolwiek operacji matematycznej lub funkcji posługującej się datą. Z tego powodu formatu DATE powinniśmy używać tylko w sytuacjach wymagających zachowania zgodności z zewnętrznymi formatami plików. Standardowo zawsze używamy typu LONG.

Porównaj: Wzorce daty, DAY, MONTH, YEAR, TODAY, SETTODAY, DATE

Standardowy czas

Standardowy czas w Clarionie jest liczbą setnych sekundy, które upłynęły od północy, plus jeden (1). Prawidłowy zakres jest zatem następujący: od 1 (północ) do 8.640.000 (czyli godzina 11:59:59.99). Standardowy czas równy 1 oznacza dokładnie północ. Taka zasada umożliwia wpisywanie do zmiennej czasowej wartości 0, która oznacz, że czas nie został wprowadzony. Chociaż czas jest wyrażany z dokładnością do najbliższej setnej sekundy, zegar systemowy aktualizuje się tylko 18,2 razy na sekundę (w przybliżeniu – co 5,5 setnej sekundy).

Typ LONG ze wzorcem formatowania czasu (@T) jest standardowo używany jako typ służący do przechowywania czasu. Typ TIME jest zaś formatem zachowanym w celu zapewnienia zgodności z Btrieve Record Manager. Pole TIME jest wewnętrznie konwertowane do typu LONG zawierającego standardowy czas Clariona przed wykonaniem jakiejkolwiek operacji matematycznej lub funkcji posługującej się czasem. Z tego powodu formatu TIME powinniśmy używać tylko w sytuacjach wymagających zachowania zgodności z Btrieve. Standardowo zawsze używamy typu LONG.

Porównaj: Wzorce czasu, CLOCK, SETCLOCK

Kody klawiszy w Clarionie

Format mapowania kodów klawiszy Windows

Każdy klawisz klawiatury ma przypisaną wartość numeryczną – kod klawisza. Kody klawiszy są liczbami 16-tkowymi, gdzie mniej znaczące 8 bitów (wartości od 0 do 255) reprezentuje kod wciśniętego klawisza, a bardziej znaczące 8 bitów określa status klawiszy Shift, Ctrl i Alt. Kody klawiszy są zwracane przez funkcje KEYCODE() i KEYBOARD() i stosują następujący format:

	A	C	S	KOD
Bity:	10	9	8	7 0

KOD – Kod wciśniętego klawisza
A - bit klawisza Alt
C - bit klawisza Ctrl
S - bit klawisza Shift

Obliczanie wartości numerycznej dla wciśniętej kombinacji klawiszy nie jest konieczne, gdyż zostały one już obliczone i zdefiniowane za pomocą instrukcji EQUATE w pliku KEYCODES.CLW.

KEYCODES.CLW

Etykiety ekwiwalentów kodów klawiszy przypisują nazwy mnemoniczne dla różnych kombinacji klawiszy. Są one zdefiniowane w pliku KEYCODES.CLW, oczywiście w języku Clarion, poprzez zastosowanie instrukcji EQUATE dla każdego z nich. Plik, o którym mowa jest zlokalizowany w katalogu \CLARIONX\LIBSRC. Jest on scalany z naszym kodem źródłowym poprzez umieszczenie następującej instrukcji w sekcji danych globalnych:

```
INCLUDE('KEYCODES.CLW')
```

Plik zawiera ekwiwalenty EQUATE dla zdecydowanej większości kodów klawiszy obsługiwanych przez Windows. Stosuje się je w celu zapewnienia czytelności kodu, eliminując przy tym konieczność żmudnego poszukiwania numerycznej wartości dla każdej kombinacji klawiszy, którą chcemy użyć (sprawdzić) w kodzie źródłowym

Porównaj: KEYCODE, KEYBOARD, KEYCHAR, KEYSTATE, SETKEYCODE, ALERT, ALRT

2 – FORMAT KODU ŹRÓDŁOWEGO PROGRAMU

Format instrukcji

Clarion jest językiem zorientowanym na instrukcje. Jak to bywa w przypadku takich języków, kod źródłowy programu jest zawarty w pliku ASCII, w którym każdy wiersz kodu stanowi oddzielny rekord. Wyznacznikiem końca rekordu jest kod CR/LF (Carriage Return/Line).

Ogólnie, format instrukcji Clariona wygląda następująco:

```
label STATEMENT[(parameters)] [,ATTRIBUTE[(parameters)]] ...
```

Atrybuty określają właściwości elementu i są wykorzystywane tylko w deklaracjach danych. Instrukcje wykonywalne przyjmują postać standardowego wywołania procedury, za wyjątkiem instrukcji przypisania (A = B) i struktur sterujących przepływem kodu, takich jak IF, CASE, czy LOOP.

Etykieta instrukcji musi zaczynać się w pierwszej kolumnie kodu źródłowego. Instrukcja nie posiadająca etykiety **nie może** zaczynać się w pierwszej kolumnie. Instrukcja jest kończona znakiem końca linii. Jeśli instrukcja jest zbyt długa, by mogła się zmieścić w jednym wierszu możemy ją kontynuować w następnym po uprzednim użyciu znaku pionowej kreski (|). Opcjonalnym znakiem separującym instrukcje jest średnik. Jego zastosowanie pozwala na wpisanie kilku instrukcji w jednym wierszu – oczywiście oddzielamy je od siebie właśnie średnikiem.

Ze względu na to, że Clarion jest językiem zorientowanym instrukcyjnie, wyeliminowano z niego większość znaków interpunkcyjnych wymaganych w innych językach do identyfikowania etykiet, czy oddzielnych instrukcji. Bloki instrukcji są inicjowane poprzez pojedynczą instrukcję złożoną, a kończone instrukcją END (bądź równoważnym znakiem kropki).

Deklaracje i etykiety instrukcji

Instrukcje języka w module kodu źródłowego można podzielić na dwie ogólne kategorie: deklaracje danych i instrukcje wykonywalne lub prościej – dane i kod.

Podczas wykonywania programu deklaracje danych rezerwują obszary pamięci, na których operują później instrukcje wykonywalne. Dana wymaga etykiety po to, by można było później odwoływać się do niej w kodzie wykonywalnym. Wszystkie zmienne, struktury danych, procedury (PROCEDURE), czy podprogramy (ROUTINE) są identyfikowane przez etykiety.

Etykieta definiuje określone miejsce programu PROGRAM. Dowolna instrukcja kodu może być identyfikowana i można się do niej odwoływać poprzez etykietę. Można dzięki temu używać instrukcji GOTO i nakazywać wykonanie kodu programu od konkretnej instrukcji identyfikowanej przez etykietę. Każda etykieta odnosząca się do instrukcji wykonywalnej zwiększa o dziesięć bajtów rozmiar kodu wykonywalnego, nawet wtedy, gdy w ogóle się do niej nie odwołujemy.

Etykieta dołączona do instrukcji PROCEDURE stanowi nazwę procedury. Umieszczenie etykiety procedury PROCEDURE w kodzie wykonywalnym powoduje jej wywołanie. W przypadku, gdy użyjemy jej w wyrażeniu lub w postaci parametru innej procedury – powoduje przypisanie wartości zwracanej przez procedurę.

Zasady budowania poprawnych etykiet Clariona są następujące:

- Etykieta MUSI zaczynać się w pierwszej kolumnie (1) kodu źródłowego.
- Etykieta może zawierać litery (małe lub wielkie), cyfry z zakresu od 0 do 9, znaki podkreślenia (_) oraz średnika (:).
- Pierwszym znakiem etykiety musi być litera bądź znak podkreślenia.
- Etykiety nie są zależne od wielkości liter (np. CurRent i CURRENT to ta sama etykieta).
- Etykietą nie może być słowo zarezerwowane.

Zakończenie struktury

Złożone struktury danych są tworzone wtedy, gdy deklaracje danych są zagnieżdżane w innych deklaracjach danych. Istnieje wiele struktur złożonych w języku Clarion: APPLICATION, WINDOW, REPORT, FILE, RECORD, GROUP, VIEW, QUEUE, itd. Definicje wymienionych struktur danych muszą zostać zakończone znakiem kropki (.) bądź słowem kluczowym END.

IF, CASE, EXECUTE, LOOP, BEGIN oraz ACCEPT są strukturami kontroli przepływu kodu. One również muszą zostać zakończone znakiem kropki (.) lub słowem kluczowym END. W przypadku pętli LOOP występuje również możliwość zakończenia jej instrukcjami WHILE lub UNTIL.

Kwalifikacja pól

Zmienne zadeklarowane jako składowe struktur złożonych (GROUP, QUEUE, FILE, RECORD, itd.) mogą posiadać zduplikowane nazwy, pod warunkiem jednak, że nie nastąpi to w ramach tej samej struktury. W celu zachowania możliwości dokładnego określenia zmiennej (przy duplikowaniu nazw), do której się odwołujemy, możemy skorzystać z atrybutu PRE nadawanego strukturze. Wówczas do zmiennej odwołujemy się poprzedzając jej nazwę prefiksem zdefiniowanym właśnie za pomocą wspomnianego atrybutu (Prefiks:NazwaZmiennej). Definiowanie atrybutu PRE nie jest obowiązkowe.

Dowolny składnik struktury złożonej może być dokładnie wskazany poprzez poprzedzenie jego nazwy nazwą struktury (rozdzielamy je znakiem kropki): NazwaStruktury.NazwaZmiennej). Nazywamy to składnią kwalifikacji pól. Musimy jej używać w odniesieniu do dowolnego pola struktury złożonej, dla której nie określono prefiksu (za pomocą atrybutu PRE). Przy tworzeniu referencji do pola struktury złożonej możemy stosować znak dwukropka (:) zamiast znaku kropki. Jedynym wyjątkiem są klasy CLASS oraz zmienne referencyjne. Należy również mieć na uwadze fakt, że wspomniana możliwość została zachowana tylko i wyłącznie w celu zachowania zgodności z wcześniejszymi wersjami Clarion.

Jeżeli zmienna jest zdefiniowana wewnątrz złożonej struktury danych, musimy dodawać etykiety kolejnych poziomów (zagnieżdżeń) struktury do nazwy zmiennej podczas tworzenia do niej odwołania (o ile zagnieżdżona struktura posiada etykietę). Jeśli dowolna z zagnieżdżonych struktur nie posiada etykiety, jest ona pomijana w tworzeniu kwalifikatora pola. Na przykład: w przypadku struktury GROUP nie posiadającej prefiksu, w której jest zagnieżdżona następna struktura GROUP posiadająca etykietę, odwołania do pól grupy wewnętrznej tworzymy w sposób następujący: *EtykietaGrupyZewnętrznej.EtykietaGrupyWewnętrznej.NazwaPola*. Jeżeli wewnętrzna grupa nie posiada etykiety, kwalifikator pola wyglądał będzie tak: *EtykietaGrupyWewnętrznej.NazwaPola*. Istnieje jeden wyjątek od tej zasady: etykieta struktury RECORD zdefiniowanej wewnątrz struktury FILE może zostać pominięta lub nie, a i tak do pól pliku odwołujemy się tak: *EtykietaPliku.NazwaPola* zamiast *EtykietaPliku.EtykietaRekordu.NazwaPola*.

Składnia kwalifikacji pól jest także stosowana w odniesieniu do wszystkich składników struktury CLASS – zarówno pól, jak i metod. Wywołanie metody danej klasy wymaga wskazania jej nazwy i, po kropce, nazwy metody: *NazwaKlasy.EtykietaMetody*.

Jeżeli chcemy utworzyć odwołanie do elementu struktury GROUP posiadającej atrybut DIM, musimy określić numer elementu tablicy (na tym poziomie, którego atrybut DIM dotyczy).

Przykład:

```
MasterFile   FILE,DRIVER('TopSpeed')
Record         RECORD
AcctNumber       LONG                    ! Referencja jako Masterfile.AcctNumber
               . .
Detail       FILE,DRIVER('TopSpeed')
               RECORD
AcctNumber       LONG                    ! Referencja jako Detail.AcctNumber
               . .
Memory       GROUP,PRE(Mem)
Message        STRING(30)                ! Może być referencja jako Mem:Message lub Memory.Message
             END
SaveQueue    QUEUE
Field1         LONG                      ! Referencja jako SaveQueue.Field1
Field2         STRING                    ! Referencja jako SaveQueue.Field2
             END
OuterGroup   GROUP
Field1         LONG                      ! Referencja jako OuterGroup.Field1
Field2         STRING                    ! Referencja jako OuterGroup.Field2
InnerGroup     GROUP
Field1           LONG                    ! Referencja jako OuterGroup.InnerGroup.Field1
Field2           STRING                  ! Referencja jako OuterGroup.InnerGroup.Field2
               END
             END
OuterGroup   GROUP,DIM(5)
Field1         LONG                      ! Referencja jako OuterGroup[1].Field1
InnerGroup     GROUP,DIM(5)              ! Referencja jako OuterGroup[1].InnerGroup
Field1           LONG                    ! Referencja jako OuterGroup[1].InnerGroup[1].Field1
               END
             END
```

Porównaj: PRE, CLASS, Zmienne referencyjne

Słowa zastrzeżone

Poniższe słowa kluczowe są zastrzeżone i nie mogą być wykorzystywane jako etykiety:

ACCEPT	**AND**	**BEGIN**	**BREAK**	**BY**
CASE	**CHOOSE**	**COMPILE**	**CYCLE**	**DO**
ELSE	**ELSIF**	**END**	**EXECUTE**	**EXIT**
FUNCTION	**GOTO**	**IF**	**INCLUDE**	**LOOP**
MEMBER	**NEW**	**NOT**	**NULL**	**OF**
OMIT	**OR**	**OROF**	**PARENT**	**PROCEDURE**
PROGRAM	**RETURN**	**ROUTINE**	**SECTION**	**SELF**
THEN	**TIMES**	**TO**	**UNTIL**	**WHILE**
XOR				

Przedstawione poniżej słowa kluczowe mogą być stosowane jako etykiety struktur danych lub instrukcji wykonywalnych. Nie mogą być jednak etykietami procedury PROCEDURE. Mogą się one pojawić jako etykiety parametrów w prototypie procedury, tylko wtedy jednak, gdy określimy również typ danych:

APPLICATION	**CLASS**	**CODE**	**DATA**	**DETAIL**
FILE	**FOOTER**	**FORM**	**GROUP**	**HEADER**
ITEM	**ITEMIZE**	**JOIN**	**MAP**	**MENU**
MENUBAR	**MODULE**	**OLECONTROL**		**OPTION**
QUEUE	**RECORD**	**REPORT**	**ROW**	**SHEET**
TAB	**TABLE**	**TOOLBAR**	**VIEW**	**WINDOW**

Znaki specjalne

Inicjatory:	!	Wykrzyknik rozpoczyna komentarz w kodzie źródłowym.
	?	Znak zapytania rozpoczyna etykietę ekwiwalentu pola.
	@	Znak „at” rozpoczyna wzorzec.
	*	Znak gwiazdki poprzedza parametr przekazywany poprzez adres w prototypie sekcji MAP.
	~	Tylda poprzedzająca nazwę pliku identyfikuje plik dołączany (linkowany) do projektu.
Terminatory:	;	Średnik służy do rozdzielania instrukcji wykonywalnych.
	CR/LF	Carriage-return/Line-feed (powrót karetki, nowy wiersz) oddziela instrukcje kodu wykonywalnego.
	.	Znak kropki kończy definicję struktury danych lub kodu (jest substytutem END).
	|	Pozioma kreska oznacza kontynuowanie kodu źródłowego w następnym wierszu.
	#	Deklaruje implikowaną zmienną typu LONG.
	$	Deklaruje implikowaną zmienną typu REAL.
	”	Deklaruje implikowaną zmienną typu STRING.
Ograniczniki:	( )	Nawiasy ograniczają listę parametrów.
	[ ]	Kwadratowe nawiasy ograniczają listę indeksów tablicy.
	’ ‘	Cudzysłowy pojedyncze ograniczają stałą łańcuchową.
	{ }	Nawiasy klamrowe ograniczają liczbę powtórzeń w stałej łańcuchowej bądź parametr stanowiący właściwość.
	< >	Nawiasy trójkątne ograniczają kod ASCII w stałej łańcuchowej bądź wskazują parametr w prototypie MAP, który może zostać pominięty.
	:	Znak dwukropka oddziela pozycję początkową i końcową fragmentu łańcucha.
	,	Znak przecinka oddziela parametry w liście parametrów.
Łączniki:	:	Dwukropek łączy prefiks z etykietą.
	.	Znak kropki oznacza pozycje kropki dziesiętnej w liczbach, służy również do łączenia etykiet złożonych struktur danych z etykietami ich elementów (pól, metod).
	$	Łączy etykietę okna WINDOW lub raportu REPORT z etykietą ekwiwalentu pola w wyrażeniach przypisania wartości właściwościom kontrolek..
Operatory:	+	Operacja dodawania.
	-	Operacja odejmowania.
	*	Operacja mnożenia.
	/	Operacja dzielenia.
	%	Operacja dzielenia całkowitego.

^	Operacją potęgowania.
<	Mniejsze niż.
>	Większe od.
=	Równe lub równoważne.
~	Logiczny (Boolean) operator NOT – negacja bitowa.
&	Operacja łączenia łańcuchów.
&=	Przypisanie referencyjne bądź równoważność referencyjna.

Format programu

PROGRAM (deklaruje program)

```
PROGRAM
MAP
prototypes
    [MODULE( )
       prototypes
    END ]
END
global data
    CODE
    statements
    [RETURN]
procedures
```

PROGRAM	Pierwsza deklaracja w module kodu źródłowego programu napisanego w języku Clarion. Wymagane.
MAP	Globalne deklaracje procedur. Wymagane.
MODULE	Deklaruje przynależne moduły kodu źródłowego.
prototypes	Deklaracje procedur.
global data	Deklaruje dane globalne widzialne dla wszystkich procedur.
CODE	Kończy sekcję deklaracji danych i zaczyna sekcję kodu wykonywalnego programu PROGRAM.
statements	Wykonywalne instrukcje programu.
RETURN	Przerywa wykonanie programu. Zwraca sterowanie do systemu operacyjnego.
procedures	Kod źródłowy dla procedur programu w module PROGRAM.

Wymaga się, by instrukcja **PROGRAM** była pierwszą deklaracją w module kodu źródłowego programu napisanego w języku Clarion. Może ona być ewentualnie poprzedzona komentarzem.

Nazwa pliku zawierającego kod źródłowy programu PROGRAM staje się nazwą pliku .OBJ oraz nazwą pliku wykonywalnego .EXE, oczywiście po kompilacji. Instrukcja PROGRAM może posiadać etykietę, jest ona jednak ignorowana przez kompilator.

Program PROGRAM z procedurami PROCEDURE musi mieć zadeklarowaną strukturę MAP. Służy ona do prototypowania procedur. Dowolna procedura zawarta w oddzielnym pliku kodu źródłowego musi zostać zadeklarowana wewnątrz struktury MODULE umieszczonej w sekcji MAP.

Dane zadeklarowane w module kodu źródłowego programu PROGRAM pomiędzy słowami kluczowymi PROGRAM i CODE są danymi globalnymi. Odwoływać się do nich może dowolna procedura programu. Pamięć dla tych danych jest przydzielana statycznie (Static).

Przykład:

```
            PROGRAM                     ! Deklaracja przykładowego programu
              INCLUDE('EQUATES.CLW')    ! Dołączenie standardowych ekwiwalentów
              MAP
    CalcTemp   PROCEDURE                ! Prototyp procedury
              END
      CODE
      CalcTemp                          ! Wywołanie procedury

CalcTemp     PROCEDURE
Fahrenheit     REAL(0)                  ! Deklaracje danych globalnych
Centigrade     REAL(0)
Window   WINDOW('Temperature Conversion'),CENTER,SYSTEM
           STRING('Enter Fahrenheit Temperature: '),AT(34,50,101,10)
           ENTRY(@N-04),AT(138,49,60,12),USE(Fahrenheit)
           STRING('Centigrade Temperature:'),AT(34,71,80,10),LEFT
           ENTRY(@N-04),AT(138,70,60,12),USE(Centigrade),SKIP
           BUTTON('Another'),AT(34,92,32,16),USE(?Another)
           BUTTON('Exit'),AT(138,92,32,16),USE(?Exit)
         END
  CODE                                  ! Początek sekcji kodu wykonywalnego
  OPEN(Window)
  ACCEPT
    CASE ACCEPTED()
    OF ?Fahrenheit
      Centigrade = (Fahrenheit - 32) / 1.8
      DISPLAY(?Centigrade)
    OF ?Another
      Fahrenheit = 0
      Centigrade = 0
      DISPLAY
      SELECT(?Fahrenheit)
    OF ?Exit
      BREAK
    END
  END
  CLOSE(Window)
  RETURN
```

Porównaj: MAP, MODULE, PROCEDURE, Deklaracje danych i przydział pamięci

MEMBER (identyfikuje plik źródłowy przynależnego modułu)

```
        MEMBER( [ program ] )
        [MAP
           prototypes
        END ]
[label] local data
        procedures
```

MEMBER Pierwsza instrukcja przynależnego modułu kodu źródłowego nie będącego plikiem źródłowym programu PROGRAM. Wymagane.

program Stała łańcuchowa zawierająca nazwę pliku (bez rozszerzenia) źródłowego programu PROGRAM. Jeśli pominiemy, dany moduł jest „uniwersalnym modułem przynależnym", który możemy wykorzystywać w dowolnym programie, po dołączeniu go do projektu.

MAP Deklaracje procedur lokalnych. Dowolna procedura tutaj zadeklarowana może być wywoływana przez inne procedury modułu MEMBER.

prototypes Deklaracje procedur.

local data Deklaruje lokalne dane statyczne, które są dostępne tylko dla procedur, których kod źródłowy znajduje się w module MEMBER.

procedures Kod źródłowy procedur modułu MEMBER.

MEMBER jest pierwszą instrukcją w module źródłowym nie będącym plikiem źródłowym programu PROGRAM. Może być ona poprzedzona jedynie komentarzem. Instrukcja ta jest wymagana na początku dowolnego pliku źródłowego zawierającego procedury PROCEDURE wykorzystywane przez program PROGRAM. Instrukcja MEMBER identyfikuje *program*, do którego moduł źródłowy MODULE przynależy.

Moduł MEMBER może posiadać lokalną strukturę MAP (może ona zawierać kolejne struktury MODULE). Procedury prototypowane w tej strukturze MAP są dostępne dla wszystkich pozostałych procedur modułu MEMBER. Kod źródłowy procedur zadeklarowanych w strukturze MAP modułu MEMBER może być umieszczony albo w pliku źródłowym MEMBER albo w innym pliku (jeśli został wskazany w strukturze MODULE umieszczonej wewnątrz MAP).

Jeśli w instrukcji MEMBER zostanie pominięty parametr *program*, musimy posiadać strukturę MAP prototypującą zawarte w module procedury. Może również zachodzić konieczność dołączenia plików definicji standardowych ekwiwalentów wykorzystywanych w kodzie źródłowym.

Jeśli kod źródłowy procedury PROCEDURE prototypowanej w sekcji MAP modułu MEMBER jest umieszczony w oddzielnym pliku, prototyp *prototype* musi się znajdować w strukturze MODULE umieszczonej wewnątrz struktury MAP. Kod źródłowy modułu MEMBER zawierający definicję procedury PROCEDURE musi również zawierać swoje własne struktury MAP, w których są zadeklarowane te same prototypy *prototype* (jest tak, ponieważ *prototype* musi się pojawić w przynajmniej dwóch strukturach MAP – kodu źródłowego modułu zawierającego procedurę i kodu źródłowego modułu wykorzystującego procedurę).

Dowolna procedura PROCEDURE nie zadeklarowana w globalnej sekcji MAP (należącej do programu PROGRAM), musi być zadeklarowana w lokalnej strukturze MAP modułu MEMBER zawierającego jej kod źródłowy.

Dane zadeklarowane w module MEMBER, po słowie kluczowym MEMBER i przed pierwszą instrukcją PROCEDURE, są lokalnymi danymi modułu. Dostęp do nich uzyskują procedury modułu (jeśli nie są przekazywane w postaci parametru). Pamięć dla tych danych jest przydzielana statycznie.

Przykład:

```
    ! Moduł Source1 zawiera:
    MEMBER('OrderSys')                  ! Moduł należący do programu OrderSys
    MAP                                 ! Deklaracje lokalnych procedur
Func1 PROCEDURE(STRING),STRING          ! Func1 jest widoczna tylko w obu modułach
      MODULE('Source2.clw')
HistOrd2  PROCEDURE                     ! HistOrd2 jest widoczna tylko w obu modułach
      END
    END

LocalData   STRING(10)                  ! Deklaracja danej lokalnej dla modułu MEMBER
HistOrd     PROCEDURE                   ! Deklaracja procedury
HistData    STRING(10)                  ! Deklaracja danej lokalnej dla procedury
  CODE
  LocalData = Func1(HistData)

Func1 PROCEDURE(RecField)               ! Deklaracja lokalnej procedury
  CODE
  ! Instrukcje kodu wykonywalnego

! Moduł Source2 zawiera:
    MEMBER('OrderSys')                  ! Moduł należy do programu OrderSys
    MAP                                 !  Deklaracje procedur lokalnych
HistOrd2  PROCEDURE                     ! HistOrd2 jest widoczna tylko w obu modułach
      MODULE('Source1.clw')
Func1   PROCEDURE(STRING),STRING        ! Func1 jest widoczna tylko w obu modułach
      END
    END

LocalData   STRING(10)                  ! Deklaracja danej lokalnej dla modułu MEMBER

HistOrd2  PROCEDURE                     ! Deklaracja procedury
  CODE
  LocalData = Func1(LocalData)
```

Porównaj: MAP, MODULE, PROCEDURE, CLASS, Deklaracje danych i przydział pamięci

MAP (deklaruje prototypy procedur)

```
MAP
   prototypes
   [MODULE( )
      prototypes
   END ]
END
```

MAP Zawiera prototypy *prototypes* deklarujące procedury i zewnętrzne moduły źródłowe wykorzystywane przez PROGRAM lub moduł przynależny MEMBER.

prototypes Deklaruje procedury.

MODULE Deklaruje moduł źródłowy zawierający definicje prototypów *prototypes* modułu MODULE..

Struktura **MAP** zawiera prototypy deklarujące procedury i zewnętrzne moduły kodu źródłowego wykorzystywane w programie PROGRAM lub module MEMBER, a które nie są elementami struktury CLASS.

Struktura MAP zadeklarowana w module źródłowym PROGRAM deklaruje prototypy *prototypes* procedur dostępnych dla całego programu. Struktura MAP w module MEMBER deklaruje prototypy *prototypes* procedur, które są dostępne w module MEMBER. Te same prototypy *prototypes* mogą zostać umieszczone w wielu modułach MEMBER i udostępnić tym samym procedury dla każdego z nich. Struktura MAP jest obowiązkowa w zasadzie dla każdego nietrywialnego programu napisanego w języku Clarion. Jest tak, ponieważ do struktury MAP programu PROGRAM jest automatycznie dołączany, przez kompilator, plik BUILTINS.CLW. plik ten zawiera prototypy większości procedur biblioteki wewnętrznej Clariona, dostępnych jako część języka Clarion. Ten plik jest wymagany również dlatego, że zawarte w nim prototypy nie zostały wbudowane w kompilator (dzięki temu jest bardziej efektywny). Ponieważ prototypy z pliku BUILTINS.CLW wykorzystują pewne stałe ekwiwalenty EQUATE zdefiniowane w pliku EQUATES.CLW, również ten plik jest automatycznie dołączany przez kompilator do programu.

Przykład:

```
        ! Pierwszy plik zawiera:
        PROGRAM                 ! Przykładowy program w sample.cla
        MAP                     ! Początek mapy deklaracji
LoadIt  PROCEDURE               ! Procedura LoadIt
        END                     ! Koniec mapy deklaracji

        ! Drugi plik zawiera:
        MEMBER('Sample')        ! Deklaracja modułu MEMBER
        MAP                     ! Początek lokalnej mapy deklaracji
ComputeIt  PROCEDURE            ! Procedura
        END                     ! Koniec mapy deklaracji
```

Porównaj: PROGRAM, MEMBER, MODULE, PROCEDURE, Prototypy procedur

MODULE (wskazuje plik źródłowy modułu MEMBER)

```
MODULE( sourcefile)
   prototype
END
```

MODULE Określa nazwę modułu MEMBER lub zewnętrznego pliku bibliotecznego.

sourcefile Stała łańcuchowa zawierająca nazwę pliku (bez rozszerzenia) z kodem źródłowym procedur w języku Clarion. Jeśli *sourcefile* jest zewnętrzną biblioteką, łańcuch ten może zawierać dowolny, unikalny, identyfikator.

prototype Prototyp procedury, której definicja znajduje się w pliku *sourcefile*.

Struktura **MODULE** wskazuje moduł MEMBER z kodem źródłowym w języku Clarion bądź plik biblioteki zewnętrznej. Zawiera ona prototypy *prototypes* procedur zawartych w pliku *sourcefile*. Struktura MODULE może być deklarowana tylko wewnątrz struktury MAP. Jej użycie jest prawidłowe dla każdej struktur MAP programu PROGRAM lub modułu MEMBER.

Przykład:

```
  !The “sample.clw” file contains:
  PROGRAM                 ! Przykładowy program w sample.clw
  MAP                     ! Początek mapy deklaracji
    MODULE(‘Loadit’)      ! Moduł kodu źródłowego loadit.clw
LoadIt PROCEDURE          ! Procedura LoadIt
    END                   ! Koniec modułu
    MODULE(‘Compute’)     ! Moduł kodu źródłowego compute.clw
ComputeIt  PROCEDURE      ! Procedura
    END                   ! Koniec modułu
  END                     ! Koniec mapy deklaracji

  !The “loadit.clw” file contains:
  MEMBER(‘Sample’)        ! Deklaracja modułu MEMBER
  MAP                     ! Początek lokalnej mapy deklaracji
    MODULE(‘Process’)     ! Moduł kodu źródłowego process.cla
ProcessIt  PROCEDURE      ! Procedura
    END                   ! Koniec modułu
  END                     ! Koniec mapy deklaracji
```

Porównaj: MEMBER, MAP, Prototypy procedur

PROCEDURE (definiuje procedurę)

```
label   PROCEDURE [ ( parameter list ) ]
        local data
        CODE
          statements
        [RETURN( [ value ] ) ]
```

PROCEDURE Rozpoczyna sekcję kodu źródłowego, która może być uruchamiana z poziomu programu PROGRAM.

label Nazwa procedury PROCEDURE. W przypadku metod klas nazwa ta może zawierać etykietę klasy poprzedzającą etykietę procedury.

parameter list Lista nazw parametrów przekazywanych do procedury oraz, opcjonalnie, ich typów, oddzielonych od siebie znakami przecinka. Nazwy te definiują lokalne referencje w procedurze przeznaczone dla przekazanych parametrów. W przypadku metod klas mogą one zawierać etykietę klasy (nazwaną SELF) jako ukryty pierwszy parametr i muszą zawsze zawierać zarówno nazwę parametru, jak i jego typ.

local data Deklaruje dane lokalne widzialne tylko dla procedury.

CODE Kończy deklarację danych i rozpoczyna sekcję kodu wykonywalnego procedury.

statements Instrukcje kodu wykonywalnego.

RETURN Przerywa wykonanie procedury. Zwraca sterowanie do miejsca, z którego procedura została wykonana. Zwraca wartość *value* do wyrażenia, w którym procedura została użyta (o ile oczywiście jej prototyp został określony tak, by można ją było wywoływać w roli funkcji).

value Stała numeryczna lub łańcuchowa bądź zmienna, w której umieszcza się rezultat zwracany przez funkcję.

PROCEDURE rozpoczyna sekcję kodu źródłowego, który może być wywoływany z poziomu programu PROGRAM. Jest on wywoływany poprzez nazwę *label* procedury wraz z ujętą w nawiasy okrągłe listą parametrów, o ile oczywiście one występują. Lista parametrów *parameter list* definiuje typ danych dla każdego z parametrów poprzedzający, opcjonalnie, etykietę parametru wykorzystywaną później w kodzie procedury. Parametry są oddzielane od siebie znakami przecinka. Typy danych dla każdego parametru (włączając w to nawiasy trójkątne dla parametrów, które można pomijać) są wymagane razem z etykietą parametru wtedy, gdy następuje przeciążenie procedury (ma to miejsce wtedy, gdy występuje wiele definicji procedury). Lista parametrów *parameter list* może być dokładnie taka sama, jak w prototypie procedury, o ile oczywiście prototyp zawiera etykiety parametrów.

Procedura może zawierać jeden lub więcej podprogramów ROUTINE w swoich instrukcjach *statements* kodu źródłowego. ROUTINE jest sekcją kodu wykonywalnego lokalną dla danej procedury. Wywołuje się ją za pomocą instrukcji DO.

Działanie procedury kończy się, a sterowanie jest przekazywane do miejsca, z którego została wywołana, w momencie, gdy zostanie napotkana instrukcja RETURN. Ukryta instrukcja RETURN występuje na końcu kodu wykonywalnego. Koniec kodu

wykonywalnego dla procedury jest zdefiniowany jako koniec pliku źródłowego, pierwsze wystąpienie ROUTINE bądź początek innej procedury PROCEDURE.

Instrukcja RETURN jest wymagana wtedy, gdy prototyp procedury wymusza zwracanie przez nią rezultatu *value*. Taka procedura może być wykorzystywana jako składnik wyrażeń, przekazywana jako parametr do innych procedur. Może ona być również wywoływana jak zwykła procedura. W tym przypadku, gdy w jej prototypie nie umieszczono atrybutu PROC, kompilator powinien wygenerować ostrzeżenie i zignorować ten błąd.

Dane zadeklarowane wewnątrz PROCEDURE, pomiędzy słowami kluczowymi PROCEDURE i CODE, są danymi lokalnymi procedury. Są one widzialne tylko dla danej procedury (za wyjątkiem sytuacji, gdy zostaną przekazane w postaci parametru do innej procedury). Dane te są alokowane w pamięci po rozpoczęciu działania procedury i zwalniane, gdy procedura zakończy działanie. Jeśli rozmiar danych jest mniejszy od rozmiaru stosu (domyślnie 5K), są one umieszczane na stosie, w przeciwnym wypadku są umieszczane na stercie (heap).

Procedura musi posiadać prototyp zadeklarowany w strukturze CLASS lub w strukturze MAP należącej do struktury PROGRAM lub MEMBER. Jeśli została zadeklarowana w strukturze MAP programu PROGRAM, jest ona dostępna dla wszystkich innych procedur programu. Jeśli została zadeklarowana w strukturze MAP modułu MEMBER, jest dostępna dla innych procedur danego modułu.

Przykład:

```
        PROGRAM
        MAP
OpenFile    PROCEDURE(FILE AnyFile)    ! Prototyp procedury z parametrami
ShoTime     PROCEDURE                  ! Prototyp procedury bez parametrów
DayString   PROCEDURE,STRING           ! Prototyp procedury z rezultatem
        END
TodayString   STRING(9)

  CODE
  TodayString = DayString()            ! Procedura wywołana z rezultatem
  OpenFile(FileOne)                    ! Wywołanie procedury do otwarcia pliku
  ShoTime                              ! Wywołanie procedury ShoTime
  !More executable statements

OpenFile  PROCEDURE(FILE AnyFile)      ! Otwarcie dowolnego pliku
  CODE                                 ! Początek sekckji kodu
    OPEN(AnyFile)                      ! Otwarcie pliku
    IF ERRORCODE() = 2                 ! Jeśli nie znaleziono pliku
      CREATE(AnyFile)                  ! tworzymy go
    END
    RETURN                             ! Powrót do miejsca wywołania

ShoTime  PROCEDURE
Time     LONG                          ! Zmienna lokalna
Window   WINDOW,CENTER
           STRING(@T3),USE(Time),AT(34,70)
           BUTTON('Exit'),AT(138,92),USE(?Exit)
         END
  CODE                                 ! Początek sekcji kodu wykonywalnego
  Time = CLOCK()                       ! Pobranie czasu systemowego
  OPEN(Window)
  ACCEPT
    CASE ACCEPTED()
    OF ?Exit
      BREAK
    END
  END
  RETURN                               ! Powrót do miejsca wywołania
```

```
DayString      PROCEDURE
ReturnString     STRING(9),AUTO          ! Niezainicjowana zmienna stosu lokalnego
  CODE                                   ! Początek sekcji kodu wykonywalnego
  EXECUTE (TODAY() % 7) + 1              ! Znajdź dzień tygodnia daty systemowej
    ReturnString = 'Sunday'
    ReturnString = 'Monday'
    ReturnString = 'Tuesday'
    ReturnString = 'Wednesday'
    ReturnString = 'Thursday'
    ReturnString = 'Friday'
    ReturnString = 'Saturday'
  END
  RETURN(ReturnString)                   ! Zwrot łańcucha
```

Porównaj: Prototypy procedur, Deklaracje danych i przydział pamięci, Przeciążanie procedur, CLASS, ROUTINE, MAP

CODE (rozpoczyna instrukcje kodu wykonywalnego)

CODE

Instrukcja **CODE** oddziela sekcję deklaracji danych od instrukcji kodu wykonywalnego wewnątrz struktur PROGRAM, PROCEDURE oraz ROUTINE. Pierwszą wykonywaną instrukcją programu, procedury lub podprogramu jest instrukcja występująca bezpośrednio po instrukcji CODE.

Przykład:

```
PROGRAM
  ! Deklaracje danych globalnych występują tutaj
CODE
  ! Instrukcje kodu występują tutaj
OrdList PROCEDURE                    ! Deklaracja procedury
  ! Deklaracje danych lokalnych występują tutaj
CODE                                 ! Początek sekcji kodu
  ! Instrukcje kodu występują tutaj
```

Porównaj: PROGRAM, PROCEDURE

DATA (rozpoczyna sekcję deklaracji danych podprogramu)

DATA

Instrukcja **DATA** rozpoczyna sekcję deklaracji danych lokalnych w podprogramie ROUTINE. Dowolne ROUTINE zawierające instrukcję DATA musi również zawierać instrukcję CODE, która kończy sekcję deklaracji danych lokalnych. Zmienne zadeklarowane w sekcji danych lokalnych ROUTINE nie mogą posiadać atrybutów STATIC lub THREAD.

Przykład:

```
SomeProc PROCEDURE
  CODE
    ! Instrukcje kodu
    DO Tally                         ! Wywyołanie podprogramu
    ! Dodatkowe instrukcje kodu
    Tally ROUTINE                    ! Początek podprogramu, koniec procedury
    DATA
CountVar BYTE                        ! Deklaracja danej lokalnej
    CODE
      CountVar += 1                  ! Licznik inkrementowany
      DO CountItAgain                ! Wywołanie innego podprogramu
      EXIT                           ! i wyjście z podprogramu
```

Porównaj: CODE, ROUTINE

ROUTINE (deklaruje lokalny podprogram)

```
label    ROUTINE
         [ DATA
         local data
         CODE ]
           statements
```

ROUTINE	Deklaruje początek lokalnego podprogramu
label	Etykieta podprogramu ROUTINE. Nie może się powtórzyć w żadnej z procedur.
DATA	Poprzedza instrukcje deklaracji danych.
local data	Deklaruje dane lokalne widzialne tylko dla podprogramu.
CODE	Poprzedza instrukcje kodu wykonywalnego.
statements	Instrukcje podprogramu.

ROUTINE deklaruje początek lokalnego podprogramu. Jest on lokalny dla procedury wewnątrz której został umieszczony i musi się znajdować na końcu sekcji CODE tej procedury. Wszystkie zmienne widzialne dla procedury są dostępne również dla jej podprogramu. Dotyczy to zmiennych lokalnych procedury, zmiennych lokalnych modułu oraz zmiennych globalnych.

ROUTINE może zawierać swoje własne zmienne lokalne. Zakres ich widzialności obejmuje tylko ROUTINE, w którym zostały zadeklarowane. Jeśli dołączamy deklaracje zmiennych lokalnych do ROUTINE, musimy je poprzedzić instrukcją DATA, za nimi zaś umieszczamy instrukcję CODE. Ponieważ podprogram posiada swój własny zakres nazw, etykiety tych zmiennych mogą być duplikatami nazw zmiennych zadeklarowanych w innych podprogramach, jak również w procedurze, do której należy podprogram.

Podprogram jest wywoływany za pomocą instrukcji DO, po której umieszczamy jego etykietę. Po wykonaniu podprogramu sterowanie jest zwracane do instrukcji występującej w kodzie bezpośrednio po instrukcji DO. Koniec podprogramu wytycza koniec modułu źródłowego, początek innego podprogramu lub początek procedury. Instrukcja EXIT również może zakończyć wykonanie podprogramu (w sposób podobny, jak instrukcja RETURN w odniesieniu do procedury).

Podprogramy posiadają kilka cech, które nie są na pierwszy rzut oka wcale oczywiste:

- Instrukcje DO i EXIT są bardzo efektywne.
- Dostęp do danych lokalnych na poziomie procedury jest mniej efektywny, niż dostęp na poziomie modułowym lub globalnym.
- Zmienne tymczasowe stosowane w podprogramach są mniej efektywne niż zmienne lokalne.
- Każda instrukcja RETURN wewnątrz podprogramu powoduje zajęcie dodatkowych 40 bajtów pamięci.

Przykład:

```
SomeProc PROCEDURE
  CODE
  ! Instrukcje kodu
  DO Tally                      ! Wywyołanie podprogramu
  ! Dodatkowe instrukcje kodu
Tally ROUTINE                   ! Początek podprogramu, koniec procedury
  DATA
CountVar BYTE                   ! Deklaracja danej lokalnej
  CODE
    CountVar += 1               ! Licznik inkrementowany
    DO CountItAgain             ! Wywołanie innego podprogramu
    EXIT                        ! I wyjście z podprogramu
```

Porównaj: PROCEDURE, EXIT, DO, DATA, CODE

END (wyznacza koniec struktury)

END

END kończy strukturę deklaracji danych lub złożoną instrukcję kodu. Jest funkcjonalnym równoważna ze znakiem kropki (.).

Przyjęto, że instrukcja END jest wyrównywana w kodzie do tej samej kolumny, w której rozpoczyna się struktura przez nią kończona; kod wewnątrz tej struktury jest przesuwany w prawo w celu ułatwienia jego analizy. END jest często stosowany do zakańczania wielowierszowych struktur, podczas gdy znak kropki używa się do zakańczania struktur jednowierszowych (w nich instrukcje rozgraniczamy znakiem średnika). Jeżeli wielowierszowe struktury złożone są zagnieżdżone, mogą zostać zakończone znakami kropki umieszczonymi w jednym wierszu.

Przykład:

```
Customer FILE,DRIVER('Clarion')      ! Deklaracja pliku
           RECORD                    ! Początek deklaracji pliku
Name         STRING(20)
Number       LONG
           END                       ! Koniec deklaracji rekordu
         END                         ! Koniec deklaracji pliku
Archive  FILE,DRIVER('Clarion')      ! Deklaracja pliku
           RECORD                    ! Początek deklaracji pliku
Name         STRING(20)
Number       LONG
           . .                       ! Koniec deklaracji rekordu i pliku
  CODE
    IF Number <> SavNumber           ! Początek struktury IF
      DO GetNumber
    END                              ! Koniec struktury IF
    IF SomeCondition THEN BREAK.     ! Zakończenie znakiem kropki
    CASE Action                      ! Początek struktury CASE
    OF 1
      DO AddRec
      IF Number <> SavNumber         ! Początek struktury IF
        DO SomeRoutine
      END                            ! Koniec struktury IF
    OF 2
      DO ChgRec
    OF 3
      DO DelRec
    END                              ! Koniec struktury CASE
```

Sekwencja wykonywania instrukcji

W sekcji CODE programu napisanego w Clarionie instrukcje są standardowo wykonywane wiersz po wierszu, w takiej kolejności, w jakiej występują w kodzie źródłowym. Do zmiany tej sekwencji służą instrukcje sterujące oraz wywołania procedur.

Wywołania procedur modyfikują sekwencję wykonywania instrukcji poprzez przekazanie sterowania do procedury. Po jej wykonaniu sterowanie wraca do instrukcji występującej w kodzie źródłowym za wywołaniem procedury.

Struktury sterujące IF, CASE, LOOP, ACCEPT oraz EXECUTE zmieniają sekwencję wykonywania instrukcji w zależności od wartości obliczonego wyrażenia. Struktura sterująca warunkowo wykonuje instrukcje w niej zawarte jeśli obliczone wyrażenie spełnia określony warunek. ACCEPT jest strukturą typu pętla, jednak nie wylicza żadnego wyrażenia warunkowego.

Skoki w sekwencji wykonywania instrukcji umożliwiają również takie instrukcje, jak GOTO, DO, CYCLE, BREAK, EXIT oraz RETURN. Wymienione instrukcje bezpośrednio i bezwarunkowo zmieniają normalną sekwencję wykonywania kodu.

Procedura START rozpoczyna działanie nowego wątku (thread), bezwarunkowo przenosząc sterowanie do tego wątku w momencie napotkania następującej po niej instrukcji ACCEPT. Jednakże użytkownik może uaktywnić kolejny wątek klikając myszką inne aktywne okno uruchomione w postaci wątku.

Przykład:

```
  PROGRAM
  MAP
ComputeTime  PROCEDURE(*GROUP)        ! Przekazanie parametru będącego grupą
MatchMaster  PROCEDURE                ! Brak przekazania parametru
  END
ParmGroup GROUP                       ! Deklaracja grupy
FieldOne     STRING(10)
FieldTwo     LONG
           END
  CODE                                ! Początek kodu wykonywalnego
    FieldTwo = CLOCK()                ! Wykonuje się jako pierwsze
    ComputeTime(ParmGroup)            ! Wykonuje się jako drugie, przekazuje sterowanie do procedury
    MatchMaster                       ! Wykonuje się, gdy zakończy się działanie procedury
```

Wywołania pocedur

Procname[(*parameters*)]
return = **funcname**[(*parameters*)]

procname Nazwa procedury (taka, jaką zadeklarowano w jej prototypie).

parameters Opcjonalna lista parametrów przekazywanych do procedury. W liście parametrów może się znaleźć etykieta jednej lub wielu zmiennych bądź wyrażeń. Parametry *parameters* są rozdzielane przez znaki przecinka i są deklarowane w prototypie.

return Etykieta zmiennej, w której jest zapisywana wartość zwracana przez procedurę.

funcname Nazwa procedury zwracającej wartość, tak, jak to zadeklarowano w prototypie.

Procedura jest wywoływana przez nazwę (włączając w to listę parametrów). Nazwę procedury umieszczamy tak, jak instrukcję w sekcji CODE programu lub procedury. Lista parametrów musi być zgodna z listą parametrów zadeklarowaną w prototypie procedury. Procedury nie mogą być wywoływane w wyrażeniach.

Procedura zwracająca rezultat (funkcja) jest wywoływana poprzez nazwę (włączając w to listę parametrów) jako element wyrażenia lub jako parametr przekazywany do innej procedury. Lista parametrów musi być zgodna z listą parametrów zadeklarowaną w prototypie procedury procedura zwracająca wartość również może być wywoływana przez nazwę umieszczoną jako instrukcja (włącznie z listą parametrów). Stosujemy to wówczas, gdy zwracana wartość nie jest nam do niczego potrzebna. Powoduje to wygenerowanie ostrzeżenia kompilatora, które możemy bez problemu zignorować (o ile w prototypie nie umieszczono atrybutu PROC).

Jeśli procedura jest metodą klasy, parametr *procname* musi zaczynać się etykietą egzemplarza obiektu klasy, po której następuje znak kropki i etykieta procedury (*nazwa_obiektu.nazwa_procedury*).

Przykład:

```
  PROGRAM
  MAP
ComputeTime  PROCEDURE(*GROUP)        ! Przekazanie procedury w postaci grupy
MatchMaster  PROCEDURE,BYTE,PROC      ! Procedura zwraca wartość i nie przekazuje parametrów
  END
ParmGroup GROUP                       ! Deklaracja grupy
FieldOne     STRING(10)
FieldTwo     LONG
           END
  CODE
    FieldTwo = CLOCK()                ! Wbudowana procedura wywołana jako wyrażenie
    ComputeTime(ParmGroup)            ! Wywołanie procedury
    MatchMaster()                     ! Wywołanie funkcji w postaci procedury
```

Porównaj: PROCEDURE

Prototypy procedur

Składnia prototypu

name **PROCEDURE** [(*parameter list*)] [, *return type*] [, *calling convention*] [,**RAW**] [,**NAME**( )] [,**TYPE**] [,**DLL**( )][,**PROC**] [,**PRIVATE**] [,**VIRTUAL**] [,**PROTECTED**] [,**REPLACE**]

name [(*parameter list*)] [, *return type*] [, *calling convention*] [,**RAW**] [,**NAME**( )] [,**TYPE**] [,**DLL**( )] [, **PROC**] [, **PRIVATE**]

name	Etykieta procedury.
PROCEDURE	Wymagane słowo kluczowe.
parameter list	Typy danych parametrów procedury. Każdy z typów może następować po etykiecie wykorzystywanej tutaj tylko i wyłącznie w celach dokumentacyjnych. Każdy parametr mający wartość numeryczną może mieć przypisaną wartość domyślną (stałą) przekazywaną do procedury w momencie, gdy zostanie pominięty w wywołaniu.
return type	Typ danych dla rezultatu zwracanego przez procedurę.
calling convention	Konwencja wywołania procedury dla parametrów przekazywanych przez stos w sposób charakterystyczny dla języka C lub PASCAL.
RAW	Powoduje, że parametry typu STRING i GROUP są przekazywane tylko przez adres w pamięci (bez określania długości przekazywanego łańcucha). Zmienia to również zachowanie parametrów ? i *?. Są one zachowane tylko w celu utrzymania zgodności z językami 3GL i nie są prawidłowe dla procedur pisanych w języku Clarion.
NAME	Określa alternatywną, „zewnętrzną” nazwę dla procedury.
TYPE	Powoduje, że prototyp staje się definicją typu dla procedur przekazywanych jako parametry.
DLL	Określa, że procedura jest zewnętrzną biblioteką .DLL.
PROC	Powoduje, że procedura z określonym typem rezultatu *return type* może być wywoływana jak zwykła procedura i nie spowoduje to błędu kompilatora.
PRIVATE	Powoduje, że procedura może być wywoływana tylko przez procedury tego samego modułu (często stosowane w przypadku metod klas).
VIRTUAL	Powoduje, że procedura staje się wirtualną metodą struktury CLASS.
PROTECTED	Powoduje, że procedura może być wywoływana tylko przez procedury tej samej klasy lub klas bezpośrednio dziedziczących.
REPLACE	Powoduje, że procedura „Construct” lub „Destruct” klasy dziedziczącej całkowicie zastępuje konstruktor i destruktor klasy nadrzędnej.

Wszystkie procedury programu PROGRAM muszą posiadać deklaracje prototypów w strukturze MAP lub CLASS. Prototyp stanowi informację dla kompilatora, jakiej formy ma się spodziewać po procedurze wywoływanej w kodzie wykonywalnym. Istnieją dwie prawidłowe formy deklaracji prototypów; zostały przedstawione w opisie składni na początku punktu. Pierwsza jest oparta na słowie kluczowym PROCEDURE i można ją stosować w każdym miejscu; jest przy tym formą preferowaną. Druga forma jest zachowana tylko i wyłącznie w celu zachowania zgodności z poprzednimi wersjami Clarion.

Prototyp zawiera:

- Nazwę *name* procedury.
- Słowo kluczowe PROCEDURE opcjonalne w strukturze MAP; wymagane w strukturze CLASS.
- Opcjonalną listę parametrów *parameter list* specyfikującą wszystkie parametry, które mogą być przekazywane do procedury.
- Typ danych rezultatu *return type* zwracanego przez procedurę; o ile prototyp definiuje procedurę zwracającą rezultat.
- Parametr określający konwencję wywołania (*calling convention*) stosowany przy linkowaniu obiektów wymagających przekazywania parametrów przez stos (mogą to być na przykład obiekty, które nie zostały skompilowane za pomocą kompilatora Clarion TopSpeed).
- Atrybuty RAW, NAME, TYPE, DLL, PROC, PRIVATE, VIRTUAL oraz PROTECTED; o ile są potrzebne.

Wartością parametru określającego konwencję wywołania (*calling convention*) bazującą na stosie może być C (parametry - od prawej do lewej) lub PASCAL (parametry - od lewej do prawej oraz kompatybilność z Windows w trybie 16-to i 32-bitowym). Dzięki temu zachowujemy kompatybilność z bibliotekami opracowanymi za pomocą innych narzędzi programistycznych. Jeśli nie określimy parametru *calling convention*, domyślnym jest wewnętrzny sposób przekazywania parametrów bazujący na rejestrach; jest on stosowany przez wszystkie kompilatory TopSpeed.

Atrybut RAW umożliwia przekazanie tylko adresu pamięci parametru *?, STRING lub GROUP (czy jest przekazywany przez wartość, czy przez referencję) do procedury lub funkcji nie napisanej w języku Clarion; standardowo, zarówno parametry STRING, jak i GROUP przekazują zarówno adres, jak i długość łańcucha. Zastosowanie atrybutu RAW ma na celu zapewnienie zgodności z funkcjami z bibliotek zewnętrznych oczekujących jedynie adresu łańcucha.

Atrybut NAME informuje linker o zewnętrznej nazwie procedury. Ma to na celu zapewnienie zgodności z bibliotekami napisanymi w innych językach programowania. Na przykład: niektóre kompilatory języka C, z konwencją wywołania C, dodają znak podkreślenia na początku nazwy funkcji. Atrybut NAME umożliwia linkerowi prawidłowe określenie nazwy funkcji.

Atrybut TYPE oznacza, że prototyp nie odnosi się do konkretnej procedury. Definiuje on nazwę prototypu, która może być używana w definiowaniu innych prototypów procedur, których parametrami są procedury.

Atrybut DLL określa, że prototyp dotyczy procedury znajdującej się w bibliotece .DLL. Atrybut DLL jest wymagany dla aplikacji 32-bitowych, ponieważ biblioteki DLL mogą być przesuwane w 32-bitowej płaskiej przestrzeni adresowej; wymaga to jednej dodatkowej operacji kompilatora w celu określenia adresu procedury.

Atrybut PRIVATE określa, że dana procedura może być wywoływana tylko przez procedury tego samego modułu. Jest on najczęściej stosowany w przypadku prototypów definiowanych w strukturze MAP modułu, może być jednak stosowany również w globalnej strukturze MAP.

W sytuacji, gdy nazwa *name* prototypu jest wykorzystywana w liście parametrów *parameter list* innego prototypu, oznacza to, że procedura, która jest prototypowana może rejestrować etykietę procedury odbierającej tą samą listę parametrów *parameter list* (i posiada ten sam typ rezultatu *return type*, o ile procedura ma zwracać rezultat). Prototyp z atrybutem TYPE nie może mieć równocześnie atrybutu NAME.

Przykład:

```
  MAP
    MODULE('Test')                                    ! 'test.clw' zawiera następujące procedury
MyProc1   PROCEDURE(LONG)                             ! Parametr typu LONG
MyProc2 PROCEDURE(<*LONG>)                            ! Pomijalny parametr typu LONG
MyProc3 PROCEDURE(LONG=23)                            ! Przekazuje 23, gdy pominie się parametr
    END
    MODULE('Party3.Obj')                              ! Biblioteka
Func46  PROCEDURE(*CSTRING),REAL,C,RAW                ! Przekazuje adres CSTRING do funkcji C
Func47  PROCEDURE(*CSTRING),*CSTRING,C,RAW            ! Zwraca wskaźnik na CSTRING
Func48  PROCEDURE(REAL),REAL,PASCAL                   ! Konwencja wywołania PASCAL
Func49  PROCEDURE(SREAL),REAL,C,NAME('_func49')       ! Konwencja C i nazwa funkcji zewnętrznej
    END
      MODULE('STDFuncs.DLL')                          ! Biblioteka DLL
Func50  PROCEDURE(SREAL),REAL,PASCAL,DLL
      END
  END
```

Porównaj: MAP, MEMBER, MODULE, NAME, PROCEDURE, RETURN, Listy parametrów prototypu, Przeciążanie procedur, CLASS

Lista parametrów prototypu

```
type [ label ]
< type [ label ] >
type [ label ] = default
```

type Typ danych parametru. Parametr może występować w postaci wartości, zmiennej, tablicy, nieokreślonego typu danych, procedury, grupy GROUP, kolejki QUEUE lub klasy CLASS.

label Opcjonalna etykieta parametru. Stosujemy je tylko i wyłącznie w celach lepszego dokumentowania prototypów.

< > Nawiasy trójkątne oznaczają, że dany parametr jest opcjonalny i można go pominąć w wywołaniu procedury. Ominięte parametry wykrywa specjalna procedura OMITTED. Wszystkie typy parametrów mogą być pomijane.

= *default* Domyślna wartość pomijalnego parametru. Jeśli zostanie on pominięty w wywołaniu, przyjmuje właśnie tą wartość. Możliwe do stosowania tylko dla prostych, numerycznych typów danych.

Lista parametrów *parameter list* w prototypie procedury jest listą typów danych *types*, oddzielonych od siebie przecinkami. Cała ta lista jest zamknięta w nawiasach okrągłych i umieszczona zaraz po słowie kluczowym PROCEDURE (lub po nazwie *name)*. Każdy typ danych parametru może występować w połączeniu z etykietą parametru (definiuje się ją po nazwie typu i oddziela od niej spacją). Etykieta parametru w prototypie jest ignorowana przez kompilator i służy wyłącznie lepszemu dokumentowaniu. Każdy parametr będący wartością numeryczną (przekazywany przez wartość) może mieć przypisaną wartość domyślną; wartość ta jest przypisywana do parametru, jeśli zostanie on pominięty w wywołaniu procedury. Każdy parametr, który może być pomijany w wywołaniu procedury musi się znaleźć w liście parametrów prototypu i zostać ujęty w trójkątne nawiasy (< >) o ile nie zdefiniowano dla niego wartości domyślnej *default*. Procedura OMITTED pozwala na sprawdzenie, w trakcie działania programu, które parametry zostały pominięte, za wyjątkiem tych, którym została przypisana wartość domyślna *default*.

Przykład:

```
    MAP
      MODULE('Test')
MyProc1  PROCEDURE(LONG)                          ! Parametr LONG
MyProc2  PROCEDURE(<LONG>)                        ! Pomijalny parametr LONG
MyProc3  PROCEDURE(LONG=23)                       ! Przekazuje 23, gdy parametr jest pominięty
MyProc4  PROCEDURE(LONG Count, REAL Sum)          ! LONG przekazuje Count,a REAL przekazuje Sum
MyProc5  PROCEDURE(LONG Count=1, REAL Sum=0)      ! Count domyślnie jest 1,a Sum - 0
      END
    END
```

Porównaj: MAP, MEMBER, MODULE, PROCEDURE, CLASS

Parametry w postaci wartości

Parametry w postaci wartości są przekazywane poprzez wartość. Kopia zmiennej przekazanej w liście parametrów do wywoływanej procedury jest wykorzystywana w tej ostatniej. Wywoływana procedura nie może zmienić wartości zmiennej przekazanej jej przez procedurę wywołującą. Obowiązują tutaj zasady konwersji danych przy prostym podstawianiu; parametry występujące w postaci wartości są konwertowane do typu wskazanego w prototypie procedury. Prawidłowymi typami parametrów wartościowych są:

```
BYTE SHORT USHORT LONG ULONG SREAL REAL DATE
TIME STRING
```

Przykład:

```
  MAP
    MODULE('Test')
MyProc1   PROCEDURE(LONG)                          ! Parametr LONG
MyProc2   PROCEDURE(<LONG>)                        ! Pomijalny parametr LONG
MyProc3   PROCEDURE(LONG=23)                       ! Przekazuje 23, gdy parametr jest pominięty
MyProc4   PROCEDURE(LONG Count, REAL Sum)          ! LONG przekazuje Count,a REAL przekazuje Sum
MyProc5   PROCEDURE(LONG Count=1, REAL Sum=0)      ! Count domyślnie jest 1,a Sum - 0
    END
    MODULE('Party3.Obj')
Func48  PROCEDURE(REAL),REAL,PASCAL                ! Konwencja wywołania PASCAL
Func49  PROCEDURE(SREAL),REAL,C,NAME('_func49')    ! Konwencja C i nazwa funkcji zewnętrznej
    END
  END
```

Parametry w postaci zmiennej

Parametry w postaci zmiennej są przekazywane poprzez adres. Zmienna przekazana przez adres posiada tylko adres w pamięci. Zmiana wartości zmiennej w wywołanej procedurze pociąga za sobą zmianę jej wartości również w procedurze wywołującej. Parametry w postaci zmiennej poprzedza się w prototypie procedury, umieszczonym w sekcji MAP, znakiem gwiazdki (*). Prawidłowymi typami dla parametrów w postaci zmiennej są:

```
*BYTE *SHORT *USHORT *LONG *ULONG *SREAL *REAL *BFLOAT4
*BFLOAT8 *DECIMAL *PDECIMAL *DATE *TIME *STRING *PSTRING *CSTRING
*GROUP
```

Przykład:

```
  MAP
    MODULE('Test')
MyProc2   PROCEDURE(<*LONG>)                  ! Pomijalny parametr LONG
MyFunc1   PROCEDURE(*SREAL),REAL,C            ! parametr SREAL, rezultat REAL, konwencja C
    END
    MODULE('Party3.Obj')
Func4    PROCEDURE(*CSTRING),REAL,C,RAW       ! Przekazuje adres CSTRING do funkcji C
Func47   PROCEDURE(*CSTRING),CSTRING,C,RAW    ! Zwraca wskaźnik na CSTRING
    END
  END
```

Przekazywanie tablic

Jeśli chcemy przekazać w postaci parametru zawartość tablicy, musimy zadeklarować w prototypie typ tablicowy jako parametr w postaci zmiennej (przekazywany przez adres) z pustą listą indeksową. Jeśli tablica posiada więcej wymiarów niż jeden, ich liczbę określamy wstawiając odpowiednią ilość znaków przecinka. Instrukcja wywołująca musi przekazać do procedury całą tablicę, nie tylko jeden jej element.

Przykład:

```
  MAP
MainProc  PROCEDURE
AddCount PROCEDURE(*LONG[,] Total,*LONG[,] Current)      ! Przekazanie tablicy dwuwymiarowej
  END
  CODE
    MainProc                                             ! Wywołanie pierwszej procedury
MainProc PROCEDURE
TotalCount  LONG,DIM(10,10)
CurrentCnt  LONG,DIM(10,10)
    CODE
      AddCount(TotalCount,CurrentCnt)                    ! Wywołanie procedury przekazującej tablice

AddCount PROCEDURE(*LONG[,] Total,*LONG[,] Current)      ! Procedura ovczekuje dwóch tablic
    CODE
      LOOP I# = 1 TO MAXIMUM(Total,1)                   ! Pętla dla pierwszego indeksu
        LOOP J# = 1 TO MAXIMUM(Total,2)                  ! Pętla dla drugiego indeksu
          Total[I#,J#] += Current[I#,J#]                 ! Zwiększenie TotalCount o CurrentCnt
    ..
    CLEAR(Current)                                       ! Wyczyszczenie tablicy CurrentCnt
```

Parametry o nieokreślonym typie danych

Możemy tworzyć procedury wykonujące określone operacje w zależności od typu danych przekazanych parametrów. By to uzyskać, musimy zastosować parametry w postaci wartości lub parametry w postaci zmiennej nieokreślonego typu danych. Są to parametry polimorficzne; mogą przyjąć dowolny prosty typ danych w zależności od tego, jaki został przekazany do procedury.

Parametry w postaci wartości nieokreślonego typu są reprezentowane w prototypie przez znak zapytania (?). Gdy procedura jest wykonywana, dla parametru jest dynamicznie określany typ i służy on jako obiekt danych o bazowym typie LONG, DECIMAL, STRING lub REAL przekazanej wartości lub typ bazowy ostatniego przypisania. Oznacza to, że przyjęty typ danych parametru może być zmieniany w procedurze i pozwala na traktowanie go jako dowolnego typu danych. Parametr w postaci wartości nieokreślonego typu jest przekazywany przez wartość; przyjęty dla niego typ danych podlega zasadom automatycznej konwersji danych Clariona. Następujące typy danych mogą być stosowane dla parametrów w postaci wartości o nieokreślonym typie danych:

BYTE SHORT USHORT LONG ULONG SREAL REAL BFLOAT4
BFLOAT8 DECIMAL PDECIMAL DATE TIME STRING PSTRING CSTRING
GROUP (traktowana jak STRING)
Untyped value-parameter (?) Untyped Variable-parameter (*?)

Atrybut RAW można stosować w połączeniu z parametrem w postaci wartości o nieokreślonym typie danych (?) jeśli ten ostatni ma być przekazywany do zewnętrznej biblioteki napisanej w innymi języku niż Clarion. Powoduje to konwersję danych na typ LONG, a następnie przekazanie parametru jako „void *” języka C/C++ (eliminuje to komunikaty o niezgodności typów).

Parametry w postaci zmiennej o nieokreślonym typie są reprezentowane w prototypie procedury przez znak gwiazdki i znak zapytania (*?). W procedurze parametry te służą jako obiekty danych o typie takim, jak typ przekazanej w czasie działania procedury zmiennej. Oznacza to, że typ danych parametru jest stały podczas wykonywania procedury.

Parametry w postaci zmiennej nieokreślonego typu danych są przekazywane przez adres. Jednakże wszystkie zmiany przekazanych parametrów dokonane w procedurze odnoszą się bezpośrednio do przekazanej zmiennej. Pozwala to na pisanie procedur polimorficznych. W procedurze, do której przekazano parametr w postaci zmiennej o nieokreślonym typie nie jest bezpiecznie przyjmować jakiekolwiek założenia co do otrzymanego typu danych. Może to doprowadzić do przekroczenia zakresów, które nie będą mogły być przetworzone w oparciu o aktualny typ danych zmiennej. Jeśli taka sytuacja będzie miała miejsce, rezultat może znacznie odbiegać od oczekiwanego.

Typy danych, które mogą być wykorzystywane dla parametrów w postaci zmiennej nieokreślonego typu:

```
BYTE SHORT USHORT LONG ULONG SREAL REAL BFLOAT4
BFLOAT8 DECIMAL PDECIMAL DATE TIME STRING PSTRING CSTRING
Untyped variable-parameter (*?)
```

W połączeniu z parametrem w postaci zmiennej nieokreślonego typu może być stosowany atrybut RAW. Ma to na celu uzyskanie efektu przekazania do zewnętrznej procedury nie napisanej w języku Clarion parametru typu „void *” języka C/C++.

Jako parametr o nieokreślonym typie danych nie może występować tablica.

Przykład:

```
    PROGRAM
    MAP
Proc1  PROCEDURE(?)              ! Parametr w postaci wartości o nieokreślonym typie
Proc2  PROCEDURE(*?)             ! Parametr w postaci zmiennej o nieokreślonym typie
Proc3  PROCEDURE(*?)             ! Parametr w postaci zmiennej o nieokreślonym typie (ma się „wysypać”)
Max    PROCEDURE(?,?),?          ! Procedura zwracająca parametr w postaci wartości o nieokreślonym
typie
    END
GlobalVar1  BYTE(3)              ! BYTE zainicjowany na 3
GlobalVar2  DECIMAL(8,2,3)
GlobalVar3  DECIMAL(8,1,3)
MaxInteger  LONG
MaxString   STRING(255)
MaxFloat    REAL

    CODE
      Proc1(GlobalVar1)          ! Przekazanie BYTE, wartość 3
      Proc2(GlobalVar2)          ! Przekazanie DECIMAL(8,2), wartość 3.00 – wyświetla 3.33
      Proc2(GlobalVar3)          ! Przekazanie DECIMAL(8,1), wartość 3.0 – wyświetla 3.3
      Proc3(GlobalVar1)          ! Przekazanie BYTE i obserwacja, jak się „sypie”
      MaxInteger = Max(1,5)      ! Funkcja Max zwraca 5
      MaxString = Max('Z','A')   ! Funkcja Max zwraca 'Z'
      MaxFloat = Max(1.3,1.25)   ! Funkcja Max zwraca 1.3

Proc1  PROCEDURE(? ValueParm)
    CODE                                ! ValueParm zaczyna się od 3 i jest LONG
      ValueParm = ValueParm & ValueParm ! teraz zawiera '33' i jest STRING
      ValueParm = ValueParm / 10        ! teraz zawiera 3.3 i jest REAL

Proc2  PROCEDURE(*? VariableParm)
    CODE
      VariableParm = 10 / 3             ! Przypisanie 3.33333333... do przekazywanej zmiennej
Proc3  PROCEDURE(*? VariableParm)
    CODE
```

```
    LOOP
      IF VariableParm > 250 THEN BREAK.          ! Jeśli przekazano BYTE, BREAK nigdy nie nastąpi
      VariableParm += 10
    END

Max  PROCEDURE(Val1,Val2)                        ! Znajdź większą z dwóch przekazancyh wartości
  CODE
    IF Val1 > Val2                               ! Porównaj pierwszą z drugą
      RETURN(Val1)                               ! zwróć pierwszą, jeśli jest większa
    ELSE !otherwise
      RETURN(Val2)                               ! zwróć drugą
    END
```

Porównaj: MAP, MEMBER, MODULE, PROCEDURE, CLASS

Parametry w postaci egzemplarza

Parametry w postaci egzemplarza przekazują nazwę struktury danych do wywoływanej procedury. Przekazanie egzemplarza pozwala wywołanej procedurze na stosowanie tych poleceń Clariona, których wymaga etykieta struktury występującej jako parametr. Parametry w postaci egzemplarza są deklarowane w prototypie procedury, w strukturze MAP, poprzez typ egzemplarza. Parametry w postaci egzemplarza są zawsze przekazywane przez adres. Właściwymi parametrami w postaci egzemplarza są:

FILE VIEW KEY INDEX QUEUE WINDOW REPORT BLOB

W postaci parametru procedury, dla którego określono w prototypie typ WINDOW może występować również raport REPORT, a to z tego względu, że wewnętrznie oba te typy korzystają z tej samej struktury przekazującej.

Przykład:

```
  MAP
    MODULE('Test')
MyFunc2  PROCEDURE(FILE),STRING          ! Parametr w postaci egzemplarza, rezultat STRING
ProcType PROCEDURE(FILE),TYPE            ! Definicja typu proceduralnego
MyFunc4  PROCEDURE(FILE),STRING,PROC     ! Może być wywołana jako procedura, bez ostrzeżeń
MyProc6  PROCEDURE(FILE),PRIVATE         ! Może być wywołana tylko przez inne procedury w TEST.CLW
    END
  END
```

Parametry w postaci procedury

Parametry w postaci procedury przekazują nazwę procedury wywoływanej procedurze. Parametry w postaci procedury są deklarowane w prototypie poprzez umieszczenie nazwy procedury również prototypowanej w sekcji MAP (może ona, choć nie musi, posiadać atrybut TYPE). Podczas wywołania procedury w kodzie wykonywalnym, należy przekazać jej jako parametr nazwę procedury, której prototyp jest dokładnie ten sam, co wskazany w prototypie wywoływanej procedury. Każdy parametr w prototypie może zawierać dodatkowo etykietę (oprócz typu danych). Etykieta ta jest ignorowana przez kompilator i służy wyłącznie do dokumentowania lub też do duplikowania instrukcji definiowania procedury. Każda definicja przekazywanego parametru może dodatkowo zawierać instrukcję przypisania stałej

wartości do typu danych (lub, jeśli występuje, etykiety). Przypisanie to definiuje domyślną wartość dla parametru, jeśli zostanie on pominięty w wywołaniu.

Przykład:

```
  MAP
    MODULE('Test')
ProcType PROCEDURE(FILE),TYPE              ! Definicja typu proceduralnego
MyFunc3  PROCEDURE(ProcType),STRING        ! parametr w postaci procedury ProcType, rezultat STRING,
    END                                    ! musi przekazywać procedurę, której parametrem jest FILE
  END
```

Przekazywanie nazwanych struktur GROUP, QUEUE, CLASS

Przekazanie struktury GROUP jako parametru w postaci wartości lub struktury QUEUE jako parametru w postaci egzemplarza, nie pozwala procedurze na odwoływanie się do ich pól składowych. By to uzyskać należy przekazywać nazwane struktury GROUP lub QUEUE. W ten sam sposób należy przekazywać struktury CLASS, jeśli chcemy uzyskać dostęp do ich pól publicznych i metod.

By móc odwoływać się do pól struktur, musimy umieścić etykietę struktury GROUP, QUEUE, czy CLASS w liście prototypów procedury jako typ danych parametru. Powoduje to przekazanie parametru poprzez adres i umożliwia procedurze odwoływanie się do pól struktury (i publicznych metod w przypadku klas).

Dane przekazywane w postaci parametru muszą zawsze posiadać podobną strukturę, zdefiniowaną w oparciu o ten sam typ danych, dla swoich pól składowych. Struktury GROUP i QUEUE przekazywane do procedury muszą mieć pierwsze pole identyczne, jak struktura GROUP lub QUEUE wskazana w prototypie. Struktury CLASS mogą być klasami dziedziczącymi klasy wskazanej w prototypie. Dodatkowe pola, w stosunku do struktury wskazanej w prototypie, są niedostępne dla procedury odbierającej parametr.

Struktury GROUP, QUEUE i CLASS wskazane w prototypie nie muszą posiadać atrybutu TYPE, nie muszą być także zadeklarowane przed prototypem procedury. Jest to jedyny przypadek w Clarionie, w którym kompilator dopuszcza stosowanie „odwołań w przód".

Przy tworzeniu odwołań do elementów przekazanych grup stosujemy składnię kwalifikacji pól (NazwaLokalna.NazwaElementu). Pola składowe przekazanych struktur korzystają z etykiet nadanych przez grupę wskazaną w prototypie jako typ danych, a nie z etykiet pól struktury przekazanej do procedury. Pozwala to procedurze na niezależność od faktycznie przekazanej struktury danych.

Przykład:

```
  PROGRAM
    MAP
MyProc    PROCEDURE
AddQue    PROCEDURE(PassGroup PassedGroup, NameQue PassedQue)
    END                                    ! AddQue otrzymuje GROUP zdefiniowaną jak PassGroup
and
                                           ! kolejka QUEUE zdefiniowana jak NameQue
PassGroup  GROUP,TYPE                      ! definicja typu – brak przydziału pamięci
F1           STRING(20)                    ! GROUP z dwoma polami STRING(20)
F2           STRING(20)
           END
NameGroup  GROUP                           ! grupa
First        STRING(20)                    ! pierwsze imię
Last         STRING(20)                    ! drugie imię
Company      STRING(30)                    ! Dodatkowe pole niedostępne do rejestrowania dla procedury
           END                             ! (AddQue) ponieważ PassGroup psiada tylko dwa pola
```

```
NameQue   QUEUE,TYPE                    ! kolejka, definicja typu -- brak przydziału pamięci
First       STRING(20)
Last        STRING(20)
          END
  CODE
    MyProc

MyProc  PROCEDURE
LocalQue  NameQue                       ! lokalna kolejka zadeklarowana tak samo, jak NameQue
  CODE
    NameGroup.First = 'Fred'
    NameGroup.Last = 'Flintstone'
    AddQue(NameGroup,LocalQue)          ! Przekazanie NameGroup i LocalQue do procedury AddQue
    NameGroup.First = 'Barney'
    NameGroup.Last = 'Rubble'
    AddQue(NameGroup,LocalQue)
    NameGroup.First = 'George'
    NameGroup.Last = 'O''Jungle'
    AddQue(NameGroup,LocalQue)
    LOOP X# = 1 TO RECORDS(LocalQue)    ! Zobacz, co teraz jest w LocalQue
      GET(LocalQue,X#)
      MESSAGE(CLIP(LocalQue.First) & ' ' & LocalQue.Last)
    END

AddQue   PROCEDURE(PassGroup PassedGroup, NameQue PassedQue)
  CODE
    PassedQue.First = PassedGroup.F1    ! Efektywnie: LocalQue.First = NameGroup.First
    PassedQue.Last = PassedGroup.F2     ! Efektywnie: LocalQue.Last = NameGroup.Last
    ADD(PassedQue)                      ! Dodanie elementu do PassedQue (LocalQue)
    ASSERT(NOT ERRORCODE())
```

Porównaj: MAP, MEMBER, MODULE, PROCEDURE, CLASS

Typy rezultatów procedury

Procedura PROCEDURE, w której prototypie określono, że ma dawać rezultat, musi zwracać wartość. Typ danych rezultatu wymienia się w prototypie po liście parametrów; oddziela się go od nawiasu zamykającego listę parametrów znakiem przecinka.

Typy zwracanych wartości:

BYTE SHORT USHORT LONG ULONG SREAL REAL DATE
TIME STRING Untyped value-parameter (?)

Rezultat w postaci wartości o nieokreślonym typie danych (?) oznacza, że typ rezultatu zwracanego przez procedurę nie jest znany. Funkcjonowanie takiego rezultatu jest takie samo, jak w przypadku parametrów w postaci wartości o nieokreślonym typie danych. Gdy rezultat zostaje otrzymany z procedury, są w odniesieniu do niego stosowane standardowe zasady konwersji danych Clariona, niezależnie od tego, jaki jest jego typ danych.

Typy zwracanych zmiennych:

CSTRING *STRING *BYTE *SHORT *USHORT *LONG
*ULONG *SREAL *REAL *DATE *TIME
Untyped variable-parameter (*?)

Rezultaty w postaci zmiennej są obsługiwane tylko dla prototypów procedur zewnętrznych (napisanych w innych językach), które dają w rezultacie adres danych; nie są one prawidłowe dla procedur napisanych w Clarionie. Funkcje zwracające wskaźnik (adres danych) powinny być prototypowane ze znakiem gwiazdki umieszczanym przed nazwą typu rezultatu (za wyjątkiem CSTRING). Kompilator automatycznie generuje obsługę zwracanego wskaźnika na czas działania programu. Funkcje prototypowane w ten sposób zachowują się jak zmienne zdefiniowane w programie – gdy są wykorzystywane w kodzie programu, dane na które wskazują są automatycznie wykorzystywane. Dane te mogą być przypisywane innym zmiennym, przekazywane w postaci parametrów do procedur; funkcja ADDRESS pozwala na określenie adresu tych danych. Typ CSTRING stanowi wyjątek z tego względu, że nie posiada stałej długości, toteż dowolna funkcja języka C zwracająca wskaźnik na daną typu CSTRING może być, od strony C, prototypowana jako „char *”; kompilator kopiuje dane na stos. Jednakże, podobnie jak w przypadku innych rezultatów będących zmiennymi wskaźnikowymi, gdy funkcja zostaje użyta w kodzie języka Clarion, dane, na które wskazuje zwrócony wskaźnik są automatycznie używane.

Jako przykład przyjmijmy, że funkcja XYZ() daje w rezultacie wskaźnik na CSTRING, CStringVar jest zmienną typu CSTRING, a LongVar jest zmienną typu LONG. Instrukcja prostego przypisania CStringVar = XYZ(), umieszcza daną wskazywaną przez zwrócony przez XYZ() wskaźnik w zmiennej CStringVar. Przypisanie LongVar = ADDRESS(XYZ()) umieszcza adres pamięci tej danej w zmiennej LongVar.

Rezultat o nieokreślonym typie (*?) oznacza, że typ danych zwracanych przez PROCEDURE jest nieznany. Znaczenie jest takie samo, jak w przypadku parametrów procedury o nieokreślonym typie.

Typy zwracanych odwołań:

*FILE *KEY *WINDOW *VIEW

Named CLASS (**ClassName*)
Named QUEUE (**QueueName*)

Procedura może dawać w rezultacie referencje, które mogą zostać przypisane do zmiennych referencyjnych lub używane w liście parametrów. Procedura dająca w rezultacie *WINDOW może także zwracać etykietę aplikacji APPLICATION lub raportu REPORT. Prawidłowym rezultatem może być wówczas także wartość nieokreślona NULL.

Przykład:

```
  MAP
    MODULE('Party3.Obj')                              ! Biblioteka pomocnicza
Func46  PROCEDURE(*CSTRING),REAL,C,RAW      ! Przekazuje adres CSTRING do funkcji C, rezultat REAL
Func47  PROCEDURE(*CSTRING),CSTRING,C,RAW ! Zwraca wskaźnik na CSTRING
Func48  PROCEDURE(REAL),REAL,PASCAL           ! Konwencja wywołania PASCAL, zwraca REAL
Func49  PROCEDURE(SREAL),REAL,C,NAME('_func49')
                                                  ! Konwencja C i nazwa funkcji zewnętrznej, zwraca REAL
    END
  END
```

Porównaj: MAP, MEMBER, MODULE, NAME, PROCEDURE, RETURN, Zmienne referencyjne

Atrybuty prototypów

C, PASCAL (konwencje przekazywania parametrów)

C
PASCAL

Atrybuty **C** lub **PASCAL** zastosowane w prototypie procedury powodują, że jej parametry będą zawsze przekazywane poprzez stos. Konwencja C powoduje przekazywanie parametrów z listy od prawej do lewej jej strony, konwencja PASCAL – od lewej do prawej. Dodatkowo, konwencja PASCAL jest całkowicie zgodna ze standardową konwencją wywołań funkcji Windows API zarówno w trybie 16-to, jak i 32-bitowym. Wymienione konwencje wywołań zapewniają zgodność z bibliotekami napisanymi w innych językach programowania (takimi, które nie zostały skompilowane za pomocą kompilatora TopSpeed).

Jeśli w prototypie nie określimy konwencji wywołania, domyślnie jest przyjmowana wewnętrzna konwencja, która opiera się na przekazaniu parametrów za pomocą rejestrów; jest ona stosowana we wszystkich kompilatorach TopSpeed.

Przykład:

```
  MAP
    MODULE('Party3.Obj') !A third-party library
Func46  PROCEDURE(*CSTRING,*REAL),REAL,C,RAW
        ! Przekazuje REAL, następnie CSTRING, tylko adres
Func49  PROCEDURE(*CSTRING,*REAL),REAL,PASCAL,RAW
        ! Przekazuje CSTRING, następnie REAL, tylkko adres
  ..
```

Porównaj: Prototypy procedur, Listy prototypów parametrów

DLL (procedura zdefiniowana w zewnętrznej bibliotece .DLL)

DLL([*flag*])

DLL Deklaruje procedurę zdefiniowaną zewnętrznie w bibliotece .DLL.

flag Stała numeryczna, ekwiwalent lub definicja projektu określająca, czy dany atrybut jest aktywny, czy nie. Jeśli *flag* jest równe zero atrybut nie jest aktywny. Jeśli wartość *flag* jest różna od zera, atrybut jest aktywny. Czasami wartość flag jest nieokreślona, wtedy atrybut jest aktywny.

Atrybut **DLL** oznacza, że procedura przy której prototypie został umieszczony, jest zaimplementowana w bibliotece zewnętrznej DLL. Atrybut DLL jest wymagany dla aplikacji 32-bitowych, ponieważ biblioteki DLL mogą być przesuwane w 32-bitowej płaskiej przestrzeni adresowej; wymaga to jednej dodatkowej operacji kompilatora w celu określenia adresu procedury.

Przykład:

```
  MAP
    MODULE('STDFuncs.DLL')                          ! Biblioteka standardowych funkcji
Func50  PROCEDURE(SREAL),REAL,PASCAL,DLL(dll_mode)
    END
  END
```

Porównaj: EXTERNAL

NAME (określenie zewnętrznej nazwy dla prototypu)

NAME(*constant*)

NAME Określa zewnętrzną nazwę dla linkera.

constant Stała łańcuchowa zawierająca przypisywaną nazwę wewnętrzną; jest ona zależna od wielkości liter.

Atrybut **NAME** określa zewnętrzną nazwę przeznaczoną dla linkera. Atrybut NAME może być umieszczany w prototypie procedury. Parametr *constant* dostarcza nazwę zewnętrzną wykorzystywaną przez linker do zidentyfikowania procedury lub funkcji znajdującej się w bibliotece zewnętrznej.

Przykład:

```
  PROGRAM
  MAP
    MODULE('External.Obj')
AddCount PROCEDURE(LONG),LONG,C,NAME('_AddCount')    ! funkcja C  nazwana '_AddCount'
  . .
```

Porównaj: Prototypy procedur

PRIVATE (ustawienie procedury jako prywatnej dla klasy lub modułu)

PRIVATE

Atrybut **PRIVATE** powoduje, że procedura, w której prototypie został użyty, może być wywoływana tylko z innej procedury tego samego modułu. Atrybut PRIVATE jest zazwyczaj stosowany w prototypach metod struktur CLASS, co oznacza, że metoda taka może być wywoływana tylko przez inną metodę danej klasy. Metody PRIVATE nie są dziedziczone przez klasy potomne, chociaż mogą one być metodami wirtualnymi (VIRTUAL) jeśli dziedziczące klasy znajdują się w tym samym module.

Przykład:

```
    MAP
      MODULE('STDFuncs.DLL')                    ! Biblioteka standardowych funkcji
Func49  PROCEDURE(SREAL),REAL,PASCAL,PROC
Proc50  PROCEDURE(SREAL),PRIVATE                ! Możliwość wywoływania tylko z Func49
      END
    END
OneClass CLASS,MODULE('OneClass.CLW'),TYPE
BaseProc PROCEDURE(REAL Parm)                   ! metoda publiczna
Proc PROCEDURE(REAL Parm),PRIVATE               ! deklaracja metody prywatnej
    END
TwoClass OneClass                               ! egzemplarz OneClass
    CODE
      TwoClass.BaseProc(1)                      ! prawidłowe wywołanie BaseProc
      TwoClass.Proc(2)                          ! nielegalne wywołanie Proc
      !In OneClass.CLW:
    MEMBER()
OneClass.BaseProc PROCEDURE(REAL Parm)
    CODE
      SELF.Proc(Parm)                           ! prawidłowe wywołanie Proc
OneClass.Proc PROCEDURE(REAL Parm)
    CODE
      RETURN(Parm)
```

Porównaj: CLASS

PROC (wywołanie funkcji jako procedury bez ostrzeżeń o błędzie)

PROC

Atrybut **PROC** może być umieszczany w prototypach procedur posiadających rezultat. Umożliwia on wywoływanie funkcji tak, jak procedur, bez przypisywania rezultatu ich działania do zmiennych. Atrybut PROC wyłącza wówczas ostrzeżenia kompilatora.

Przykład:

```
    MAP
      MODULE('STDFuncs.DLL')
Func50  PROCEDURE(SREAL),REAL,PASCAL,PROC
      END
    END
```

Porównaj: PROCEDURE

PROTECTED (procedura prywatna dla klasy i klas dziedziczących)

PROTECTED

Atrybut **PROTECTED** określa, że procedura, w której prototypie został użyty jest widoczna tylko dla procedur zadeklarowanych w tej samej strukturze CLASS (czyli dla innych metod danej klasy) oraz dla metod wszystkich klas dziedziczących. Zapewnia to enkapsulację procedury, zapobiegając możliwości jej wywołania z poziomu kodu zewnętrznego w stosunku do danej klasy bądź klas dziedziczących.

Przykład:

```
OneClass CLASS,MODULE('OneClass.CLW'),TYPE
BaseProc   PROCEDURE(REAL Parm)                          ! metoda publiczna
Proc       PROCEDURE(REAL Parm),PROTECTED                ! deklaracja metody chronionej
           END
TwoClass OneClass                                        ! egzemplarz OneClass
ThreeClass CLASS(OneClass),MODULE('ThreeClass.CLW')      ! dziedziczone z OneClass
ThreeProc    PROCEDURE(REAL Parm)                        ! deklaracja metody publicznej
             END
  CODE
    TwoClass.BaseProc(1)                                 ! prawidłowe wywołanie BaseProc
    TwoClass.Proc(2)                                     ! nielegalne wywołanie Proc
    !In OneClass.CLW:
  MEMBER()
OneClass.BaseProc PROCEDURE(REAL Parm)
  CODE
    SELF.Proc(Parm)                                      ! prawidłowe wywołanie Proc
OneClass.Proc PROCEDURE(REAL Parm)
  CODE
    RETURN(Parm)
    !In ThreeClass.CLW:
  MEMBER()
ThreeClass.NewProc  PROCEDURE(REAL Parm)
  CODE
    SELF.Proc(Parm)                                      ! prawidłowe wywołanie Proc
```

Porównaj: CLASS

RAW (przekazanie samego adresu)

RAW

Atrybut **RAW** zastosowany w prototypie procedury oznacza, że parametry typu STRING lub GROUP przekazują jedynie adres pamięci. Umożliwia to po prostu przekazywanie adresu pamięci dla parametrów *?, STRING lub GROUP, niezależnie od tego, czy są przekazywane jako wartość, czy jako referencja, do procedur zapisanych w innych językach programowania. Standardowo, parametry STRING lub GROUP przekazują adres i długość łańcucha. Atrybut RAW eliminuje tę część, która wskazuje długość łańcucha. W przypadku prototypu z parametrem ?, jest on pobierany jako LONG, a przekazywany jako "void *" co eliminuje ostrzeżenia generowane przez linker. Ta cecha jest dostarczana w celu zachowania kompatybilności z funkcjami pochodzącymi z bibliotek zewnętrznych.

Przykład:

```
MAP
  MODULE('Party3.Obj')                          ! biblioteka pomocnicza
Func46  PROCEDURE(*CSTRING),REAL,C,RAW    ! Przekazuje adres CSTRING do funkcji C
  ..
```

Porównaj: Prototypy procedur, listy prototypów parametrów

REPLACE (ustawia zastąpienie konstruktora lub destruktora)

REPLACE

Atrybut **REPLACE** określa, że procedura, w prototypie której został zastosowany, całkowicie zastępuje konstruktor lub destruktor klasy nadrzędnej. Stosowanie atrybutu REPLACE jest właściwe tylko dla procedur o etykietach "Construct" lub "Destruct" zadeklarowanych w klasach dziedziczących od klas, w których zadeklarowano odpowiednie procedury "Construct" lub "Destruct". Jeśli etykietą procedury jest "Construct" stanowi ona metodę będącą konstruktorem – automatycznie wywoływaną gdy obiekt jest powoływany – pojawia się „w zasięgu" kodu lub jest tworzony za pomocą instrukcji NEW. Jeżeli etykietą procedury jest "Destruct", stanowi ona metodę będącą destruktorem – wywoływaną automatycznie, gdy obiekt jest niszczony – znika „z zasięgu" kodu lub jest usuwany za pomocą instrukcji DISPOSE.

Przykład:

```
   PROGRAM
SomeQueue  QUEUE,TYPE
F1           STRING(10)
           END
OneClass   CLASS,MODULE('OneClass.CLW'),TYPE
ObjectQueue  &SomeQueue                          ! deklaracja referencji do kolejki
Construct    PROCEDURE                           ! deklaracja konstruktora
Destruct     PROCEDURE                           ! deklaracja destruktora
         END
TwoClass CLASS(OneClass),MODULE('TwoClass.CLW'),TYPE
Construct  PROCEDURE,REPLACE                     ! deklaracja zastępnika konstruktora
         END
MyClass  OneClass                                ! egzemplarz OneClass
YourClass  &TwoClass                             ! referencja do TwoClass
  CODE                                           ! obiekt MyClass pojawia się w zasięgu widzialności,
                                                 ! automatyczne wywołanie OneClass.Construct
  YourClass &= NEW(TwoClass)                     ! obiekt YourClass pojawia się w zasięgu widzialności,
                                                 ! automatyczne wywołanie TwoClass.Construct
  DISPOSE(YourClass)                             ! obiekt YourClass pojawia się w zasięgu widzialności,
                                                 ! automatyczne wywołanie OneClass.Destruct
  RETURN                                         ! obiekt MyClass pojawia się w zasięgu widzialności,
                                                 ! automatyczne wywołanie OneClass.Destruct
  !OneClass.CLW zawiera:
OneClass.Construct  PROCEDURE
  CODE
    SELF.ObjectQueue = NEW(SomeQueue)            ! tworzy kolejkę obiektów
OneClass.Destruct   PROCEDURE
  CODE
    FREE(SELF.ObjectQueue)                       ! zwalnia elementy kolejki
    DISPOSE(SELF.ObjectQueue)                    ! usuwa kolejkę
    !TwoClass.CLW zawiera:
TwoClass.Construct  PROCEDURE
  CODE
    SELF.ObjectQueue = NEW(SomeQueue)            ! tworzy kokejkę obiektów
    SELF.ObjectQueue.F1 = 'First Entry'
    ADD(SELF.ObjectQueue)
```

Porównaj: NEW, DISPOSE, CLASS

TYPE (określa typ proceduralny)

TYPE

Atrybut **TYPE** zastosowany w prototypie procedury oznacza, że nie odnosi się on do konkretnej procedury. Zamiast tego definiuje typ proceduralny umożliwiający przekazywanie procedury, w postaci parametru, do innej procedury.

Gdy parametr *name* takiego prototypu jest wykorzystany w liście parametrów innego prototypu procedury, ta ostatnia otrzymuje etykietę procedury, która musi posiadać taką sama listę parametrów i dawca rezultat takiego samego typu, jak ta, dla której użyto TYPE.

Przykład:

```
    MAP
ProcType  PROCEDURE(FILE),TYPE              ! definicja typu proceduralnego
MyFunc3   PROCEDURE(ProcType),STRING        ! Parametr w postaci procedury ProcType, rezulat STRING,
  END                                       ! musi przekazywać etykietę procedury, której wymaganym
                                            ! parametrem jest FILE
```

Porównaj: Prototypy procedur, Listy parametrów prototypów

VIRTUAL (metoda wirtualna)

VIRTUAL

Atrybut **VIRTUAL** określa, że dana procedura jest metoda wirtualną. Umożliwia to metodom klasy nadrzędnej dostęp do metod klas dziedziczących. Atrybut VIRTUAL musi być umieszczony zarówno w prototypie metody klasy nadrzędnej, jak I w prototypie metody klasy dziedziczącej.

Przykład:

```
OneClass CLASS                                  ! klasa bazowa
BaseProc   PROCEDURE(REAL Parm)                 ! metoda nie-wirtualna
Proc       PROCEDURE(REAL Parm),VIRTUAL         ! deklaracja metody wirtualnej
           END
TwoClass CLASS(OneClass)                        ! klasa dziedzicząca od OneClass
Proc       PROCEDURE(REAL Parm),VIRTUAL         ! deklaracja metody wirtualnej
           END
ClassThree OneClass                             ! kolejny egzemplarz obiektu OneClass
ClassFour  TwoClass                             ! kolejny egzemplarz obiektu TwoClass
  CODE
    OneClass.BaseProc(1)                        ! BaseProc wywołuje OneClass.Proc
    TwoClass.BaseProc(2)                        ! BaseProc wywołuje TwoClass.Proc
    ClassThree.BaseProc(3)                      ! BaseProc wywołuje OneClass.Proc
    ClassFour.BaseProc(4)                       ! BaseProc wywołuje TwoClass.Proc
OneClass.BaseProc  PROCEDURE(REAL Parm)
  CODE
    SELF.Proc(Parm)                             ! wywołanie metody wirtualnej, oprócz OneClass.Proc
    ! TwoClass.Proc, w zależności od tego, który egzemplarz jest wykonywany
```

Porównaj: CLASS

Przeciążenie procedur

Przeciążenie procedur (procedure overloading) oznacza możliwość stosowania jednej nazwy przez liczne definicje procedur. Jest to jedna z form polimorfizmu. Każda z procedur współdzielących tę samą nazwę musi używać różnej listy parametrów. Kompilator decyduje, której z nich użyć właśnie na podstawie listy parametrów umieszczonych w wywołaniu.

Cel, do którego dążymy, to możliwość posiadania wielu procedur o takiej samej nazwie, ale o różnych prototypach i działających podobnie, lecz oczywiście każda nieco inaczej, w zależności od typu danych przez nie przetwarzanych.

Z punktu widzenia efektywności, przeciążanie procedur jest znacznie bardziej efektywne, niż kodowanie jednej procedury z wieloma pomijalnymi parametrami i wykonywaniem kodu w zależności od zakresu przekazanych parametrów.

Język Clarion umożliwia oczywiście stosowanie procedur polimorficznych poprzez zastosowanie parametrów ? oraz *?, jednakże przeciążanie procedur rozszerza tą funkcjonalność dzięki dopuszczeniu stosowania w roli parametrów egzemplarzy obiektów, czy struktur złożonych.

Jednym z przykładów przeciążenia procedur może być instrukcja OPEN inicjująca egzemplarz do wykorzystania w programie. W zależności od typu egzemplarza przekazanego w postaci parametru (plik FILE, okno WINDOW, czy widok VIEW) wykonuje ona różne działania.

Zasady przeciążania procedur

Język Clarion posiada wbudowane konwencje konwersji danych, które mogą przeciążać rozwiązania skomplikowane dla kompilatora. Jednakże występują reguły kierujące sposobem rozwiązywania przez kompilator przeciążeń funkcjonalnych. Są one uwzględniane w następującym porządku:

1. Parametry będące egzemplarzami są analizowane jako FILE, KEY, WINDOW lub QUEUE. Jeśli prototyp może być określony na podstawie jednego z nich, kompilator go akceptuje (większość wbudowanych procedur Clariona mieści się w tej kategorii). Należy zauważyć, że KEY i VIEW dziedziczą bezpośrednio z klasy FILE, tak jak APPLICATION i REPORT dziedziczą bezpośrednio z klasy WINDOW.
2. Wszystkie parametry typu “nazwana grupa” muszą być zgodne ze strukturą grupy. Parametry proceduralne są dopasowywana na podstawie struktury. Klasy muszą być zgodne na poziomie nazwy, niekoniecznie struktury.
3. Liczba niepomijalnych parametrów i ich położenie musi być zgodna z prototypem. Jest to trzecie kryterium, tak więc kompilator może zazwyczaj określić, który prototyp użytkownik zamierzał zastosować i wygenerować właściwy komunikat błędu.
4. Jeśli nie występują pasujące prototypy, brane jest pod uwagę dziedziczenie. W tym punkcie KEY może zostać potraktowany jako pasujący do FILE a grupa może zostać dopasowana do klasy, z której dziedziczy. Jeśli pierwszy poziom dziedziczenia nie przyniesie efektu, kompilator kontynuuje operację dopasowywania poruszając się w kierunku korzenia klasy. Wszystkie kolejki QUEUE mogą być wtedy określone jako zgodne z QUEUE bądź GROUP etc.

5. Parametry w postaci zmiennej (o nieokreślonej nazwie) muszą dokładnie pasować do aktualnego typu danych. Typ *GROUP pasuje do typu *STRING. Dowolny parametr w postaci zmiennej pasuje do *?.
6. Parametry w postaci wartości muszą być określonego typu.

Przykład:

```
  MAP
Func   PROCEDURE(WINDOW)        ! 1
Func   PROCEDURE(FILE)          ! 2
Func   PROCEDURE(KEY)           ! 3
Func   PROCEDURE(FILE,KEY)      ! 4
Func   PROCEDURE(G1)            ! 5
Func   PROCEDURE(G0)            ! 6
Func   PROCEDURE(KEY,G0)        ! 7
Func   PROCEDURE(FILE,G1)       ! 8
Func   PROCEDURE(SHORT = 10)    ! 9
Func   PROCEDURE(LONG)          ! 10
Func   PROCEDURE()              ! nielegalne, nieodróżnialne od 9
Func   PROCEDURE(*SHORT)        ! 11
Func1 PROCEDURE(*SHORT)
Func1a  PROCEDURE(*SHORT)
Func2 PROCEDURE(*LONG)
Func   PROCEDURE(Func1)         ! 12
Func   PROCEDURE(Func1a)        ! nielegalne, to samo, co 12
Func   PROCEDURE(Func2)         ! 13
  END
G0   GROUP
     END
G1   GROUP(G0)
     END
  CODE
    Func(A:Window)              ! wywołanie 1 zgodnie z regułą 1
    Func(A:File)                ! wywołanie 2 zgodnie z regułą 1
    Func(A:Key)                 ! wywołanie 3 zgodnie z regułą 1
    Func(A:View)                ! wywołanie 2 zgodnie z regułą 4
    Func(A:Key,A:Key)           ! wywołanie 4 zgodnie z regułą 4 (może wywołać key,key jeśli występuje)
    Func(A:G0)                  ! wywołanie 6 zgodnie z regułą 2
    Func(A:G1)                  ! wywołanie 5 zgodnie z regułą 2
    Func(A:Func2)               ! wywołanie 13 zgodnie z regułą 2
    Func(A:Key,A:G1)            ! błąd - dwuznaczność. Jeśli reguła 4 jest używana, to 7 & 8 są obie możliwe
    Func(A:Short)               ! błąd - dwuznaczność. Wywołuje 9 lub 11
    Func(A:Real)                ! wywołanie 9 zgodnie z regułą 6
    Func                        ! wywołanie 9 zgodnie z regułą 3
```

Porównaj: CLASS

Zniekształcenie nazw i zgodność z C++

Każda przeciążona funkcja posiada nazwę nadaną na czas linkowania skomponowaną w oparciu o nazwę procedury i złączoną listę argumentów (atrybut NAME może być użyty do wyłączenia zniekształcania nazw). Jest to wprowadzone w celu umożliwienia stosowania krzyżowych odwołań pomiędzy C++ i Clarion. Od strony C++ potrzebujemy:

```
#pragma name(prefix=>"")
```

i nazwy określonej wielkimi literami. Od strony Clariona potrzebujemy struktury modułu MODULE z pustym ciągiem znaków w roli parametru:

```
MODULE('')
END
```

Jedynymi procedurami, które mogą być wywoływane krzyżowo są te, których prototypy zawierają tylko typy danych z przedstawionej listy. Parametry w postaci zmiennych (przekazywane przez adres) odpowiadają parametrom referencyjnym od strony C za wyjątkiem tych, które są pomijalne (w tym wypadku odpowiadają one parametrom wskaźnikowym)

Clarion	C++
BYTE	unsigned char
USHORT	unsigned short
SHORT	short
LONG	long
ULONG	unsigned long
SREAL	float
REAL	double
*CSTRING (z RAW)	char&
<*CSTRING> (z RAW)	char*
<*GROUP> (z RAW)	void*

Przykład:

```
//C++ prototypes:
#pragma name(prefix=>"")
void HADD(short,short);
void HADD(long*,unsigned char);
void HADD(short unsigned &);
void HADD(char *,void *);
!Clarion prototypes:
MODULE('')
hADD(short,short)
HaDD(<*long>,byte)
HAdD(*ushort)
HADd(<*CSTRING>,<*GROUP>),RAW
END
```

Porównaj: NAME

Dyrektywy kompilatora

Dyrektywy kompilatora są instrukcjami nakazującymi mu wykonanie określonych czynności w trakcie trwania kompilacji. Instrukcje te nie są umieszczane w kodzie wykonywalnym.

ASSERT (ustawia założenie dla debugowania)

ASSERT(*expression*)

ASSERT Określa założenie dla celów debugowania.

expression Wyrażenie logiczne, które *powinno* być zawsze wyliczane jako prawda (dowolna wartość różna od spacji lub zera).

Instrukcja **ASSERT** określa wyrażenie *expression* do wyliczenia dokładnie w tym miejscu programu, w którym jest ona umieszczona. Może to być wyrażenie logiczne dowolnego typu i powinno być sformułowane tak, by oczekiwany wynik wyliczenia dawał zawsze prawdę (dowolną wartość różna od spacji lub zera). Celem zastosowania ASSERT jest przechwycenie błędnych założeń przyjętych przez programistę.

Jeśli tryb debugowania jest włączony i wyrażenie *expression* daje rezultat – fałsz (spacja bądź zero), jest wyświetlany komunikat o błędzie ze wskazaniem konkretnej linii kodu i modułu źródłowego. Użytkownik jest pytany, czy ma zostać wygenerowany błąd GPF (General Protection Fail) co umożliwi uaktywnienie debuggera post-mortem.

Jeśli tryb debugowania jest wyłączony, wyrażenie *expression* jest również wyliczane, ale nie pociąga to za sobą żadnych komunikatów o błędach.

Przykład:

```
MyQueue QUEUE
F1        LONG
        END
  CODE
    LOOP X# = 1 TO 10
      MyQueue.F1 = X#
      ADD(MyQueue)
      IF ERRORCODE() THEN STOP(ERROR()).
    END
    LOOP X# = 1 TO 10
      GET(MyQueue, X#)
      ASSERT(~ERRORCODE())                ! Nigdy nie powinno być błędu przy tym GET
    END
```

BEGIN (definiuje strukturę kodu)

```
BEGIN
   statements
END
```

BEGIN Deklaruje pojedynczą strukturę kodu.

statements Instrukcje wykonywalne.

Dyrektywa kompilatora **BEGIN** informuje kompilator, że ma traktować instrukcje *statements* jako pojedynczą strukturę. Struktura BEGIN musi być zakończona znakiem kropki lub instrukcją END. BEGIN jest zazwyczaj stosowany w EXECUTE w celu umożliwienia traktowania kilku linii kodu, jako linii pojedynczej.

Przykład:

```
EXECUTE Value
  Proc1                ! wykonaj, jeśli Value = 1
  BEGIN                ! wykonaj, jeśli Value = 2
    Value += 1
    Proc2
  END
  Proc3                ! wykonaj, jeśli Value = 3
END
```

Porównaj: EXECUTE

COMPILE (określa kod źródłowy do kompilacji)

COMPILE(*terminator* [, *expression*])

COMPILE Określa blok kodu źródłowego, który ma być dołączony do kompilacji.

terminator Stała łańcuchowa oznaczająca ostatnią linię bloku kodu źródłowego.

expression Wyrażenie umożliwiające warunkowe wykonanie COMPILE. Wyrażenie jest ekwiwalentem EQUATE o wartości 0 lub 1 , bądź EQUATE = integer.

Dyrektywa kompilatora **COMPILE** określa blok kodu źródłowego, który ma zostać dołączony do kompilacji. Dołączany blok zaczyna się dyrektywą COMPILE, a kończy linią zawierającą znaki określone w parametrze *terminator*. Zawartość kończącej linii jest dołączana do bloku podlegającego kompilacji.

Opcjonalny parametr *expression* umożliwia warunkowe wykonywanie COMPILE. Format wyrażenia *expression* jest stały. Jest to albo etykieta instrukcji EQUATE, albo przełącznik warunkowy ustawiony w parametrach naszego projektu (Project System), po którym następuje znak równości (=) i stała całkowita. Kod pomiędzy COMPILE a *terminator* jest kompilowany tylko wtedy, gdy wyrażenie *expression* jest prawdziwe. Jeśli wyrażenie *expression* zawiera ekwiwalent EQUATE, który aktualnie jest nieokreślony, przyjmuje się, że jego wartość jest równa zero (0).

Mimo, że występowanie wyrażenia *expression* nie jest obowiązkowe, użycie COMPILE bez wyrażenia *expression* nie daje żadnego efektu, gdyż kod źródłowy będzie zawsze kompilowany. Przeciwieństwem dyrektywy COMPILE jest dyrektywa OMIT.

Przykład:

```
OMIT('***',_WIDTH32_)                ! OMIT tylko, gdy aplikacja jest 32-bitowa
SIGNED      EQUATE(SHORT)
UNSIGNED    EQUATE(USHORT)
***
COMPILE('***',_WIDTH32_)             ! COMPILE tylko, gdy aplikacja jest 32-bitowa
SIGNED      EQUATE(LONG)
UNSIGNED    EQUATE(ULONG)
***
COMPILE('EndOfFile',OnceOnly = 0)    ! COMPILE tylko przy pierwszym wystąpieniu ponieważ
OnceOnly    EQUATE(1)                ! ekwiwalent OnceOnly jest zdefiniowany po COMPILE, które się
                                     ! do niego odnosi, tak więc przy drugim przebiegu tej samej kompilacji
                                     ! kod nie będzie kompilowany
Demo        EQUATE(1)                ! określa wartość ekwiwalentu Demo
CODE
COMPILE('EndDemoChk',Demo = 1)       ! COMPILE tylko wtedy, gdy Demo jest włączone
  DO DemoCheck                       ! sprawdzenie ograniczeń wersji demo
  ! EndDemoChk                       ! koniec warunkowego kodu COMPILE
  ! EndOfFile
```

Porównaj: OMIT, EQUATE

INCLUDE (kompilacja kodu z innego pliku)

INCLUDE(*filename* [, *section*])

INCLUDE Wskazuje kod źródłowy, który ma zostać skompilowany, a który jest zapisany w oddzielnym pliku nie będącym modułem MEMBER.

filename Stała łańcuchowa stanowiąca nazwę pliku dyskowego zawierającego dołączany kod. Jeśli zostanie pominięte rozszerzenie nazwy tego pliku, przyjmuje się, że jest nim .CLW.

section Stała łańcuchowa, odpowiadająca parametrowi *string* dyrektywy SECTION, wyznaczająca początek kodu, który ma zostać dołączony do kompilacji.

Dyrektywa kompilatora **INCLUDE** określa kod źródłowy, który pochodzi z odrębnego pliku nie należącego do modułu MEMBER. Począwszy od linii zawierającej dyrektywę INCLUDE, plik źródłowy (lub określona jego sekcja *section*) jest kompilowany w takiej kolejności, w jakiej się pojawia wewnątrz modułu. Możemy zagnieżdżać dyrektywy INCLUDEs do trzech poziomów, tak więc możliwe jest dołączenie pliku, który dołącza kolejny plik, który z kolei również dołącza następny plik – ten ostatni nie może już dołączać żadnego pliku.

Kompilator stosuje plik przekierunkowania (redirection file) - C*urrentReleaseName*.RED do wyszukania pliku *filename*, przeszukując ścieżkę określoną dla plików danego typu (określonego zazwyczaj w oparciu o rozszerzenie nazwy pliku). Nie ma zatem konieczności wskazywania pełnej ścieżki dostępu do pliku *filename*, który ma zostać dołączony. Opis pliku przekierunkowań (redirection file) znajduje się w podręczniku *User's Guide* oraz w rozdziale *Project System* podręcznika *Programmer's Guide*.

Przykład:

```
GenLedger PROCEDURE                ! deklaracja procedury
INCLUDE('filedefs.clw')            ! dołączenie pliku definicji
  CODE                             ! początek sekcji kodu
    INCLUDE('Setups','ChkErr')     ! dołączenie sekcji z pliku setups.clw
```

Porównaj: SECTION

EQUATE (przypisanie etykiety)

```
                    | label      |
label    EQUATE(    | [ constant ] | )
                    | picture    |
                    | type       |
```

EQUATE Przypisuje etykietę innej etykiecie lub stałej.

label Etykieta dowolnej instrukcji poprzedzająca instrukcję EQUATE. Jest stosowana do deklarowania alternatywnej etykiety dla instrukcji. Nie może to być etykieta procedury PROCEDURE ani podprogramu ROUTINE.

constant Stała numeryczna lub łańcuchowa. Jest stosowana do deklarowania podręcznej etykiety dla wartości stałej. Ułatwia również lokalizację i zmianę stałej. Może być pomijana tylko w strukturach ITEMIZE.

picture Wzorzec wprowadzania. Jest wykorzystywany do deklarowania podręcznych etykiet dla wzorców wprowadzania. Jednakże ani edytor ekranów, ani edytor raportów Clariona nie potrafią rozpoznać takiej etykiety jako prawidłowego wzorca wprowadzania.

type Typ danych. Wykorzystywany zazwyczaj do deklarowania pojedynczej metody deklarowania zmiennej jako jednej z kilku typów danych w zależności od ustawień kompilatora (podobnie do typedef w języku C++ dla prostych typów danych).

Dyrektywa **EQUATE** przypisuje etykietę innej etykiecie lub stałej. Nie powoduje ona zajęcia pamięci w czasie działania programu. Etykieta dyrektywy EQUATE nie może być taka sama, jak jej parametr.

Przykład:

```
Init          EQUATE(SetUpProg)                  ! ustawia nazwę aliasową
Off           EQUATE(0)                          ! Off oznacza zero
On            EQUATE(1)                          ! On oznacza jeden
PI            EQUATE(3.1415927)                  ! wartość PI
EnterMsg      EQUATE('Press Ctrl-Enter to SAVE')
SocSecPic     EQUATE(@P###-##-####P)             ! wzorzec numeru
OMIT('End16BitChk',Flag32Bit = 0)                ! OMIT jeśli kompilacja 32-bitowa jest wyłączona
SIGNED        EQUATE(LONG)                       ! SIGNED = LONG w kompilacji 32-bitowej
                                                 ! End16BitChk
OMIT('End32BitChk',Flag32Bit = 1)                ! OMIT jeśli kompilacja 32-bitowa jest włączona
SIGNED        EQUATE(SHORT)                      ! SIGNED = SHORT w kompilacji 16-bitowej
                                                 ! End32BitChk
```

Porównaj: Słowa zastrzeżone, ITEMIZE

ITEMIZE (wyliczeniowa struktura danych)

```
[ label]    ITEMIZE( [ seed ] ) [,PRE( )]
              equates
            END
```

label Opcjonalna etykieta struktury ITEMIZE.

ITEMIZE Wyliczeniowa struktura danych.

seed Stała całkowita lub stałe wyrażenie określające wartość pierwszego ekwiwalentu EQUATE struktury.

PRE Prefiks dla zmiennych struktury.

equates Kolejne deklaracje EQUATE określające dodatnie wartości całkowite z zakresu od 0 do 65535.

Struktura **ITEMIZE** deklaruje wyliczeniową strukturę danych. Jeśli pierwszy ekwiwalent *equate* nie deklaruje wartości i nie określono wartości dla *seed*, jest przyjmowana dla niego wartość jeden (1). Wszystkie następne ekwiwalenty *equates* są zwiększane o jeden (1) w stosunku do poprzednika; o ile nie określono wartości dla następnego ekwiwalentu *equate*. W przypadku, gdy określono wartość dla ekwiwalentu *equate*, wszystkie następujące po nim ekwiwalenty *equates* zwiększają swą wartość o jeden (1) w stosunku do poprzednika i zaczynając od wartości początkowej.

Do ekwiwalentów e*quates* struktury ITEMIZE odwołujemy się stosując prefiks poprzedzający etykietę *equate* (prefiks ten określa atrybut PRE). Jeśli prefiks struktury wyliczeniowej jest pusty, do ekwiwalentów odwołujemy się poprzedzając ich etykietę etykietą *label* struktury ITEMIZE (*label*:EtykietaEkwiwalentu). Jeśli nie ma ani prefiksu, ani etykiety *label*, do ekwiwalentów *equates* odwołujemy się stosując ich etykiety.

Przykład:

```
   ITEMIZE
False  EQUATE(0)              ! False = 0
True   EQUATE                 ! True = 1
   END

Color  ITEMIZE(0),PRE         ! wartością początkową jest zero
Red      EQUATE               ! Color:Red = 0
White    EQUATE               ! Color:White = 1
Blue     EQUATE               ! Color:Blue = 2
Pink     EQUATE(5)            ! Color:Pink = 5
Green    EQUATE               ! Color:Green = 6
Last     EQUATE
       END

Stuff  ITEMIZE(Color:Last + 1),PRE(My)      ! wyrażenie stałe, jako początkowe
X        EQUATE                             ! My:X = Color:Last + 1
Y        EQUATE                             ! My:Y = Color:Last + 2
Z        EQUATE                             ! My:Z = Color:Last + 3
       END
```

Porównaj: EQUATE, PRE

OMIT (określenie bloku kodu, który nie ma być kompilowany)

OMIT(*terminator* [, *expression*])

OMIT Wskazuje blok kodu źródłowego, który ma zostać pominięty przy kompilacji.

terminator Stała łańcuchowa zaznaczająca ostatnią linię bloki kodu źródłowego.

expression Wyrażenie pozwalające na warunkowe wykonywanie OMIT. Wyrażenie jest albo ekwiwalentem EQUATE, którego wartość jest równa 0, albo ekwiwalentem QUATE o wartości całkowitej.

Dyrektywa **OMIT** wyznacza blok kodu źródłowego, który ma być pomijany podczas kompilacji. Blok ten może zawierać linie kodu dołączone na przykład w celach testowych. Pomijany blok rozpoczyna się po dyrektywie OMIT, a kończy w wierszu zawierającym ten sam łańcuch co *terminator*. Linia kończąca jest częścią bloku OMIT.

Opcjonalny parametr *expression* pozwala na warunkowe wykonywanie OMIT. Format tego wyrażenia jest stały. Jest to etykieta instrukcji EQUATE lub warunkowy przełącznik (Conditional Switch) ustawiony w projekcie (Project System), po którym może następować znak równości (=) i stała całkowita. Dyrektywa OMIT jest wykonywana tylko wtedy, gdy wyrażenie *expression* jest prawdziwe. Oznacza to, że kod umieszczony pomiędzy OMIT a *terminator* jest kompilowany tylko wtedy, gdy *expression* nie jest prawdziwe. Jeżeli wyrażenei *expression* zawiera ekwiwalent EQUATE, który nie został jeszcze zdefiniowany, jest dla niego przyjmowana wartość zerowa.

Dyrektywy COMPILE i OMIT są sobie przeciwstawne.

Przykład:

```
  OMIT('**END**')                  ! OMIT bezwarunkowy
* Main Program Loop
**END**
  OMIT('***',_WIDTH32_)            ! OMIT tylko, gdy aplikacja jest 32-bitowa
SIGNED    EQUATE(SHORT)
  ***
  COMPILE('***',_WIDTH32_)         ! COMPILE tylko, gdy aplikacja jest 32-bitowa
SIGNED    EQUATE(LONG)
  ***
  OMIT('EndOfFile',OnceOnly)       ! COMPILE tylko przy pierwszym wystąpieniu ponieważ
OnceOnly  EQUATE(1)                ! ekwiwalent OnceOnly jest zdefiniowany po COMPILE, które się
                                   ! do niego odnosi, tak więc przy drugim przebiegu tej samej kompilacji
                                   ! kod nie będzie kompilowany
Demo   EQUATE(0)                   ! określa wartość ekwiwalentu Demo
  CODE
  OMIT('EndDemoChk',Demo = 0)      ! OMIT tylko, gdy Demo jest wyłączone
    DO DemoCheck                   ! sprawdzenie przekazanych ograniczeń wersji demo
    ! EndDemoChk
  ! Koniec omijanego kodu
! EndOfFile
```

Porównaj: COMPILE, EQUATE

SECTION (określenie sekcji kodu źródłowego)

SECTION(*string*)

SECTION Identyfikuje początek bloku wykonywalnego kodu źródłowego lub deklaracji danych.

string Stała łańcuchowa stanowiąca nazwę sekcji SECTION.

Dyrektywa kompilatora **SECTION** identyfikuje początek bloku wykonywalnego kodu źródłowego lub deklaracji danych, które mogą być dołączone, za pomocą INCLUDE, do kodu źródłowego zapisanego w innym pliku.

Parametr *string* dyrektywy SECTION jest stosowany jako parametr opcjonalny dyrektywy INCLUDE dołączającej określony blok kodu źródłowego. SECTION jest kończone przez następne SECTION lub koniec pliku.

Przykład:

```
  SECTION('FirstSection')          ! początek sekcji
FieldOne    STRING(20)
FieldTwo    LONG
  SECTION('SecondSection')         ! koniec poprzedniej sekcji, początek nowej sekcji
    IF Number <> SavNumber
      DO GetNumber
    END

  SECTION('ThirdSection')          ! koniec poprzedniej sekcji, początek nowej sekcji
    CASE Action
    OF 1
      DO AddRec
    OF 2
      DO ChgRec
    OF 3
      DO DelRec
    END                            ! trzecia sekcja kończy się wraz z końcem pliku
```

Porównaj: INCLUDE

SIZE (rozmiar pamięci w bajtach)

```
          | variable |
SIZE(     | constant |)
          | picture  |
```

SIZE Określa rozmiar pamięci wykorzystywanej do przechowywania.

variable Etykieta wcześniej zadeklarowanej zmiennej.

constant Stała numeryczna lub łańcuchowa.

picture Wzorzec.

SIZE nakazuje kompilatorowi zapewnienie odpowiedniej ilości pamięci, określonej w bajtach, przeznaczonej do przechowywania zmiennej *variable*, stałej *constant* lub wzorca *picture*.

Przykład:

```
SavRec    STRING(1),DIM(SIZE(Cus:Record)          ! wymiar tablicy łańcuchowej ustawiony na rozmiar rekordu
StringVar STRING(SIZE('TopSpeed Corporation'))    ! łańcuch wystarczająco długi dla stałej
LOOP I# = 1 TO SIZE(ParseString)                  ! powtarzanie dla liczby bajtów w łańcuchu
PicLen = SIZE(@P(###)###-####P)                   ! zachowanie rozmiaru wzorca
```

Porównaj: LEN

3 – DEKLARACJE ZMIENNYCH

Proste typy danych

BYTE (liczba całkowita, 1-bajtowa, bez znaku)

label **BYTE**(*initial value*) [,**DIM**( )] [,**OVER**( )] [,**NAME**( )] [,**EXTERNAL**] [,**DLL**] [,**STATIC**] [,**THREAD**] [,**AUTO**] [,**PRIVATE**] [,**PROTECTED**]

BYTE Liczba całkowita bez znaku o rozmiarze jednego bajtu.

```
Format:        wielkość
               |. . . . . . . .|
Bity:          7               0
Zakres:  0 do 255
```

initial value Stała numeryczna stanowiąca wartość początkową. Jeśli zostanie pominięta, wartością początkową jest 0; za wyjątkiem sytuacji, gdy jest określony atrybut AUTO.

DIM Rozmiar zmiennej wystepującej w postaci tablicy.

OVER Współdzielenie tego samego obszaru pamięci z inną zmienną.

NAME Określa alternatywną, „zewnętrzną" nazwę dla pola.

EXTERNAL Określa, że dana zmienna jest zdefiniowana, a pamięć dla niej zaalokowana, w zewnętrznej bibliotece. Nie dotyczy deklaracji FILE, QUEUE i GROUP.

DLL Określa, że zmienna jest zdefiniowana w bibliotece dynamicznej DLL. Występuje w połączeniu z atrybutem EXTERNAL.

STATIC Pamięć dla zmiennej jest przydzielona statycznie, na stałe.

THREAD Pamięć dla zmiennej jest przydzielana oddzielnie dla każdego wątku. Również, w sposób niejawny nadaje atrybut STATIC lokalnym zmiennym procedury.

AUTO Powoduje, że zmienna nie posiada wartości początkowej.

PRIVATE Powoduje, że zmienna nie jest widoczna poza modułem zawierającym metody klasy CLASS, w której została zadeklarowana. Dotyczy tylko zmiennych użytych w deklaracjach klas.

PROTECTED Powoduje, że zmienna nie jest widoczna poza metodami klasy bazowej, w której została zadeklarowana, i klas dziedziczących. Dotyczy tylko zmiennych użytych w deklaracjach klas.

Przykład:

```
Count1   BYTE                ! deklaruje jedno-bajtową liczbę całkowitą
Count2   BYTE,OVER(Count1)   ! deklaruje zmienną nałożoną na jedno-bajtową liczbę całkowitą
Count4   BYTE,DIM(5)         ! deklaruje jako tablicę 5-elementową
Count4   BYTE(5)             ! deklaruje z początkową wartością
```

SHORT (liczba całkowita, 2-bajtowa, ze znakiem)

label **SHORT**([*initial value*]) [,**DIM**( )] [,**OVER**( )] [,**NAME**( )] [,**EXTERNAL**] [,**DLL**] [,**STATIC**] [,**THREAD**] [,**AUTO**] [,**PRIVATE**] [,**PROTECTED**]

SHORT Liczba całkowita ze znakiem o rozmiarze dwóch bajtów.

```
Format:    ±    wielkość
           | . |. . . . . . . . . . . . . . . |
Bity:     15 14                    0
Zakres:   -32,768 do 32,767
```

initial value Stała numeryczna stanowiąca wartość początkową. Jeśli zostanie pominięta, wartością początkową jest 0; za wyjątkiem sytuacji, gdy jest określony atrybut AUTO.

DIM Rozmiar zmiennej wystepującej w postaci tablicy.

OVER Współdzielenie tego samego obszaru pamięci z inną zmienną.

NAME Określa alternatywną, „zewnętrzną" nazwę dla pola.

EXTERNAL Określa, że dana zmienna jest zdefiniowana, a pamięć dla niej zaalokowana, w zewnętrznej bibliotece. Nie dotyczy deklaracji FILE, QUEUE i GROUP.

DLL Określa, że zmienna jest zdefiniowana w bibliotece dynamicznej DLL. Występuje w połączeniu z atrybutem EXTERNAL.

STATIC Pamięć dla zmiennej jest przydzielona statycznie, na stałe.

THREAD Pamięć dla zmiennej jest przydzielana oddzielnie dla każdego wątku. Również, w sposób niejawny nadaje atrybut STATIC lokalnym zmiennym procedury.

AUTO Określa, że zmienna nie posiada wartości początkowej.

PRIVATE Powoduje, że zmienna nie jest widoczna poza modułem zawierającym metody klasy CLASS, w której została zadeklarowana. Dotyczy tylko zmiennych użytych w deklaracjach klas.

PROTECTED Powoduje, że zmienna nie jest widoczna poza metodami klasy bazowej, w której została zadeklarowana, i klas dziedziczących. Dotyczy tylko zmiennych użytych w deklaracjach klas.

SHORT deklaruje liczbę całkowitą ze znakiem o rozmiarze dwóch bajtów, wykorzystując format całkowitego słowa procesora Intel 8086. Najbardziej znaczący bit jest bitem znaku (0 = liczba dodatnia, 1 = liczba ujemna). Liczby ujemne są reprezentowane w standardowej notacji dopełnienia dwójkowego.

Przykład:

```
Count1       SHORT                    ! Deklaruje dwu-bajtową liczbę całkowitą ze znakiem
Count2       SHORT,OVER(Count1)       ! Deklaruje zmienną nałożoną na dwu-bajtową liczbę całkowitą ze znakiem
Count3       SHORT,DIM(4)             ! Deklaruje jako tablicę 4-elementową liczba całkowitych krótkich
Count4       SHORT(5)                 ! Deklaruje z wartością początkową
Count5       SHORT,EXTERNAL           ! Deklaruje jako zmienną zewnętrzną
Count6       SHORT,EXTERNAL,DLL       ! Deklaruje jako zmienną zewnętrzną w .DLL
Count7       SHORT,NAME('SixCount')   ! Deklaruje z nazwą zewnętrzną
ExampleFile  FILE,DRIVER('Clarion')   ! Deklaruje plik
Record         RECORD
Count7           SHORT,NAME('Counter')      ! Deklaruje z nazwą zewnętrzną
                 . .
```

USHORT (liczba całkowita, 2-bajtowa, bez znaku)

label **USHORT**([*initial value*]) [,**DIM**( )] [,**OVER**( )] [,**NAME**( )] [,**EXTERNAL**] [,**DLL**] [,**STATIC**] [,**THREAD**] [,**AUTO**] [,**PRIVATE**] [,**PROTECTED**]

USHORT Liczba całkowita bez znaku o rozmiarze dwóch bajtów.

```
Format:          wielkość
                 |. . . . . . . . . . . . . . . .|
Bity:            15                             0
Zakres:    0 do 65,535
```

initial value Stała numeryczna stanowiąca wartość początkową. Jeśli zostanie pominięta, wartością początkową jest 0; za wyjątkiem sytuacji, gdy jest określony atrybut AUTO.

DIM Rozmiar zmiennej wystepującej w postaci tablicy.

OVER Współdzielenie tego samego obszaru pamięci z inną zmienną.

NAME Określa alternatywną, „zewnętrzną" nazwę dla pola.

EXTERNAL Określa, że dana zmienna jest zdefiniowana, a pamięć dla niej zaalokowana, w zewnętrznej bibliotece. Nie dotyczy deklaracji FILE, QUEUE i GROUP.

DLL Określa, że zmienna jest zdefiniowana w bibliotece dynamicznej DLL. Występuje w połączeniu z atrybutem EXTERNAL.

STATIC Pamięć dla zmiennej jest przydzielona statycznie, na stałe.

THREAD Pamięć dla zmiennej jest przydzielana oddzielnie dla każdego wątku. Również, w sposób niejawny nadaje atrybut STATIC lokalnym zmiennym procedury.

AUTO Określa, że zmienna nie posiada wartości początkowej.

PRIVATE Powoduje, że zmienna nie jest widoczna poza modułem zawierającym metody klasy CLASS, w której została zadeklarowana. Dotyczy tylko zmiennych użytych w deklaracjach klas.

PROTECTED Powoduje, że zmienna nie jest widoczna poza metodami klasy bazowej, w której została zadeklarowana, i klas dziedziczących. Dotyczy tylko zmiennych użytych w deklaracjach klas.

USHORT deklaruje liczbę całkowitą bez znaku o rozmiarze dwóch bajtów wykorzystując format słowa procesora Intel 8086. W tej konfiguracji nie występuje bit znaku.

Przykład:

```
Count1      USHORT                  ! Deklaruje dwu-bajtową liczbę całkowitą bez znaku
Count2      USHORT,OVER(Count1)     ! Deklaruje zmienbną nałożoną na dwu-bajtową liczbę całkowitą bez
                                    ! znaku
Count3      USHORT,DIM(4)           ! Deklaruje jako tablicę 4 liczb całkowitych krótkich bez znaku
Count4      USHORT(5)               ! Deklaruje z wartością początkową
Count5      USHORT,EXTERNAL         ! Deklaruje jako zmienną zewnętrzną
Count6      USHORT,EXTERNAL,DLL     ! Deklaruje jako zmienną zewnętrzną w .DLL
Count7      USHORT,NAME('SixCount') ! Deklaruje z nazwą zewnętrzną
ExampleFile FILE,DRIVER('Btrieve')  ! Deklaruje plik
Record        RECORD
Count8          USHORT,NAME('Counter')    ! Deklaruje z nazwą zewnętrzną
              . .
```

LONG (liczba całkowita, 4-bajtowa, ze znakiem)

label **LONG**([*initial value*]) [,**DIM**( )] [,**OVER**( )] [,**NAME**( )] [,**EXTERNAL**] [,**DLL**] [,**STATIC**] [,**THREAD**] [,**AUTO**] [,**PRIVATE**] [,**PROTECTED**]

LONG Liczba całkowita ze znakiem o rozmiarze czterech bajtów.

```
Format:      ±            wielkość
             | . |. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . |
Bity:        31  30                                                          0
Zakres:    -2,147,483,648 do 2,147,483,647
```

initial value Stała numeryczna stanowiąca wartość początkową. Jeśli zostanie pominięta, wartością początkową jest 0; za wyjątkiem sytuacji, gdy jest określony atrybut AUTO.

DIM Rozmiar zmiennej wystepującej w postaci tablicy.

OVER Współdzielenie tego samego obszaru pamięci z inną zmienną.

NAME Określa alternatywną, „zewnętrzną" nazwę dla pola.

EXTERNAL Określa, że dana zmienna jest zdefiniowana, a pamięć dla niej zaalokowana, w zewnętrznej bibliotece. Nie dotyczy deklaracji FILE, QUEUE i GROUP.

DLL Określa, że zmienna jest zdefiniowana w bibliotece dynamicznej DLL. Występuje w połączeniu z atrybutem EXTERNAL.

STATIC Pamięć dla zmiennej jest przydzielona statycznie, na stałe.

THREAD Pamięć dla zmiennej jest przydzielana oddzielnie dla każdego wątku. Również, w sposób niejawny nadaje atrybut STATIC lokalnym zmiennym procedury.

AUTO Określa, że zmienna nie posiada wartości początkowej.

PRIVATE Powoduje, że zmienna nie jest widoczna poza modułem zawierającym metody klasy CLASS, w której została zadeklarowana. Dotyczy tylko zmiennych użytych w deklaracjach klas.

PROTECTED Powoduje, że zmienna nie jest widoczna poza metodami klasy bazowej, w której została zadeklarowana, i klas dziedziczących. Dotyczy tylko zmiennych użytych w deklaracjach klas.

LONG deklaruje liczbę całkowitą ze znakiem o rozmiarze czterech bajtów wykorzystując format długiej liczby całkowitej procesora Intel 8086. Najbardziej znaczący bit jest bitem znaku (0 = liczba dodatnia, 1 = liczba ujemna). Liczby ujemne są reprezentowane w standardowej notacji dopełnienia dwójkowego.

Przykład:

```
Count1      LONG                    ! Deklaruje cztero-bajtową liczbę całkowitą ze znakiem
Count2      LONG,OVER(Count1)       ! Deklaruje zmienną nałożoną na cztero-bajtową liczbę całkowitą
                                    ! ze znakiem
Count3      LONG,DIM(4)             ! Deklaruje jako tablicę 4 liczb całkowitych długich
Count4      LONG(5)                 ! Deklaruje z wartością początkową
Count5      LONG,EXTERNAL           ! Deklaruje jako zmienną zewnętrzną
Count6      LONG,EXTERNAL,DLL       ! Deklaruje jako zmienną zewnętrzną w .DLL
Count7      LONG,NAME('SixCount')   ! Deklaruje z nazwą zewnętrzną
ExampleFile FILE,DRIVER('Clarion')  ! Deklaruje plik
Record        RECORD
Count8          LONG,NAME('Counter')    ! Deklaruje z nazwą zewnętrzną
              . .
```

ULONG (liczba całkowita, 4-bajtowa, bez znaku)

label **ULONG**([*initial value*]) [,**DIM**( )] [,**OVER**( )] [,**NAME**( )] [,**EXTERNAL**] [,**DLL**] [,**STATIC**] [,**THREAD**] [,**AUTO**] [,**PRIVATE**] [,**PROTECTED**]

ULONG Liczba całkowita bez znaku o rozmiarze czterech bajtów.

```
Format:              wielkość
                  |.................................|
Bity:             31                                0
Zakres:           0 do 4,294,967,295
```

initial value Stała numeryczna stanowiąca wartość początkową. Jeśli zostanie pominięta, wartością początkową jest 0; za wyjątkiem sytuacji, gdy jest określony atrybut AUTO.

DIM Rozmiar zmiennej wystepującej w postaci tablicy.

OVER Współdzielenie tego samego obszaru pamięci z inną zmienną.

NAME Określa alternatywną, „zewnętrzną” nazwę dla pola.

EXTERNAL Określa, że dana zmienna jest zdefiniowana, a pamięć dla niej zaalokowana, w zewnętrznej bibliotece. Nie dotyczy deklaracji FILE, QUEUE i GROUP.

DLL Określa, że zmienna jest zdefiniowana w bibliotece dynamicznej DLL. Występuje w połączeniu z atrybutem EXTERNAL.

STATIC Pamięć dla zmiennej jest przydzielona statycznie, na stałe.

THREAD Pamięć dla zmiennej jest przydzielana oddzielnie dla każdego wątku. Również, w sposób niejawny nadaje atrybut STATIC lokalnym zmiennym procedury.

AUTO Określa, że zmienna nie posiada wartości początkowej.

PRIVATE Powoduje, że zmienna nie jest widoczna poza modułem zawierającym metody klasy CLASS, w której została zadeklarowana. Dotyczy tylko zmiennych użytych w deklaracjach klas.

PROTECTED Powoduje, że zmienna nie jest widoczna poza metodami klasy bazowej, w której została zadeklarowana, i klas dziedziczących. Dotyczy tylko zmiennych użytych w deklaracjach klas.

ULONG deklaruje liczbę całkowitą bez znaku o rozmiarze czterech bajtów wykorzystującą format liczby całkowitej długiej procesora Intel 8086. W tej konfiguracji nie występuje bit znaku.

Przykład:

```
Count1      ULONG                    ! Deklaruje cztero-bajtową liczbę całkowitą bez znaku
Count2      ULONG,OVER(Count1)       ! Deklaruje zmienną nałożoną na cztero-bajtową liczbę całkowitą
                                     ! bez znaku
Count3      ULONG,DIM(4)             ! Deklaruje jako tablicę 4 liczb całkowitych bez znaku
Count4      ULONG(5)                 ! Deklaruje z wartością początkową
Count5      ULONG,EXTERNAL           ! Deklaruje jako zmienną zewnętrzną
Count6      ULONG,EXTERNAL,DLL       ! Deklaruje jako zmienną zewnętrzną w  .DLL
Count7      ULONG,NAME('SixCount')   ! Deklaruje z nazwą zewnętrzną
ExampleFile FILE,DRIVER('Btrieve')   ! Deklaruje plik
Record        RECORD
Count8          ULONG,NAME('Counter')      ! Deklaruje z nazwą zewnętrzną
              . .
```

SIGNED (liczba całkowita, 16/32-bitowa, ze znakiem)

label **SIGNED**([*initial value*]) [,**DIM**( )] [,**OVER**( )] [,**NAME**( )] [,**EXTERNAL**] [,**DLL**] [,**STATIC**] [,**THREAD**] [,**AUTO**] [,**PRIVATE**] [,**PROTECTED**]

SIGNED	Liczba całkowita ze znakiem, SHORT lub LONG w zależności od tego, czy kod jest kompilowany jako 16-o, czy jako 32-bitowy.
initial value	Stała numeryczna stanowiąca wartość początkową. Jeśli zostanie pominięta, wartością początkową jest 0; za wyjątkiem sytuacji, gdy jest określony atrybut AUTO.
DIM	Rozmiar zmiennej wystepującej w postaci tablicy.
OVER	Współdzielenie tego samego obszaru pamięci z inną zmienną.
NAME	Określa alternatywną, „zewnętrzną” nazwę dla pola.
EXTERNAL	Określa, że dana zmienna jest zdefiniowana, a pamięć dla niej zaalokowana, w zewnętrznej bibliotece. Nie dotyczy deklaracji FILE, QUEUE i GROUP.
DLL	Określa, że zmienna jest zdefiniowana w bibliotece dynamicznej DLL. Występuje w połączeniu z atrybutem EXTERNAL.
STATIC	Pamięć dla zmiennej jest przydzielona statycznie, na stałe.
THREAD	Pamięć dla zmiennej jest przydzielana oddzielnie dla każdego wątku. Również, w sposób niejawny nadaje atrybut STATIC lokalnym zmiennym procedury.
AUTO	Określa, że zmienna nie posiada wartości początkowej.
PRIVATE	Powoduje, że zmienna nie jest widoczna poza modułem zawierającym metody klasy CLASS, w której została zadeklarowana. Dotyczy tylko zmiennych użytych w deklaracjach klas.
PROTECTED	Powoduje, że zmienna nie jest widoczna poza metodami klasy bazowej, w której została zadeklarowana, i klas dziedziczących. Dotyczy tylko zmiennych użytych w deklaracjach klas.

SIGNED deklaruje liczbę całkowitą ze znakiem, SHORT lub LONG w zależności od tego, czy kod jest kompilowany jako 16-o, czy jako 32-bitowy. Nie jest to dokładnie typ danych, a ekwiwalent EQUATE zdefiniowany w pliku EQUATES.CLW w sposób następujący:

```
OMIT('***',_WIDTH32_)
SIGNED EQUATE(SHORT)
***
COMPILE('***',_WIDTH32_)
SIGNED EQUATE(LONG)
***
```

Typ SIGNED jest przydatny głównie przy prototypowaniu funkcji Windows API, których parametrami powinny być wartości typu SHORT w wersji 16-bitowej lub LONG w wersji 32-bitowej.

Przykład:

```
Count1    SIGNED          ! deklaruje SHORT w trybie 16-bitowym i LONG w trybie 32-bitowym
```

UNSIGNED (liczba całkowita, 16/32-bitowa, bez znaku)

label **UNSIGNED**([*initial value*]) [,**DIM**( )] [,**OVER**( )] [,**NAME**( )] [,**EXTERNAL**] [,**DLL**] [,**STATIC**] [,**THREAD**] [,**AUTO**] [,**PRIVATE**] [,**PROTECTED**]

UNSIGNED	Liczba całkowita bez znaku, USHORT lub ULONG w zależności od tego, czy kod jest kompilowany jako 16-o, czy jako 32-bitowy.
initial value	Stała numeryczna stanowiąca wartość początkową. Jeśli zostanie pominięta, wartością początkową jest 0; za wyjątkiem sytuacji, gdy jest określony atrybut AUTO.
DIM	Rozmiar zmiennej wystepującej w postaci tablicy.
OVER	Współdzielenie tego samego obszaru pamięci z inną zmienną.
NAME	Określa alternatywną, „zewnętrzną” nazwę dla pola.
EXTERNAL	Określa, że dana zmienna jest zdefiniowana, a pamięć dla niej zaalokowana, w zewnętrznej bibliotece. Nie dotyczy deklaracji FILE, QUEUE i GROUP.
DLL	Określa, że zmienna jest zdefiniowana w bibliotece dynamicznej DLL. Występuje w połączeniu z atrybutem EXTERNAL.
STATIC	Pamięć dla zmiennej jest przydzielona statycznie, na stałe.
THREAD	Pamięć dla zmiennej jest przydzielana oddzielnie dla każdego wątku. Również, w sposób niejawny nadaje atrybut STATIC lokalnym zmiennym procedury.
AUTO	Określa, że zmienna nie posiada wartości początkowej.
PRIVATE	Powoduje, że zmienna nie jest widoczna poza modułem zawierającym metody klasy CLASS, w której została zadeklarowana. Dotyczy tylko zmiennych użytych w deklaracjach klas.
PROTECTED	Powoduje, że zmienna nie jest widoczna poza metodami klasy bazowej, w której została zadeklarowana, i klas dziedziczących. Dotyczy tylko zmiennych użytych w deklaracjach klas.

UNSIGNED deklaruje liczbę całkowitą bez znaku, USHORT lub LONG w zależności od tego, czy kod jest kompilowany jako 16-o, czy jako 32-bitowy. Nie jest to dokładnie typ danych, a ekwiwalent EQUATE zdefiniowany w pliku EQUATES.CLW w sposób następujący:

```
OMIT('***',_WIDTH32_)
UNSIGNED EQUATE(USHORT)
***
COMPILE('***',_WIDTH32_)
UNSIGNED EQUATE(LONG)
***
```

Typ UNSIGNED jest przydatny głównie przy prototypowaniu funkcji Windows API, których parametrami powinny być wartości typu USHORT w wersji 16-bitowej lub LONG (bądź ULONG) w wersji 32-bitowej.

Przykład:

```
Count1    UNSIGNED        ! deklaruje USHORT w trybie 16-bitowym, a LONG w trybie 32-bitowym
```

SREAL (liczba zmiennoprzecinkowa, 4-bajtowa, ze znakiem)

label **SREAL**([*initial value*]) [,**DIM**( )] [,**OVER**( )] [,**NAME**( )] [,**EXTERNAL**] [,**DLL**] [,**STATIC**] [,**THREAD**] [,**AUTO**] [,**PRIVATE**] [,**PROTECTED**]

SREAL Liczba zmiennoprzecinkowa o rozmiarze czterech bajtów.

```
Format:    ±    eksponent          wartość znacząca
           | . |. . . . . . . .|. . . . . . . . . . . . . . . . . . . . . . . |
Bity:      31 30            23                                              0
Zakres:    0, ± 1.175494e-38 .. ± 3.402823e+38 (6 cyfr znaczących)
```

initial value Stała numeryczna stanowiąca wartość początkową. Jeśli zostanie pominięta, wartością początkową jest 0; za wyjątkiem sytuacji, gdy jest określony atrybut AUTO.

DIM Rozmiar zmiennej wystepującej w postaci tablicy.

OVER Współdzielenie tego samego obszaru pamięci z inną zmienną.

NAME Określa alternatywną, „zewnętrzną" nazwę dla pola.

EXTERNAL Określa, że dana zmienna jest zdefiniowana, a pamięć dla niej zaalokowana, w zewnętrznej bibliotece. Nie dotyczy deklaracji FILE, QUEUE i GROUP.

DLL Określa, że zmienna jest zdefiniowana w bibliotece dynamicznej DLL. Występuje w połączeniu z atrybutem EXTERNAL.

STATIC Pamięć dla zmiennej jest przydzielona statycznie, na stałe.

THREAD Pamięć dla zmiennej jest przydzielana oddzielnie dla każdego wątku. Również, w sposób niejawny nadaje atrybut STATIC lokalnym zmiennym procedury.

AUTO Określa, że zmienna nie posiada wartości początkowej.

PRIVATE Powoduje, że zmienna nie jest widoczna poza modułem zawierającym metody klasy CLASS, w której została zadeklarowana. Dotyczy tylko zmiennych użytych w deklaracjach klas.

PROTECTED Powoduje, że zmienna nie jest widoczna poza metodami klasy bazowej, w której została zadeklarowana, i klas dziedziczących. Dotyczy tylko zmiennych użytych w deklaracjach klas.

SREAL deklaruje liczbę zmiennoprzecinkową ze znakiem o rozmiarze czterech bajtów zgodną z formatem liczby całkowitej krótkiej (pojedynczej precyzji) procesora 8087.

Przykład:

```
Count1      SREAL                   ! Deklaruje cztero-bajtową liczbę zmiennoprzecinkową ze znakiem
Count2      SREAL,OVER(Count1)      ! Deklaruje zmienną nałożoną na cztero-bajtową liczbę
                                    ! zmiennoprzecinkową ze znakiem
Count3      SREAL,DIM(4)            ! Deklaruje jako tablicę 4 liczb zmiennoprzecinkowych ze znakiem
Count4      SREAL(5)                ! Deklaruje z wartością początkową
Count5      SREAL,EXTERNAL          ! Deklaruje jako zmienną zewnętrzną
Count6      SREAL,EXTERNAL,DLL      ! Deklaruje jako zmienną zewnętrzną w .DLL
Count7      SREAL,NAME('SixCount')  ! Deklaruje z nazwą zewnętrzną
ExampleFile FILE,DRIVER('Btrieve')  ! Deklaruje plik
Record        RECORD
Count8          SREAL,NAME('Counter')  ! Deklaruje z nazwą zewnętrzną
              . .
```

REAL (liczba zmiennoprzecinkowa, 8-bajtowa, ze znakiem)

label **REAL**([*initial value*]) [,**DIM**( )] [,**OVER**( )] [,**NAME**( )] [,**EXTERNAL**] [,**DLL**] [,**STATIC**] [,**THREAD**] [,**AUTO**] [,**PRIVATE**] [,**PROTECTED**]

REAL Liczba zmiennoprzecinkowa o rozmiarze ośmiu bajtów.

```
Format:    ±     eksponent              wartość znacząca
           | . |. . . . . . . . . . . .|. . ..... . . . . . . . . . . . . . . . . . . . . . . . . . . . . |
Bity :    63  62              52                                                                        0
Zakres:   0, ± 2.225073858507201e-308 .. ± 1.79769313496231e+308
(15 cyfr znaczących)
```

initial value Stała numeryczna stanowiąca wartość początkową. Jeśli zostanie pominięta, wartością początkową jest 0; za wyjątkiem sytuacji, gdy jest określony atrybut AUTO.

DIM Rozmiar zmiennej wystepującej w postaci tablicy.

OVER Współdzielenie tego samego obszaru pamięci z inną zmienną.

NAME Określa alternatywną, „zewnętrzną" nazwę dla pola.

EXTERNAL Określa, że dana zmienna jest zdefiniowana, a pamięć dla niej zaalokowana, w zewnętrznej bibliotece. Nie dotyczy deklaracji FILE, QUEUE i GROUP.

DLL Określa, że zmienna jest zdefiniowana w bibliotece dynamicznej DLL. Występuje w połączeniu z atrybutem EXTERNAL.

STATIC Pamięć dla zmiennej jest przydzielona statycznie, na stałe.

THREAD Pamięć dla zmiennej jest przydzielana oddzielnie dla każdego wątku. Również, w sposób niejawny nadaje atrybut STATIC lokalnym zmiennym procedury.

AUTO Określa, że zmienna nie posiada wartości początkowej.

PRIVATE Powoduje, że zmienna nie jest widoczna poza modułem zawierającym metody klasy CLASS, w której została zadeklarowana. Dotyczy tylko zmiennych użytych w deklaracjach klas.

PROTECTED Powoduje, że zmienna nie jest widoczna poza metodami klasy bazowej, w której została zadeklarowana, i klas dziedziczących. Dotyczy tylko zmiennych użytych w deklaracjach klas.

REAL deklaruje liczbę zmiennoprzecinkową ze znakiem o rozmiarze ośmiu bajtów zgodną z formatem liczby całkowitej długiej (podwójnej precyzji) procesora 8087

Przykład:

```
Count1      REAL                ! Deklaruje ośmio-bajtową liczbę zmiennoprzecinkową ze znakiem
Count2      REAL,OVER(Count1)   ! Deklaruje zmienną nałożoną na ośmio-bajtową liczbę zmiennoprzecinkową
                                ! ze znakiem
Count3      REAL,DIM(4)         ! Deklaruje jako tablicę 4 liczb zmiennoprzecinkowych
Count4      REAL(5)             ! Deklaruje z wartością początkową
Count5      REAL,EXTERNAL       ! Deklaruje jako zmienną zewnętrzną
Count6      REAL,EXTERNAL,DLL   ! Deklaruje jako zmienną zewnętrzną w .DLL
Count7      REAL,NAME('SixCount')  ! Deklaruje z nazwą zewnętrzną
ExampleFile FILE,DRIVER('Clarion') ! Deklaruje plik
Record        RECORD
Count8          REAL,NAME('Counter')  ! Deklaruje z nazwą zewnętrzną
              . .
```

BFLOAT4 (liczba zmiennoprzecinkowa, 4-bajtowa, ze znakiem)

label **BFLOAT4**([*initial value*]) [,**DIM**( )] [,**OVER**( )] [,**NAME**( )] [,**EXTERNAL**] [,**DLL**] [,**STATIC**] [,**THREAD**] [,**AUTO**] [,**PRIVATE**] [,**PROTECTED**]

BFLOAT4 Liczba zmiennoprzecinkowa o rozmiarze czterech bajtów.

```
Format:      eksponent  ±          wartość znacząca
             |..........|.|.............................|
Bity:        31         23 22                         0
Zakres:  0, ± 5.87747e-39 .. ± 1.70141e+38 (6 cyfr znaczących)
```

initial value Stała numeryczna stanowiąca wartość początkową. Jeśli zostanie pominięta, wartością początkową jest 0; za wyjątkiem sytuacji, gdy jest określony atrybut AUTO.

DIM Rozmiar zmiennej wystepującej w postaci tablicy.

OVER Współdzielenie tego samego obszaru pamięci z inną zmienną.

NAME Określa alternatywną, „zewnętrzną" nazwę dla pola.

EXTERNAL Określa, że dana zmienna jest zdefiniowana, a pamięć dla niej zaalokowana, w zewnętrznej bibliotece. Nie dotyczy deklaracji FILE, QUEUE i GROUP.

DLL Określa, że zmienna jest zdefiniowana w bibliotece dynamicznej DLL. Występuje w połączeniu z atrybutem EXTERNAL.

STATIC Pamięć dla zmiennej jest przydzielona statycznie, na stałe.

THREAD Pamięć dla zmiennej jest przydzielana oddzielnie dla każdego wątku. Również, w sposób niejawny nadaje atrybut STATIC lokalnym zmiennym procedury.

AUTO Określa, że zmienna nie posiada wartości początkowej.

PRIVATE Powoduje, że zmienna nie jest widoczna poza modułem zawierającym metody klasy CLASS, w której została zadeklarowana. Dotyczy tylko zmiennych użytych w deklaracjach klas.

PROTECTED Powoduje, że zmienna nie jest widoczna poza metodami klasy bazowej, w której została zadeklarowana, i klas dziedziczących. Dotyczy tylko zmiennych użytych w deklaracjach klas.

BFLOAT4 deklaruje liczbę zmiennoprzecinkową ze znakiem o rozmiarze czterech bajtów zgodną z formatem Microsoft BASIC (pojedynczej precyzji) Ten typ danych ma na celu zachowanie kompatybilności z istniejącymi danymi ponieważ jest wewnętrznie konwertowany do typu REAL przed wykonaniem operacji arytmetycznej.

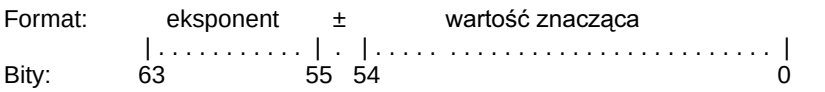
Przykład:

```
Count1      BFLOAT4                    ! Deklaruje cztero-bajtową liczbę zmiennoprzecinkową ze znakiem
Count2      BFLOAT4,OVER(Count1)       ! Deklaruje zmienną nałożoną na cztero-bajtową liczbę
                                       ! zmiennoprzecinkową ze znakiem
Count3      BFLOAT4,DIM(4)             ! Deklaruje tablicę 4 liczb rzeczywistych pojedynczej precyzji
Count4      BFLOAT4(5)                 ! Deklaruje z wartością początkową
Count5      BFLOAT4,EXTERNAL           ! Deklaruje jako zmienną zewnętrzną
Count6      BFLOAT4,EXTERNAL,DLL       ! Deklaruje jako zmienną zewnętrzną w .DLL
Count7      BFLOAT4,NAME('SixCount')   ! Deklaruje z nazwą zewnętrzną
ExampleFile FILE,DRIVER('Btrieve')     ! Deklaruje plik
Record        RECORD
Count8          BFLOAT4,NAME('Counter')  ! Deklaruje z nazwą zewnętrzną
              . .
```

BFLOAT8 (liczba zmiennoprzecinkowa, 8-bajtowa, ze znakiem)

label **BFLOAT8**([*initial value*]) [,**DIM**( )] [,**OVER**( )] [,**NAME**( )] [,**EXTERNAL**] [,**DLL**] [,**STATIC**] [,**THREAD**] [,**AUTO**] [,**PRIVATE**] [,**PROTECTED**]

BFLOAT8 Liczba zmiennoprzecinkowa o rozmiarze ośmiu bajtów.



Zakres: 0, ± 5.877471754e-39.. ± 1.7014118346e+38 (15 cyfr znaczących)

initial value Stała numeryczna stanowiąca wartość początkową. Jeśli zostanie pominięta, wartością początkową jest 0; za wyjątkiem sytuacji, gdy jest określony atrybut AUTO.

DIM Rozmiar zmiennej wystepującej w postaci tablicy.

OVER Współdzielenie tego samego obszaru pamięci z inną zmienną.

NAME Określa alternatywną, „zewnętrzną" nazwę dla pola.

EXTERNAL Określa, że dana zmienna jest zdefiniowana, a pamięć dla niej zaalokowana, w zewnętrznej bibliotece. Nie dotyczy deklaracji FILE, QUEUE i GROUP.

DLL Określa, że zmienna jest zdefiniowana w bibliotece dynamicznej DLL. Występuje w połączeniu z atrybutem EXTERNAL.

STATIC Pamięć dla zmiennej jest przydzielona statycznie, na stałe.

THREAD Pamięć dla zmiennej jest przydzielana oddzielnie dla każdego wątku. Również, w sposób niejawny nadaje atrybut STATIC lokalnym zmiennym procedury.

AUTO Określa, że zmienna nie posiada wartości początkowej.

PRIVATE Powoduje, że zmienna nie jest widoczna poza modułem zawierającym metody klasy CLASS, w której została zadeklarowana. Dotyczy tylko zmiennych użytych w deklaracjach klas.

PROTECTED Powoduje, że zmienna nie jest widoczna poza metodami klasy bazowej, w której została zadeklarowana, i klas dziedziczących. Dotyczy tylko zmiennych użytych w deklaracjach klas.

BFLOAT8 deklaruje liczbę zmiennoprzecinkową ze znakiem o rozmiarze ośmiu bajtów zgodną z formatem Microsoft BASIC (podwójnej precyzji) Ten typ danych ma na celu zachowanie kompatybilności z istniejącymi danymi ponieważ jest wewnętrznie konwertowany do typu REAL przed wykonaniem operacji arytmetycznej.

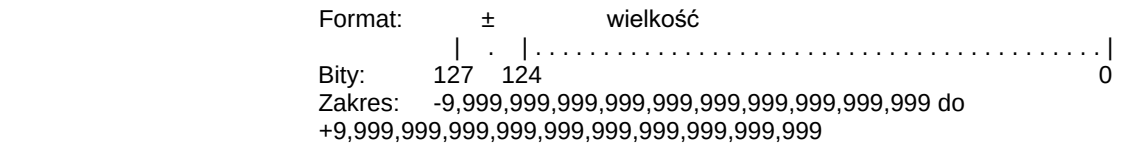
Przykład:

```
Count1        BFLOAT8                      ! Deklaruje ośmio-bajtową liczbę zmiennoprzecinkową ze
                                           !  znakiem
Count2        BFLOAT8,OVER(Count1)         ! Deklaruje zmienną nałożoną
Count3        BFLOAT8,DIM(4)               ! Deklaruje jako tablicę 4 liczb rzeczywistych
Count4        BFLOAT8(5)                   ! Deklaruje z wartością początkową
Count5        BFLOAT8,EXTERNAL             ! Deklaruje jako zmienną zewnętrzną
Count6        BFLOAT8,EXTERNAL,DLL         ! Deklaruje jako zmienną zewnętrzną w .DLL
Count7        BFLOAT8,NAME('SixCount')     ! Deklaruje z nazwą zewnętrzną
ExampleFile   FILE,DRIVER('Btrieve')       ! Deklaruje plik
Record          RECORD
Count8             BFLOAT8,NAME('Counter') ! Deklaruje z nazwą zewnętrzną
                . .
```

DECIMAL (upakowana liczba dziesiątkowa ze znakiem)

label **DECIMAL**(*length* [, *places*] [, *initial value*]) [,**DIM**( )] [,**OVER**( )] [,**NAME**( )] [,**EXTERNAL**] [,**DLL**] [,**STATIC**] [,**THREAD**] [,**AUTO**] [,**PRIVATE**] [,**PROTECTED**]

DECIMAL Upakowana, dziesiątkowa liczba zmiennoprzecinkowa.



length Wymagana stała numeryczna zawierająca całkowitą liczbę cyfr dziesiętnych (połączona część całkowita i ułamkowa) w zmiennej. Maksymalna wartość *length* to 31.

places Stała numeryczna określająca stałą liczbę cyfr dziesiętnych części ułamkowej (na prawo od kropki dziesiętnej) w zmiennej. Musi ona być mniejsza bądź równa parametrowi *length*. Jeśli pominiemy ten parametr, zmienna jest deklarowana jako liczba całkowita.

initial value Stała numeryczna stanowiąca wartość początkową. Jeśli zostanie pominięta, wartością początkową jest 0; za wyjątkiem sytuacji, gdy jest określony atrybut AUTO.

DIM Rozmiar zmiennej wystepującej w postaci tablicy.

OVER Współdzielenie tego samego obszaru pamięci z inną zmienną.

NAME Określa alternatywną, „zewnętrzną” nazwę dla pola.

EXTERNAL Określa, że dana zmienna jest zdefiniowana, a pamięć dla niej zaalokowana, w zewnętrznej bibliotece. Nie dotyczy deklaracji FILE, QUEUE i GROUP.

DLL Określa, że zmienna jest zdefiniowana w bibliotece dynamicznej DLL. Występuje w połączeniu z atrybutem EXTERNAL.

STATIC Pamięć dla zmiennej jest przydzielona statycznie, na stałe.

THREAD Pamięć dla zmiennej jest przydzielana oddzielnie dla każdego wątku. Również, w sposób niejawny nadaje atrybut STATIC lokalnym zmiennym procedury.

AUTO Określa, że zmienna nie posiada wartości początkowej.

PRIVATE Powoduje, że zmienna nie jest widoczna poza modułem zawierającym metody klasy CLASS, w której została zadeklarowana. Dotyczy tylko zmiennych użytych w deklaracjach klas.

PROTECTED Powoduje, że zmienna nie jest widoczna poza metodami klasy bazowej, w której została zadeklarowana, i klas dziedziczących. Dotyczy tylko zmiennych użytych w deklaracjach klas.

DECIMAL deklaruje upakowaną, dziesiątkową liczbę zmiennoprzecinkową ze znakiem o zmiennej długości. Każdy bajt w DECIMAL przechowuje dwie cyfry dziesiętne (4 bity na cyfrę). Bajt znajdujący się na skraju lewej strony, w swej bardziej znaczącej części, przechowuje znak (0 = liczba dodatnia, inna wartość = liczba ujemna).

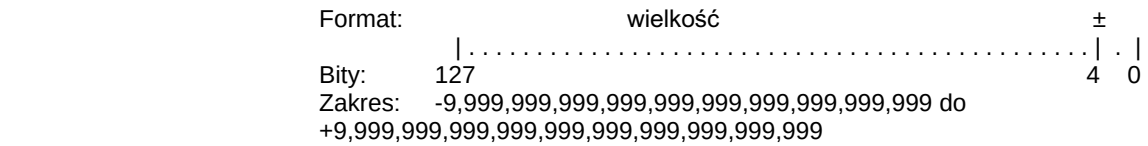
Przykład:

```
Count1      DECIMAL(5,0)                ! Deklaruje trzy-bajtową upakowaną liczbę dziesiętną ze znakiem
Count2      DECIMAL(5),OVER(Count1)      ! Deklaruje zmienną nałożoną na trzy-bajtową upakowaną
                                         ! liczbę dziesiętną
Count3      DECIMAL(5,0),DIM(4)          ! Deklaruje jako tablicę 4 upakowanych liczb dziesiętnych
Count4      DECIMAL(5,0,5)               ! Deklaruje z wartością początkową
Count5      DECIMAL(5,0),EXTERNAL        ! Deklaruje jako zmienną zewnętrzną
Count6      DECIMAL(5,0),EXTERNAL,DLL    ! Deklaruje jako zmienną zewnętrzną w .DLL
Count7      DECIMAL(5,0),NAME('SixCount') ! Deklaruje z nazwą zewnętrzną
ExampleFile FILE,DRIVER('TopSpeed')      ! Deklaruje plik
Record        RECORD
Count8          DECIMAL(5,0),NAME('Counter')  ! Deklaruje z nazwą zewnętrzną
              . .
```

PDECIMAL (upakowana liczba dziesiątkowa ze znakiem)

label **PDECIMAL**(*length* [, *places*] [, *initial value*]) [,**DIM**( )] [,**OVER**( )] [,**NAME**( )] [,**EXTERNAL**] [,**DLL**] [,**STATIC**] [,**THREAD**] [,**AUTO**] [,**PRIVATE**] [,**PROTECTED**]

PDECIMAL Upakowana, dziesiątkowa liczba zmiennoprzecinkowa.



length Wymagana stała numeryczna zawierająca całkowitą liczbę cyfr dziesiętnych (połączona część całkowita i ułamkowa) w zmiennej. Maksymalna wartośc *length* to 31.

places Stała numeryczna określająca stałą liczbę cyfr dziesiętnych części ułamkowej (na prawo od kropki dziesiętnej) w zmiennej. Musi ona być mniejsza bądź równa parametrowi *length*. Jeśli pominiemy ten parametr, zmienna jest deklarowana jako liczba całkowita.

initial value Stała numeryczna stanowiąca wartość początkową. Jeśli zostanie pominięta, wartością początkową jest 0; za wyjątkiem sytuacji, gdy jest określony atrybut AUTO.

DIM Rozmiar zmiennej wystepującej w postaci tablicy.

OVER Współdzielenie tego samego obszaru pamięci z inną zmienną.

NAME Określa alternatywną, „zewnętrzną” nazwę dla pola.

EXTERNAL Określa, że dana zmienna jest zdefiniowana, a pamięć dla niej zaalokowana, w zewnętrznej bibliotece. Nie dotyczy deklaracji FILE, QUEUE i GROUP.

DLL Określa, że zmienna jest zdefiniowana w bibliotece dynamicznej DLL. Występuje w połączeniu z atrybutem EXTERNAL.

STATIC Pamięć dla zmiennej jest przydzielona statycznie, na stałe.

THREAD Pamięć dla zmiennej jest przydzielana oddzielnie dla każdego wątku. Również, w sposób niejawny nadaje atrybut STATIC lokalnym zmiennym procedury.

AUTO Określa, że zmienna nie posiada wartości początkowej.

PRIVATE Powoduje, że zmienna nie jest widoczna poza modułem zawierającym metody klasy CLASS, w której została zadeklarowana. Dotyczy tylko zmiennych użytych w deklaracjach klas.

PROTECTED Powoduje, że zmienna nie jest widoczna poza metodami klasy bazowej, w której została zadeklarowana, i klas dziedziczących. Dotyczy tylko zmiennych użytych w deklaracjach klas.

PDECIMAL deklaruje upakowaną, dziesiątkową liczbę zmiennoprzecinkową ze znakiem o zmiennej długości stosowaną w formatach Btrieve i IBM/EBCDIC. Każdy bajt w PDECIMAL przechowuje dwie cyfry dziesiętne (4 bity na cyfrę). Bajt znajdujący się na skraju prawej strony, w swej mniej znaczącej części, przechowuje znak (0Fh lub 0Ch = liczba dodatnia, 0Dh = liczba ujemna).

Przykład:

```
Count1       PDECIMAL(5,0)                  ! Deklaruje trzy-bajtową upakowaną liczbę dziesiętną ze znakiem
Count2       PDECIMAL(5),OVER(Count1)       ! Deklaruje zmienną nałożoną na trzy-bajtową
                                            ! upakowaną liczbę dziesiętną ze znakiem
Count3       PDECIMAL(5,0),DIM(4)           ! Deklaruje jako tablicę 4 upakowanych liczb dziesiętnych
Count4       PDECIMAL(5,0,5)                ! Deklaruje z wartością początkową
Count5       PDECIMAL(5,0),EXTERNAL         ! Deklaruje jako zmienną zewnętrzną
Count6       PDECIMAL(5,0),EXTERNAL,DLL     ! Deklaruje jako zmienną zewnętrzną w .DLL
Count7       PDECIMAL(5,0),NAME('SixCount') ! Deklaruje z nazwą zewnętrzną
ExampleFile  FILE,DRIVER('Btrieve')         ! Deklaruje plik
Record         RECORD
Count8           PDECIMAL(5,0),NAME('Counter')  ! Deklaruje z nazwą zewnętrzną
               . .
```

STRING (łańcuch stałej długości)

```
                | length          |
label  STRING(  | string constant |) [,DIM( )][,OVER( )] [,NAME( )] [,EXTERNAL] [,DLL] [,STATIC]
                | picture         |        [,THREAD] [,AUTO] [,PRIVATE] [,PROTECTED]
```

STRING Łańcuch znaków.

Format: stała liczba znaków
Rozmiar: 1 do 65,520 bajtów w trybie 16-bitowym lub 4MB w trybie 32-bitowym.

length Stała numeryczna określająca liczbę bajtów w łańcuchu STRING. Zmienne łańcuchowe nie są inicjowane jeśli nie jest określony parametr *string constant*.

string constant Początkowa wartość łańcucha STRING. Długość łańcucha STRING (w bajtach) jest ustawiana, jako długość parametru *string constant*.

picture Parametr formatujący wartości przypisywane do zmiennej typu STRING. Długością jest liczba bajtów niezbędnych do przechowania sformatowanego łańcucha STRING.

DIM Rozmiar zmiennej wystepującej w postaci tablicy.

OVER Współdzielenie tego samego obszaru pamięci z inną zmienną.

NAME Określa alternatywną, „zewnętrzną" nazwę dla pola.

EXTERNAL Określa, że dana zmienna jest zdefiniowana, a pamięć dla niej zaalokowana, w zewnętrznej bibliotece. Nie dotyczy deklaracji FILE, QUEUE i GROUP.

DLL Określa, że zmienna jest zdefiniowana w bibliotece dynamicznej DLL. Występuje w połączeniu z atrybutem EXTERNAL.

STATIC Pamięć dla zmiennej jest przydzielona statycznie, na stałe.

THREAD Pamięć dla zmiennej jest przydzielana oddzielnie dla każdego wątku. Również, w sposób niejawny nadaje atrybut STATIC lokalnym zmiennym procedury.

AUTO Określa, że zmienna nie posiada wartości początkowej.

PRIVATE Powoduje, że zmienna nie jest widoczna poza modułem zawierającym metody klasy CLASS, w której została zadeklarowana. Dotyczy tylko zmiennych użytych w deklaracjach klas.

PROTECTED Powoduje, że zmienna nie jest widoczna poza metodami klasy bazowej, w której została zadeklarowana, i klas dziedziczących. Dotyczy tylko zmiennych użytych w deklaracjach klas.

STRING deklaruje ciąg znaków o stałej długości. Pamięć przydzielana na STRING jest inicjowana znakami spacji; o ile nie występuje atrybut AUTO. W uzupełnieniu do jawnej deklaracji, wszystkie dane typu STRING są niejawnie deklarowane jako tablica STRING(1),DIM(*długość_łańcucha*). Umożliwia to na dostęp do każdego znaku łańcucha STRING poprzez adresowanie go jako elementu tablicy. Jeśli STRING posiada dodatkowo atrybut DIM, niejawna deklaracja tablicy jest ostatnim

(opcjonalnym) wymiarem tablicy (na prawo od określonego w atrybucie DIM jawnego wymiaru).

Istnieje również możliwość bezpośredniego adresowania wielu znaków w łańcuchu STRING poprzez zastosowanie techniki „string slicing – fragmentowanie łańcucha". Przeprowadza ona podobne działania, co funkcja SUB, jest jednak bardziej elastyczna i efektywna (jednakże nie zapewnia kontroli zakresu). Jej większa elastyczność wynika z tego, że "fragment łańcucha" może być zastosowany po obu stronach instrukcji przypisania, a funkcji SUB możemy używać tylko po stronie źródłowej. Większa efektywność jest spowodowana mniejszym zużyciem pamięci.

Jeśli chcemy uzyskać "fragment łańcucha", numer jego pierwszego i ostatniego znaku oddzielamy od siebie znakiem dwukropka (:) i umieszczamy w indeksie niejawnej tablicy ([]) naszego łańcucha STRING. Numery pozycji mogą być stałymi lub całkowitymi, wyrażeniami. Jeśli stosujemy nazwy zmiennych, przed i po znaku dwukropka musimy umieścić przynajmniej jedną spację, w przeciwnym razie kompilator potraktuje cały zapis jako pojedynczą zmienną z prefiksem.

Przykład:

```
Name        STRING(20)                      ! Deklaruje pole 20 bajtowe
ArrayString STRING(5),DIM(20)               ! Deklaruje tablicę
Company     STRING('TopSpeed Corporation')  ! Nazwa firmy - 20 bajtów
Phone       STRING(@P(###)###-####P)        ! Numer telefonu - 13 bajtów
ExampleFile FILE,DRIVER('Clarion')          ! Deklaruje plik
Record        RECORD
NameField       STRING(20),NAME('Name')     ! Deklaruje z nazwą zewnętrzną
              . .
  CODE
    NameField = 'Tammi'                     ! przypisanie wartości
    NameField[5] = 'y'                      ! zmiana piątej litery
    NameField[5:6] = 'ie'                   ! zmiana fragmentu
                                            ! – piątej i szóstej litery
    ArrayString[1] = 'First'                ! przypisanie wartości do pierwszego elementu
    ArrayString[1,2] = 'u'                  ! zmiana drugiegio znaku pierwszego elementu
    ArrayString[1,2:3] = NameField[5:6]     ! przypisanie fragmentu do fragmentu
```

CSTRING (łańcuch null terminated stałej długości)

```
                    | length          |
label   CSTRING (   | string constant |) [,DIM( )] [,OVER( )] [,NAME( )] [,EXTERNAL] [,DLL]
                    | picture         |    [,STATIC] [,THREAD] [,AUTO] [,PRIVATE] [,PROTECTED]
```

CSTRING Łańcuch znakowy.

Format: Stała liczba bajtów
Rozmiar: 2 do 65,521 w 16-bitach, bez ograniczeń w 32-bitach.

length Stała numeryczna określająca liczbę bajtów w łańcuchu STRING włączając w to pozycję zajmowaną przez znak NULL kończący łańcuch. Zmienne łańcuchowe nie są inicjowane jeśli nie jest określony parametr *string constant*.

string constant Początkowa wartość łańcucha CSTRING. Długość łańcucha CSTRING (w bajtach) jest ustawiana, jako długość parametru *string constant* powiększona o znak NULL kończący łańcuch.

picture Parametr formatujący wartości przypisywane do danej typu CSTRING. Długością jest liczba bajtów niezbędnych do przechowania sformatowanego łańcucha CSTRING powiększona o znak NULL kończący łańcuch.

DIM Rozmiar zmiennej wystepującej w postaci tablicy.

OVER Współdzielenie tego samego obszaru pamięci z inną zmienną.

NAME Określa alternatywną, „zewnętrzną" nazwę dla pola.

EXTERNAL Określa, że dana zmienna jest zdefiniowana, a pamięć dla niej zaalokowana, w zewnętrznej bibliotece. Nie dotyczy deklaracji FILE, QUEUE i GROUP.

DLL Określa, że zmienna jest zdefiniowana w bibliotece dynamicznej DLL. Występuje w połączeniu z atrybutem EXTERNAL.

STATIC Pamięć dla zmiennej jest przydzielona statycznie, na stałe.

THREAD Pamięć dla zmiennej jest przydzielana oddzielnie dla każdego wątku. Również, w sposób niejawny nadaje atrybut STATIC lokalnym zmiennym procedury.

AUTO Określa, że zmienna nie posiada wartości początkowej.

PRIVATE Powoduje, że zmienna nie jest widoczna poza modułem zawierającym metody klasy CLASS, w której została zadeklarowana. Dotyczy tylko zmiennych użytych w deklaracjach klas.

PROTECTED Powoduje, że zmienna nie jest widoczna poza metodami klasy bazowej, w której została zadeklarowana, i klas dziedziczących. Dotyczy tylko zmiennych użytych w deklaracjach klas.

CSTRING deklaruje łańcuch znaków zakończony znakiem NULL (ASCII zero). Pamięć przydzielana na łańcuch CSTRING jest inicjowana z zerową długością; o ile nie występuje atrybut AUTO.

CSTRING jest zgodny z łańcuchowym typem danych stosowanym w języku C oraz z typem „ZSTRING" wykorzystywanym przez Btrieve Record Manager. Rozmiar pamięci wykorzystywanej przez CSTRING jest stałej długości, na końcu zawsze jest umieszczany znak NULL. Typu CSTRING używamy, jeśli chcemy zachować kompatybilność z zewnętrznymi plikami, czy procedurami.

W uzupełnieniu do jawnej deklaracji, wszystkie dane typu CSTRING są niejawnie deklarowane jako tablica STRING(1),DIM(*długość_łańcucha*). Umożliwia to na dostęp do każdego znaku łańcucha CSTRING poprzez adresowanie go jako elementu tablicy. Jeśli CSTRING posiada dodatkowo atrybut DIM, niejawna deklaracja tablicy jest ostatnim (opcjonalnym) wymiarem tablicy (na prawo od określonego w atrybucie DIM jawnego wymiaru).

Istnieje również możliwość bezpośredniego adresowania wielu znaków w łańcuchu CSTRING poprzez zastosowanie techniki „string slicing – fragmentowanie łańcucha". Przeprowadza ona podobne działania, co funkcja SUB, jest jednak bardziej elastyczna i efektywna (jednakże nie zapewnia kontroli zakresu). Jej większa elastyczność wynika z tego, że "fragment łańcucha" może być zastosowany po obu stronach instrukcji przypisania, a funkcji SUB możemy używać tylko po stronie źródłowej. Większa efektywność jest spowodowana mniejszym zużyciem pamięci.

Jeśli chcemy uzyskać "fragment łańcucha", numer jego pierwszego i ostatniego znaku oddzielamy od siebie znakiem dwukropka (:) i umieszczamy w indeksie niejawnej tablicy ([]) naszego łańcucha CSTRING. Numery pozycji mogą być stałymi lub całkowitymi, wyrażeniami. Jeśli stosujemy nazwy zmiennych, przed i po znaku dwukropka musimy umieścić przynajmniej jedną spację, w przeciwnym razie kompilator potraktuje cały zapis jako pojedynczą zmienną z prefiksem.

Ponieważ CSTRING jest łańcuchem typu null-terminated, programista jest odpowiedzialny za zapewnienie wystąpienia znaku NULL na końcu łańcucha, który uzyskuje poprzez fragmentowanie lub adresowanie indeksu tablicy. Jeśli nie zapewni właściwej obsługi, może to doprowadzić do pozostawienia "śmieci" po znaku kończącym łańcuch. Z tego między innymi powodu dane typu CSTRING nie powinny być stosowane w strukturach typu GROUP.

Przykład:

```
Name        CSTRING(21)                        ! Deklaruje 21-bajtowe pole - 20 bajtów danych
OtherName CSTRING(21),OVER(Name)               ! Deklaruje pole nałożone na pole Name
Contact     CSTRING(21),DIM(4)                 ! tablica 21-bajtowych pól - 80 bajtów danych
Company   CSTRING('TopSpeed Corporation')      ! 21-bajtowy łańcuch - 20 bajtów danych
Phone       CSTRING(@P(###)###-####P)          ! Deklaruje 14 bajtów - 13 bajtów danych
ExampleFile  FILE,DRIVER('Btrieve')            ! Deklaruje plik
Record          RECORD
NameField       CSTRING(21),NAME('ZstringField')    ! Deklaruje z nazwą zewnętrzną
          . .
  CODE
    Name = 'Tammi'                             ! przypisanie wartości
    Name[5] = 'y'                              ! zmiana piątej litery
    Name[6] = 's'                              ! dodanie litery
    Name[7] = '<0>'                            ! obsługa terminatora null
    Name[5:6] = 'ie'                           ! zmiana fragmentu
                                               ! – piątej i szóstej litery
    Contact[1] = 'First'                       ! przypisanie wartości do pierwszego elementu
    Contact[1,2] = 'u'                         ! zmiana drugiego znaku pierwszego elementu
    Contact[1,2:3] = Name[5:6]                 ! przypisanie fragmentu do fragmentu
```

PSTRING (łańcuch z osadzonym bajtem długości)

```
                | length          |
label  PSTRING ( | string constant |) [,DIM( )] [,OVER( )] [,NAME( )] [,EXTERNAL] [,DLL] [,STATIC]
                | picture         |       [,THREAD] [,AUTO] [,PRIVATE] [,PROTECTED]
```

PSTRING Łańcuch znakowy.

Format: Stała liczba bajtów.
Rozmiar: 2 do 256 bajtów.

length Stała numeryczna określająca liczbę bajtów w łańcuchu STRING włączając w to pozycję zajmowaną przez bajt zawierający długość łańcucha.

string constant Początkowa wartość łańcucha PSTRING. Długość łańcucha PSTRING (w bajtach) jest ustawiana, jako długość parametru *string constant* powiększona o bajt zawierający długość łańcucha.

picture Parametr formatujący wartości przypisywane do danej typu PSTRING. Długością jest liczba bajtów niezbędnych do przechowania sformatowanego łańcucha PSTRING powiększona o bajt długości (na pierwszej pozycji). Zmienne łańcuchowe nie są inicjowane, jeśli nie został określony parametr *string constant*.

DIM Rozmiar zmiennej wystepującej w postaci tablicy.

OVER Współdzielenie tego samego obszaru pamięci z inną zmienną.

NAME Określa alternatywną, „zewnętrzną" nazwę dla pola.

EXTERNAL Określa, że dana zmienna jest zdefiniowana, a pamięć dla niej zaalokowana, w zewnętrznej bibliotece. Nie dotyczy deklaracji FILE, QUEUE i GROUP.

DLL Określa, że zmienna jest zdefiniowana w bibliotece dynamicznej DLL. Występuje w połączeniu z atrybutem EXTERNAL.

STATIC Pamięć dla zmiennej jest przydzielona statycznie, na stałe.

THREAD Pamięć dla zmiennej jest przydzielana oddzielnie dla każdego wątku. Również, w sposób niejawny nadaje atrybut STATIC lokalnym zmiennym procedury.

AUTO Określa, że zmienna nie posiada wartości początkowej.

PRIVATE Powoduje, że zmienna nie jest widoczna poza modułem zawierającym metody klasy CLASS, w której została zadeklarowana. Dotyczy tylko zmiennych użytych w deklaracjach klas.

PROTECTED Powoduje, że zmienna nie jest widoczna poza metodami klasy bazowej, w której została zadeklarowana, i klas dziedziczących. Dotyczy tylko zmiennych użytych w deklaracjach klas.

PSTRING deklaruje łańcuch znaków, gdzie pierwsza pozycja jest zajmowana przez bajt zawierający długość łańcucha. Pamięć przydzielana na daną typu PSTRING jest inicjowana z zerową długością, o ile nie został określony atrybut AUTO.

PSTRING jest zgodny z łańcuchowym typem danych stosowanym w języku Pascal oraz z typem „LSTRING" wykorzystywanym przez Btrieve Record Manager. Rozmiar pamięci wykorzystywanej przez CSTRING jest stałej długości, na początku zawsze jest umieszczany bajt zawierający wartość odpowiadającą aktualnej długości łańcucha. PSTRING jest wewnętrznie konwertowany na STRING dla operacji wykonywanych w trakcie działania programu. Typu PSTRING używamy, jeśli chcemy zachować kompatybilność z zewnętrznymi plikami, czy procedurami.

W uzupełnieniu do jawnej deklaracji, wszystkie dane typu PSTRING są niejawnie deklarowane jako tablica PSTRING(1),DIM(*długość_łańcucha*). Umożliwia to na dostęp do każdego znaku łańcucha PSTRING poprzez adresowanie go jako elementu tablicy. Jeśli PSTRING posiada dodatkowo atrybut DIM, niejawna deklaracja tablicy jest ostatnim (opcjonalnym) wymiarem tablicy (na prawo od określonego w atrybucie DIM jawnego wymiaru).

Istnieje również możliwość bezpośredniego adresowania wielu znaków w łańcuchu PSTRING poprzez zastosowanie techniki „string slicing – fragmentowanie łańcucha". Przeprowadza ona podobne działania, co funkcja SUB, jest jednak bardziej elastyczna i efektywna (jednakże nie zapewnia kontroli zakresu). Jej większa elastyczność wynika z tego, że "fragment łańcucha" może być zastosowany po obu stronach instrukcji przypisania, a funkcji SUB możemy używać tylko po stronie źródłowej. Większa efektywność jest spowodowana mniejszym zużyciem pamięci.

Jeśli chcemy uzyskać "fragment łańcucha", numer jego pierwszego i ostatniego znaku oddzielamy od siebie znakiem dwukropka (:) i umieszczamy w indeksie niejawnej tablicy ([]) naszego łańcucha PSTRING. Numery pozycji mogą być stałymi lub całkowitymi, wyrażeniami. Jeśli stosujemy nazwy zmiennych, przed i po znaku dwukropka musimy umieścić przynajmniej jedną spację, w przeciwnym razie kompilator potraktuje cały zapis jako pojedynczą zmienną z prefiksem.

Ponieważ PSTRING musi posiadać na wiodącej pozycji bajt przechowujący aktualną długość łańcucha, programista jest odpowiedzialny za jego aktualizację przy operacjach fragmentowania. Bajt zawierający długość łańcucha PSTRING jest adresowany jako zerowy (0) element tablicy (Jedyny przypadek, gdy Clarion pozwala na odwołanie do zerowego elementu tablicy). Z tego powodu prawidłowym zakresem dla danej typu PSTRING(30) jest zakres od 0 do 29. Stosowanie PSTRING może pociągnąć za sobą pozostawienie "śmieci" poza aktywną częścią łańcucha, z tego powodu nie zaleca się go dla zmiennych wchodzących w skład struktury GROUP.

Przykład:

```
Name        PSTRING(21)                          ! Deklaruje 21-bajtowe pole - 20 bajtów danych
OtherName PSTRING(21),OVER(Name)                 ! Deklaruje pole nałożone na pole Name
Contact     PSTRING(21),DIM(4)                   ! tablica 21-bajtowych pól - 80 bajtów danych
Company   PSTRING('TopSpeed Corporation')        ! łańcuch 21-bajtowy - 20 bajtów danych
Phone       PSTRING(@P(###)###-####P)            ! Deklaruje 14 bajtów - 13 bajtów danych
ExampleFile  FILE,DRIVER('Btrieve')              ! Deklaruje plik
Record          RECORD
NameField         PSTRING(21),NAME('LstringField')   ! Deklaruje z nazwą zewnętrzną
                . .
  CODE
    Name = 'Tammi'                               ! przypisanie wartości
    Name[5] = 'y'                                ! zmiana piątej litery
    Name[6] = 's'                                ! dodanie litery
    Name[0] = '<6>'                              ! zmiana bajtu długości
    Name[5:6] = 'ie'                             ! zmiana fragmentu -- piątej i szóstej litery
    Contact[1] = 'First'                         ! przypisanie wartości do pierwszego elementu
    Contact[1,2] = 'u'                           ! zmiana drugiego znaku pierwszego elementu
    Contact[1,2:3] = Name[5:6]                   ! przypisanie fragmentu do fragmentu
```

Niejawne tablice znaków oraz „string slicing”

W uzupełnieniu ich bezpośrednich deklaracji, wszystkie zmienne typu STRING, CSTRING oraz PSTRING posiadają niejawne deklaracje jako tablice, których elementami są pojedyncze znaki, a rozmiarem – długość łańcucha. Jest to równoważne zadeklarowaniu drugiej zmiennej jako:

```
StringVar  STRING(10)
StringArray  STRING(1), DIM(SIZE(StringVar)),OVER(StringVar)
```

Ta niejawna deklaracja tablicy umożliwia adresowanie każdego znaku łańcucha jako elementu tablicy, bez konieczności stosowania tej drugiej deklaracji.

Bajt długości łańcucha typu PSTRING jest adresowany jako zerowy (indeks równy 0) element tablicy, podobnie jak pierwszy bajt zmiennej typu BLOB (są to jedyne dwa przypadki, w których Clarion dopuszcza stosowanie zerowego elementu tablicy).

Jeśli łańcuch posiada dodatkowo atrybut DIM, niejawna deklaracja tablicy oznacza ostatni, dodatkowy wymiar tak zdefiniowanej tablicy. Na niejawnych wymiarach nie operuje funkcja MAXIMUM, w takich przypadkach należy stosować funkcję SIZE.

Mamy możliwość bezpośredniego adresowania wielu znaków w łańcuchu za pomocą techniki o nazwie „string slicing – fragmentowanie łańcucha”. Wykonuje ona kilka działań funkcjonalnie podobnych do działania funkcji SUB, z tym że o wiele bardziej elastycznych i efektywnych (nie przeprowadza jednak kontroli zakresów). Jest bardziej elastyczna, gdyż może być używana w instrukcjach przypisać zarówno ze strony źródła, jak i wyniku; funkcja SUB może być stosowana tylko od strony źródła. Jest bardziej efektywna, gdyż absorbuje mniejszą liczbę pamięci, niż przypisanie pojedynczego znaku bądź wywołanie funkcji SUB. Jeśli chcemy uzyskać fragment łańcucha („slice”), numer początkowego i końcowego znaku muszą być oddzielone od siebie znakiem dwukropka, a całość należy zamknąć w indeksie niejawnej tablicy, czyli w nawiasach kwadratowych ([]). Liczbami określającymi pozycje mogą być stałe i zmienne całkowite lub wyrażenia, których rezultatem są wartości całkowite. Jeśli stosujemy zmienne, musimy umieścić przynajmniej jedną spację odstępu pomiędzy nazwą zmiennej a znakiem dwukropka (zapobiega to zinterpretowaniu jej jako prefiksu).

Przykład:

```
Name        STRING(15)
CONTACT  STRING(15),DIM(4)
  CODE
    Name = ‘Tammi’                    ! przypisanie wartości
    Name[5] = ‘y’                     ! zmiana piątej litery
    Name[6] = ‘s’                     ! dodanie litery
    Name[0] = ‘<6>’                   ! obsługa bajtu długości
    Name[5:6] = ‘ie’                  ! zmiana fragmentu -- piątej i szóstej litery
    Contact[1] = ‘First’              ! przypisanie wartości do pierwszego elementu
    Contact[1,2] = ‘u’                ! zmiana drugiego znaku pierwszego elementu
    Contact[1,2:3] = Name[5:6]        ! przypisanie fragmentu do drugiej i trzeciej litery pierwszego elementu
```

Porównaj: STRING, CTRING, PSTRING, BLOB

DATE (data - czterobajtowa)

label **DATE** [,**DIM**( )] [,**OVER**( )] [,**NAME**( )] [,**EXTERNAL**] [,**DLL**] [,**STATIC**] [,**THREAD**] [,**AUTO**] [,**PRIVATE**] [,**PROTECTED**]

DATE Data - czterobajtowa.

```
Format:        rok          mm      dd
               |. . . . . . . . |. . . .  |. . . . |
Bity:          31            15       7       0
Zakres:        rok: 1 do 9999
               miesiąc: 1 do 12
               dzień: 1 do 31
```

DIM Rozmiar zmiennej wystepującej w postaci tablicy.

OVER Współdzielenie tego samego obszaru pamięci z inną zmienną.

NAME Określa alternatywną, „zewnętrzną" nazwę dla pola.

EXTERNAL Określa, że dana zmienna jest zdefiniowana, a pamięć dla niej zaalokowana, w zewnętrznej bibliotece. Nie dotyczy deklaracji FILE, QUEUE i GROUP.

DLL Określa, że zmienna jest zdefiniowana w bibliotece dynamicznej DLL. Występuje w połączeniu z atrybutem EXTERNAL.

STATIC Pamięć dla zmiennej jest przydzielona statycznie, na stałe.

THREAD Pamięć dla zmiennej jest przydzielana oddzielnie dla każdego wątku. Również, w sposób niejawny nadaje atrybut STATIC lokalnym zmiennym procedury.

AUTO Określa, że zmienna nie posiada wartości początkowej.

PRIVATE Powoduje, że zmienna nie jest widoczna poza modułem zawierającym metody klasy CLASS, w której została zadeklarowana. Dotyczy tylko zmiennych użytych w deklaracjach klas.

PROTECTED Powoduje, że zmienna nie jest widoczna poza metodami klasy bazowej, w której została zadeklarowana, i klas dziedziczących. Dotyczy tylko zmiennych użytych w deklaracjach klas.

DATE deklaruje datę - czterobajtową. Format ten jest zgodny z typem "DATE" stosowanym przez Btrieve Record Manager. Zmienna typu DATE stosowana w wyrażeniach numerycznych jest konwertowana na liczbę stanowiącą liczbę dni, które minęły od dnia 28 grudnia 1800 r. (standardowa data Clariona jest przechowywana zazwyczaj jako wartość typu LONG). Zakres prawidłowych standardowych dat Clariona to przedział od 1 stycznia 1801 r. do 31 grudnia 9999 r. Stosowanie dat leżących poza zakresem może doprowadzić do nieoczekiwanych rezultatów. Pola typu

DATE są przeznaczone do stosowania w wypadku konieczności zachowania kompatybilności z zewnętrznymi formatami plików.

Przykład:

```
DueDate      DATE                        ! Deklaruje pole daty
OtherDate    DATE,OVER(DueDate)          ! Deklaruje pole nałożone na pole daty
ContactDate  DATE,DIM(4)                 ! tablica 4 pól daty
ExampleFile  FILE,DRIVER('Btrieve')      ! Deklaruje plik
Record         RECORD
DateRecd         DATE,NAME('DateField')  ! Deklaruje z nazwą zewnętrzną
             . .
```

Porównaj: Standardowa data

TIME (czas - czterobajtowy)

label **TIME** [,**DIM**( )] [,**OVER**( )] [,**NAME**( )] [,**EXTERNAL**] [,**DLL**] [,**STATIC**] [,**THREAD**] [,**AUTO**] [,**PRIVATE**] [,**PROTECTED**]

TIME Czas - czterobajtowy.

```
Format:      hh      mm      ss      hs
             |. . . . |. . . . |. . . . |. . . . |
Bity:        31    23      15    7       0
Zakres:      godziny (hh): 0 do 23
             minuty (mm): 0 do 59
             sekundy (ss): 0 do 59
             sekundy/100: 0 do 99
```

DIM Rozmiar zmiennej wystepującej w postaci tablicy.

OVER Współdzielenie tego samego obszaru pamięci z inną zmienną.

NAME Określa alternatywną, „zewnętrzną” nazwę dla pola.

EXTERNAL Określa, że dana zmienna jest zdefiniowana, a pamięć dla niej zaalokowana, w zewnętrznej bibliotece. Nie dotyczy deklaracji FILE, QUEUE i GROUP.

DLL Określa, że zmienna jest zdefiniowana w bibliotece dynamicznej DLL. Występuje w połączeniu z atrybutem EXTERNAL.

STATIC Pamięć dla zmiennej jest przydzielona statycznie, na stałe.

THREAD Pamięć dla zmiennej jest przydzielana oddzielnie dla każdego wątku. Również, w sposób niejawny nadaje atrybut STATIC lokalnym zmiennym procedury.

AUTO Określa, że zmienna nie posiada wartości początkowej.

PRIVATE Powoduje, że zmienna nie jest widoczna poza modułem zawierającym metody klasy CLASS, w której została zadeklarowana. Dotyczy tylko zmiennych użytych w deklaracjach klas.

PROTECTED Powoduje, że zmienna nie jest widoczna poza metodami klasy bazowej, w której została zadeklarowana, i klas dziedziczących. Dotyczy tylko zmiennych użytych w deklaracjach klas.

TIME deklaruje czterobajtową zmienną reprezentującą czas. Format ten jest zgodny z formatem „TIME” stosowanym przez Btrieve Record Manager. Dana typu TIME użyta w wyrażeniu numerycznym jest konwertowana do liczby setnych sekundy, które upłynęły od północy - standardowy czas Clariona (Clarion Standard Time)

przechowywana w formacie LONG. Pola typu TIME powinny być stosowane w celu zachowania kompatybilności z zewnętrznymi plikami i procedurami.

Przykład:

```
ChkoutTime   TIME                          ! Deklaruje pole czasu
OtherTime    TIME,OVER(CheckoutTime)       ! Deklaruje pole nałożone na pole czasu
ContactTime  TIME,DIM(4)                   ! tablica 4 pól czasu
ExampleFile  FILE,DRIVER('Btrieve')        ! Deklaruje plik
Record         RECORD
TimeRecd         TIME,NAME('TimeField')    ! Deklaruje z nazwą zewnętrzną
             . .
```

Porównaj: Standardowy czas

Specjalne typy danych

ANY (dowolny prosty typ danych)

label **ANY** [,**DIM**( )] [,**OVER**( )] [,**NAME**( )] [,**EXTERNAL**] [,**DLL**] [,**STATIC**] [,**THREAD**] [,**PRIVATE**] [,**PROTECTED**]

ANY	Zmienna mogąca zawierać wartość dowolnego typu (numeryczną lub łańcuchową) lub referencję do dowolnego prostego typu danych.
NAME	Określa alternatywną, „zewnętrzną" nazwę dla pola.
EXTERNAL	Określa, że dana zmienna jest zdefiniowana, a pamięć dla niej zaalokowana, w zewnętrznej bibliotece. Nie dotyczy deklaracji FILE, QUEUE i GROUP.
DLL	Określa, że zmienna jest zdefiniowana w bibliotece dynamicznej DLL. Występuje w połączeniu z atrybutem EXTERNAL.
STATIC	Pamięć dla zmiennej jest przydzielona statycznie, na stałe.
THREAD	Pamięć dla zmiennej jest przydzielana oddzielnie dla każdego wątku. Również, w sposób niejawny nadaje atrybut STATIC lokalnym zmiennym procedury.
PRIVATE	Powoduje, że zmienna nie jest widoczna poza modułem zawierającym metody klasy CLASS, w której została zadeklarowana. Dotyczy tylko zmiennych użytych w deklaracjach klas.
PROTECTED	Powoduje, że zmienna nie jest widoczna poza metodami klasy bazowej, w której została zadeklarowana, i klas dziedziczących. Dotyczy tylko zmiennych użytych w deklaracjach klas.

ANY deklaruje zmienną, która może zawierać dowolną wartość (numeryczną lub łańcuchową) bądź też referencję do dowolnego prostego typu danych. Umożliwia to stosowanie zmiennych typu ANY jako zmiennych "ogólnego" typu danych. Zmienna typu ANY może zostać zadeklarowana wewnątrz struktury CLASS, GROUP lub QUEUE. Nie można jej natomiast zadeklarować wewnątrz struktury FILE lub też w roli atrybutu USE dowolnej kontroli okna bądź raportu.

Gdy zmienna ANY jest *celem* instrukcji prostego przypisania (*cel = źródło*) otrzymuje ona wartość wyrażenia *źródło*. Zmienna ANY używa typu REAL jako typu bazowego dla operacji numerycznych, co oznacza utratę precyzji przy operacji przypisania wartości DECIMAL o większej niż 14 liczbie cyfr znaczących. Gdy zmienna ANY jest *celem* instrukcji przypisania referencyjnego (*cel &= źródło*) rejestruje ona referencję do zmiennej *źródło*. Nie jest możliwe przekazywanie zmiennych ANY w roli parametru-zmiennej (poprzez adres) jeśli procedura nie została prototypowana z odpowiednim parametrem o jieokreślonym typie (*?) i przekazywana wartość ANY zawiera referencję.

Jeśli zmienna ANY jest zadeklarowana wewnątrz kolejki QUEUE, istnieje kilka uwarunkowań, które muszą zostać spełnione. Wynika to z wewnętrznej reprezentacji wartości typu ANY i jej polimorficznego charakteru.

- Musimy wyczyścić (za pomocą CLEAR) kolejkę QUEUE albo przypisać referencyjnie NULL do zmiennej ANY (ZmiennaAny &= NULL) przed dodaniem nowego elementu do kolejki QUEUE. Gdy zmienna ANY kolejki QUEUE posiada przypisaną wartość (proste przypisanie ZmiennaAny = JakaśWartość) kolejne proste przypisanie może nadać jej nową wartość. Oznacza to, że poprzednia wartość jest zwalniana i zastępowana przez tą nową wartość. Jeśli pierwsza wartosc została już dołączona do kolejki QUEUE, odpowiedni jej element „wskazuje" na wartość już nieistniejącą. Gdy zmienna ANY kolejki QUEUE otrzymuje referencyjne przypisanie zmiennej (ZmiennaAny &= Jakaś zmienna), kolejne przypisanie referencyjne przypisuje jej nową zmienną. Oznacza to, że poprzedni „wskaźnik" jest zwalniany i zastępowany przez ten nowy „wskaźnik". Jeśli pierwsza referencja została już dodana do kolejki, odpowiedni jej element „wskazuje" na nie istniejący już „wskaźnik". W obu tych scenariuszach musimy stosować czyszczenie (CLEAR) kolejki QUEUE lub referencyjne przypisanie wartości NULL do zmiennej typu ANY przed dodaniem nowego elementu do kolejki QUEUE - w celu zapobieżenia sytuacji, w której w kolejce będą przechowywane „bzdurne" dane.
- Musimy wyczyścić (za pomocą CLEAR) kolejkę QUEUE albo przypisać referencyjnie NULL do zmiennej ANY (ZmiennaAny &= NULL) przed usunięciem elementu z kolejki QUEUE. Jak zostało już wyjaśnione powyżej, zmienna ANY zarządza swoim własnym obszarem danych, gdzie przechowuje wartości lub „wskaźniki" do zmiennych. Czyszczenie zmiennej ANY jest wymagane w celu zapobieżenia blokowaniu pamięci przez „zagubione" referencje.

Przykład:

```
MyQueue    QUEUE
AnyField     ANY                          ! Deklaruje zmienną zawierającą dowolną wartość
Type         STRING(1)
           END
DueDate  DATE                             ! Deklaruje pole daty
  CODE
    MyQueue.AnyField = 'TopSpeed'         ! przypisanie łańcucha
    MyQueue.Type = 'S'                    ! dana pełniąca rolę flagi
    ADD(MyQueue)
    CLEAR(MyQueue)                        ! wyczyszczenie referencji
    MyQueue.AnyField &= DueDate           ! przypisanie referencyjne DATE
    MyQueue.Type = 'R'                    ! dana pełniąca rolę flagi
    ADD(MyQueue)
    MyQueue.AnyField &= NULL              ! przypisanie referencyjne NULL w celu wyczyszczenia
    LOOP X# = RECORDS(MyQueue) TO 1 BY -1 ! przetwarzanie kolejki QUEUE
      GET(MyQueue,X#)
      ASSERT(~ERRORCODE())
      CASE MyQueue.Type
      OF 'S'
        DO StringRoutine
      OF 'R'
        DO ReferenceRoutine
      END
      MyQueue.AnyField &= NULL            ! przypisanie referencyjne NULL przed usunięciem
      DELETE(MyQueue)
      ASSERT(~ERRORCODE())
    END
```

Porównaj: Instrukcje prostego przypisania, Instrukcje przypisania referencyjnego

LIKE (dziedziczony typ danych)

new declaration **LIKE**(*like declaration*) [,**DIM**( )] [,**OVER**( )] [,**PRE**( )] [,**NAME**( )] [,**EXTERNAL**] [,**DLL**] [,**STATIC**] [,**THREAD**] [,**BINDABLE**]

LIKE	Deklaruje zmienną, która typ dziedziczy od innej zmiennej.
new declaration	Etykieta nowej zmiennej.
like declaration	Etykieta już zdefiniowanej zmiennej, której definicja ma zostać zastosowana. Może to być dowolny prosty typ danych, referencja na taki typ (za wyjątkiem &STRING) lub etykieta grupy GROUP bądź kolejki QUEUE.
DIM	Rozmiar zmiennej wystepującej w postaci tablicy.
OVER	Współdzielenie tego samego obszaru pamięci z inną zmienną lub strukturą.
PRE	Deklaruje prefiks nazwy dla zmiennych struktury *new declaration* (jeśli *like declaration* jest złożoną struktura danych). Nie jest to wymagane, o ile nie używamy *new declaration* w kontekście składni wymagającej kwalifikowania pól właśnie poprzez prefiks.
NAME	Określa alternatywną, „zewnętrzną” nazwę dla pola.
EXTERNAL	Określa, że dana zmienna jest zdefiniowana, a pamięć dla niej zaalokowana, w zewnętrznej bibliotece. Nie dotyczy deklaracji FILE, QUEUE i GROUP.
DLL	Określa, że zmienna jest zdefiniowana w bibliotece dynamicznej DLL. Występuje w połączeniu z atrybutem EXTERNAL.
STATIC	Pamięć dla zmiennej jest przydzielona statycznie, na stałe.
THREAD	Pamięć dla zmiennej jest przydzielana oddzielnie dla każdego wątku. Również, w sposób niejawny nadaje atrybut STATIC lokalnym zmiennym procedury.
BINDABLE	Umożliwia stosowanie wszystkich zmiennych grupy w wyrażeniach dynamicznych.

LIKE informuje kompilator, że definicja *new declaration* ma zostać oparta na definicji *like declaration*, włączając w to wszystkie jej atrybuty. Jeśli oryginalna *like declaration* ulegnie zmianie, pociąga to za sobą zmianę *new declaration*. Deklaracja *new declaration* może stosować atrybuty DIM i OVER. Jeśli *like declaration* posiada atrybut DIM, *new declaration* automatycznie staje się tablicą. Jeśli dołączymy atrybut DIM do *new declaration* staje się ona tablicą lub uzyskuje dodatkowy wymiar.

Atrybuty PRE oraz NAME mogą być stosowane, jeśli zachodzi taka potrzeba. Jeśli parametr *like declaration* posiada już takie atrybuty, *new declaration* może je odziedziczyć, co spowoduje powstanie błędu kompilacji. Możemy temu zapobiec określając atrybuty PRE lub NAME dla *new declaration* przykrywając w ten sposób dziedziczone atrybuty.

Jeśli *like declaration* jest strukturą QUEUE, LIKE nie tworzy nowej kolejki QUEUE, ponieważ *like declaration* jest po prostu traktowana jako grupa GROUP. Kolejka *like declaration* jest konwertowana do grupy *new declaration*. To samo zachodzi, gdy *like*

declaration jest strukturą typu RECORD. Podobnie, jeśli *like declaration* jest typu MEMO, *new declaration* otrzymuje typ STRING o maksymalnym rozmiarze MEMO.

Możemy stosować LIKE do tworzenia nowych egzemplarzy klasy CLASS. Z drugiej strony, proste zadeklarowanie nowego egzemplarza ze wskazaniem nazwy klasy CLASS jako typu danych, pociąga za sobą niejawne wykonanie LIKE. Dla obu typów deklaracji egzemplarzy niewłaściwe są atrybuty DIM, OVER, PRE oraz NAME; wszystkie pozostałe są dopuszczalne.

Przykład:

```
Amount        REAL                              ! definicja pola
QTDAmount     LIKE(Amount)                      ! zastosowanie tej samej definicji
YTDAmount     LIKE(QTDAmount)                   ! ponowne zastosowanie tej samej definicji
MonthlyAmts   LIKE(Amount),DIM(12)              ! zastosowanie tej samej definicji dla tablicy 12-elementowej
AmtPrPerson   LIKE(MonthlyAmts),DIM(10)         ! zastosowanie tej samej definicji dla tablicy
                                                !120-elementowej (12,10)
Construct     GROUP                             ! definicja grupy
Field1          LIKE(Amount)                    ! Construct.field1 - real
Field2          STRING(10)                      ! Construct.field2 - string(10)
              END
NewGroup    LIKE(Construct)                     ! definicja nowej grupy zawierającej
                                                ! NewGroup.field1 - real
                                                ! NewGroup.field2 - string(10)
MyQue   QUEUE                                   ! definicja kolejki
Field1    STRING(10)
Field2    STRING(10)
        END
MyGroup  LIKE(MyQue)                            ! definicja nowej grupy, takiej jak kolejka QUEUE
AmountFile  FILE,DRIVER('Clarion'),PRE(Amt)
Record        RECORD
Amount          REAL                            ! definicja pola
QTDAmount       LIKE(Amount)                    ! zastosowanie tej samej definicji
              . .
Animal   CLASS
Feed       PROCEDURE(short amount),VIRTUAL
Die        PROCEDURE
Age        LONG
Weight     LONG
         END
Cat      LIKE(Animal)                             ! nowy egzemplarz klasy Animal
Bird     Animal                                   ! nowy egzemplarz klasy Animal (jawny LIKE)
```

Porównaj: DIM, OVER, PRE, NAME, Kwalifikacja pól

Zmienne niejawne

Zmienne niejawne nie są deklarowane w sekcji deklaracji danych. Są one tworzone przez kompilator w momencie, gdy napotka on pierwsze wystąpienie tego typu zmiennej. Zmienne niejawne są automatycznie inicjowane wartością 0 lub łańcuchem pustym; nie ma możliwości nadania im wartości zanim zostaną użyte. Można śmiało przyjąć, że są równe 0 lub łańcuchem pustym dopóty, dopóki nie zostanie wykonana instrukcja przypisania im jakiejś wartości. Zmienne niejawne stosuje się zazwyczaj przy tworzeniu podzbiorów tablic, jako przełączniki prawda/fałsz, jako zmienne pośrednie w złożonych obliczeniach, jako liczniki pętli itp. itd.

Język Clarion obsługuje trzy typy zmiennych niejawnych:

#	Etykieta zakończona znakiem # oznacza niejawną zmienną typu LONG.
$	Etykieta zakończona znakiem $ oznacza niejawną zmienną typu REAL.
“	Etykieta zakończona znakiem “ oznacza niejawną zmienną typu STRING(32).

Dowolna zmienna niejawna użyta w obszarze deklaracji zmiennych globalnych (pomiędzy słowami kluczowymi PROGRAM i CODE) jest zmienną globalną. Jest jej przydzielana pamięć statyczna i jest ona widoczna z każdego punktu programu.

Dowolna zmienna niejawna użyta pomiędzy słowami kluczowymi MEMBER i PROCEDURE jest zmienną modułową. Jest jej przydzielana pamięć statyczna i jest ona widoczna dla każdej procedury zdefiniowanej w danym module.

Pozostałe zmienne niejawne są zmiennymi lokalnymi, którym jest przydzielana pamięć dynamiczna w stosie programu i które są widoczne tylko dla procedur.

Zmienne niejawne użyte w podprogramach ROUTINE mogą prowadzić do nieoczekiwanych błędów, zaleca się ich stosowanie z dużą ostrożnością i tylko wtedy, gdy jest to naprawdę konieczne.

Ponieważ kompilator tworzy zmienne niejawne w momencie ich wystąpienia, trudne jest ich śledzenie. Jest to koszt, który ponosimy w zamian za brak błędów kompilacji i kontroli typów. Na przykład, jeśli pomylimy się w nazwie już użytej wcześniej zmiennej niejawnej, kompilator nie wygeneruje błędu, a utworzy zamiast tego nową zmienna niejawną. Gdy program będzie sprawdzał wartość tej właściwej zmiennej niejawnej, okaże się, że będzie ona odbiegała od oczekiwanej; wykrycie tego typu błędu jest bardzo trudne i wymaga żmudnego sprawdzania kodu programu.

Przykład:

```
LOOP Counter# = 1 TO 10                          ! bezwarunkowy licznik powtórzeń
  ArrayField[Counter#] = Counter# * 2            ! inicjowanie tablicy
END
Address” = CLIP(City) & ‘, ‘ & State & ‘ ‘ & Zip  ! niejawny STRING(32)
MESSAGE(Address”)                                ! wykorzystywany do wyświetlenia wartości tymczasowej
Percent$ = ROUND((Quota / Sales),.1) * 100       ! niejawny REAL
MESSAGE(FORMAT(Percent$,@P%<<<.##P))             ! wykorzystywany do wyświetlenia wartości tymczasowej
```

Porównaj: Deklaracje danych i przydział pamięci

Zmienne referencyjne

Zmienna referencyjna zawiera referencję do innej deklaracji danych (swój "cel"). Zmienną referencyjną deklarujemy poprzez umieszczenie znaku ampersand a(&) przed nazwą typu danych (np. &BYTE, &FILE, &LONG, etc.) lub poprzez zadeklarowanie zmiennej typu ANY. W zależności od typu danych „celu" zmienna referencyjna może zawierać adres pamięci lub złożoną, wewnętrzną strukturę danych (opisującą lokalizację i typ danych „celu").

Prawidłowe deklaracje zmiennych referencyjncyh:

&BYTE	&SHORT	&USHORT	&LONG	&ULONG
&DATE	&TIME	&REAL	&SREAL	&BFLOAT8
&BFLOAT4	&DECIMAL	&PDECIMAL	&STRING	&CSTRING
&PSTRING	&GROUP	&QUEUE	&FILE	&KEY
&BLOB	&VIEW	&WINDOW	ANY	

Deklaracje &STRING, &CSTRING, &PSTRING, &DECIMAL i &PDECIMAL nie wymagają parametru określającego długość, gdyż wszelkie niezbędne informacje są zawarte w referencji. Oznacza to, że zmienna referencyjna &STRING może zawierać referencję na zmienną STRING dowolnej długości.

Zmienna referencyjna zadeklarowana jako &WINDOW może być wskazywać na strukturę APPLICATION, WINDOW lub REPORT. Referencje do tych struktur są wewnętrznie traktowane jako takie same przez bibliotekę runtime Clariona.

Zmienna ANY może zawierać referencję do dowolnego prostego typu danych i, jako taka, dotyczy wszystkich przedstawionych wcześniej, za wyjątkiem &GROUP, &QUEUE, &FILE, &KEY, &BLOB, &VIEW oraz &WINDOW.

Przypisanie referencyjne

Operator &= wykonuje instrukcję przypisania referencyjnego (*cel* &= *źródło*) powodującą umieszczenie referencji do *źródła* w zmiennej referencyjnej *cel*. Instrukcję przypisania referencyjnego możemy stosować także w wyrażeniach warunkowych.

Zmienna wbudowana NULL jest wykorzystywana do „wyzerowania" referencji umieszczonej w zmiennej referencyjnej oraz do sprawdzenia, czy w zmiennej takiej znajduje się jakaś referencja.

Stosowanie zmiennych referencyjnych

Użycie etykiety zmiennej referencyjnej jest dopuszczalne w każdym miejscu kodu, w którym jest widoczna zmienna, na którą ona wskazuje. Oznacza to, że dowolna instrukcja pobierająca etykietę okna WINDOW jako swój parametr może również pobierać etykietę zmiennej referencyjnej &WINDOW związanej z danym oknem.

W momencie użycia zmiennej referencyjnej w instrukcji kodu jest ona automatycznie "de-referencjonowana" w celu uzyskania wartości przechowywanej w zmiennej, na którą ona wskazuje. Jedynym wyjątkiem od tej reguły jest instrukcja przypisania referencyjnego. Na przykład:

```
Var1      LONG                ! Var1 jest typu LONG
RefVar1   &LONG               ! RefVar1 jest referencją na LONG
RefVar2   &LONG               ! RefVar2 także jest referencją na LONG
  CODE
    RefVar1 &= Var1           ! RefVar1 jest teraz referencją na Var1
    RefVar2 &= RefVar1        ! RefVar2 także jest teraz referencją na Var1
    RefVar1 &= NULL           ! RefVar1 nie wskazuje na nic
```

Deklaracje zmiennych referencyjnych

Zmienne referencyjne nie mogą być deklarowane wewnątrz struktur FILE i VIEW, można je natomiast deklarować w ramach struktur GROUP, QUEUE oraz CLASS.

Wydanie polecenia CLEAR(NazwaStruktury) dla struktury GROUP, QUEUE lub CLASS zawierającej zmienną referencyjną jest równoznaczne z przypisaniem NULL do tejże zmiennej.

Referencje globalne mogą być stosowane do umożliwienia odwoływania się do struktur innych wątków.

Referencje do nazw kolejek QUEUE i klas CLASS

W uzupełnieniu do typów danych wymienionych już wcześniej, istnieje możliwość definiowania referencji do nazwanych kolejek QUEUE (&*NazwaKolejki*) oraz nazwanych klas CLASS (&*NazwaKlasy*). Umożliwia to stosowanie referencji do przekazywania grupowych parametrów i odwoływania się do ich składników wewnątrz procedur, do których zostały przekazane.

Referencja do nazwanej kolejki QUEUE lub klasy CLASS może być referencją "z wyprzedzeniem". Oznacza to, że nazwana kolejka QUEUE lub klasa CLASS nie musi zostać zdefiniowana przed zmienną referencyjną, która na nią wskazuje. Jednakże zmienna referencyjna „z wyprzedzeniem" musi zostać określona zanim zostanie użyta w kodzie.

W przypadkach, gdzie zmienna referencyjna znajduje się wewnątrz struktury CLASS, referencja „z wyprzedzeniem" musi być określona przed zainicjowaniem obiektu; w przeciwnym wypadku stanie się pusta i nie do użycia. Ze stosowania zmiennych referencyjnych „z wyprzedzeniem" wynika kilka korzyści. Możemy uzyskać kolejkę QUEUE referencji, z których każda wskazuje na kolejkę QUEUE referencji, z których każda wskazuje na kolejkę QUEUE referencji, z których każda wskazuje na ..... Dla przykładu, możemy utworzyć kolejkę rodzeństwa wewnątrz struktury CLASS:

```
FamilyQ     QUEUE
Sibling       &FamilyClass        ! referencja z wyprzedzeniem
            END
FamilyClass CLASS
Family        &FamilyQ
            END
```

Inną korzyścią jest zdolność do prawdziwego ukrycia celów referencji PRIVATE struktury CLASS. Na przykład:

```
! plik dołączany (MyFile.inc) zawiera:
WidgetManager   CLASS,TYPE
WidgetList        &WidgetQ,PRIVATE
DoSomething       PROCEDURE
                  END

! inny plik dołączany (MyFile.CLW) zawiera:
  MEMBER('MyApp')
  INCLUDE('MyFile.INC')
WidgetQ   QUEUE,TYPE
Widget          STRING(40)
WidgetNumber LONG
             END
MyWidget WidgetManager                  ! aktualne powołanie egzemplarza musi
                                        ! następować po referencji z wyprzedzeniem
MyWidget.DoSomething   PROCEDURE
  CODE
    SELF.WidgetList &= NEW(WidgetQ)     ! prawidłowy kod
    SELF.WidgetList.Widget = 'Widget One'
    SELF.WidgetList.WidgetNumber = 1
    ADD(SELF.WidgetList)
```

W tym przykładzie referencje do SELF.WidgetList są prawidłowe tylko wewnątrz pliku MyFile.CLW.

Przykład:

```
App1   APPLICATION('Hello')
       END
App2   APPLICATION('Buenos Dias')
       END
AppRef  &WINDOW                         ! referencja do APPLICATION, WINDOW lub REPORT
Animal   CLASS
Feed       PROCEDURE(SHORT amount),VIRTUAL
Die        PROCEDURE
Age      LONG
Weight   LONG
       END
Carnivore CLASS(Animal),TYPE
Feed         PROCEDURE(Animal)
             END
Cat   CLASS(Carnivore)
Feed   PROCEDURE(SHORT amount),VIRTUAL
Potty   BYTE
      END
Bird          Animal                    ! egzemplarz klasy Animal
AnimalRef   &Animal                     ! referencja do klasy Animal
  CODE
    IF CTL:Language = 'Spanish'         ! jeśli język hiszpański
      AppRef &= App2                    ! referencja do okna w języku hiszpańskim
    ELSE
      AppRef &= App1                    ! w przeciwnym wypadku referencja do okna w języku angielskim
    END
    OPEN(AppRef)                        ! otwarcie okna
    IF SomeCondition
      AnimalRef &= Cat                  ! referencja do Cat
    ELSE
      AnimalRef &= Bird                 ! referencja do Bird
    END
    AnimalRef.Feed(10)
```

Porównaj: Instrukcje przypisania referencyjnego, CLASS, GROUP, QUEUE, ANY

Deklaracje danych i alokacja pamięci

Dane globalne, lokalne, statyczne i dynamiczne

Deklaracje danych pociągają za sobą automatyczną alokację pamięci, w której będą przechowywane ich wartości.

Global, Local, Static oraz Dynamic są terminami opisującymi typy alokacji pamięci.

Terminy "Global" i "Local" odnoszą się do zakresu "widzialności" danych (ich zasięgu):

- "Global" oznacza, że dana jest widoczna dla wszystkich procedur programu.
- "Local" oznacza, że dana jest widoczna w ograniczonym zakresie: tylko dla procedury lub podprogramu ROUTINE, bądź też dla określonego zestawu procedur w ramach jednego modułu kodu źródłowego.

Terminy "Static" i "Dynamic" odnoszą się do sposobu przechowywania pamięci przydzielonej dla danej:

- "Static" oznacza, że dana jest alokowana w pamięci, która nie jest zwalniana dopóty, dopóki program nie zakończy swego działania.
- "Dynamic" oznacza, że dana jest alokowana w pamięci wydzielonej na stosie programu. Pamięć dynamiczna jest zwalniana, gdy procedura lub podprogram ROUTINE, który dokonał jej alokacji, nie zakończy swego działania.

Sekcje deklaracji danych

Dane w programie Clarion mogą być deklarowane w kilku obszarach:

- W module PROGRAM, po słowie kluczowym PROGRAM,a przed instrukcją CODE. Jest to sekcja danych globalnych (**Global data**).
- W module MEMBER, po słowie kluczowym MEMBER, a przed pierwszą instrukcją PROCEDURE. Jest to sekcja danych modułu (**Module data**).
- W procedurze, po słowie kluczowym PROCEDURE, a przed instrukcją CODE. Jest to sekcja danych lokalnych (**Local data**).
- W podprogramie ROUTINE, po słowie kluczowym DATA, a przed instrukcją COD. Jest to sekcja danych podprogramu (**Routine Local data**).

Dane globalne (**Global data**) są widoczne dla wszystkich wykonywalnych instrukcji i wyrażeń procedur programu. Dane globalne zawsze są w zasięgu widzialności. Dane globalne są alokowane w pamięci statycznej i są dostępne dla wszystkich procedur w programie.

Dane modułu (**Module data**) są widoczne dla zestawu procedur określonego modułu MEMBER. Mogą one być przekazywane, w postaci parametrów, do procedur innych modułów MEMBER. Dane modułu pojawiają się w zasięgu, gdy dowolna procedura modułu zostaje wywołana. Dane modułu również są alokowane w pamięci statycznej.

Dane lokalne (**Local data**) są widoczne tylko w procedurze, w której zostały zadeklarowane lub też w lokalnie dziedziczonych metodach zadeklarowanych w procedurze. Dane lokalne pojawiają się w zasięgu widzialności w momencie

wywołania procedury, znikają z niego, gdy ma miejsce zwrot sterowania (instrukcja RETURN wywołana w sposób jawny lub niejawny). Dane lokalne mogą być przekazywane, w postaci parametrów, do innych procedur.

Dane lokalne są alokowane w pamięci dynamicznej. Pamięć dla zmiennych mniejszych od progu stosu (domyślnie 5K) jest przydzielana ze stosu programu, w przeciwnym wypadku jest przydzielana na stercie. Możemy wymusić zmianę stosując atrybut STATIC, w ten sposób wartość zmiennej jest zachowana pomiędzy różnymi wywołaniami procedury.

Deklaracje plików FILE są zawsze alokowane w pamięci statycznej (na stercie), nawet wtedy, gdy są umieszczone w sekcji danych lokalnych.

Alokacja pamięci dynamicznej dla zmiennych lokalnych umożliwia zapewnienie pełnej rekurencyjności procedury, która przy każdym wywołaniu rejestruje nowe kopie swoich zmiennych lokalnych

Dane podprogramu (**Routine Local data**) są widoczne tylko w podprogramie ROUTINE, w którym zostały zadeklarowane. Mogą one być przekazywane w postaci parametrów do dowolnej procedury.

Dane podprogramu pojawiają się w zasięgu widzialności w chwili wywołania podprogramu, znikają – gdy zwraca on sterowanie instrukcją EXIT (wywołaną jawnie lub niejawnie).

Dane podprogramu są alokowane w pamięci dynamicznej. Pamięć dla zmiennych mniejszych od progu stosu (domyślnie 5K) jest przydzielana ze stosu programu, w przeciwnym wypadku jest przydzielana na stercie.

Podprogram ROUTINE posiada swoją własną przestrzeń nazw, tak więc etykiety użyte w sekcji danych podprogramu mogą duplikować etykiety stosowane w innym podprogramie, czy nawet w procedurze, do której dany podprogram należy. Zmienne podprogramu nie mogą posiadać atrybutów STATIC i THREAD.

Porównaj: PROGRAM, MEMBER, PROCEDURE, CLASS, Prototypy procedur, STATIC, THREAD

NEW (przydział pamięci sterty)

reference &= **NEW**(*datatype*)

reference Etykieta zmiennej referencyjnej.

NEW Tworzy nowy egzemplarz *datatype* na stercie.

datatype Etykieta wcześniej zadeklarowanej klasy CLASS lub kolejki QUEUE, bądź też deklaracja dowolnego prostego typu danych. Możemy zastosować zmienną w roli parametru *datatype* w celu uzyskania prawdziwie dynamicznych deklaracji.

Instrukcja **NEW** tworzy nowy egzemplarz *datatype* na stercie. NEW jest prawidłowa tylko po stronie źródłowej instrukcji przypisania referencyjnego. Pamięć alokowana przez NEW jest automatycznie inicjowana spacjami lub wartościami zerowymi. Musi ona być jawnie zwolniona za pomocą instrukcji DISPOSE (jeśli tego nie zrobimy, doprowadzimy do powstania zablokowanych „dziur" w pamięci, których już nie będzie można wykorzystać).

Przykład:

```
StringRef    &STRING                          ! referencja do dowolnej zmiennej STRING
LongRef      &LONG                            ! referencja do dowolnej zmiennej LONG
Animal   CLASS
Feed       PROCEDURE(short amount)
Weight     LONG
         END
AnimalRef   &Animal                           ! referencja do klasy Animal
NameQ    QUEUE
Name       STRING(30)
           END
QueRef    &NameQ                              ! referencja do kolejki QUEUE tylko z STRING(30)
  CODE
    AnimalRef &= NEW(Animal)                  ! tworzy nowy egzemplarz klasy Animal
    QueRef &= NEW(NameQ)                      ! tworzy nowy egzemplarz kolejki NameQ
    StringRef &= NEW(STRING(50))              ! tworzy nową zmienną STRING(50)
    X# = 35                                   ! przypisanie 35 do zmiennej, następnie
    StringRef &= NEW(STRING(X#))              ! użycie tej zmiennej do utworzenia STRING(35)
    LongRef &= NEW(LONG)                      ! utworzenie nowej zmiennej LONG
```

Porównaj: DISPOSE

DISPOSE (zwolnienie pamięci sterty)

DISPOSE(*reference*)

DISPOSE Zwalnia pamięć sterty zaalokowaną przez instrukcję NEW.

reference Etykieta zmiennej referencyjnej użytej wcześniej w instrukcji NEW. Referencja ta może mieć wartość NULL i nie pociągnie to za sobą złych skutków.

Instrukcja **DISPOSE** zwalnia pamięć sterty zaalokowana wcześniej za pomocą instrukcji NEW. Jeśli DISPOSE nie zostanie wywołane, pamięć nie jest zwracana systemowi operacyjnemu do powtórnego użycia. Z drugiej strony, jeśli zwolnimy za pomocą DISPOSE pamięć referencji *reference* będącej cały czas w użyciu (na przykład kolejki, której zawartość jest wyświetlana w liście LIST) doprowadzimy do powstania błędu GPF (General Proptection Failure).

DISPOSE(SELF) jest prawidłowym wywołaniem prowadzącym do usunięcia bieżącego obiektu. Jeśli je stosujemy, musi to być ostatnia instrukcja procedury, w przeciwnym wypadku następujące później referencje do obiektu doprowadzą do powstania błędów.

Przykład:

```
StringRef  &STRING                          ! referencja do dowolnej zmiennej STRING
Animal     CLASS,TYPE
Feed         PROCEDURE(short amount),VIRTUAL
Weight       LONG
           END
AnimalRef  &Animal                          ! referencja do klasy Animal
NameQ      QUEUE
Name         STRING(30)
           END
QueRef     &NameQ                           ! referencja do dowolnej kolejki QUEUE tylko z STRING(30)
  CODE
    AnimalRef &= NEW(Animal)                ! tworzy nowy egzemplarz klasy Animal
    DISPOSE(AnimalRef)                      ! zwalnia Animal
    QueRef &= NEW(NameQ)                    ! tworzy nowy egzemplarz kolejki NameQ
    DISPOSE(QueRef)                         ! zwalnia kolejkę
    StringRef &= NEW(STRING(50())           ! tworzy nową zmienną STRING(50)
    DISPOSE(StringRef)                      ! zwalnia STRING(50)
```

Porównaj: NEW

Wzorce formatowania

Wzorce formatowania umożliwiają przechowywanie i wyświetlanie wartości w różnych postaciach.

Istnieje siedem typów wzorców: numeryczne i walutowe, notacji naukowej, łańcuchowe, daty, czasu, szablonowe, klawiszowe.

Wzorce numeryczne i walutowe

@**N** [*currency*] [*sign*] [f*ill*] *size* [*grouping*] [*places*] [*sign*] [*currency*] [**B**]

@N Wszystkie wzorce numeryczne i walutowe zaczynają się na @N.

currency Znak dolara ($) lub dowolna stała łańcuchowa zamknięta w znakach tyldy (~). Jeśli znak waluty poprzedza indykator znaku *sign* oraz nie występuje indykator *fill*, symbol *currency* "pływa" po lewej stronie cyfry znajdującej się w najwyższej pozycji. W przypadku, gdy indykator *fill* występuje, symbol *currency* pozostaje w stałej pozycji, po lewej stronie liczby. Jeśli indykator *currency* następuje za *size* oraz *grouping*, pojawia się na końcu liczby.

sign Określa sposób wyświetlania liczb ujemnych. Jeśli znak łącznika (-) poprzedza indykatory *fill* i *size*, liczby ujemne są poprzedzane znakiem minus. Jeśli znak łącznika (-) następuje po indykatorach *size*, *grouping*, *places*, czy *currency*, liczby ujemne mają na końcu znak minus. Jeśli w obu pozycjach umieścimy nawiasy, liczby ujemne będą ujmowane w nawiasy. W celu uniknięcia niejasności, ujemny znak *sign* umieszczany na końcu liczby powinien występować zawsze przy określonym indykatorze *grouping*.

fill Określa występowanie zer, spacji lub znaków asterisk (*) we wszystkich wiodących pozycjach, w których występuje cyfra zero, wymuszając przy tym ignorowanie *grouping*. Jeśli *fill* zostanie pominięte, wiodące zera są ukrywane.

0 (zero)	Wiodące zera
_ (podkreślenie)	Wiodące spacje
***** (asterisk)	Wiodące znaki asterisks

size Parametr *size* jest wymagany do określenia całkowitej liczby znaczących cyfr do wyświetlenia, włączając w to liczbę cyfr w *places i* wszystkie znaki formatujące.

grouping Symbol grupujący, inny niż przecinek (domyślny). Może pojawiać się po prawej stronie indykatora *size* w celu określenia trzycyfrowego separatora grupy. W celu uniknięcia niejasności, występujący w roli indykatora *grouping* znak łącznika (-) wymaga określenia indykatora *sign*.

. (kropka)	Grupujące kropki
- (łącznik)	Grupujące łączniki (minusy)
_ (podkreślenie)	Grupujące spacje

places Określa symbol oddzielający część dziesiętną i liczbę cyfr części dziesiętnej. Liczba cyfr części dziesiętnej musi być mniejsza od *size*. Separatorem części dziesiętnej może być kropka (.), grave accent (‘) (domyślnie pociąga za sobą *grouping* w postaci kropki), bądź też litera “v” (używana tylko w przypadku przechowywania w postaci łańcucha STRING – nie przy wyświetlaniu).

. (kropka)	Znak kropki
‘ (grave accent)	Znak przecinka
v	Brak separatora dziesiętnego

B Powoduje brak wyświetlania liczby, jeśli jej wartość jest równa zero.

Wzorce numeryczne i walutowe formatują wartości numeryczne wyświetlane w oknach lub drukowane w raportach. Jeśli wartość liczby jest większa niż maksymalna wartość, którą dopuszcza wzorzec, we wszystkich pozycjach są wyświetlane znaki (#).

Przykład:

Numerycznie	*Rezultat*	*Format*
@N9	4,550,000	Dziewięć cyfr, grupowanie przecinkiem(domyślnie)
@N_9B	4550000	Dziewięć cyfr, bez grupowania, wiodące spacje, jeśli zero
@N09	004550000	Dziewięć cyfr, wiodące zero
@N*9	***45,000	Dziewięć cyfr, wypełnianie gwiazdkami, grupowanie przecinkami
@N9_	4 550 000	Dziewięć cyfr, grupowanie spacjami
@N9.	4.550.000	Dziewięć cyfr, grupowanie kropkami

Formaty dziesiętne

@N9.2	4,550.75	Dwie pozycje dziesiętne, separator dziesiętny w postaci kropki
@N_9.2B	4550.75	Dwie pozycje dziesiętne, separator dziesiętny w postaci kropki, bez grupowania, puste - jeśli zero
@N_9‘2	4550,75	Dwie pozycje dziesiętne, separator dziesiętny w postaci przecinka
@N9.‘2	4.550,75	Separator dziesiętny w postaci przecinka, grupowanie kropkami
@N9_‘2	4 550,75	Separator dziesiętny w postaci przecinka, grupowanie spacjami,

formaty ze znakiem

@N-9.2B	-2,347.25	wiodący znak minus, puste - jeśli zero
@N9.2-	2,347.25-	znak minus na końcu
@N(10.2)	(2,347.25)	liczba ujemna zamykana w nawiasy

formaty walutowe

@N$9.2B	$2,347.25	wiodący znak dolara, puste - jeśli zero
@N$10.2-	$2,347.25-	wiodący znak dolara, znak minus na końcu
@N$(11.2)	$(2,347.25)	wiodący znak dolara, liczba ujemna zamykana w nawiasy

międzynarodowe formaty walutowe

```
@N12_‘2~ F~ 1     5430,50 F      France
@N~L. ~12‘ L.     1.430.050      Italy
@N~Ł~12.2        Ł1,240.50       United Kingdom
@N~kr~12‘2       kr1.430,50      Norway
@N~DM~12‘2         DM1.430,50    Germany
@N12_‘2~ mk~       1 430,50 mk   Finland
@N12‘2~ kr~      1.430,50 kr     Sweden
```

wzorce przechowywania

```
Variable1  STRING(@N_6v2)      ! Deklaruje jako 6 bajtów przechowywanych bez kropki dziesiętnej
  CODE
    Variable1 = 1234.56                          ! przypisanie wartości, przechowuje ‘123456’ w pliku
    MESSAGE(FORMAT(Variable1,@N_7.2))    ! wyświetla z kropką dziesiętną: ‘1234.56’
```

Wzorce notacji naukowej

@**E***msn*[**B**]

@E Wszystkie wzorce notacji naukowej zaczynają się na @E.

m Określa całkowitą liczbę znaków.

s Określa znak separujący część dziesiętną i znak grupujący, jeśli wartość **n** jest większa od 3.

. (kropka)	kropka i przecinek
.. (kropka kropka)	kropka i kropka
‘ (grave accent)	przecinek i kropka
_.(podkreślenie kropka)	kropka i spacja

n Określa liczbę cyfr na lewo od kropki dziesiętnej.

B Powoduje brak wyświetlania liczby, jeśli jej wartość jest równa zero.

Wzorce notacji naukowej formatują liczby o bardzo dużych lub bardzo małych wartościach. Ich format to liczba dziesiętna pomnożona przez potęgę liczby dziesięć.

Przykład:

Wzorzec	Wartość	Rezultat
@E9.0	1,967,865	.20e+007
@E12.1	1,967,865	1.9679e+006
@E12.1B	0	
@E12.1	-1,967,865	-1.9679e+006
@E12.1	.000000032	3.2000e-008
@E12_.4	1,967,865	1 967.865e+003

Wzorce łańcuchowe

@**S** *length*

@S Wszystkie wzorce łańcuchowe zaczynają się na @S.

length Określa liczbę wyświetlanych znaków.

Wzorzec łańcuchowy opisuje niesformatowany łańcuch określonej długości *length*.

Przykład:

```
Name STRING(@S20)                ! 20-znakowe pole łańcuchowe
```

Wzorce daty

@**D***n* [*s*] [*direction* [*range*]] [**B**]

@D Wszystkie wzorce daty zaczynają się na @D.

n Określa format daty. Zakres dostępnych wartości to: 1 do 18. Wiodące zero (0) powoduje uzupełnianie znakiem zera jednocyfrowego numeru dnia i miesiąca.

s Znak separujący miesiąc, dzień i rok. Domyślnie (przy pominięciu) jest to slash (/).

. (kropka)	Separator w postaci kropki
‘ (grave accent)	Separator w postaci przecinków
- (łącznik)	Separator w postaci łączników
_ (podkreślenie)	Separator w postaci spacji

direction Prawy (>) lub lewy (<) nawias trójkątny określający kierunek “Intellidate” (> oznacza przyszłość, < oznacza przeszłość) dla parametru *range*. Prawidłowe tylko dla wzorców dat z dwucyfrowym rokiem.

range Stała całkowita z zakresu od 0 do 99 określająca stulecie “Intellidate” dla parametru *direction*. Prawidłowe tylko dla wzorców dat z dwucyfrowym rokiem. Jeśli pominiemy, domyślną wartością jest 80.

B Powoduje brak wyświetlania liczby, jeśli jej wartość jest równa zero.

Daty mogą być przechowywane w zmiennych numerycznych (zazwyczaj typu LONG) lub w polach typu DATE (kompatybilność z Btrieve) bądź też w postaci łańcuchów STRING zadeklarowanych ze wzorcem daty.

Datę przechowywaną w zmiennej numerycznej nazywamy standardową datą Clariona (Clarion Standard Date). Przechowywana wartość odpowiada liczbie dni, które minęły od 28 grudnia 1800 r. Wzorzec daty konwertuje wartość w jeden ze zdefiniowanych formatów.

Stulecie dla dat w dowolnym wzorcu z dwucyfrowym rokiem jest określane w oparciu o specjalną technikę “Intellidate”. Wzorce daty, dla których nie określono parametrów *direction* i *range* przyjmują, że daty leżą w zakresie następnych 20 lub poprzednich 80 lat. Parametry *direction* i *range* umożliwiają zmianę tych domyślnych wartości. Parametr *direction* określa, czy *range* odnosi się do przyszłości, czy też do przeszłości. Przeciwne *direction* powoduje otrzymanie przeciwnej wartości (100-*range)* dzięki czemu dowolny dwucyfrowy rok daje w rezultacie właściwe stulecie. Na przykład, wzorzec @D1>60 określa użycie właściwego stulecia dla każdego roku do 60 lat w przód i do 40 lat wstecz. Jeśli bieżącym rokiem jest 1996, wprowadzenie przez użytkownika daty “5/01/40” powoduje, że przyjmowany jest rok 2040, a wprowadzenie daty “5/01/60” – rok 1960.

Dla tych wzorców daty, w których miesiąc jest reprezentowany słownie, właściwe nazwy miesięcy są pobierane z pliku środowiskowego (.ENV).

Przykład:

Wzorzec	Format	Rezultat
@D1	mm/dd/yy	10/31/59
@D1>40	mm/dd/yy	10/31/59
@D01	mm/dd/yy	01/01/95
@D2	mm/dd/yyyy	10/31/1959
@D3	mmm dd, yyyy	OCT 31,1959
@D4	mmmmmmmmm dd, yyyy	October 31, 1959
@D5	dd/mm/yy	31/10/59
@D6	dd/mm/yyyy	31/10/1959
@D7	dd mmm yy	31 OCT 59
@D8	dd mmm yyyy	31 OCT 1959
@D9	yy/mm/dd	59/10/31
@D10	yyyy/mm/dd	1959/10/31
@D11	yymmdd	591031
@D12	yyyymmdd	19591031
@D13	mm/yy	10/59
@D14	mm/yyyy	10/1959
@D15	yy/mm	59/10
@D16	yyyy/mm	1959/10
@D17	Ustawienie Panelu sterowania Windows dla daty krótkiej	
@D18	Ustawienie Panelu sterowania Windows dla daty długiej	
Alternatywne separatory		
@D1.	mm.dd.yy	separator - kropka
@D2-	mm-dd-yyyy	separator - myślnik
@D5_	dd mm yy	separator - spacja
@D6‘	dd,mm,yyyy	separator - przecinek

Porównaj: Standardowa data, FORMAT, DEFORMAT, Pliki środowiskowe

Wzorce czasu

@T*n*[*s*][**B**]

@T Wszystkie wzorce czasu zaczynają się na @T.

n Określa format daty. Zakres dostępnych wartości to: 1 do 8. Wiodące zero (0) powoduje uzupełnianie znakiem zera jednocyfrowej godziny

s Znak separujący. Domyślnie pomiędzy godziną, minutą, sekundą pojawiają się znaki dwukropka (:). Alternatywą są następujące separatory:

. (kropka)	Separator w postaci kropki
‘ (grave accent)	Separator w postaci przecinków
- (łącznik)	Separator w postaci łączników
_ (podkreślenie)	Separator w postaci spacji

B Powoduje brak wyświetlania liczby, jeśli jej wartość jest równa zero.

Czas może być przechowywany w zmiennych numerycznych (zazwyczaj LONG) w polu TIME (kompatybilność Btrieve) lub w łańcuchu STRING zadeklarowanym ze wzorcem czasu.

Czas przechowywany w zmiennych numerycznych jest nazywany standardowym czasem Clariona (Standard Time). Przechowywana wartość jest liczbą setnych sekundy, które minęły od północy. Wzorzec konwertuje wartość do jednego z ośmiu formatów czasu.

Dla tych wzorców czasu, które zawierają dane łańcuchowe, są one pobierane z pliku środowiskowego (.ENV).

Przykład:

Wzorzec	Format	Rezultat
@T1	hh:mm	17:30
@T2	hhmm	1730
@T3	hh:mmXM	5:30PM
@T03	hh:mmXM	05:30PM
@T4	hh:mm:ss	17:30:00
@T5	hhmmss	173000
@T6	hh:mm:ssXM	5:30:00PM
@T7	krótki format czasu z Panelu sterowania	
@T8	długi format czasu z Panelu sterowania	
Alternatywne separatory		
@T1.	hh.mm	kropka
@T1-	hh-mm	myślnik
@T3_	hh mmXM	spacja
@T4‘	hh,mm,ss	przecinek

Porównaj: Standardowy czas, FORMAT, DEFORMAT, Pliki środowiskowe

Wzorce szablonowe

@**P**[*<*][*#*][*x*]**P**[**B**]

@P Wszystkie wzorce szablonowe zaczynają się na @P i kończą na P. Wielkość liter P musi być taka sama (obie – „P” lub obie - „p”).

< Określa, że w danej pozycji występuje cyfra lub puste miejsce, w przypadku zera.

Określa, że w danej pozycji występuje cyfra

x Reprezentuje opcjonalne znaki wyświetlania. Znaki te pojawiają się w finalnym łańcuchu.

P Końcowy znak wzorca szablonowego.

B Powoduje brak wyświetlania liczby, jeśli jej wartość jest równa zero.

Wzorce szablonowe zawierają opcjonalne pozycje cyfrowe oraz opcjonalne znaki edycyjne. Dowolny znak różny od < i # jest traktowany jako znak edycyjny mogący być użyty w łańcuchu wzorca.

Ograniczniki @P i P muszą być tej samej wielkości. Duża litera “P” może być użyta w roli znaku edycyjnego, wtedy jednak ogranicznikami muszą być małe “p”; zachodzi oczywiście również sytuacja odwrotna.

Wzorce szablonowe nie rozpoznają znaków kropki dziesiętnej w celu umożliwienia używania go w roli znaku edycyjnego. Z tego powodu wartość formatowana przez wzorzec szablonowy powinna być wartością całkowitą. Użycie kropki dziesiętnej spowoduje, że w rezultacie zostanie umieszczona tylko część całkowita liczby.

Przykład:

Wzorzec	Wartość	Rezultat
@P###-##-####P	215846377	215-84-6377
@P<#/##/##P	103159	10/31/59
@P(###)###-####P	3057854555	(305)785-4555
@P###/###-####P	7854555	000/785-4555
@p<#:##PMp	530	5:30PM
@P<#' <#”P	506	5' 6"
@P<#lb. <#oz.P	902	9lb. 2oz.
@P4##A-#P	112	411A-2
@PA##.C#P	312.45	A31.C2

Wzorce klawiszowe

@**K**[@][*#*][*<*][*x*][**][*?*][*^*][*_*]*[|]*[**K**[**B**]

@K Wszystkie wzorce klawiszowe zaczynają się na @K i kończą na K. Wielkość liter K musi być taka sama (obie – „K” lub obie - „k”).

@ Powoduje stosowanie tylko wielkich lub tylko małych liter alfabetu.

Cyfra z zakresu 0 do 9.

< Cyfra lub spacja jeśli w tej pozycji znajdzie się zero wiodące.

x Opcjonalny, stały znak (dowolny znak, który może być wyświetlany). Znaki takie pojawiają się w łańcuchu wynikowym.

**** Określa, że następny znak jest znakiem podlegającym wyświetlaniu. Umożliwia to stosowanie we wzorcach takich znaków, jak @,#,<,\,?,^,_,|.

? Określa, że w danej pozycji może występować dowolny znak.

^ Określa, że w danej pozycji mogą występować tylko wielkie litery alfabetu.

_ Określa, że w danej pozycji mogą występować tylko małe litery alfabetu.

| Umożliwia operatorowi przerwanie wprowadzania w danym miejscu. W łańcuchu wynikowym znajdą się tylko wprowadzone do tej pory dane i stałe znaki.

K Końcowy ogranicznik wzorca.

B Powoduje brak wyświetlania liczby, jeśli jej wartość jest równa zero.

Wzorce klawiszowe mogą zawierać pozycje całkowite (# <), litery alfabetu (@ ^ _), znaki dowolne(?) i znaki wyświetlane na stałe. Dowolny znak różny od indykatora formatowania jest traktowany jako znak wyświetlany, który pojawia się w łańcuchu wynikowym.

Ograniczniki @K i K muszą być tej samej wielkości. Duża litera “K” może być użyta w roli znaku wyświetlanego, wtedy jednak ogranicznikami muszą być małe “k”; zachodzi oczywiście również sytuacja odwrotna.

Wzorce klawiszowe stosuje się zazwyczaj dla pól STRING, PSTRING oraz CSTRING w celu umożliwienia definiowania własnej kontroli i walidacji wprowadzanych danych. Stosowanie wzorców klawiszowych zawierających indykatory alfabetyczne (@ ^ _) dla numerycznych pól wprowadzania może prowadzić do nieoczekiwanych rezultatów.

Nieoczekiwane rezultaty otrzymamy również wtedy, gdy w polu z formatowaniem klawiszowym ustalimy tryb dopisywania (insert mode). Z tego powodu wzorce klawiszowe zawsze rejestrują dane w trybie nadpisywania (overwrite mode), niezależnie od tego, czy występuje atrybut INS, czy też nie.

Przykład:

Wzorzec	Wprowadzona wartość	Łańcuch wynikowy
@K###-##-####K	215846377	215-84-6377
@K#####|-####K	33064	33064
@K#####|-####K	330643597	33064-3597
@K<# ^^^ ##K	10AUG59	10 AUG 59
@K(###)@@@-##\@##K	305abc4555	(305)abc-45@55
@K###/?##-####K	7854555	000/785-4555
@k<#:##^Mk	530P	5:30PM
@K<#' <#"K	506	5' 6"
@K4#_#A-#K	1g12	41g1A-200

4 – DEKLARACJE EGZEMPLARZY

Złożone struktury danych

GROUP (złożona struktura danych)

```
label   GROUP( [ group ] ) [,PRE( )] [,DIM( )] [,OVER( )] [,NAME( )] [,EXTERNAL] [,DLL] [,STATIC]
                                [,THREAD] [,BINDABLE] [, TYPE] [,PRIVATE] [,PROTECTED]
          declarations
        END
```

GROUP	Grupa - złożona struktura danych.
group	Etykieta wcześniej zdefiniowanej grupy GROUP lub kolejki QUEUE, której strukturę dziedziczy bieżąca deklaracja. Może to być grupa GROUP lub kolejka QUEUE posiadająca atrybut TYPE.
PRE	Deklaruje prefiks dla zmiennych występujących wewnątrz struktury. Nie dotyczy struktury GROUP zadeklarowanej wewnątrz struktury FILE.
DIM	Rozmiar zmiennej w tablicy.
OVER	Współdzieli ten sam obszar pamięci z inną zmienną lub strukturą.
NAME	Określa alternatywną, „zewnętrzną” nazwę dla pola.
EXTERNAL	Określa, że dana zmienna jest zdefiniowana, a pamięć dla niej zaalokowana, w zewnętrznej bibliotece. Nie dotyczy deklaracji FILE, QUEUE i GROUP.
DLL	Określa, że zmienna jest zdefiniowana w bibliotece dynamicznej DLL. Występuje w połączeniu z atrybutem EXTERNAL.
STATIC	Pamięć dla zmiennej jest przydzielona statycznie, na stałe.
THREAD	Pamięć dla zmiennej jest przydzielana oddzielnie dla każdego wątku. Dodatkowo, w sposób niejawny, nadaje atrybut STATIC lokalnym zmiennym procedury.
BINDABLE	Powoduje, że wszystkie zmienne grupy mogą być wykorzystywane w budowaniu wyrażeń dynamicznych.
TYPE	Określa, że grupa GROUP jest definicją typu dla grup przekazywanych w postaci parametrów.
PRIVATE	Powoduje, że grupa i jej pola nie są widoczne poza modułem zawierającym metody klasy CLASS, w której została zadeklarowana. Dotyczy tylko grup użytych w deklaracjach klas.
PROTECTED	Powoduje, że zmienna nie jest widoczna poza metodami klasy bazowej, w której została zadeklarowana, i klas dziedziczących. Dotyczy tylko zmiennych użytych w deklaracjach klas.

declarations Deklaracje zmiennych wewnętrznych grupy.

Struktura **GROUP** umożliwia deklarowanie grupy wielu zmiennych, do których można odwoływać się poprzez pojedynczą etykietę. Może ona być wykorzystywana do organizowania zbiorów zmiennych, porównywania zbiorów zmiennych, przypisywania zestawów zmiennych za pomocą pojedynczej instrukcji. W dużych, skomplikowanych programach, struktura GROUP jest przydatna właśnie przy organizowaniu danych. Struktura GROUP musi być zakończona znakiem kropki lub instrukcją END.

Struktura grupy GROUP zadeklarowanej z parametrem *group* zaczyna się tak samo, jak struktura wskazanej grupy *group*; GROUP dziedziczy wszystkie pola grupy *group*. Grupa GROUP może zawierać również własne deklaracje *declarations,* które następują po polach odziedziczonych. Jeśli parametr *group* wskazuje na strukturę QUEUE lub RECORD, są dziedziczone tylko pola; nie jest dziedziczona funkcjonalność wymienionych struktur.

Gdy do grupy GROUP odwołujemy się w instrukcjach lub wyrażeniach jest ona traktowana jak łańcuch zestawiony ze wszystkich zmiennych wchodzących w jej skład. Struktura GROUP może być zagnieżdżana wewnątrz innych struktur danych, takich jak RECORD, czy inna grupa GROUP.

Ze względu na wewnętrzny format jej przechowywania zmienne numeryczne (inne niż DECIMAL) zadeklarowane w grupie, nie układają się prawidłowo podczas traktowania jej jako łańcucha. Z tego względu budowanie kluczy KEY w oparciu o grupy GROUP zawierające zmienne numeryczne może prowadzić do powstania błędów.

Grupa GROUP z atrybutem BINDABLE pozwala na zastosowanie wszystkich jej pól w wyrażeniach dynamicznych. Zawartość atrybutu NAME każdego z pól jest nazwą logiczną stosowaną w wyrażeniach dynamicznych. Jeśli nie występuje atrybut NAME, stosowana jest etykieta pola włącznie z prefiksem. W pliku .EXE jest alokowany obszar na nazwy zbindowanych zmiennych. Rozmiar programu staje się przez to większy i pociąga to za sobą zwiększenie zapotrzebowania na pamięć. Z tego względu atrybut BINDABLE powinien być stosowany tylko wtedy, gdy w wyrażeniach dynamicznych korzystamy praktycznie ze wszystkich pól elementu grupy.

Grupie GROUP z atrybutem TYPE nie jest przydzielana pamięć; stanowi on jedynie definicję dla grup, które będą przekazywane w postaci parametrów do procedur. Umożliwia to procedurze bezpośrednie adresowanie pól będących składnikami przekazanej grupy GROUP. Deklaracja parametru w instrukcji PROCEDURE może nadawać lokalny prefiks przekazywanej grupie GROUP, nie jest to jednak konieczne jeśli stosujemy kwalifikatory pól zamiast prefiksów. Na przykład, PROCEDURE(LOC:PassedGroup) deklaruje procedurę stosującą prefiks LOC: (wraz z indywidualnymi nazwami pól określonymi w definicji typu) do bezpośredniego adresowania pól składowych grupy GROUP przekazanej jako parametr.

Do elementów danych grupy GROUP posiadającej atrybut DIM odwołujemy się poprzez standardową składnię kwalifikacji pól w połączeniu z odpowiednimi indeksami identyfikującymi numer elementu w tablicy.

Procedury WHAT i WHERE umożliwiają dostęp do pól w oparciu o ich względną pozycję w ramach struktury GROUP.

Przykład:

```
   PROGRAM
PassGroup   GROUP,TYPE          ! definicja typu dla parametrów w postaci grupy
F1            STRING(20)        ! pierwsze pole
F2            STRING(1)         ! środkowe pole
F3            STRING(20)        ! ostatnie pole
            END
MAP
  MyProc1(PassGroup)            ! przekazuje grupę zdefiniowaną tak samo, jak PassGroup
END

NameGroup   GROUP               ! grupa
First         STRING(20)        ! pierwsze imię
Middle        STRING(1)         ! inicjał drugiego imienia
Last          STRING(20)        ! nazwisko
            END                 ! koniec deklaracji grupy

NameGroup2  GROUP(PassGroup)    !grupa dziedzicząca pola PassGroup
                                ! co daje w rezultacie pola NameGroup2.F1, NameGroup2.F2,
                                ! NameGroup2.F3 w niej zadeklarowane
            END

DateTimeGrp GROUP,DIM(10)       ! tablica daty I czasu
Date          LONG              ! referencjonowana jako DateTimeGrp[1].Date
StartStopTime LONG,DIM(2)       ! referencjonowana jako DateTimeGrp[1].Time[1]
            END                 ! koniec deklaracji grupy

FileNames   GROUP,BINDABLE                  ! grupa zbindowana
FileName      STRING(8),NAME('FILE')        ! dynamiczna nazwa: FILE
Dot           STRING('.')                   ! dynamiczna nazwa: Dot
Extension     STRING(3),NAME('EXT')         ! dynamiczna nazwa: EXT
            END

  CODE
    MyProc1(NameGroup)      ! wywołuje procedurę przekazując NameGroup jako parametr
    MyProc1(NameGroup2)     ! wywołuje procedurę przekazując NameGroup2 jako parametr

MyProc1   PROCEDURE(PassedGroup)    ! procedura odbierająca parametr GROUP
LocalVar    STRING(20)
  CODE
  LocalVar = PassedGroup.F1         ! przypisanie wartości z pierwszego pola
                                    ! przekazanego parametru do LocalVar
```

Porównaj: Kwalifikacja pól, WHAT, WHERE

CLASS (deklaracja obiektu)

```
label   CLASS( [ parentclass ] ) [,EXTERNAL] [,DLL] [,STATIC] [,THREAD] [,BINDABLE] [,MODULE( )]
                                 [, LINK( )] [, TYPE]
          [ data members and methods ]
        END
```

CLASS Obiekt zawierający właściwości (*data members*) i metody operujące na tych właściwościach (*methods*).

parentclass Etykieta wcześniej zdefiniowanej struktury CLASS, której dane i metody są dziedziczone w nowej klasie. Może to być struktura CLASS posiadająca atrybut TYPE.

EXTERNAL Określa, że dany obiekt jest zdefiniowany, a pamięć dla niego przydzielona, w zewnętrznej bibliotece.

DLL Określa, że obiekt jest zdefiniowany w bibliotece dynamicznej DLL. Występuje w połączeniu z atrybutem EXTERNAL.

STATIC Pamięć dla właściwości *data members'* jest przydzielona statycznie, na stałe.

THREAD Pamięć dla zmiennej jest przydzielana oddzielnie dla każdego wątku. Również, w sposób niejawny, nadaje atrybut STATIC lokalnym zmiennym procedury. Nie dotyczy TYPE.

BINDABLE Powoduje, że wszystkie zmienne klasy mogą być wykorzystywane w budowaniu wyrażeń dynamicznych.

MODULE Wskazuje moduł źródłowy, w którym umieszczono definicje metod danej struktury CLASS. Spełnia tę samą funkcję, co MODULE wewnątrz struktury MAP. Jeśli pominiemy ten parametr, definicje metod muszą znaleźć się w tym samym module źródłowym, co deklaracja klasy.

LINK Powoduje, że moduł źródłowy, w którym umieszczono definicje metod danej struktury CLASS jest automatycznie dołączany do listy linkowania tworzonej dla kompilatora. Eliminuje to konieczność „ręcznego" dołączania pliku kodu źródłowego do projektu.

TYPE Określa, że klasa CLASS jest jedynie definicją typu, a nie jednocześnie egzemplarzem obiektu danej klasy.

data members and methods Deklaracje danych i prototypy procedur. *Data members* mogą być jedynie deklaracjami danych analogicznymi do struktury GROUP, mogą zawierać one odniesienia do tej samej klasy (klasy rekurencyjne) Metody WHAT i WHERE umożliwiają dostęp do *data members* poprzez ich względną pozycję w strukturze CLASS.

Struktura **CLASS** deklaruje obiekt o określonych właściwościach *data members* i metodach *methods* operujących na danych. Struktura CLASS musi być zakończona znakiem kropki lub równoważną instrukcją END.

Dziedziczenie klas

Klasa CLASS zadeklarowana z parametrem *parentclass* tworzy klasę dziedziczącą *derived class*, która dziedziczy wszystkie pola *data members* i metody *methods* klasy *parentclass*. Klasa dziedzicząca *derived class* może posiadać również swoje własne

pola *data members* i metody *methods*. Wszystkie pola *data members* jawnie zadeklarowane w klasie dziedziczącej *derived class* powodują utworzenie nowych zmiennych, które nie mogą być deklarowane z takimi samymi etykietami, jako pola *data members* w klasie nadrzędnej *parentclass*.

Dowolna metoda *method* prototypowana w klasie dziedziczącej *derived class* z taką samą nazwą jak metoda *method* klasy nadrzędnej *parentclass* przykrywa metodę dziedziczoną, o ile obie posiadają taką samą listę parametrów. Jeśli listy parametrów są różne, jest tworzona w klasie dziedziczącej *derived class* metoda polimorficzna, która musi spełniać wszystkie reguły związane z przeciążaniem procedur.

Właściwości obiektu (enkapsulacja)

Każdy egzemplarz klasy, niezależnie od tego, czy jest to klasa bazowa, dziedzicząca, czy zadeklarowany egzemplarz którejś z nich, zawiera swój własny zestaw właściwości *data members* specyficznych tylko dla niego. Mogą to być właściwości prywatne lub publiczne. Jednakże istnieje tylko jedna kopia każdej dziedziczonej metody *method* (rezydująca w klasie, w której została zadeklarowana), która może być wywoływana przez dowolny egzemplarz klasy CLASS lub klasy dziedziczącej *derived class*.

Metody klasy CLASS posiadającej atrybut TYPE, nie mogą być wywoływane bezpośrednio (w postaci *NazwaKlasy.NazwaMetody*) – muszą być wywoływane jako metody egzemplarza takiej klasy: *Obiekt.NazwaMetody*.

Metody wirtualne (polimorfizm)

Jeśli występuje metoda *method* prototypowana w klasie dziedziczącej *derived class* o takiej samej nazwie, jak metoda *w* klasie nadrzędnej *parentclass* posiadająca atrybut VIRTUAL, musi ona także być prototypowana z atrybutem VIRTUAL.

Atrybut VIRTUAL w obu prototypach tworzy metody wirtualne umożliwiające metodom klasy nadrzędnej *parentclass* wywoływanie tak samo nazwanych, wirtualnych metod, klas podrzędnych *derived class*. Dzięki temu jest możliwe wykonywanie funkcji specyficznych dla klas dziedziczących *derived class,* które nie są określone dla klasy nadrzędnej *parentclass*.

Metody wirtualne klasy dziedziczącej mogą bezpośrednio wywoływać metody klasy nadrzędnej *parentclass* o takiej samej nazwie, poprzedzając tę nazwę słowem kluczowym PARENT (i znakiem kropki). Umożliwia to przyrostowe dziedziczenie, przy którym metoda klasy dziedziczącej wywołuje metodę klasy nadrzędnej w celu przeprowadzenia specyficznych dla niej funkcji oraz dodaje do tego część kodu specyficzną tylko dla niej samej.

Zasięg widzialności

Zasięg widzialności obiektu zależy od miejsca, w którym został zadeklarowany. Generalnie, zadeklarowany obiekt pojawia się w zasięgu po instrukcji CODE występującej po jego deklaracji. Obiekt powoływany dynamicznie (za pomocą funkcji NEW) znajduje się w zasięgu widzialności sekcji kodu, w której został powołany

Obiekt zadeklarowany jako:

- Dana globalna – znajduje się w zasięgu widzialności całej aplikacji.
- Dana modułu - znajduje się w zasięgu widzialności całego modułu.
- Dana lokalna - znajduje się w zasięgu widzialności procedury, z pewnymi wyjątkami …

Metody prototypowane w dziedziczonych deklaracjach klas CLASS wewnątrz sekcji danych lokalnych procedury są lokalnie dziedziczonymi metodami i współdzielą zasięg widzialności procedury dla deklaracji wszystkich danych lokalnych i podprogramów. Metody muszą być zadeklarowane w tym samym module kodu źródłowego, co procedura wewnątrz której została zadeklarowana klasa CLASS i muszą występować bezpośrednio po tej procedurze – muszą się pojawić po podprogramach ROUTINE i przed kolejną procedurą. Oznacza to, że deklaracje danych lokalnych procedury i podprogramy ROUTINE są widoczne i mogą być wykorzystywane wewnątrz kodu metod. Na przykład:

```
SomeProc        PROCEDURE
MyLocalVar      LONG
MyDerivedClass  CLASS(MyClass)              ! dziedziczna klasa z metodą wirtualną
MyProc            PROCEDURE,VIRTUAL
                  END
  CODE
    ! kod główny procedury SomeProc

! a tutaj podprogram procedury SomeProc
MyRoutine  ROUTINE
  ! kod podprogramu

! za nim występują metody MyDerivedClass:
MyDerivedClass.MyProc PROCEDURE
  CODE
    MyLocalVar = 10          ! MyLocalVar jest cały czas w zasięgu, możliwa do użycia
    DO MyRoutine             ! MyRoutine jest cały czas w zasięgu, możliwa do użycia

! dowolne pozostałe proceury tego samego modułu występuja tutaj, po metodach
! klasy dziedziczącej
```

Powoływanie obiektów

Egzemplarz klasy CLASS (obiekt) powołujemy po prostu nadając nazwę typowi CLASS lub też wykonując procedurę NEW w przypisaniu referencyjnym obiektu do zmiennej referencyjnej wskazującej na daną, nazwaną klasę. Nowy egzemplarz obiektu dziedziczy wszystkie pola *data members* i metody *methods* klasy, której jest egzemplarzem. Wszystkie atrybuty klasy CLASS, za wyjątkiem MODULE oraz TYPE, są prawidłowe w deklaracji egzemplarza.

Jeśli w powiązaniu z deklaracją CLASS nie występuje atrybut TYPE, oznacza to, że jest nie tylko definiowana sama klasa, ale automatycznie jest także powoływany jej egzemplarz; o nazwie takiej samej, jak nazwa klasy. Deklaracja klasy CLASS posiadająca atrybut TYPE nie tworzy obiektu będącego egzemplarzem danej klasy. Na przykład, następująca deklaracja CLASS definiuje klasę jako typ danych i automatycznie powołuje obiekt tego typu:

```
MyClass  CLASS          ! jednoczesna deklaracja typu danych I egzemplarza obiektu
MyField    LONG
MyProc     PROCEDURE
         END
```

podczas, gdy poniższa deklaracja powoduje zdefiniowanie jedynie typu, bez powoływania egzemplarza:

```
MyClass  CLASS,TYPE     ! to jest jedynie deklaracja typu danych
MyField    LONG
MyProc     PROCEDURE
         END
```

Jest preferowane bezpośrednie deklarowanie egzemplarzy obiektów jako typu CLASS zamiast deklarowania ich jako referencji na typ CLASS. Kod jest wówczas nieco szybszy i nie jest wymagane stosowanie funkcji NEW oraz DISPOSE w celu powoływania i usuwania obiektów. Z kolei zaletą użycia NEW i DISPOSE jest uzyskanie pełnej kontroli nad „czasem życia" obiektu. Na przykład:

```
MyClass     CLASS,TYPE
MyField       LONG
MyProc        PROCEDURE
              END
OneClass    MyClass          ! zadeklarowany egzemplarz obiektu, mniejszy I szbyszy
TwoClass    &MyClass         ! referencja do obiektu, musi stosować NEW i DISPOSE
  CODE
    ! jakiś kod wykonywalny
    TwoClass &= NEW(MyClass)   ! tu zaczyna się czas życia obiektu
    ! jakiś kod wykonywalny
    DISPOSE(TwoClass)          ! I trwa tylko do tego miejsca
    ! jakiś kod wykonywalny
```

Inną zaletą deklarowania obiektów jest możliwość nadawania im tych samych atrybutów (za wyjątkiem TYPE i MODULE), co klasie CLASS. Możemy na przykład zadeklarować egzemplarz obiektu z atrybutem THREAD, niezależnie od tego, czy klasa CLASS posiada taki atrybut, czy też nie.

Czas życia obiektu zależy od sposobu, w jaki jest on powoływany:

- Obiekt zadeklarowany w sekcji danych globalnych lub w sekcji danych modułu, jest powoływany w momencie wystąpienia pierwszej instrukcji CODE następującej po instrukcji PROGRAM. Obiekt taki jest usuwany, gdy aplikacja kończy swe działanie.
- Referencja do obiektu jest powoływana przez instrukcję NEW, a usuwana przez instrukcję DISPOSE.
- Obiekt zadeklarowany w sekcji danych lokalnych procedury, jest powoływany w momencie wystąpienia instrukcji CODE danej procedury (następującej po instrukcji PROCEDURE). Obiekt taki jest usuwany, gdy procedura kończy swe działanie.

Inicjalizacja danych (właściwości)

Polom *data members* prostych typów danych pamięć jest przydzielana automatycznie i są one inicjowane łańcuchami pustymi lub wartością zero (o ile nie posiadają atrybutu AUTO) w momencie, gdy obiekt pojawi się w zasięgu widzialności. Alokowana pamięć jest zwracana systemowi operacyjnemu do powtórnego użycia wtedy, gdy obiekt zniknie z zakresu widzialności.

Pola *data members* będące zmiennymi referencyjnymi nie otrzymują automatycznie przydziału pamięci i nie są inicjowane wartościami, gdy obiekt pojawi się w zasięgu widzialności. Musimy sami zadbać o wykonanie odpowiedniego przypisania referencyjnego lub instrukcji NEW. Zmienne referencyjne nie są automatycznie czyszczone, gdy obiekt znika z zasięgu widzialności – by to uzyskać, musimy zwolnić, za pomocą DISPOSE, wszystkie pola, które zostały zainicjowane za pomocą NEW i to zanim obiekt zniknie z zasięgu widzialności.

Konstruktory i destruktory

Metoda *method* klasy CLASS o nazwie "Construct" jest metodą-konstruktorem, która jest automatycznie wykonywana w momencie, gdy obiekt pojawia się w zasięgu widzialności, bezpośrednio po alokowaniu i inicjalizacji właściwości *data members*. Metoda "Construct" nie może otrzymywać żadnych parametrów i nie może być metodą wirtualną VIRTUAL. Jest możliwe również bezpośrednie wywoływanie konstruktora.

Jeśli obiekt jest egzemplarzem dziedziczącej klasy CLASS, przy czym klasa nadrzędna *parentclass* i klasa dziedzicząca posiadają konstruktory, a konstruktor klasy dziedziczącej nie posiada atrybutu REPLACE, wtedy konstruktorem wykonywanym automatycznie podczas powoływania obiektu jest konstruktor klasy nadrzędnej *parentclass*. Jeśli konstruktor klasy dziedziczącej posiada atrybut REPLACE, wtedy wykonywany jest tylko konstruktor klasy dziedziczącej (jeśli chcemy wywołać konstruktor klasy nadrzędnej, posługujemy się wówczas PARENT.Construct).

Metoda *method* klasy CLASS o nazwie "Destruct" jest metodą-destruktorem, która jest automatycznie wykonywana w momencie, gdy obiekt znika z zasięgu widzialności, bezpośrednio przed usuwaniem właściwości *data members*. Metoda "Destruct" nie może otrzymywać żadnych parametrów. Jest możliwe bezpośrednie wywoływanie destruktora.

Jeśli obiekt jest egzemplarzem dziedziczącej klasy CLASS, przy czym klasa nadrzędna *parentclass* i klasa dziedzicząca posiadają destruktory, a destruktor klasy dziedziczącej nie posiada atrybutu REPLACE, wtedy destruktorem wykonywanym automatycznie podczas usuwania obiektu jest destruktor klasy nadrzędnej *parentclass*. Jeśli destruktor klasy dziedziczącej posiada atrybut REPLACE, wtedy wykonywany jest tylko destruktor klasy dziedziczącej (jeśli chcemy wywołać destruktor klasy nadrzędnej, posługujemy się wówczas PARENT.Destruct).

Publiczne, prywatne i chronione (enkapsulacja)

Publiczne właściwości *data members* i metody *methods* klasy CLASS lub klasy dziedziczącej muszą być zadeklarowane zarówno bez atrybutu PRIVATE, jak i bez atrybutu PROTECTED. Właściwości i metody publiczne są widzialne (dostępne) dla wszystkich metod danej klasy, klas dziedziczących oraz kodu, w którego zasięgu widzialności obiekt się znajduje.

Prywatne właściwości *data members* i metody *methods* są deklarowane z atrybutem PRIVATE. Właściwości i metody prywatne są widzialne (dostępne) tylko dla metod klasy, w której zostały zdefiniowane oraz dla procedur znajdujących się w tym samym module kodu źródłowego.

Chronione właściwości *data members* i metody *methods* są deklarowane z atrybutem PROTECTED. Właściwości i metody chronione są widzialne (dostępne) tylko dla metod klasy, w której zostały zdefiniowane oraz dla metod klas dziedziczących.

Definicja metody

Definicja procedury PROCEDURE stanowiącej metodę *method* (jej kodu wykonywalnego, nie prototypu) jest zewnętrzna w stosunku do struktury CLASS. Definicja metody *method* albo musi się zaczynać nazwą klasy CLASS poprzedzającą etykietę procedury, albo zawierać nazwę klasy CLASS jako typ i SELF – jako parametr, w postaci pierwszego (wymaganego) parametru w liście parametrów przekazywanych do tej procedury.

Należy pamiętać, że w instrukcji PROCEDURE definiującej procedurę nadajemy etykiety wszystkim parametrom, które będą używane w kodzie metody. Ponieważ

etykieta klasy CLASS jest typem danych dla wymaganego pierwszego parametru, musimy stosować SELF jako etykietę przydzieloną naszej klasie. Na przykład, dla następujących deklaracji klasy:

```
MyClass   CLASS
MyProc      PROCEDURE(LONG PassedVar)        ! metoda otrzymuje 1 parametr
          END
```

możemy zdefiniować metodę MyProc PROCEDURE tak:

```
MyClass.MyProc PROCEDURE(LONG PassedVar)  ! poprzedź nazwą klasy nazwę metody
  CODE
```

Albo tak:

```
MyProc PROCEDURE(MyClass SELF, LONG PassedVar)   ! nazwa klasy jest typem
  CODE                                           ! danych pierwszego parametru
                                                 ! nazwanego SELF
```

Referencje do właściwości i metod obiektu w kodzie źródłowym

Referencji do właściwości *data members* klasy w kodzie źródłowym dokonujemy posługując się składnią kwalifikacji pól. W tym celu należy poprzedzić etykietę właściwości (pola) etykietą obiektu i oddzielić je od siebie znakiem kropki. Na przykład, dla następujących deklaracji klas:

```
MyClass     CLASS                ! bez TYPE, jest to także egzemplarz obiektu
MyField       LONG               ! w uzupełnieniu do deklaracji typu klasy
MyProc        PROCEDURE
            END
MyClass2    MyClass              ! deklaracja innego egzemplarza obiektu MyClass
```

możemy odwoływać się do pól MyField w następujący sposób:

```
MyClass.MyField = 10             ! referencja do obiektu MyClass
MyClass2.MyField = 10            ! referencja do obiektu MyClass2
```

Metody klasy mogą być wywoływane albo w oparciu o składnie kwalifikacji pól (etykieta metody jest poprzedzana etykietą obiektu i znakiem kropki), albo poprzez przekazanie etykiety klasy CLASS jako pierwszego (wymaganego) parametru procedury. Na przykład, dla następującej deklaracji klasy:

```
MyClass   CLASS
MyProc      PROCEDURE
          END
```

możemy wywołać metodę MyProc tak:

```
CODE
  MyClass.MyProc
```

lub tak:

```
CODE
MyProc(MyClass)
```

SELF i PARENT

Wewnątrz metod danej klasy a CLASS, do właściwości tej klasy i innych jej metod możemy się odwoływać poprzedzając ich etykiety słowem kluczowym SELF (po którym umieszczamy oczywiście znak kropki). Umożliwia to polom i metodom

uogólnienie referencji do aktualnego egzemplarza klasy, bez rozstrzygania problemu, czy jest to obiekt klasy nadrzędnej *parentclass*, klasy dziedziczącej *derived class*, czy dowolny egzemplarz jednej z nich.

Jest to także mechanizm pozwalający klasie nadrzędnej *parentclass* na wywoływanie metod wirtualnych klasy dziedziczącej *derived class*.

Na przykład, rozszerzając poprzedni przykład, odwołanie do MyField w metodzie MyClass.MyProc wygląda tak:

```
MyClass.MyProc    PROCEDURE
  CODE
    SELF.MyField = 10          ! przypisanie do właściwości bieżącego egzemplarza obiektu
```

Pola i metody klasy nadrzędnej mogą być bezpośrednio wywoływane z poziomu metod klasy dziedziczącej poprzez zastosowanie słowa kluczowego PARENT oddzielonego od nazwy pola lub metody znakiem kropki. Na przykład:

```
MyDerivedClass.MyProc PROCEDURE
  CODE
    ! jakiś kod wykonywalny
    PARENT.MyProc                    ! wywołanie metody klasy bazowej
    ! jakiś kod wykonywalny
```

Przykład:

```
! The ClassPrg.CLW file contains:
  PROGRAM
  MAP.                          ! MAP wymagane do pobrania BUILTINS.CLW
OneClass    CLASS               ! klasa bazowa
NameGroup   GROUP               ! referencja jako OneClass.NameGroup
First         STRING(20)        ! referencja jako OneClass.NameGroup.First
Last          STRING(20)        ! referencja jako OneClass.NameGroup.Last
            END

BaseProc    PROCEDURE(REAL Parm)                    ! deklaruje prototyp metody
Func        PROCEDURE(REAL Parm),STRING,VIRTUAL     ! deklaruje prototyp metody wirtualnej
Proc        PROCEDURE(REAL Parm),VIRTUAL            ! deklaruje prototyp metody wirtualnej
          END                                       ! koniec deklaracji klasy

TwoClass CLASS(OneClass),MODULE('TwoClass.CLW')     ! dziedziczy z OneClass
Func       PROCEDURE(LONG Parm),STRING              ! zastępuje OneClass.Func
Proc       PROCEDURE(STRING Msg,LONG Parm)          ! funkcjonalnie przeciążone
         END

ClassThree CLASS(TwoClass),MODULE('Class3.CLW')     ! dziedziczy z TwoClass
Func         PROCEDURE(<STRING Msg>,LONG Parm),STRING,VIRTUAL
Proc         PROCEDURE(REAL Parm),VIRTUAL
           END

ClassFour   ClassThree                              ! deklaruje egzemplarz ClassThree
ClassFive   ClassThree                              ! deklaruje egzemplarz ClassThree

  CODE
  OneClass.NameGroup = '|OneClass Method'           ! przypisuje wartości do każdego
  TwoClass.NameGroup = '|TwoClass Method'           ! egzemplarza NameGroup
  ClassThree.NameGroup = '|ClassThree Method'
  ClassFour.NameGroup = '|ClassFour Method'
  MESSAGE(OneClass.NameGroup & OneClass.Func(1.0))  ! wywołuje OneClass.Func
  MESSAGE(TwoClass.NameGroup & TwoClass.Func(2))    ! wywołuje TwoClass.Func
  MESSAGE(ClassThree.NameGroup & ClassThree.Func('|Call ClassThree.Func',3.0))
                                                    ! wywołuje ClassThree.Func
  MESSAGE(ClassFour.NameGroup & ClassFour.Func('|Call ClassFour.Func',4.0))
                                                    ! także wywołuje ClassThree.Func
  OneClass.BaseProc(5)                              ! BaseProc wywołuje OneClass.Proc & Func
```

```
  BaseProc(TwoClass,6)                                  ! BaseProc również wywołuje OneClass.Proc & Func
  TwoClass.Proc('Second Class',7)                       ! wywołuje TwoClass.Proc (przeciążoną)
  ClassThree.BaseProc(8)                                ! BaseProc wywołuje ClassThree.Proc & Func
  ClassFour.BaseProc(9)                                 ! BaseProc także wywołuje ClassThree.Proc & Func
  Proc(ClassFour,'Fourth Class',10)                     ! wywołuje TwoClass.Proc (przeciążoną)

OneClass.BaseProc  PROCEDURE(REAL Parm)                 ! definicja OneClass.BaseProc
  CODE
  MESSAGE(Parm & SELF.NameGroup &'|BaseProc executing|calling SELF.Proc Virtual method')
  SELF.Proc(Parm)                                       ! wywołuje metodę wirtualną
  MESSAGE(Parm & SELF.NameGroup&'|BaseProc executing|calling SELF.Func Virtual method')
  MESSAGE(SELF.NameGroup & SELF.Func(Parm))             ! wywołuje metodę wirtualną

OneClass.Func    PROCEDURE(REAL Parm)                   ! definicja OneClass.Func
  CODE
  RETURN('|Executing OneClass.Func - ' & Parm)

Proc    PROCEDURE(OneClass SELF,REAL Parm)              ! definicja OneClass.Proc
  CODE
  MESSAGE(SELF.NameGroup & ' |Executing OneClass.Proc - ' & Parm)

! plik TwoClass.CLW zawiera:
  MEMBER('ClassPrg')
Func   PROCEDURE(TwoClass SELF,LONG Parm)               ! definicja TwoClass.Func
  CODE
  RETURN('|Executing TwoClass.Func - ' & Parm)

TwoClass.Proc PROCEDURE(STRING Msg,LONG Parm)           ! definicja TwoClass.Proc
  CODE
  MESSAGE(Msg & '|Executing TwoClass.Proc - ' & Parm)
  ! plik Class3.CLW zawiera:
  MEMBER('ClassPrg')

ClassThree.Func PROCEDURE(<STRING Msg>,LONG Parm)       ! definicja ClassThree.Func
  CODE
  SELF.Proc(Msg,Parm) !Call TwoClass.Proc (overloaded)
  RETURN(Msg & '|Executing ClassThree.Func - ' & Parm)

ClassThree.Proc PROCEDURE(REAL Parm)                    ! definicja ClassThree.Proc
  CODE
  SELF.Proc('Called from ClassThree.Proc',Parm)         ! wywołuje TwoClass.Proc
  MESSAGE(SELF.NameGroup &' |Executing ClassThree.Proc - ' & Parm)
```

Porównaj: Kwalifikacja pól, MODULE, Prototypy procedur, Przeciążanie procedur, WHAT, WHERE

Struktury plików

FILE (deklaruje strukturę pliku danych)

```
label       FILE ,DRIVER( ) [,CREATE] [,RECLAIM] [,OWNER( )] [,ENCRYPT] [,NAME()] [,PRE( )]
                                [,BINDABLE] [,THREAD] [,EXTERNAL] [,DLL] [,OEM]
label           [INDEX( )]
label           [KEY( )]
label           [MEMO( )]
label           [BLOB]
[ label]        RECORD
[ label]            fields
                END
            END
```

label Etykieta FILE, INDEX, KEY, MEMO, BLOB, RECORD lub pola *field* (PROP:Label).

FILE Deklaruje plik danych.

DRIVER Określa typ pliku danych (PROP:DRIVER). Atrybut DRIVER jest wymagany dla wszystkich deklaracji struktur FILE.

CREATE Umożliwia tworzenie pliku za pomocą instrukcji CREATE w trakcie działania programu (PROP:CREATE).

RECLAIM Powoduje zagospodarowanie wolnego miejsca pozostałego po usuniętych rekordach (PROP:RECLAIM).

OWNER Określa hasło, w oparciu o które odbywa się szyfrowanie danych zapisywanych w pliku (PROP:OWNER).

ENCRYPT Wymusza szyfrowanie pliku (PROP:ENCRYPT).

NAME Określa nazwę pliku DOS (PROP:NAME).

PRE Deklaruje prefiks dla struktury.

BINDABLE Powoduje, że wszystkie pola rekordu mogą być wykorzystywane w budowaniu wyrażeń dynamicznych.

THREAD Pamięć dla bufora rekordu jest przydzielana oddzielnie dla każdego wątku, gdy plik jest w nim otwierany (PROP:THREAD).

EXTERNAL Określa, że struktura FILE jest zdefiniowana, a pamięć dla jej bufora rekordu przydzielona, w zewnętrznej bibliotece.

DLL Określa, że struktura FILE jest zdefiniowana w bibliotece dynamicznej DLL. Występuje w połączeniu z atrybutem EXTERNAL.

OEM Powoduje konwersję danych łańcuchowych z OEM ASCII na ANSI przy odczycie z dysku i z ANSI na OEM ASCII przed zapisem na dysk (PROP:OEM).

INDEX Deklaruje klucz statyczny, który wymaga odbudowy w trakcie działania.

KEY Deklaruje klucz dynamiczny odbudowywany automatycznie.

MEMO Deklaruje pole memo zmiennej długości o rozmiarze mogącym wynosić maksymalnie 64KB.

BLOB Deklaruje pole memo zmiennej długości, którego rozmiar może przekraczać 64KB.

RECORD Deklaruje strukturę rekordu dla *fields*. Struktura RECORD jest obowiązkowa dla wszystkich deklaracji struktur FILE.

fields Deklaracje pól w strukturze RECORD.

FILE deklaruje strukturę dyskowego pliku danych. Etykieta pliku FILE jest stosowana w instrukcjach przetwarzania plików i przez procedury operujące na danych zapisanych na dysku. Struktura FILE musi być zakończona znakiem kropki lub instrukcją END.

Wszystkie atrybuty FILE, KEY, INDEX, MEMO, deklaracji danych, typów danych, zależą od stosowanego sterownika pliku (formatu, w jakim będzie zapisywany na dysku). Jeśli użyjemy w deklaracji FILE elementu niedozwolonego dla danego sterownika DRIVER spowoduje powstanie błędu raportowanego przy próbie otwarcia pliku. Rodzaje dostępnych atrybutów i typów danych dla poszczególnych sterowników plików można znaleźć w ich dokumentacji.

W czasie działania aplikacji struktura RECORD otrzymuje przydział pamięci, w której jest przechowywana zawartość aktualnego rekordu wczytanego z dysku. Bufor rekordu jest zawsze alokowany w pamięci statycznej, na stercie – również wtedy, gdy plik jest zadeklarowany w sekcji danych lokalnych. W strukturze FILE jest wymagane wystąpienie struktury RECORD. Pamięć na bufor danych dla pól MEMO jest alokowana tylko wtedy, gdy plik FILE jest otwierany, a zwalniana w momencie jego zamknięcia. Pamięć dla pól BLOB jest alokowana, jeśli zajdzie taka potrzeba, po otwarciu pliku.

Plik FILE z atrybutem BINDABLE pozwala na zastosowanie wszystkich jego pól w wyrażeniach dynamicznych, bez konieczności bindowania każdego z nich za pomocą instrukcji BIND (dopuszczalne jest też u życie BIND(plik) w celu zbindowania wszystkich pól). Zawartość atrybutu NAME każdego z pól jest nazwą logiczną stosowaną w wyrażeniach dynamicznych. Jeśli nie występuje atrybut NAME, stosowana jest etykieta pola włącznie z prefiksem. W pliku .EXE jest alokowany obszar na nazwy zbindowanych zmiennych. Rozmiar programu staje się przez to większy i pociąga to za sobą zwiększenie zapotrzebowania na pamięć. Z tego względu atrybut BINDABLE powinien być stosowany tylko wtedy, gdy w wyrażeniach dynamicznych korzystamy praktycznie ze wszystkich pól rekordu pliku.

Plik FILE posiadający atrybut THREAD deklaruje oddzielny bufor rekordu (oraz blok kontrolny pliku - file control block) dla każdego wątku, który go otwiera. Jeśli wątek nie otwiera pliku, nie jest dla niego alokowany bufor rekordu.

Plik FILE zadeklarowany z atrybutem EXTERNAL może być wywoływany z poziomu kodu Clariona, jednak nie jest mu przydzielana pamięć. Pamięć dla bufora rekordu takiego pliku jest alokowana przez bibliotekę zewnętrzną. Pozwala to na dostęp, z poziomu aplikacji Clariona, do plików publicznych zadeklarowanych w bibliotekach zewnętrznych.

Powiązane procedury: BUFFER, BUILD, CLOSE, COPY, CREATE, EMPTY, FLUSH, LOCK, NAME, OPEN, PACK, RECORDS, REMOVE, RENAME, SEND, SHARE, STATUS, STREAM, UNLOCK, ADD, APPEND, BOF, BYTES, DELETE, DUPLICATE, EOF, GET, HOLD, NEXT, NOMEMO, POINTER, POSITION, PREVIOUS, PUT, RELEASE, REGET, RESET, SET, SKIP, WATCH

Przykład:

```
Names  FILE,DRIVER('Clarion')      ! deklaracja struktury pliku
Rec      RECORD                    ! wymagana struktura rekordu
Name       STRING(20)              ! zawierająca jeden lub więcej elementów danych
         . .                       ! koniec deklaracji rekordu i pliku
```

Porównaj: KEY, INDEX, MEMO, BLOB, RECORD

INDEX (deklaruje klucz statyczny)

label **INDEX**([-/+][*field*],...,[-/+][*field*]) [,**NAME**( )] [,**NOCASE**] [,**OPT**]

label Etykieta klucza INDEX (PROP:Label).

INDEX Deklaruje statyczny klucz pliku danych.

-/+ Znak poprzedzający pole *field* będące składnikiem klucza, a oznaczający porządek sortowania. Jeśli pominiemy, domyślnie jest przyjmowany znak + (plus) oznaczający sortowanie rosnące. Znak – (minus) oznacza sortowanie malejące.

field Etykieta pola struktury RECORD pliku FILE stanowiącego składnik klucza INDEX. Pola zadeklarowane z atrybutem DIM (tablice) nie mogą być używane jako składniki klucza.

NAME Określa plik dyskowy dla klucza INDEX (PROP:NAME).

OPT Powoduje wyłączenie z klucza rekordów, których wszystkie pola będące składnikami klucza posiadają wartości nieokreślone (zerowe lub puste) (PROP:OPT).

NOCASE Powoduje, że przy sortowaniu nie jest brana pod uwagę wielkość liter (PROP:NOCASE).

Klucz **INDEX** deklaruje statyczny klucz dla struktury FILE. Jego aktualizacja jest wykonywana tylko poprzez wywołanie procedury BUILD. Stosuje się go w celu umożliwienia dostępu do pliku w różnych logicznych porządkach, innych niż porządek fizyczny. Klucz INDEX zapewnia zarówno dostęp sekwencyjny, jak i bezpośredni – do wybranego rekordu.

Klucz INDEX zawsze pozwala na tworzenie duplikatów. Klucz INDEX może posiadać więcej niż jeden składnik *field*. Znaczenie składników w kluczu zależy od kolejności, w jakiej zostały ujęte podczas jego definiowania. Pierwszy komponent jest brany pod uwagę w pierwszej kolejności, ostatni – w ostatniej.

Generalnie plik danych może posiadać do 255 kluczy (typu KEY i INDEX), każdy klucz może liczyć do 255 bajtów, dokładna wielkość zależy od stosowanego sterownika pliku danych.

Klucz INDEX zadeklarowany bez żadnego pola *field* powoduje utworzenie tzw. indeksu dynamicznego. Indeks taki może być budowany w oparciu o dowolne pole (lub pola) rekordu RECORD (za wyjątkiem tablic). Składniki indeksu dynamicznego definiuje się w czasie działania programu, jako drugi parametr instrukcji BUILD. Do tworzenia roboczych kluczy o różnych składnikach może dzięki temu służyć cały czas jedna i ta sama deklaracja INDEX.

Przykład:

```
Names       FILE,DRIVER('TopSpeed'),PRE(Nam)
NameNdx       INDEX(Nam:Name),NOCASE              ! deklaracja indeksu
NbrNdx        INDEX(Nam:Number),OPT               ! deklaracja numerycznego indeksu
DynamicNdx  INDEX()                               ! deklaracja dynamicznego indeksu
Rec           RECORD
Name            STRING(20)
Number          SHORT
              . .
```

Porównaj: SET, GET, KEY, BUILD

KEY (deklaruje klucz dynamiczny)

label **KEY**([-/+] *field*,...,[-/+][*field*]) [,**DUP**] [,**NAME**( )] [,**NOCASE**] [,**OPT**] [,**PRIMARY**]

label	Etykieta klucza KEY (PROP:Label).
KEY	Deklaruje dynamiczny klucz pliku danych.
-/+	Znak poprzedzający pole *field* będące składnikiem klucza, a oznaczający porządek sortowania. Jeśli pominiemy, domyślnie jest przyjmowany znak + (plus) oznaczający sortowanie rosnące. Znak – (minus) oznacza sortowanie malejące.
field	Etykieta pola struktury RECORD pliku FILE stanowiącego składnik klucza INDEX. Pola zadeklarowane z atrybutem DIM (tablice) nie mogą być używane jako składniki klucza.
NAME	Określa plik dyskowy dla klucza KEY (PROP:NAME).
DUP	Umożliwia duplikowanie w kluczu rekordów o tych samych wartościach pól stanowiących składniki klucza (PROP:DUP).
NOCASE	Powoduje, że przy sortowaniu nie jest brana pod uwagę wielkość liter (PROP:NOCASE).
OPT	Powoduje wyłączenie z klucza rekordów, których wszystkie pola będące składnikami klucza posiadają wartości nieokreślone (zerowe lub puste) (PROP:OPT).
PRIMARY	Określa, że klucz KEY jest podstawowym kluczem relacji dla pliku (unikalnym kluczem, w którym znajdują się wszystkie rekordy pliku) (PROP:PRIMARY).

Klucz **KEY** jest indeksem nałożonym na plik danych, który aktualizuje się automatycznie przy każdej operacji aktualizacji danych (dopisywanie, poprawianie, usuwanie rekordów). Stosuje się go w celu umożliwienia dostępu do pliku w różnych logicznych porządkach, innych niż porządek fizyczny. Klucz KEY zapewnia zarówno dostęp sekwencyjny, jak i bezpośredni – do wybranego rekordu.

Klucz KEY może posiadać więcej niż jeden składnik *field*. Znaczenie składników w kluczu zależy od kolejności, w jakiej zostały ujęte podczas jego definiowania. Pierwszy komponent jest brany pod uwagę w pierwszej kolejności, ostatni – w ostatniej.

Generalnie plik danych może posiadać do 255 kluczy (typu KEY i INDEX), każdy klucz może liczyć do 255 bajtów, dokładna wielkość zależy od stosowanego sterownika pliku danych.

Przykład:

```
Names     FILE,DRIVER('Clarion'),PRE(Nam)
NameKey   KEY(Nam:Name),NOCASE,DUP        ! deklaracja klucza
NbrKey    KEY(Nam:Number),OPT             ! deklaracja numerycznego klucza
Rec       RECORD
Name        STRING(20)
Number      SHORT
          . .
  CODE
  Nam:Name = 'Clarion Software'           ! zainicjowanie pola klucza
  GET(Names,Nam:NameKey)                  ! pobranie rekordu
  SET(Nam:NbrKey)                         ! ustawienie sekwencji wg numeru
```

Porównaj: SET, GET, INDEX, BUILD, PACK

MEMO (deklaruje pole tekstowe)

label **MEMO**(*length*) [,**BINARY**] [,**NAME**( )]

label Etykieta pola MEMO (PROP:Label).

MEMO Deklaruje łańcuch stałej długości, który jest na dysku przechowywany ze zmienną długością.

length Stała numeryczna określająca maksymalną liczbę znaków. Dostępny zakres wynosi od 1 do 65.,520 bajtów w trybie 16-bitowym, nie jest zaś ograniczony w trybie 32-bitowym. (zależy tylko od rodzaju wybranego sterownika pliku danych).

BINARY Deklaruje MEMO jako obszar do przechowywania danych binarnych (PROP:BINARY).

NAME Określa plik dyskowy dla pola MEMO (PROP:NAME).

MEMO deklaruje łańcuch stałej długości, który jest na dysku przechowywany ze zmienną długością. Parametr *length* definiuje maksymalny rozmiar pola memo. Pole MEMO musi być deklarowane przed strukturą RECORD. Bufor pamięci dla pól MEMO jest przydzielany po otwarciu pliku, a zwalniany po jego zamknięciu. Pola MEMO są zazwyczaj wyświetlane w kontrolkach TEXT na ekranie SCREEN i w strukturach REPORT.

Generalnie w strukturze FILE można zadeklarować do 255 pól MEMO. Dokładny rozmiar i ilość pól MEMO, sposób ich przechowywania na dysku, ściśle zależą od stosowanego formatu plików.

Przykład:

```
Names     FILE,DRIVER('Clarion'),PRE(Nam)
NameKey     KEY(Nam:Name)
NbrKey      KEY(Nam:Number)
Notes       MEMO(4800)                    ! Memo, 4800 bajtów
Rec         RECORD
Name          STRING(20)
Number        SHORT
            . .
```

BLOB (deklaruje pole o zmiennej długości)

label **BLOB** [,**BINARY**] [,**NAME**( )]

label Etykieta pola BLOB (PROP:Label).

BLOB Deklaruje łańcuch zmiennej długości, którego rozmiar może być większy od 64K (zarówno w przypadku aplikacji 16-bitowych, jak i 32-bitowych).

BINARY Deklaruje BLOB jako obszar do przechowywania danych binarnych (PROP:BINARY).

NAME Określa plik dyskowy dla pola BLOB (PROP:NAME).

BLOB (Binary Large OBject) deklaruje pole łańcuchowe o zmiennej długości, którego rozmiar może przekraczać 64K zarówno w przypadku aplikacji 16-bitowych, jak i 32-bitowych. Pole BLOB musi być deklarowane przed strukturą RECORD.

Generalnie w strukturze FILE można zadeklarować do 255 pól BLOB. Dokładna ilość pól MEMO i sposób ich przechowywania na dysku, ściśle zależą od stosowanego formatu plików.

BLOB nie może być stosowane jako zmienna – nie możemy określić BLOB, jako źródło danych dla kontrolki (pole USE). Nie jest również możliwe bezpośrednie przypisywanie wartości jednego pola BLOB do drugiego. Jeżeli chcemy uzyskać uchwyt (handle) do egzemplarza BLOB, stosujemy właściwość PROP:Handle. Operacja przypisania wartości pola BLOB do innego pola BLOB powinna przebiegać następująco: pobieramy uchwyty do obu egzemplarzy BLOB, przypisujemy uchwyt egzemplarza źródłowego do docelowego. Przy dostępie do danych zapisanych w BLOB nie możemy traktować go jako jednej całości, musimy stosować albo technikę fragmentowania łańcucha (do 64K w 16-bitach, bez limitu – w 32-bitach), albo właściwość PROP:ImageBlob. Bajty w polu BLOB są numerowane od zera (początek), a nie od 1.

Procedura SIZE daje w rezultacie liczbę bajtów zapisanych w polu BLOB – dla rekordu, który aktualnie znajduje się w pamięci. Możemy zarówno odczytać, jak i ustawić rozmiar pola BLOB, posługując się właściwością PROP:Size. Jest możliwe określenie rozmiaru pola BLOB przed wstawieniem do niego danych za pomocą operacji fragmentowania łańcucha. Nie jest to jednak konieczne, gdyż rozmiar jest określany automatycznie przez operację wykonującą fragmentowanie. Jest także możliwe zastosowanie właściwości PROP:ImageBlob pozwalającej na przechowywanie i odtwarzanie plików graficznych bez potrzeby ręcznego określania rozmiaru poprzez ustawienie właściwości PROP:Size. Dobrym rozwiązaniem jest ustawianie początkowej wartości PROP:Size na zero (0) przed wykonaniem operacji przypisania danych do pola BLOB; dzięki temu wyeliminujemy zjawisko polegające na pozostawianiu „śmieci” przez poprzednio wykorzystywane pola BLOB. Podczas przypisywania danych z jednego pola BLOB do drugiego, w oparciu o właściwość PROP:Handle, może zachodzić potrzeba wykorzystania właściwości PROP:Size w celu dopasowania rozmiaru pola docelowego do rozmiaru pola źródłowego.

Jeśli chcemy sprawdzić, czy zawartość pola BLOB uległa zmianie od czasu wczytania go z dysku, posługujemy się właściwością PROP:Touched.

Przykład:

```
ArchiveFile  PROCEDURE
Names          FILE,DRIVER('TopSpeed')
NaneKey          KEY(Name)
Notes            BLOB                                  ! może być większe niż 64K
Rec              RECORD
Name               STRING(20)
                 . .

ArcNames   FILE,DRIVER('TopSpeed')
Notes          BLOB
Rec            RECORD
Name             STRING(20)
                 . .
  CODE
  SET(Names)
  LOOP
    NEXT(Names)
    IF ERRORCODE() THEN BREAK.
    ArcNames.Rec = Names.Rec                                    ! przypisanie danej Rec do Arc
    ArcNames.Notes{PROP:Handle} = Names.Notes{PROP:Handle}      ! przypisanie BLOB do Arc
    ArcNames.Notes{PROP:Size} = Names.Notes{PROP:Size}          ! i dopasowanie rozmiaru
    ADD(ArcNames)
  END

StoreFileInBlob   PROCEDURE                                 ! zachowuje dowolny plik dyskowy w BLOB
DosFileName         STRING(260),STATIC
LastRec             LONG
SavPtr              LONG(1)                                 ! start od 1
FileSize            LONG
DosFile             FILE,DRIVER('DOS'),PRE(DOS),NAME(DosFileName)
Record                RECORD
F1                      STRING(2000)
                      . .
BlobStorage         FILE,DRIVER('TopSpeed'),PRE(STO)
File                  BLOB,BINARY
Record                RECORD
FileName                STRING(64)
                      . .
  CODE
  IF NOT FILEDIALOG('Choose File to Store',DosFileName,,0010b) THEN RETURN.
  OPEN(BlobStorage)                                         ! otwarcie pliku BLOB
  STO:FileName = DosFileName                                ! i zachowanie jego nazwy
  OPEN(DosFile)                                             ! otwarcie pliku
  FileSize = BYTES(DosFile)                                 ! pobranie rozmiaru pliku
  STO:File{PROP:Size} = FileSize                            ! i ustawienie BLOB do przechowania pliku
  LastRec = FileSize % SIZE(DOS:Record)                     ! sprawdzenie krótkiego rekordu na końcu pliku
  LOOP INT(FileSize/SIZE(DOS:Record)) TIMES
    GET(DosFile,SavPtr)                                     ! pobranie każdego rekordu
    ASSERT(NOT ERRORCODE())
    STO:File[SavPtr - 1 : SavPtr + SIZE(DOS:Record) - 2] = DOS:Record
    ! fragmentowanie danych łańcuchowych do BLOB
    SavPtr += SIZE(DOS:Record)                              ! wylicza wskaźnik następnego rekordu
  END
  IF LastRec                                                ! jeśli jest krótki record na końcu pliku
    GET(DosFile,SavPtr)                                     ! pobierz ostatni rekord
    ASSERT(BYTES(DosFile) = LastRec)                        ! czy odczytany rozmiar równa się wyliczonemu
    STO:File[SavPtr - 1 : SavPtr + LastRec - 2] = DOS:Record
  END
  ADD(BlobStorage)
  ASSERT(NOT ERRORCODE())
  CLOSE(DosFile);CLOSE(BlobStorage)
```

Porównaj: PROP:ImageBlob, PROP:Size

RECORD (deklaruje strukturę rekordu)

```
[label]   RECORD [,PRE( )] [,NAME( )]
            fields
          END
```

RECORD Deklaruje początek struktury danych wewnątrz deklaracji FILE.

fields Deklaracje pól.

PRE Deklaruje prefiks dla struktury.

NAME Określa zewnętrzną nazwę dla struktury RECORD.

Instrukcja **RECORD** deklaruje początek struktury danych wewnątrz deklaracji FILE. Struktura RECORD jest wymagana w deklaracjach FILE. Każde pole *field* jest elementem struktury RECORD. Długość struktury RECORD jest sumą długości wszystkich jej pól.

Gdy etykieta struktury RECORD jest stosowana w instrukcjach przypisania, wyrażeniach lub w listach parametrów, jest traktowana tak, jak dana typu GROUP.

W czasie działania programu jest przydzielany bufor pamięci, w pamięci statycznej, przeznaczony na przechowywanie danych struktury RECORD. Pola *fields* w buforze rekordu są dostępne niezależnie od tego, czy plik jest otwarty, czy zamknięty.

Jeśli pola *fields* zawierają deklaracje zmiennych z określonymi wartościami początkowymi, wartości te są wykorzystywane tylko do obliczenia niezbędnego rozmiaru dla zmiennej – bufor rekordu nie jest inicjowany wartościami początkowymi. Na przykład, deklaracja pola STRING('abc') tworzy łańcuch trzybajtowy, ale odpowiadająca mu zmienna nie jest inicjowana wartością 'abc' dopóty, dopóki nie zostanie to wykonane odpowiednią instrukcją w kodzie wykonywalnym. Rekordy z dyskowego pliku danych są wczytywane do bufora rekordu za pomocą instrukcji NEXT, PREVIOUS, GET, czy REGET. Dane w polach *fields* są przetwarzane, następnie są aktualizowane w dyskowym pliku danych jako jednostka stanowiąca pojedynczy rekord RECORD – za pomocą poleceń ADD, APPEND, PUT lub DELETE.

Procedury WHAT i WHERE umożliwiają uzyskanie dostępu do pól *fields* w oparciu o ich względną pozycję w strukturze RECORD.

Przykład:

```
Names   FILE,DRIVER('Clarion')      ! deklaruje strukturę pliku
Record    RECORD                   ! początek deklaracji rekordu
Name        STRING(20)             ! deklaracja pola łańcuchowego
Number      SHORT                  ! deklaracja pola numerycznego
          . .                      ! koniec deklaracji rekordu I pliku
```

Porównaj: FILE, NEXT, PREVIOUS, GET, REGET, ADD, APPEND, PUT, DELETE, WHAT, WHERE

Przetwarzanie danych o nieokreślonych wartościach (null)

Koncepcja wartości nieokreślonej “null” w polu struktury FILE polega na przyjęciu, że do pola takiego użytkownik nigdy nie wprowadził żadnej wartości. Null oznacza wartość nieokreśloną, nieznaną. Jest to zasadnicza różnica w porównaniu z wartością pustą (łańcuch pusty), czy zerową i umożliwia określenie, czy pole zawiera wartość zerową (pustą) i nieokreśloną.

W wyrażeniach, null nie oznacza wartości zerowej lub pustej. Z tego względu dowolne wyrażenie porównujące nieokreśloną wartość pola pliku FILE z inną wartością da zawsze rezultat nieokreślony. Tak samo będzie również w przypadku, gdy obie porównywane wartości są nieokreślone (null). Na przykład, warunkowe wyrażenie Pre:Field1 = Pre:Field2 jest wyliczone jako prawdziwe tylko wtedy, gdy oba pola zawierają określone wartości. Jeśli wartości obu lub jednego z tych pól są nieokreślone, również rezultat jest nieokreślony.

```
Znana = Znana                 ! Wyliczona jako Prawda lub Fałsz
Znana = Nieokreślona          ! Wyliczona jako nieokreślona
Nieokreślona = Nieokreślona   ! Wyliczona jako nieokreślona
Nieokreślona <> 10            ! Wyliczona jako nieokreślona
1 + Nieokreślona              ! Wyliczona jako nieokreślona
```

Jedyne cztery wyjątki od przedstawionej zasady obejmują wyrażenia boolowskie wykorzystujące OR i AND, gdzie tylko jedna część całego wyrażenia może być nieokreślona, a druga część spełnia określone kryteria.

```
Nieokreślona OR Prawda        ! Wyliczona jako Prawda
Prawda OR Nieokreślona        ! Wyliczona jako Prawda
Nieokreślona AND Fałsz        ! Wyliczona jako Fałsz
Fałsz AND Nieokreślona        ! Wyliczona jako Fałsz
```

Obsługa wartości nieokreślonych w danym pliku FILE ściśle zależy od sterownika pliku. Niektóre sterowniki plików umożliwiają obsługę wartości nieokreślonych zgodnie z przedstawioną koncepcją (sterowniki SQL – w większości przypadków), niektóre - nie. Należy zawsze sprawdzić w dokumentacji wybranego sterownika, czy pozwala on na posługiwanie się wartościami nieokreślonymi..

Porównaj: NULL, SETNULL, SETNONULL

Właściwości struktury FILE

Przedstawione poniżej właściwości są atrybutami struktury FILE. Opisują one atrybuty, pola, klucze, pola MEMO, pola BLOB, które mogą występować wewnątrz struktury FILE. Wszystkie te właściwości struktury FILE są właściwościami tylko-do-odczytu, za wyjątkiem: PROP:NAME (pozwala na zmianę nazwy pola), PROP:OWNER oraz PROP:DriverString. Przypisanie wartości do właściwości powoduje zastąpienie początkowo zadeklarowanych atrybutów.

Niektóre z właściwości są specyficzne dla struktury FILE, jako cel *target* występuje dla nich etykieta pliku FILE. Inne są specyficzne dla klucza KEY (lub INDEX), dla nich celem *target* jest etykieta klucza KEY (lub INDEX). Jeszcze inne są specyficzne dla BLOB i posługują się BLOB w roli celu *target*.

Część właściwości jest tablicami pobierającymi numer określonego pola lub klucza. Każde pole struktury RECORD otrzymuje dodatni numer. W strukturze RECORD deklaracje pola zaczynają się na 1 i są zwiększane o 1 dla każdego kolejnego pola w takiej kolejności, w jakiej pojawiają się one w strukturze RECORD. Instrukcja END kończąca grupę GROUP nie jest numerowana, jako że nie jest ona deklaracją pola.

Pola MEMO są numerowane liczbami ujemnymi. Deklaracje MEMO zaczynają się od –1 i są zmniejszane o 1 dla każdego kolejnego pola MEMO, w takiej kolejności, w jakiej występują one w strukturze FILE. Pola BLOB są numerowane liczbami dodatnimi. Deklaracje BLOB zaczynają się na 1 i są zwiększane o 1 dla każdego kolejnego pola BLOB, w takiej kolejności, w jakiej występują one w strukturze FILE.

Właściwości uniwersalne

PROP:Label

Zwraca w rezultacie etykietę.

Jeśli nie został wyspecyfikowany numer elementu tablicy i *target* jest etykietą klucza KEY (lub INDEX), PROP:Label daje w rezultacie etykietę klucza KEY (lub INDEX).

Gdy jest określony dodatni numer elementu tablicy i *target* jest plikiem FILE, PROP:Label daje w rezultacie etykietę pola wewnątrz struktury RECORD.

Gdy jest określony ujemny numer elementu tablicy i *target* jest plikiem FILE, PROP:Label daje w rezultacie etykietę pola MEMO wewnątrz struktury FILE.

Gdy jest określony dodatni numer elementu tablicy i *target* jest BLOB, PROP:Label daje w rezultacie etykietę BLOB.

PROP:NAME

Atrybut NAME instrukcji deklaracji.

Jeśli nie został wyspecyfikowany numer elementu tablicy i *target* jest etykietą pliku FILE, PROP:Name daje w rezultacie zawartość atrybutu NAME pliku FILE.

Gdy jest określony dodatni numer elementu tablicy i *target* etykietą pliku FILE, PROP:Name daje w rezultacie zawartość atrybutu NAME określonego pola struktury RECORD.

Gdy jest określony ujemny numer elementu tablicy i *target* etykietą pliku FILE, PROP:Name daje w rezultacie zawartość atrybutu NAME określonego pola MEMO struktury FILE.

Jeśli nie został wyspecyfikowany numer elementu tablicy i *target* jest etykietą klucza KEY (lub INDEX), PROP:Name daje w rezultacie zawartość atrybutu NAME określonego klucza KEY (lub INDEX).

Gdy jest określony dodatni numer elementu tablicy i *target* etykietą BLOB, PROP:Name daje w rezultacie zawartość atrybutu NAME określonego pola BLOB.

PROP:Type

Typ danych instrukcji deklaracji.

Jeśli nie został wyspecyfikowany numer elementu tablicy i *target* jest etykietą klucza KEY (lub INDEX), PROP:Type daje w rezultacie "KEY" lub "INDEX".

Gdy jest określony dodatni numer elementu tablicy i *target* etykietą pliku FILE, PROP:Type daje w rezultacie typ danych określonego pola struktury RECORD.

Właściwości instrukcji FILE

Celem *target* przedstawionych poniżej właściwości jest zawsze etykieta pliku FILE

PROP:DRIVER

Atrybut DRIVER. Określa sterownik pliku FILE.

PROP:CREATE

Atrybut CREATE instrukcji FILE. Atrybut przełącznikowy zawierający pusty łańcuch (") – jeśli nie występuje lub '1' – jeśli występuje.

PROP:RECLAIM

Atrybut RECLAIM instrukcji FILE. Atrybut przełącznikowy zawierający pusty łańcuch (") – jeśli nie występuje lub '1' – jeśli występuje.

PROP:OWNER

Atrybut OWNER instrukcji FILE.

PROP:ENCRYPT

Atrybut ENCRYPT instrukcji FILE. Atrybut przełącznikowy zawierający pusty łańcuch (") – jeśli nie występuje lub '1' – jeśli występuje.

PROP:THREAD

Atrybut THREAD instrukcji FILE. Atrybut przełącznikowy zawierający pusty łańcuch (") – jeśli nie występuje lub '1' – jeśli występuje.

PROP:OEM

Atrybut OEM instrukcji FILE. Atrybut przełącznikowy zawierający pusty łańcuch (") – jeśli nie występuje lub '1' – jeśli występuje.

PROP:Keys

Daje w rezultacie liczbę deklaracji KEY i INDEX w strukturze FILE.

PROP:Key

Tablica zwracająca referencję do określonego klucza KEY lub INDEX w strukturze FILE. Referencja ta może być używana jako strona źródłowa instrukcji przypisania referencyjnego.

Właściwości indeksu

Celem *target* przedstawionych poniżej właściwości jest zawsze etykieta klucza KEY (lub INDEX)

PROP:PRIMARY

Atrybut PRIMARY instrukcji KEY. Atrybut przełącznikowy zawierający pusty łańcuch (") – jeśli nie występuje lub '1' – jeśli występuje.

PROP:DUP

Atrybut DUP instrukcji KEY. Atrybut przełącznikowy zawierający pusty łańcuch (") – jeśli nie występuje lub '1' – jeśli występuje.

PROP:NOCASE

Atrybut NOCASE instrukcji KEY lub INDEX. Atrybut przełącznikowy zawierający pusty łańcuch (") – jeśli nie występuje lub '1' – jeśli występuje.

PROP:OPT

Atrybut OPT instrukcji KEY lub INDEX. Atrybut przełącznikowy zawierający pusty łańcuch (") – jeśli nie występuje lub '1' – jeśli występuje.

PROP:Components

Zwraca liczbę składników (pól) klucza KEY lub INDEX.

PROP:Field

Tablica określająca numer pola (wewnątrz struktury RECORD) określonego pola składnikowego klucza KEY lub INDEX. Numer pola może być używany jako numer elementu tablicy PROP:Label lub PROP:Name.

PROP:Ascending

Tablica dająca w rezultacie '1' jeśli określony składnik klucza posiada porządek rosnący lub łańcuch pusty (") – przy porządku malejącym.

Właściwości pola

Celem *target* przedstawionych poniżej właściwości jest zawsze etykieta pliku FILE

PROP:Memos

Zwraca liczbę pól MEMO w strukturze FILE.

PROP:Blobs

Zwraca liczbę pól BLOB w strukturze FILE.

PROP:BINARY

Atrybut BINARY powiązany z MEMO lub BLOB w strukturze FILE. Atrybut przełącznikowy zawierający pusty łańcuch (") – jeśli nie występuje lub '1' – jeśli występuje.

PROP:Fields

Zwraca liczbę pól zadeklarowanych w strukturze RECORD.

PROP:Size

Tablica zwracająca zadeklarowany rozmiar pola MEMO, STRING, CSTRING, PSTRING, DECIMAL lub PDECIMAL.

PROP:Places

Tablica zwracająca liczbę pozycji dziesiętnych zadeklarowaną dla określonego pola DECIMAL lub PDECIMAL.

PROP:Dim

Właściwość tablicowa dająca w wyniku iloczyn wymiarów tablicy DIM określonej dla danego pola. Na przykład, dla pola DIM(3,2) - PROP:Dim wyniesie 6.

PROP:Over

Właściwość tablicowa numer pola referencyjnego wskazywanego przez atrybut OVER nałożony na określone pole.

Przykład:

```
    MEMBER
      MAP
DumpGroupDetails   PROCEDURE(USHORT start, USHORT total)
DumpFieldDetails   PROCEDURE(USHORT indent, USHORT FieldNo)
DumpToFile         PROCEDURE
SetAttribute       PROCEDURE(SIGNED Prop,STRING Value)
StartLine          PROCEDURE(USHORT indent,STRING label, STRING type)
Concat             PROCEDURE(STRING s)
      END
LineSize           EQUATE(255)
FileIndent         EQUATE(20)
DestName           STRING(FILE:MaxFilePath)
DestFile           FILE,DRIVER('ASCII'),CREATE,NAME(DestName)
Record               RECORD
Line                   STRING(LineSize)
                     ..
TheFile            &FILE
ABlob              &BLOB
AKey               &KEY
Line               STRING(LineSize)

PrintFile    PROCEDURE(*FILE F)
  CODE
  IF NOT FILEDIALOG('Choose Output File',DestName,'Text|*.TXT|Source|*.CLW',0100b)
    RETURN
  END
  OPEN(DestFile)
  IF ERRORCODE()
    CREATE(DestFile)
    OPEN(DestFile)
  END
  ASSERT(ERRORCODE()=0)
  TheFile &= F
  DO DumpFileDetails
  DO DumpKeys
  DO DumpMemos
  DO DumpBlobs
  DumpGroupDetails(0, F{PROP:Fields})
  StartLine(FileIndent,'','END')
  DumpToFile

DumpFileDetails  ROUTINE
  StartLine(FileIndent,TheFile{PROP:label},'FILE')
  Concat(',DRIVER("' & CLIP(TheFile{PROP:Driver}))
```

```
    IF TheFile{PROP:DriverString}
      Concat(',' & CLIP(TheFile{PROP:DriverString}))
    END
    Concat('')')
    SetAttribute(TheFile{PROP:Create},'CREATE')
    SetAttribute(TheFile{PROP:Reclaim},'RECLAIM')
    IF TheFile{PROP:Owner}
      Concat(',OWNER('' & CLIP(TheFile{PROP:Owner}) & '')')
    END
    SetAttribute(TheFile{PROP:Encrypt},'ENCRYPT')
    Concat(',NAME('' & CLIP(TheFile{PROP:Name}) & '')')
    SetAttribute(TheFile{PROP:Thread},'THREAD')
    SetAttribute(TheFile{PROP:OEM},'OEM')
    DumpToFile

  DumpMemos ROUTINE
    LOOP X# = 1 TO TheFile{PROP:Memos}
      StartLine(FileIndent+2,TheFile{PROP:label,-X#},'MEMO(')
      Concat(CLIP(TheFile{PROP:Size,-X#})&')')
      SetAttribute(TheFile{PROP:Binary,-X#},'BINARY')
      IF TheFile{PROP:Name,-X#}
        Concat(',NAME(' & CLIP(TheFile{PROP:Name,-X#}) & ')')
      END
      DumpToFile
    END

  DumpBlobs    ROUTINE
    Blobs = TheFile{PROP:Blobs}
    LOOP X# = 1 TO TheFile{PROP:Blobs}
      ABlob &= TheFile{PROP:Blob,X#}
      StartLine(FileIndent+2,ABlob{PROP:label},'BLOB')
      SetAttribute(ABlob{PROP:Binary},'BINARY')
      IF ABlob{PROP:Name}
        Concat(',NAME(' & CLIP(ABlob{PROP:Name}) & ')')
      END
      DumpToFile
    END

  DumpKeys  ROUTINE
    LOOP X# = 1 TO TheFile{PROP:Keys}
      AKey &= TheFile{PROP:Key,X#}
      StartLine(FileIndent+2,AKey{PROP:label},AKey{PROP:Type})
      Concat('(')
      LOOP Y# = 1 TO AKey{PROP:Components}
      IF Y# > 1 THEN Concat(',').
        IF Key{PROP:Ascending,Y#}
          Concat('+')
        ELSE
          Concat('-')
        END
        Concat(TheFile{PROP:Label,key{PROP:Field,Y#}})
      END
      Concat(')')
      SetAttribute(AKey{PROP:Dup},'DUP')
      SetAttribute(AKey{PROP:NoCase},'NOCASE')
      SetAttribute(AKey{PROP:Opt},'OPT')
      SetAttribute(AKey{PROP:Primary},'PRIMARY')
      IF AKey{PROP:Name}
        Concat(',NAME('' & CLIP(AKey{PROP:Name}) & '')')
      END
      DumpToFile
    END

  DumpGroupDetails   PROCEDURE(USHORT start, USHORT total)
  fld                  USHORT
  fieldsInGroup        USHORT
```

```
GroupIndent              USHORT,STATIC
  CODE
  IF start = 0 THEN
    GroupIndent = FileIndent+2
    StartLine(GroupIndent,'RECORD','RECORD')
    DumpToFile
  END
  GroupIndent += 2
  LOOP fld = start+1 TO start+total
    DumpFieldDetails(GroupIndent,fld)
    IF TheFile{PROP:Type,fld} = 'GROUP'
      fieldsInGroup = TheFile{PROP:Fields,fld}
      DumpGroupDetails (fld, fieldsInGroup)
      fld += fieldsInGroup
    END
  END
  GroupIndent -= 2
  StartLine(GroupIndent,'','END')
  DumpToFile

DumpFieldDetails  PROCEDURE(USHORT indent, USHORT FieldNo)
FldType    STRING(20)
  CODE
  FldType = TheFile{PROP:Type,FieldNo}
  StartLine(indent,TheFile{PROP:Label,FieldNo},FldType)
  IF INSTRING('STRING',FldType,1,1) OR INSTRING('DECIMAL',FldType,1,1)
    Concat('(' & TheFile{PROP:Size,FieldNo})
    IF FldType = 'DECIMAL' OR FldType = 'PDECIMAL'
      Concat(',' & TheFile{PROP:Places,FieldNo})
    END
    Concat(')')
  END
  IF TheFile{PROP:Dim,FieldNo} <> 0
    Concat(',DIM(' & CLIP(TheFile{PROP:Dim,FieldNo}) & ')')
  END
  IF TheFile{PROP:Over,FieldNo} <> 0
    Concat(',OVER(' & CLIP(TheFile{PROP:Label,TheFile{PROP:Over,FieldNo}}) & ')')
  END
  IF TheFile{PROP:Name,FieldNo}
    Concat(',NAME(''' & CLIP(TheFile{PROP:Name,FieldNo}) & ''')')
  END
  DumpToFile

SetAttribute PROCEDURE (Prop,Value)
  CODE
    IF Prop THEN Line = CLIP(Line) & ',' & CLIP(Value).
StartLine    PROCEDURE (USHORT indent,STRING label, STRING type)
TypeStart    USHORT
  CODE
    Line = label
    IF LEN(CLIP(Line)) < Indent
      TypeStart = Indent
    ELSE
      TypeStart = LEN(CLIP(Line)) + 4
    END
    Line[TypeStart : LineSize] = type

Concat PROCEDURE (STRING s)
  CODE
    Line = CLIP(Line) & s

DumpToFile PROCEDURE
  CODE
    DestFile.Line = Line
    ADD(DestFile)
    ASSERT(ERRORCODE()=0)
```

Pliki ustawień środowiskowych (narodowych)

Plik środowiskowy zawiera parametry narodowe dla aplikacji. W momencie inicjowania programu, biblioteka runtime Clariona próbuje zlokalizować plik środowiskowy, który będzie miał taką samą nazwę, jako plik wykonywalny (*program*.ENV). Jeśli plik środowiskowy nie zostanie odnaleziony, biblioteka runtime domyślnie przyjmie ustawienia dla języka angielskiego (English/ASCII).

Przedstawione tu ustawienia mogą być wykorzystane do przystosowania do wymogów narodowych samego środowiska Clarion. Wystarczy w tym celu zmienić plik CLARION*X*.ENV (gdzie *X* jest numerem wersji Clarion).

Plik .ENV jest kompatybilny z plikiem .INI, który znamy z Clarion for DOS (wersje 3 i 3.1), jeśli parametr CLACHARSET jest ustawiony na OEM. Jest tak dlatego, bo pliki .INI Clarion for DOS są zazwyczaj tworzone w oparciu o zestawy znaków OEM ASCII, a nie ANSI.

Pliki ustawień środowiskowych mogą być ładowane dynamicznie w czasie działania programu, służy do tego procedura LOCALE. LOCALE może służyć również do ustawiania wybranych parametrów środowiskowych. Obsługa znaków narodowych zależy od stosowanego sterownika pliku (zazwyczaj powinien on mieć wtedy atrybut OEM); charakterystyki poszczególnych sterowników należy sprawdzać w dokumentacji.

Poniżej przedstawiono parametry, które można ustawiać w plikach środowiskowych:

CLACHARSET=WINDOWS

CLACHARSET=OEM

Wskazuje zestaw znaków, który będzie wykorzystywany przez wpisy pliku .ENV. Domyślnym zestawem jest WINDOWS – jeśli omawiany parametr zostanie pominięty. Jeśli edytujemy plik .ENV za pomocą edytora MS-DOS – wybieramy ustawienie OEM. Wybieramy je również wtedy, gdy chcemy zachować zgodność z Clarion for DOS. W pozostałych przypadkach powinniśmy wybierać ustawienie WINDOWS lub pomijać dany wpis. Parametr CLACHARSET powinien występować w pliku .ENV jako pierwszy.

CLACOLSEQ=WINDOWS

CLACOLSEQ=*"string"*

Definiuje sekwencję sortowania znaków wykorzystywana w czasie działania programu. Sekwencja ta jest wykorzystywana do budowania kluczy KEY i INDEX, jak również przy sortowaniu elementów kolejek QUEUE, czy w operacjach porównywania ciągów znaków.

Jeśli jest wykorzystywane ustawienie WINDOWS, domyślną sekwencją sortowania będzie ta, która została zdefiniowana w Panelu sterowania Windows (ustawienia regionalne). Domyślnym ustawieniem, przyjmowanym w wypadku ominięcia omawianego parametru, jest sekwencja sortowania ANSI.

Zastosowanie ustawienia WINDOWS powoduje „przeplatanie" wielkich i małych liter (AaBbCc ...), tak więc kod:

```
CASE SomeString[1]
OF 'A' TO 'Z'
```

oznacza włączenie w warunek również znaków z zakresu od 'a' do 'y'. Jeśli stosujemy ustawienie WINDOWS, przy porównywaniu ciągów znaków możemy stosować funkcje ISUPPER i ISLOWER

W uzupełnieniu do ustawienia WINDOWS, istnieje możliwość zdefiniowania własnego łańcucha *string* określającego sekwencję sortowania znaków. Łańcuch ten zamykamy w podwójnych cudzysłowach. W łańcuchu string wpisujemy znaki w takiej kolejności, w jakiej mają być brane pod uwagę przy sortowaniu. Nie musimy wpisywać wszystkich znaków, wystarczą te, których kolejność jest nietypowa dla ustawień standardowych.

Na przykład, jeśli CLACOLSEQ="CA", a standardowe sortowanie języka angielskiego wygląda tak: (ABCD ...), to w efekcie zdefiniowania powyższego ustawienia jest ono modyfikowane do postaci (CBAD ...). Taki sposób definiowania sekwencji sortowania różni się od tego, który znamy z Clarion for DOS, gdzie wymagane było określenie w łańcuchu dokładnie 222 znaków, ale jest przy tym kompatybilny ze wszystkimi poprzednimi wersjami Clariona.

UWAGA: Plik danych powinien być zawsze odczytywany i zapisywany w oparciu o tę samą sekwencję sortowania.

Stosowanie różnych sekwencji może doprowadzić do błędów w kluczu i braku dostępu do rekordów.

Ustawienie CLACOLSEQ=WINDOWS oznacza, że sekwencja sortowania może się zmieniać wraz ze zmianą ustawień w Panelu sterowania. W momencie, gdy taka zmiana nastąpi, należy użyć BUILD do odbudowania kluczy plików danych.

CLAAMPM=WINDOWS

CLAAMPM=*"AMstring","PMstring"*

Definiuje tekst stosowany do oznaczenia pory przed- i popołudniowej podczas wyświetlania czasu w formacie skróconym. Ustawienie WINDOWS powoduje, że obowiązują odpowiednie ustawienia z Panelu sterowania Windows (Ustawienia regionalne, Godzina). Łańcuchy *AMstring* i *PMstring* są takie same, jak w Clarion for DOS, różnicą jest to, że mogą one przyjmować ustawienia na podstawie parametru CLACHARSET.

CLAMONTH="Month1","Month2", ... ,"Month12"

Definiuje teksty reprezentujące pełne nazwy miesięcy zwracane przez odpowiednie procedury oraz wzorce wyświetlania.

CLAMON="AbbrevMonth1","AbbrevMonth2", ... ,"AbbrevMonth12"

Definiuje teksty reprezentujące skrócone nazwy miesięcy zwracane przez odpowiednie procedury oraz wzorce wyświetlania.

CLADIGRAPH="*DigraphChar1Char2, ...* "

Umożliwia definiowanie par znaków reprezentujących pojedynczy znak, którego nie można wyświetlić w danym alfabecie. Przykładem może być niemieckie „a umlaut" pisane jako para „ae". Oznacza to, że *Digraph* jest logicznie pojedynczym znakiem, zapisanym fizycznie jako połączenie

dwóch znaków: *Char1* i *Char2*. Możemy definiować wiele kombinacji *DigraphChar1Char2*; oddzielamy je od siebie znakiem przecinka. Ten parametr bierze pod uwagę ustawienie CLACHARSET.

CLACASE=WINDOWS

CLACASE=*"UpperString","LowerString"*

Pozwala na zdefiniowanie par wielka-mała litera. Ustawienie WINDOWS powoduje przyjęcie par zdefiniowanych dla kraju określonego w Panelu sterowania Windows. Pominięcie tego parametru w pliku środowiskowym pociąga za sobą przyjęcie sortowania opartego o ANSI

Parametru *UpperString* i *LowerString* definiują odpowiednio zestawy wielkich i małych liter. Pary tworzą litery z tych samych pozycji, tak więc długości obu zestawów znaków musza być jednakowe. parametr bierze pod uwagę ustawienie CLACHARSET. Znaki ANSI poniżej kodu 127 nie są brane pod uwagę.

CLABUTTON="OK","&Yes","&No","&Abort","&Retry","&Ignore",Cancel", "&Help"

Definiuje teksty dla przycisków, które mogą być umieszczane w okienku komunikatu wyświetlanym przez procedurę MESSAGE. Poszczególne teksty zamyka się w cudzysłowach i oddziela od siebie przecinkami. Odpowiadają one, w kolejności występowania, przyciskom: OK, YES, NO, ABORT, RETRY, IGNORE, CANCEL, HELP. Domyślne teksty zostały pokazane powyżej.

CLAMSG*errornumber="ErrorMessage"*

Pozwala na zdefiniowanie komunikatów błędów działania programu (run-time error messages) zastępujących komunikaty oryginalne. Stała *errornumber* jest standardowym kodem błędu dodawanym do CLAMSG. Tekst *ErrorMessage* jest wartością łańcuchową zastępującą domyślny komunikat o błędzie. Na przykład, CLAMSG2="Nie znaleziono pliku" powoduje, że tekstem zwracanym przez funkcję ERROR() w momencie wystąpienia błędu ERRORCODE() = 2, jest właśnie „Nie znaleziono pliku".

CLALFN=OFF

Wyłącza obsługę długich nazw plików w naszym programie.

Przykład:

```
CLACHARSET=WINDOWS
CLACOLSEQ="AÄĹĆaŕáâäĺćBbCÇcçDdEÉečéęëFfGgHhIiěíîďJjKkLlMmNŃnńOÖoňóôöPpQqRrSsßTtUÜuůúűüV
          vWwXxYyZz˙"
CLAAMPM="AM","PM"
CLAMONTH="January","February","March","April","May","June","July","August","September","October",
         "November","December"
CLAMON="Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"
CLADIGRAPH="ĆAe,ćae"
CLACASE="ÄĹĆÇÉŃÖÜ","äĺćçéńňü"
CLABUTTON="OK","&Si","&No","&Abortar","&Volveratratar","&Ignora","Cancelar","&Ayuda"
CLAMSG2="No File Found"
```

Widoki

VIEW (deklaruje plik "wirtualny")

```
label    VIEW( primary file) [,FILTER( )] [,ORDER( )]
           [PROJECT( )]
           [JOIN( )
             [PROJECT( )]
             [JOIN( )
               [PROJECT( )]
             END]
           END]
         END
```

VIEW Deklaruje widok - plik „wirtualny" - jako kompozyt wzajemnie powiązanych plików.

label Etykieta widoku VIEW.

primary file Etykieta nadrzędnego pliku FILE widoku VIEW.

FILTER Deklaruje wyrażenie filtrujące rekordy w widoku VIEW (PROP:FILTER).

ORDER Deklaruje wyrażenie lub listę wyrażeń stosowanych do definiowania porządku sortowania rekordów w widoku VIEW (PROP:ORDER lub PROP:SQLOrder).

PROJECT Określa pola z pliku nadrzędnego *primary file* lub z powiązanego pliku wskazanego w strukturze JOIN, które będą pobierane do widoku VIEW. Jeśli pominiemy ten parametr, pobierane są wszystkie pola.

JOIN Deklaruje powiązany relacją plik podrzędny.

VIEW deklaruje plik „wirtualny" będący kompozytem powiązanych relacjami plików. Elementy danych zadeklarowane w widoku VIEW nie istnieją fizycznie, gdyż VIEW jest konstrukcją logiczną. VIEW jest spójną metodą adresowania danych fizycznie istniejących w wielu, powiązanych wzajemnie, plikach FILE. W czasie działania programu struktura VIEW nie otrzymuje przydziału pamięci na bufor danych, pola wykorzystywane w widoku VIEW rezydują w buforach rekordów plików, z których są zostają pobierane.

Struktura VIEW musi być w sposób jawny otwarta (OPEN), zanim zaczniemy jej używać, podobnie jak wszystkie nadrzędne i podrzędne pliki, z których ona korzysta.

W celu określenia początkowego porządku sortowania elementów widoku VIEW musi zostać zastosowana instrukcja SET odnosząca się do pliku nadrzędnego w widoku (przed otwarciem widoku - OPEN(view)), bądź tez instrukcja SET(*view*) wywoływana po otwarciu widoku. Sekwencyjny dostęp do elementów widoku zapewniają NEXT(view) oraz PREVIOUS(view).

Struktura VIEW jest w zasadzie przeznaczona do przetwarzania sekwencyjnego. Umożliwia jednak ona także dostęp losowy za pomocą instrukcji REGET. Instrukcja

REGET jest dostępna dla widoku po to, by można było określić rekordy pliku nadrzędnego i plików podrzędnych, które powinny być rekordami bieżącymi w odpowiednich buforach rekordów, po zamknięciu widoku VIEW. Jeśli bezpośrednio przed instrukcją CLOSE(view) nie występuje instrukcja REGET, w buforach rekordów nie będą się znajdowały rekordy wskazywane w widoku.

Sekwencja przetwarzania pliku nadrzędnego i plików podrzędnych pozostaje niezdefiniowana po zamknięciu widoku VIEW. Z tego względu, po zamknięciu widoku, należy użyć SET lub RESET w celu ustalenia porządku sortowania

Struktura danych VIEW jest stworzona w celu ułatwienia dostępu do bazy danych w systemach klient-serwer. Realizuje ona dwie relacyjne operacje jednocześnie: powiązanie „Join" i wybieranie pól „Project". W systemach klient-serwer operacje te są wykonywane przez serwer bazy danych, do klienta są przekazywane tylko ich rezultaty. Dzięki temu bardzo znacząco poprawia się wydajność aplikacji sieciowych. Powiązanie „Join" pobiera dane z różnych plików, na podstawie zdefiniowanych między nimi relacji. Struktura JOIN w strukturze VIEW definiuje operację relacyjną „Join". W jednej strukturze VIEW może występować wiele struktur JOIN, mogą one być jednocześnie wzajemnie zagnieżdżane w celu zapewnienia powiązań wielopoziomowych. Struktura VIEW zawiera domyślnie powiązania typu „left outer join," w których pobierane są wszystkie rekordy pliku nadrzędnego *primary file* widoku, niezależnie od tego, czy w pliku podrzędnym występują powiązane z nimi rekordy, czy też nie. Pola pliku podrzędnego są bezwarunkowo czyszczone (CLEAR) na wartości zerowe lub łańcuchy puste, dla tych rekordów pliku nadrzędnego, do których nie są dowiązane żadne rekordy pliku podrzędnego. Możemy zastąpić domyślne powiązania „left outer join" nadając strukturze JOIN atrybut INNER; otrzymujemy wtedy powiązanie typu „inner join". W powiązaniu takim są pobierane tylko te rekordy pliku nadrzędnego *primary file*, do których są dowiązane rekordy pliku podrzędnego.

Relacyjna operacja "Project" pobiera tylko określone pola *fields* z pliku, a nie całą strukturę rekordu. Tylko pola jawnie zadeklarowane instrukcją PROJECT w strukturze VIEW są wczytywane, jeśli są zadeklarowane instrukcje PROJECT. Dlatego relacyjna operacja "Project" jest automatycznie implementowana w strukturze VIEW. Zawartość dowolnych pól, które nie zostały umieszczone w instrukcji PROJECT, pozostaje niezdefiniowana.

Atrybut FILTER ogranicza widok VIEW do podzbioru rekordów. Wyrażenie FILTER może zawierać dowolne pola jawnie zadeklarowane w strukturze VIEW i ograniczać widok w oparciu o wartość dowolnych z nich. Pozwala to na operowanie filtrem FILTER na wszystkich poziomach powiązań "Join".

Powiązane procedury: BUFFER, CLOSE, FLUSH, OPEN, RECORDS, DELETE, HOLD, NEXT, POSITION, PREVIOUS, PUT, RELEASE, REGET, RESET, SET, SKIP, WATCH

Przykład:

```
Customer    FILE,DRIVER('Clarion'),PRE(Cus)
AcctKey       KEY(Cus:AcctNumber)
Record        RECORD
AcctNumber      LONG
OrderNumber     LONG
Name            STRING(20)
Addr            STRING(20)
City            STRING(20)
State           STRING(20)
Zip             STRING(20)
                . .

Header      FILE,DRIVER('Clarion'),PRE(Hea)
AcctKey       KEY(Hea:AcctNumber)
OrderKey      KEY(Hea:OrderNumber)
Record        RECORD
AcctNumber      LONG
OrderNumber     LONG
ShipToName      STRING(20)
ShipToAddr      STRING(20)
ShipToCity      STRING(20)
ShipToState     STRING(20)
ShipToZip       STRING(20)
              . .

Detail        FILE,DRIVER('Clarion'),PRE(Dtl)
OrderKey        KEY(Dtl:OrderNumber)
Record          RECORD
OrderNumber       LONG
Item              LONG
Quantity          SHORT
                . .

Product       FILE,DRIVER('Clarion'),PRE(Pro)
ItemKey         KEY(Pro:Item)
Record          RECORD
Item              LONG
Description       STRING(20)
Price             DECIMAL(9,2)
                . .

ViewOrder     VIEW(Customer)                          ! deklaruje strukturę VIEW
                PROJECT(Cus:AcctNumber,Cus:Name)
                JOIN(Hea:AcctKey,Cus:AcctNumber)      ! dołącza plik Header
                  PROJECT(Hea:OrderNumber)
                  JOIN(Dtl:OrderKey,Hea:OrderNumber)  ! dołącza plik Detail
                    PROJECT(Det:Item,Det:Quantity)
                    JOIN(Pro:ItemKey,Dtl:Item)        ! dołącza plik Product
                      PROJECT(Pro:Description,Pro:Price)
                    END
                  END
                END
              END
```

Porównaj: JOIN, PROJECT

PROJECT (ustawia pola widoku)

PROJECT(*fields*)

PROJECT Deklaruje pola pobierane do widoku VIEW.

fields Lista pól (włącznie z prefiksami) oddzielonych od siebie przecinkami pliku nadrzędnego lub pliku drugorzędnego (powiązanego) określonego w strukturze JOIN zawierającej deklaracje PROJECT.

Instrukcja **PROJECT** deklaruje pola *fields* wczytywane dla relacyjnej operacji "Project". Relacyjna operacja "Project" pobiera tylko określone pola *fields* z pliku, a nie całą strukturę rekordu.

Instrukcja PROJECT może być zadeklarowana w widoku VIEW lub też wewnątrz jednego z jego elementów JOIN. Jeśli w strukturze VIEW lub JOIN nie występuje żadna deklaracja PROJECT, są pobierane wszystkie pola odpowiedniego pliku.

Jeśli instrukcja PROJECT występuje w strukturze VIEW lub JOIN, pobierane są tylko pola *fields* jawnie zadeklarowane w PROJECT, zawartość pozostałych pól pozostaje niezdefiniowana.

Przykład:

```
Detail          FILE,DRIVER('Clarion'),PRE(Dtl)
OrderKey          KEY(Dtl:OrderNumber)
Record            RECORD
OrderNumber         LONG
Item                LONG
Quantity            SHORT
Description         STRING(20)
                . .

Product         FILE,DRIVER('Clarion'),PRE(Pro)
ItemKey           KEY(Pro:Item)
Record            RECORD
Item                LONG
Description         STRING(20)
Price               DECIMAL(9,2)
                . .

ViewOrder       VIEW(Detail)
                  PROJECT(Det:OrderNumber,Det:Item,Det:Description)
                    JOIN(Pro:ItemKey,Det:Item)
                    PROJECT(Pro:Description,Pro:Price)
                  END
                END
```

JOIN (deklaruje powiązania)

```
JOIN(   | secondary key ,linking fields | ) [, INNER ]
        | secondary file ,expression |
   [PROJECT( )]
   [JOIN( )
     [PROJECT( )]
   END]
END
```

JOIN Deklaruje drugi plik stosowany w powiązaniu relacyjnym.

secondary key Etykieta klucza KEY, który definiuje drugorzędny plik FILE i jego klucz dostępu.

linking fields Lista pól powiązanego pliku, oddzielonych od siebie przecinkami, zawierająca wartości dla klucza *secondary key* wykorzystywane do pobierania rekordów.

secondary file Etykieta drugorzędnego pliku FILE.

expression Stała łańcuchowa zawierająca pojedyncze logiczne wyrażenie służące do wiązania plików (PROP:JoinExpression lub PROP:SQLJoinExpression). Wyrażenie to może zawierać dowolne operatory logiczne i boolowskie.

INNER Wymusza powiązanie „inner join" zamiast domyślnego „left outer join" – jedynymi rekordami wczytywanymi z nadrzędnego pliku *primary file* widoku są te, które posiadają przynajmniej jeden rekord powiązany z nimi w pliku drugorzędnym *secondary file*.

PROJECT Określa pola pliku drugorzędnego powiązanego dołączonego za pomocą struktury JOIN, które mają być dołączone do widoku VIEW. Jeśli pominiemy, pobierane są wszystkie pola.

Struktura **JOIN** deklaruje plik podrzędny dla relacyjnej operacji „Join". Powiązanie „Join" pobiera dane z różnych plików, na podstawie zdefiniowanych między nimi relacji. W jednej strukturze VIEW może występować wiele struktur JOIN, mogą one być jednocześnie wzajemnie zagnieżdżane w celu zapewnienia powiązań wielopoziomowych.

Klucz *secondary key* definiuje klucz dostępu dla pliku podrzędnego. Lista *linking fields* określa nazwy pól w pliku, z którymi jest powiązany plik podrzędny, zawierają one wartości wykorzystywane do pobierania powiązanych rekordów. Dla JOIN znajdującego się bezpośrednio wewnątrz VIEW, pola te pochodzą z pliku nadrzędnego dla widoku. W przypadku zagnieżdżonych JOIN (w innych strukturach JOIN), pola te pochodzą z plików podrzędnych określonych w JOIN, w którym występuje zagnieżdżenie. Nie powiązane pola w kluczu *secondary key* są dopuszczalne, jednak muszą występować w liście składników po polach łączących relacje. Gdy dane są pobierane, w przypadku nie wystąpienia powiązanych rekordów w pliku podrzędnym, odpowiednie pola określone przez PROJECT są czyszczone (wartość zerowa lub łańcuch pusty). Taki typ operacji relacyjnej jest określany jako „left outer join."

Parametr *expression* umożliwia dołączanie plików posiadających powiązane wartości, dla których jednak nie zdefiniowano klucza pozwalającego na zestawienie relacji. Właściwości tablicowe PROP:JoinExpression oraz PROP:SQLJoinExpression w elementach swoich tablic przechowują pozycje struktur JOIN widoku VIEW, których dotyczą. Właściwość PROP:SQLJoinExpression jest wersją właściwości PROP:JoinExpression dostępna tylko dla sterowników SQL-owych. Jeśli pierwszy znak wyrażenia przypisanego do PROP:JoinExpression lub PROP:SQLJoinExpression jest znakiem plusa (+), nowe wyrażenie jest dołączane do już istniejącego.

Przykład:

```
Customer    FILE,DRIVER('Clarion'),PRE(Cus)
AcctKey       KEY(Cus:AcctNumber)
Record        RECORD
AcctNumber      LONG
OrderNumber     LONG
Name            STRING(20)
              . .

Header      FILE,DRIVER('Clarion'),PRE(Hea)
AcctKey       KEY(Hea:AcctNumber)
OrderKey      KEY(Hea:AcctNumber,Hea:OrderNumber)
Record        RECORD
AcctNumber      LONG
OrderNumber     LONG
Total           DECIMAL(11,2)
Discount        DECIMAL(11,2)
OrderDate       LONG
              . .

Detail      FILE,DRIVER('Clarion'),PRE(Dtl)
OrderKey      KEY(Dtl:AcctNumber,Dtl:OrderNumber)
Record        RECORD
AcctNumber      LONG
OrderNumber     LONG
Item            LONG
Quantity        SHORT
              . .

Product     FILE,DRIVER('Clarion'),PRE(Pro)
ItemKey       KEY(Pro:Item)
Record        RECORD
Item            LONG
Description     STRING(20)
Price           DECIMAL(9,2)
              . .

ViewOrder1  VIEW(Header)
              PROJECT(Hea:AcctNumber,Hea:OrderNumber)
                JOIN(Dtl:OrderKey,Hea:AcctNumber,Hea:OrderNumber)   ! dołącza plik Detail
                PROJECT(Dtl:ItemDtl:Quantity)
                  JOIN(Pro:ItemKey,Dtl:Item)                        ! dołącza plik Product
                  PROJECT(Pro:Description,Pro:Price)
              . . .

ViewOrder2    VIEW(Customer)                                        ! deklaruje strukturę VIEW
                JOIN(Header,'Cus:AcctNumber = Hea:AcctNumber AND ' & |
                    ' (Hea:Discount + Hea:Total) * .1 > Hea:Discount')
                  PROJECT(Hea:AcctNumber,Hea:OrderNumber)
                    JOIN(Dtl:OrderKey,Hea:AcctNumber,Hea:OrderNumber)   ! dołącza plik Detail
                    PROJECT(Dtl:ItemDtl:Quantity)
                . . .
```

Porównaj: INNER

Kolejki

QUEUE (deklaruje strukturę kolejki)

```
label       QUEUE( [ group ] ) [,PRE] [,STATIC] [,THREAD] [,TYPE] [,BINDABLE] [,EXTERNAL] [,DLL]
fieldlabel      variable [,NAME( )]
            END
```

QUEUE Deklaruje kolejkę elementów przechowywaną w pamięci.

label Nazwa kolejki QUEUE.

group Etykieta wcześniej zdeklarowanej struktury GROUP lub QUEUE, która jest dziedziczona do bieżącej definicji. Mogą to być struktury GROUP lub QUEUE posiadające, bądź nie posiadające atrybut TYPE.

PRE Deklaruje prefiks *fieldlabel* dla struktury.

STATIC Deklaruje kolejkę QUEUE, lokalną dla danej procedury, dla której bufor jest przydzielony w pamięci statycznej.

THREAD Pamięć dla kolejki jest przydzielana oddzielnie dla każdego wątku. Również, w sposób niejawny nadaje atrybut STATIC lokalnym zmiennym procedury.

TYPE Określa, że kolejka QUEUE jest jedynie definicją typu dla innych deklaracji kolejek.

BINDABLE Powoduje, że wszystkie pola kolejki mogą być wykorzystywane w budowaniu wyrażeń dynamicznych.

EXTERNAL Określa, że kolejka QUEUE jest zdefiniowana, a pamięć dla niej przydzielona, w zewnętrznej bibliotece.

DLL Określa, że kolejka QUEUE jest zdefiniowana w bibliotece dynamicznej DLL. Występuje w połączeniu z atrybutem EXTERNAL.

fieldlabel Nazwa zmiennej *variables* w kolejce.

variable Deklaracje danych. Suma pamięci wymagana dla wszystkich deklaracji danych *variables* w kolejce nie może być większa niż 65.520 bajtów w aplikacjach 16-bitowych i 4MB w aplikacjach 32-bitowych.

QUEUE deklaruje kolejkę (listę elementów) przechowywaną w pamięci. Etykieta *label* kolejki QUEUE jest wykorzystywana przez instrukcje i procedury operujące na kolejce i jej elementach. Gdy kolejka QUEUE jest używana w instrukcjach przypisania, wyrażeniach, w listach parametrów – jest traktowana dana typu GROUP.

Struktura kolejki QUEUE zadeklarowanej z parametrem *group* rozpoczyna się tą samą strukturą, co *group* o określonej nazwie; kolejka QUEUE dziedziczy pola nazwanej grupy *group*. Kolejka QUEUE może również zawierać swoje własne deklaracje *declarations*, które następują po dziedziczonych polach. Jeśli kolejka QUEUE nie ma zawierać żadnych dodatkowych pól, nazwa grupy *group*, od której ona dziedziczy może być stosowana jako typ danych, bez konieczności użycia słowa kluczowego QUEUE oraz END.

Kolejka QUEUE może być traktowana jako plik zlokalizowany w pamięci (zamiast na dysku); wewnętrznie jest ona zaimplementowana jako tablica dynamiczna składająca się z dowolnej liczby elementów. Gdy kolejka QUEUE zostaje zadeklarowana, jest jej przydzielany bufor danych (podobnie, jak w przypadku pliku). Każdy element kolejki ma zmienną długość QUEUE, gdyż jest kompresowany przez polecenia ADD, czy PUT w celu maksymalnego zmniejszenia pamięci zajmowanej przez kolejkę. W momencie pobrania elementu kolejki za pomocą GET, następuje jego dekompresja. Bufor danych dla kolejki będącej lokalną dla danej procedury (zadeklarowanej w jej sekcji danych lokalnych) jest alokowany na stosie; o ile kolejka nie posiada atrybutu STATIC lub jej element nie jest zbyt duży. Pamięć przydzielana elementom kolejki lokalnej dla procedury (i nie posiadającej atrybutu STATIC) jest zwalniana w momencie użycia funkcji FREE lub wyjścia (RETURN) z procedury – wtedy kolejka jest automatycznie zwalniana.

W przypadku kolejek globalnych, kolejek modułu, czy też kolejek lokalnych ale z atrybutem STATIC, bufor danych jest alokowany w pamięci statycznej, a dane w nim przechowywane są trwałe i nie zmieniają się pomiędzy wywołaniami procedury. Pamięć przydzielona elementom kolejki pozostaje zaalokowana dopóty, dopóki w sposób świadomy nie użyjemy funkcji FREE w celu zwolnienia (usunięcia z pamięci) kolejki QUEUE.

Zmienne *variables* w buforze kolejki QUEUE nie są automatycznie inicjowane żadnymi wartościami; musimy je przypisać w sposób bezpośredni. Nie można przyjmować, że posiadają one wartości puste lub zerowe przed pierwszym do nich dostępem.

Jak tylko element zostaje dołączony do kolejki, jest mu dynamicznie przydzielana pamięć, a następnie kopiowane są do niej i kompresowane dane znajdujące się w buforze kolejki. Gdy element jest usuwany z kolejki QUEUE, pamięć przez niego zajmowana jest zwalniana. Maksymalna liczba elementów w kolejce to teoretycznie 2^{32}, ale w rzeczywistości zależy od rozmiarów posiadanej pamięci wirtualnej. Rozmiar aktualnie wykorzystywanej pamięci przez każdy element kolejki zależy od współczynnika kompresji danych narzuconego przez bibliotekę runtime.

Kolejka QUEUE z atrybutem BINDABLE pozwala na zastosowanie wszystkich jej pól w wyrażeniach dynamicznych, bez konieczności bindowania każdego z nich za pomocą instrukcji BIND (dopuszczalne jest też u życie BIND(kolejka) w celu zbindowania wszystkich jej pól). Zawartość atrybutu NAME każdego z pól jest nazwą logiczną stosowaną w wyrażeniach dynamicznych. Jeśli nie występuje atrybut NAME, stosowana jest etykieta pola włącznie z prefiksem. W pliku .EXE jest alokowany obszar na nazwy zbindowanych zmiennych. Rozmiar programu staje się przez to większy i pociąga to za sobą zwiększenie zapotrzebowania na pamięć. Z tego względu atrybut BINDABLE powinien być stosowany tylko wtedy, gdy w wyrażeniach dynamicznych korzystamy praktycznie ze wszystkich pól elementu kolejki.

Kolejka QUEUE posiadająca atrybut TYPE nie jest alokowana w pamięci, stanowi ona jedynie definicję typu dla kolejek, które maja być przekazywane do procedur w postaci ich parametrów. Dzięki temu umożliwiamy procedurze bezpośrednie adresowanie pól stanowiących składniki elementów przekazanej kolejki QUEUE. Deklaracja parametru w instrukcji PROCEDURE przydziela lokalny prefiks dla przekazanej kolejki QUEUE. Na przykład, PROCEDURE(LOC:PassedGroup) deklaruje procedurę stosującą prefiks LOC: (wspólnie z indywidualnymi nazwami pól określonymi w definicji typu) w celu umożliwienia adresowania pól składowych kolejki QUEUE przekazanej jako parametr.

Procedury WHAT i WHERE umożliwiają do pól na podstawie ich względnej pozycji w strukturze QUEUE.

Powiązane procedury: ADD, CHANGES, DELETE, FREE, GET, POINTER, PUT, RECORDS, SORT

Przykład:

```
NameQue   QUEUE,PRE(Nam)                  ! deklaruje kolejkę
Name        STRING(20)
Zip         DECIMAL(5,0),NAME('SortField')
          END                             ! koniec struktury

NameQue2  QUEUE(NameQue),PRE(Nam2)        ! kolejka dziedzicząca pola Name i Zip
Phone       STRING(10)                    ! I wprowadzająca pole Phone
          END
NameQue3  NameQue2                        ! deklaruje drugą kolejkę z tą samą
                                          ! strukturą, co NameQue2
```

Porównaj: PRE, STATIC, NAME, FREE, THREAD, WHAT, WHERE

5 – ATRYBUTY DEKLARACJI

Atrybuty zmiennych i egzemplarzy

AUTO (niezainicjowana zmienna lokalna)

AUTO

Atrybut **AUTO** umożliwia przydzielenie pamięci dla zmiennej, zadeklarowanej wewnątrz procedury, na stosie – bez inicjowania jej wartości. Jeśli zmienna nie posiada atrybutu AUTO, wtedy jest inicjowana z wartością 0 (w przypadku zmiennych numerycznych) lub wypełniana spacjami (w przypadku zmiennych łańcuchowych). Atrybutu AUTO używamy wtedy, gdy nie chcemy by zmienna była inicjowana zerem lub spacjami, innymi słowy, gdy mamy zamiar nadać jej inne wartości początkowe. Dzięki temu zaoszczędzamy czas, który byłby niezbędny na wykonanie automatycznej inicjalizacji zmiennej.

Przykład:

```
SomeProc    PROCEDURE
SaveCustID  LONG,AUTO        ! niezainicjowana zmienna lokalna
```

Porównaj: Deklaracje danych i przydział pamięci

BINARY (zawartość binarna pola memo)

BINARY

Atrybut **BINARY** (PROP:BINARY) użyty w deklaracji pola MEMO lub BLOB określa, że pole takie może służyć do przechowywania danych składających się nie tylko ze znaków ASCII. Ten atrybut stosuje się zazwyczaj wtedy, gdy w polu MEMO lub BLOB przechowujemy zawartość pliku graficznego wyświetlanego na ekranie w polu IMAGE. Dla pól MEMO lub BLOB z atrybutem BINARY nie stosuje się konwersji OEM. Niektóre sterowniki plików (Clarion, Btrieve, xBase) zakładają, że dane w polu MEMO lub BLOB z atrybutem BINARY są uzupełniane zerami, podczas, gdy pola bez atrybutu BINARY są uzupełniane spacjami.

Przykład:

```
Names    FILE,DRIVER('Clarion'),PRE(Nam)
NbrKey     KEY(Nam:Number)
Picture    MEMO(48000),BINARY              ! binarne memo - 48,000 bajtów
Rec        RECORD
Number       SHORT
           . .
```

Porównaj: MEMO, BLOB, IMAGE. OEM

BINDABLE (ustawienie zmiennej dla wyrażeń dynamicznych)

BINDABLE

Atrybut **BINDABLE** zadeklarowany dla struktur GROUP, QUEUE, FILE lub VIEW powoduje, że wszystkie ich pola stają się dostępne do użycia w wyrażeniach dynamicznych (runtime expression) budowanych w czasie działania programu. Zawartość atrybutu NAME każdej ze zmiennych staje się logiczną nazwą stosowaną w wyrażeniu dynamicznym. Jeżeli atrybut NAME nie występuje, wykorzystywana jest etykieta zmiennej (łącznie z jej prefiksem). Dla nazw wszystkich zmiennych występujących w strukturze jest rezerwowane miejsce w pliku .EXE. Oczywiście prowadzi to do zwiększenia jego rozmiaru, a co za tym idzie, również zwiększenia pamięci operacyjnej przez niego zajmowanej. Z tego powodu atrybut BINDABLE powinien być używany raczej tylko wtedy, gdy zdecydowana większość pól danej struktury będzie używana w wyrażeniach dynamicznych. Należy pamiętać, że instrukcja BIND(group) musi być zastosowana w kodzie wykonywalnym zanim nastąpi odwołanie do któregokolwiek pola danej struktury.

Przykład:

```
Names     QUEUE,BINDABLE                ! struktura zbindowana
Name        STRING(20)
FileName    STRING(8),NAME('FName')     ! dynamiczna nazwa: FName
Dot         STRING(1)                   ! dynamiczna nazwa: Dot
Extension   STRING(3),NAME('EXT')       ! dynamiczna nazwa: EXT
          END
  CODE
    BIND(Names)

Names     FILE,DRIVER('Clarion'),BINDABLE  ! struktura zbindowana
Record      RECORD
Name          STRING(20)
FileName      STRING(8),NAME('FName')    ! dynamiczna nazwa: FName
Dot           STRING(1)                  ! dynamiczna nazwa: Dot
Extension     STRING(3),NAME('EXT')      ! dynamiczna nazwa: EXT
            . .
  CODE
    OPEN(Names)
    BIND(Names)

FileNames  GROUP,BINDABLE                ! grupa zbindowana
FileName     STRING(8),NAME('FILE')      ! dynamiczna nazwa: FILE
Dot          STRING('.')                 ! dynamiczna nazwa: Dot
Extension    STRING(3),NAME('EXT')       ! dynamiczna nazwa: EXT
           END
```

Porównaj: BIND, UNBIND, EVALUATE

CREATE (umożliwienie tworzenia pliku)

CREATE

Atrybut **CREATE** (PROP:CREATE) zastosowany w deklaracji FILE umożliwia programowi utworzenie pliku dyskowego jeśli on nie istnieje; wygenerowany kod źródłowy stosuje do tego instrukcję CREATE. Wszystkie informacje o danym pliku są wtedy umieszczane w kodzie wykonywalnym programu.

Przykład:

```
Names   FILE,DRIVER('Clarion'),CREATE          ! deklaracja pliku umożliwiająca tworzenie
Rec       RECORD
Name        STRING(20)
          . .
```

DIM (ustawienie rozmiaru tablicy)

DIM(dimension,...,dimension)

DIM Deklaruje zmienną w postaci tablicy.

Dimension Stała numeryczna określająca liczbę elementów danego wymiaru tablicy.

Atrybut **DIM** deklaruje zmienną w postaci tablicy. Zmienna jest „powtarzana" określoną przez parametry *dimension* liczbę razy. Tablice wielowymiarowe można traktować jako zagnieżdżenia. Każdy wymiar (*dimension*) w tablicy posiada odpowiedni indeks. Zatem odwołanie do elementu trzywymiarowej tablicy wymaga użycia trzech indeksów. W zasadzie nie ma ograniczenia na liczbę wymiarów, jednak całkowity rozmiar tablicy nie może przekraczać 65520 bajtów w przypadku aplikacji 16-bitowej; w przypadku aplikacji 32-bitowych nie ma żadnych ograniczeń.

Indeksy identyfikują jednoznacznie, do którego elementu tablicy się odwołujemy. Oddzielamy je od siebie za pomocą znaku przecinka i umieszczamy je w nawiasach kwadratowych występujących zaraz po etykiecie zmiennej tablicowej (bez odstępu). Indeks może być stałą numeryczną, wyrażeniem lub funkcją. Do całej tablicy odwołujemy się poprzez jej etykietę, bez indeksów.

Specjalnym przypadkiem jest tablica typu GROUP. Jej każdy poziom zagnieżdżenia dołącza indeks do grupy GROUP. Do pól zadeklarowanych wewnątrz struktury GROUP odwołujemy się w oparciu o standardową składnię dołączając indeks właściwy dla danego poziomu zagnieżdżenia. Dokładniej zostało to pokazane w oparciu o przykład przedstawiony poniżej.

Przykład:

```
Scr   GROUP                    ! znaki na ekranie w trybie znakowym
Row    GROUP,DIM(25)           ! 25 wierszy
Pos      GROUP,DIM(80)         ! 2000 pozycji
Attr       BYTE                ! bajt atrybutu
Char       BYTE                ! bajt znaku
       . . .                   ! koniec grupy
! w grupie powyżej:
! Scr jest grupą 4,000 bajtów
! Scr.Row jest grupą 4,000 bajtów
! Scr.Row[1] jest grupą 160 bajtów
! Scr.Row[1].Pos jest grupą 160 bajtów
! Scr.Row[1].Pos[1] jest grupą 2 bajtów
! Scr.Row[1].Pos[1].Attr jest pojedynczym bajtem
! Scr.Row[1].Pos[1].Char jest pojedynczym bajtem

Month    STRING(10),DIM(12)        ! rozmiar – 12 elementów
  CODE
    CLEAR(Month)                   ! przypisanie spacji do całej tablicy
    Month[1] = 'January'           ! ładowanie nazw miesięcy do tablicy
    Month[2] = 'February'
    Month[3] = 'March'
```

Porównaj: MAXIMUM, Lista parametrów prototypu (przekazywanie tablic)

DLL (ustawienie zmiennej jako zdefiniowanej w bibliotece DLL)

DLL([*flag*])

DLL Deklaruje zmienną, FILE, QUEUE, GROUP lub CLASS zdefiniowaną w zewnętrznej bibliotece .DLL.

flag Stała numeryczna, ekwiwalent lub definicja projektu określająca, czy atrybut jest aktywny, czy też nie. Jeśli *flag* wynosi zero, atrybut nie jest aktywny – tak, jakby nie występował. Jeśli *flag* jest wartością różną od zera, atrybut jest aktywny.

Atrybut **DLL** określa, że dana deklaracja (może to być deklaracja dowolnej zmiennej, pliku FILE, kolejki QUEUE, grupy GROUP, czy klasy CLASS) znajduje się w bibliotece dynamicznej .DLL. Deklaracja z atrybutem DLL musi dodatkowo posiadać atrybut EXTERNAL. Atrybut DLL jest wymagany dla aplikacji 32-bitowych, gdyż biblioteki .DLL są przemieszczane w 32-bitowej płaskiej, przestrzeni adresowej. Wymaga to od kompilatora stosowania dodatkowych odnośników do adresowania zmiennych.

Atrybut DLL nie jest poprawny dla zmiennych zadeklarowanych wewnątrz struktur FILE, QUEUE, CLASS i GROUP.

Deklaracje we wszystkich bibliotekach i programach muszą być IDENTYCZNE (oczywiście za wyjątkiem atrybutów EXTERNAL oraz DLL). Jeżeli nie będą takie same, dane mogą zostać uszkodzone. Wszelkie niezgodności pomiędzy bibliotekami nie mogą zostać wykryte przez kompilator i linker, z tego powodu odpowiedzialność za zachowanie zgodności deklaracji spoczywa na programiście. Gdy używamy atrybutów EXTERNAL i DLL w deklaracjach współdzielonych przez biblioteki .DLL i pliki wykonywalne .EXE, tylko jedna biblioteka .DLL powinna zawierać deklarację danej zmiennej, pliku FILE, klasy CLASS, czy kolejki QUEUE pozbawioną atrybutów EXTERNAL i DLL. Wszystkie pozostałe biblioteki .DLL (i pliki wykonywalne .EXE) powinny zawierać deklarację tej zmiennej, pliku FILE, klasy CLASS, czy kolejki QUEUE z atrybutami EXTERNAL i DLL. Zapewnia to, że istnieje tylko jedna alokacja pamięci przeznaczona dla tej zmiennej, pliku, klasy, czy kolejki i wszystkie pozostałe biblioteki i programy odwołują się właśnie do niej. Z tego powodu sugeruje się, by duże systemy złożone z wielu bibliotek dynamicznych i wielu programów wykonywalnych posiadały jedną wydzieloną bibliotekę .DLL, w której deklaruje się pliki i zmienne globalne wykorzystywane przez wszystkie z nich. Dzięki temu uzyskamy centralną bibliotekę zarządzającą aktualnymi definicjami plików. Jest ona konsolidowana ze wszystkimi programami z nich korzystającymi. Wszystkie pozostałe biblioteki .DLL i pliki wykonywalne .EXE tworzące nasz system powinny deklarować wspólne definicje z atrybutami EXTERNAL i DLL.

Przykład:

```
TotalCount  LONG,EXTERNAL,DLL(dll_mode)          ! zmienna zadeklarowana w zewnętrznym .DLL
Cust        FILE,PRE(Cus),EXTERNAL(''),DLL(1)    ! plik zdefiniowany w module PROGRAM .DLL
CustKey       KEY(Cus:Name)
Record        RECORD
Name            STRING(20)
            . .
DLLQueue    QUEUE,PRE(Que),EXTERNAL,DLL(1)       ! kolejka zadeklarowana w zewnętrznym .DLL
TotalCount    LONG
            END
```

Porównaj: EXTERNAL

DRIVER (określenie typu struktury danych)

DRIVER(*filetype* [, *driver string*])

DRIVER Wskazuje sterownik (format) dla pliku danych.

filetype Łańcuch zawierający nazwę sterownika pliku danych (Btrieve, Clarion, itp.).

driver string Stała łańcuchowa lub zmienna zawierająca dodatkowe instrukcje kierowane do sterownika pliku danych. Wszystkie właściwe wartości są wyszczególnione w opisach dokumentacji poszczególnych sterowników plików danych.

Atrybut **DRIVER** (PROP:DRIVER) określa, który z dostępnych sterowników plików danych ma być stosowany dla danego pliku. Atrybut DRIVER jest wymagany we wszystkich deklaracjach plików FILE. Programy napisane w Clarionie wykorzystują sterowniki plików danych przy fizycznym dostępie do danych. Sterownik pliku danych pełni role tłumacza pomiędzy programem napisanym w Clarionie a systemem plików, eliminując trudności związane z różnymi zestawami poleceń dostępu w różnych systemach plików. Dzięki temu program może uzyskiwać dostęp do plików różnych formatów bez zmiany składni poleceń Clariona. Konkretna specyfikacja metody, w oparciu o którą uzyskuje się dostęp do danych, zależy od rodzaju użytego sterownika plików danych. Należy pamiętać, że niektóre z poleceń nie będą dostępne dla wybranych systemów plików z uwagi na ich ograniczenia. Każdy sterownik plików danych jest szczegółowo udokumentowany w podręczniku *User's Guide*. Są tam wymienione wszelkie niedostępne dla danego formatu polecenia, atrybuty deklaracji pliku, typy danych itp.

Przykład:

```
Names   FILE,DRIVER('Clarion')          ! początek deklaracji pliku
Record    RECORD
Name        STRING(20)
          . .
```

DUP (dopuszczenie powtórzeń w kluczu)

DUP

Atrybut **DUP** (PROP:DUP) użyty w deklaracji klucza KEY dopuszcza umieszczanie w kluczu rekordów o powtarzających się wartościach składników klucza. Jeżeli atrybut DUP jest pominięty próba utworzenia nowego (ADD) lub zaktualizowania istniejącego (PUT) rekordu, w którym zostaną powtórzone wartości składników klucza zakończy się niepowodzeniem i wygenerowaniem komunikatu o błędzie "Creates Duplicate Key". Podczas sekwencyjnego przetwarzania rekordów wg klucza, rekordy o powtórzonych wartościach składników klucza są pobierane w oparciu o fizyczne ich położenie w pliku. Instrukcje GET i SET na ogół pobierają pierwszy rekord w zbiorze zduplikowanych rekordów. Atrybut DUP nie jest potrzebny w deklaracjach kluczy typu INDEX gdyż te zawsze dopuszczają duplikowanie wartości składników klucza.

Przykład:

```
Names    FILE,DRIVER('Clarion'),PRE(Nam)
NameKey    KEY(Nam:Name),DUP              ! deklaracja klucza, dopuszcza duplikaty
NbrKey     KEY(Nam:Number)                ! deklaracja klucza, nie dopuszcza duplikatów
Rec        RECORD
Name         STRING(20)
Number       SHORT
           . .
```

Porównaj: KEY, GET, SET

ENCRYPT (szyfrowanie pliku danych)

ENCRYPT

Atrybut **ENCRYPT** (PROP:ENCRYPT) jest stosowany w połączeniu z atrybutem OWNER w celu zaszyfrowania informacji zapisywanych w pliku danych. Po zaszyfrowaniu dane są trudne do odczytania nawet za pomocą edytorów binarnych.

Przykład:

```
Names  FILE,DRIVER('Clarion'),OWNER('Clarion'),ENCRYPT
Record   RECORD
Name     STRING(20)
         . .
```

Porównaj: OWNER, EXTERNAL

EXTERNAL (ustawienie jako definicji zewnętrznej)

EXTERNAL(*member*)

EXTERNAL Określa, że dana zmienna, plik FILE, referencja pliku &FILE, kolejka QUEUE, grupa GROUP, czy klasa CLASS jest zdefiniowana w bibliotece zewnętrznej.

member Stała łańcuchowa (prawidłowa tylko w deklaracjach plików FILE lub referencji plików &FILE) określająca nazwę pliku (bez rozszerzenia), w którym jest zapisany moduł MEMBER zawierający aktualną definicję pliku FILE (ta definicja - bez atrybutu EXTERNAL). Jeżeli plik FILE jest zdefiniowany w module PROGRAM, łańcuch *member* musi być pusty (‘’).

Atrybut **EXTERNAL** określa, że zmienna, plik FILE, kolejka QUEUE, grupa GROUP lub klasa CLASS, dla której został użyty, jest zdefiniowana w bibliotece zewnętrznej. Jednakże może ona być również zadeklarowana w kodzie źródłowym Clarion, z tym, że wtedy nie jest jej przydzielana pamięć – pamięć ta jest przydzielana w bibliotece zewnętrznej. Dzięki temu program napisany w Clarionie ma możliwość uzyskania dostępu do dowolnej zmiennej, pliku, kolejki, grupy, czy klasy zadeklarowanej jako publiczna w bibliotece zewnętrznej. Atrybut EXTERNAL nie jest prawidłowy dla zmiennych zadeklarowanych w strukturach FILE, QUEUE, GROUP lub CLASS.

Gdy używamy atrybutu EXTERNAL w postaci EXTERNAL(*member*) w celu zadeklarowania pliku FILE współdzielonego przez wiele bibliotek (.LIB, .DLL oraz .EXE), tylko jedna z nich powinna zawierać definicje pliku FILE pozbawioną atrybutu EXTERNAL. Wszystkie pozostałe biblioteki (oraz pliki .EXE) powinny zawierać deklaracje pliku FILE z atrybutem EXTERNAL. Zapewniamy w ten sposób przydzielenie pamięci dla jednego bufora rekordu danego pliku dostępnego dla wszystkich bibliotek i programów.

Deklaracje we wszystkich bibliotekach (i plikach .EXE) muszą być identyczne (za wyjątkiem atrybutów EXTERNAL i DLL). Dla przykładu, deklaracje pliku FILE we wszystkich bibliotekach i programach, które odwołują się do wspólnych plików muszą zawierać dokładnie te same klucze, pola memo, deklaracje pól. Co więcej, muszą one występować w jednakowej kolejności. W przypadku rozbieżności, mogą wystąpić uszkodzenia danych.

Wszelkie niezgodności pomiędzy bibliotekami nie mogą zostać wykryte przez kompilator i linker, z tego powodu odpowiedzialność za zachowanie zgodności deklaracji spoczywa na programiście.

Plik z atrybutem EXTERNAL nie może mieć atrybutów OWNER, ENCRYPT lub NAME. Te atrybuty mogą wystąpić jedynie wtedy, gdy deklarujemy plik bez atrybutu EXTERNAL.

Zaleca się, by duże systemy złożone z wielu bibliotek dynamicznych i wielu programów wykonywalnych posiadały jedną wydzieloną bibliotekę .DLL, w której deklaruje się pliki i zmienne globalne wykorzystywane przez wszystkie z nich. Dzięki temu uzyskamy centralną bibliotekę zarządzającą aktualnymi definicjami plików. Jest

ona konsolidowana ze wszystkimi programami z nich korzystającymi. Wszystkie pozostałe biblioteki .DLL i pliki wykonywalne .EXE tworzące nasz system powinny deklarować wspólne definicje z atrybutem EXTERNAL.

Przykład:

```
    PROGRAM
    MAP
    MODULE('LIB.LIB')
AddCount    PROCEDURE                          ! procedura biblioteki zewnętrznej
    . .
TotalCount    LONG,EXTERNAL                    ! zmienna zadeklarowana w bibliotece zewnętrznej
Cust      FILE,PRE(Cus),EXTERNAL('')           ! plik zdefiniowany w module PROGRAM,
CustKey   KEY(Cus:Name)                        ! którego .LIB jest linkowany do programu
Record    RECORD
Name        STRING(20)
          . .
Contact     FILE,PRE(Con),EXTERNAL('LIB01')    ! plik zdefiniowany w module MEMBER module,
ContactKey  KEY(Con:Name)                      ! którego .LIB jest linkowany do programu
Record        RECORD
Name          STRING(20)
              . .
! plik LIB.CLW zawiera:
    PROGRAM
    MAP
    MODULE('LIB01')
AddCount    PROCEDURE                          ! procedura biblioteczna
    . .
TotalCount  LONG                               ! definicja zmiennej TotalCount
Cust      FILE,PRE(Cus)                        ! definicja Cust File, gdzie jest alokowany
CustKey   KEY(Cus:Name)                        ! bufor rekordu
Record      RECORD
Name        STRING(20)
          . .
    CODE
    ! kod wykonywalny...
    ! plik LIB01.CLW zawiera:
    MEMBER('LIB')
Contact     FILE,PRE(Con)                      ! definicja pliku Contact, gdzie jest
ContactKey  KEY(Con:Name)                      ! alokowany bufor rekordu
Record        RECORD
Name          STRING(20)
              . .
AddCount    PROCEDURE
    CODE
      TotalCount += 1
```

Porównaj: DLL, NAME

FILTER (ustawienie wyrażenia filtrującego widok)

FILTER(*expression*)

FILTER Określa wyrażenie *expression* definiujące filtr stosowany przy pobieraniu rekordów do wyświetlania w widoku VIEW.

expression Stała łańcuchowa zawierająca wyrażenie logiczne.

Atrybut **FILTER** (PROP:FILTER) określa wyrażenie filtrujące rekordy (*expression*) wykorzystywane przy tworzeniu widoku VIEW. Parametr *expression* może odnosić się do dowolnego pola widoku VIEW, na wszystkich poziomach struktur JOIN. Gdy rezultat wyrażenia *expression* jest prawdziwy, dany rekord jest włączany do widoku VIEW. Wyrażenie *expression* może być zbudowane w oparciu o dowolne wyrażenie logiczne języka Clarion. Wartość wyrażenia *expression* jest obliczana w czasie działania programu (podobnie, jak procedury EVALUATE), z tego powodu wszystkie zmienne stosowane w wyrażeniu muszą być bindowane (BIND).

Przykład:

```
BRW1::View:Browse   VIEW(Members)
                      PROJECT(Mem:MemberCode,Mem:LastName,Mem:FirstName)
                    END
KeyValue            STRING(20)

! pobiera tylko zamówienia klienta 9999 o d zamówienia numer 100

ViewOrder   VIEW(Customer),FILTER('Cus:AcctNumber = 9999 AND Hea:OrderNumber > 100')
              PROJECT(Cus:AcctNumber,Cus:Name)
              JOIN(Hea:AcctKey,Cus:AcctNumber)              ! dołącza plik Header
                PROJECT(Hea:OrderNumber)
                JOIN(Dtl:OrderKey,Hea:OrderNumber)          ! dołącza plik Detail
                  PROJECT(Det:Item,Det:Quantity)
                  JOIN(Pro:ItemKey,Dtl:Item)                ! dołącza plik Product
                    PROJECT(Pro:Description,Pro:Price)
            ....

  CODE
  BIND('KeyValue',KeyValue)
  BIND(Mem:Record)
  KeyValue = 'Smith'
  BRW1::View:Browse{PROP:Filter} = 'Mem:LastName = KeyValue'     ! określa filtr
  OPEN(BRW1::View:Browse)                                       ! otwiera widok
  SET(BRW1::View:Browse)                                        ! i ustawia sie na początku filtrowanego
  CODE                                                          ! i uporządkowanego zbioru wynikowego
  OPEN((Customer,22h); OPEN((Header,22h); OPEN((Product,22h); OPEN(Detail,22h)
  BIND('Cus:AcctNumber',Cus:AcctNumber)
  BIND('Hea:OrderNumber',Hea:OrderNumber)
  SET(Cus:AcctKey)
  OPEN(ViewOrder)
  LOOP
    NEXT(ViewOrder)
    IF ERRORCODE() THEN BREAK.
      ! przetwarza właściwy rekord
    END
    UNBIND('Cus:AcctNumber',Cus:AcctNumber)
    UNBIND('Hea:AcctNumber',Hea:AcctNumber)
    CLOSE(Header); CLOSE(Customer); CLOSE(Product); CLOSE(Detail)
```

Porównaj: BIND, UNBIND, EVALUATE

INNER (ustawienie operacji typu inner join)

INNER

Atrybut **INNER** (PROP:INNER) określa, że struktura JOIN deklaruje „inner join" zamiast domyślnego „left outer join." Domyślnie struktura VIEW stosuje „left outer join," przy którym wszystkie rekordy głównego pliku widoku VIEW są wczytywane niezależnie od tego, czy wskazane pliki znajdujące się w relacji zawierają powiązane rekordy, czy też nie. Nadając atrybut INNER powodujemy zastosowanie „inner join", przy którym wczytywane są tylko te rekordy pliku głównego, dla których wskazane pliki pozostające z nim w relacji, zawierają powiązane rekordy. Powiązania typu „inner join" są zazwyczaj bardziej efektywne, niż powiązania „outer join".

Właściwość PROP:INNER jest tablicą określającą dla widoku VIEW istnienie lub brak atrybutu INNER dla struktury JOIN. Każdy element tablicy daje w rezultacie ('1') jeśli JOIN posiada atrybut INNER lub łańcuch pusty ('') w przeciwnym przypadku. Struktury JOIN są wymieniane dla widoku VIEW począwszy od 1 w kolejności ich występowania w strukturze VIEW.

Przykład:

```
AView VIEW(BaseFile)
        JOIN(ParentFile,'BaseFile.parentID = ParentFile.ID')          ! JOIN 1
          JOIN(GrandParent.PrimaryKey, ParentFile.GrandParentID)      ! JOIN 2
          END
        END
        JOIN(OtherParent.PrimaryKey,BaseFile.OtherParentID),INNER     ! JOIN 3
        END
      END

  ! AView{PROP:Inner,1} zwraca ''
  ! AView{PROP:Inner,2} zwraca ''
  ! AView{PROP:Inner,3} zwraca '1'

ViewOrder  VIEW(Customer),ORDER('-Hea:OrderDate,Cus:Name')
             PROJECT(Cus:AcctNumber,Cus:Name,Cus:Zip)
             JOIN(Hea:AcctKey,Cus:AcctNumber),INNER          ! wewnętrzne dołączenie Header
               PROJECT(Hea:OrderNumber,Hea:OrderDate)        ! pobiera tylko klientów z zamówieniami
               JOIN(Dtl:OrderKey,Hea:OrderNumber),INNER      ! wewnętrzne dołączenie Detail
                 PROJECT(Det:Item,Det:Quantity)              ! jest naturalne I efektywne
                 JOIN(Pro:ItemKey,Dtl:Item),INNER            ! wewnętrzne dołączenie Product
                   PROJECT(Pro:Description,Pro:Price)        ! jest naturalne I efektywne
                   END
                 END
               END
             END
```

Porównaj: JOIN

LINK (określenie klasy linkowanej do projektu)

LINK(*linkfile*, [*flag*])

LINK Określa nazwę pliku, który jest dołączany do listy powiązań bieżącego projektu.

linkfile Stała łańcuchowa stanowiąca nazwę pliku bez rozszerzenia (automatycznie jest przyjmowane rozszerzenie .OBJ). Standardowo powinna być to ta sama nazwa, co w przypadku parametru atrybutu MODULE, z tym, że wskazuje na plik .LIB lub .OBJ.

flag Stała numeryczna, ekwiwalent, lub definicja projektu określająca, czy atrybut jest aktywny, czy też nie. Jeżeli *flag* jest równe 0 lub zostało pominięte, atrybut nie jest aktywny- tak, jakby nie występował. Jeżeli *flag* jest dowolną wartością różną od 0, atrybut jest aktywny.

Atrybut **LINK** struktury CLASS określa nazwę pliku (*linkfile*), który należy dołączyć do listy konsolidowanych plików kompilatora. LINK jest właściwy tylko dla struktur CLASS.

Przykład:

```
OneClass   CLASS,MODULE('OneClass'),LINK('OneClass',1)      ! łączenie w OneClass.OBJ
LoadIt       PROCEDURE
ComputeIt    PROCEDURE
           END
```

Porównaj: CLASS, MEMBER, MODULE

MODULE (określenie modułu zawierającego definicję klasy)

MODULE(*sourcefile*)

MODULE Określa nazwę modułu MEMBER lub pliku zawierającego bibliotekę zewnętrzną.

sourcefile Stała łańcuchowa. Jeżeli plik źródłowy zawiera kod źródłowy zapisany w języku programowania Clarion, parametr wskazuje nazwę pliku (bez rozszerzenia) zawierającego kod źródłowy procedur. Jeśli plik źródłowy jest zewnętrzną biblioteką, parametr może wskazywać dowolny, unikalny identyfikator.

Atrybut **MODULE** zastosowany w definicji struktury CLASS określa nazwę modułu MEMBER lub pliku zawierającego bibliotekę zewnętrzną z definicjami metod danej klasy. Atrybut MODULE jest poprawny tylko dla deklaracji CLASS.

Przykład:

```
OneClass   CLASS,MODULE('OneClass')      ! definicje metod w OneClass.CLW
LoadIt       PROCEDURE                   ! prototyp procedury LoadIt
ComputeIt    PROCEDURE                   ! prototyp procedury ComputeIt
           END
```

Porównaj: CLASS, MEMBER, LINK, Prototypy procedur

NAME (ustawienie nazwy zewnętrznej)

NAME([*name*])

NAME Określa nazwę zewnętrzną.

name Łańcuch zawierający nazwę wewnętrzną lub etykietę statycznej zmiennej łańcuchowej. Może być ona zadeklarowana jako globalna, lokalna i powinna posiadać atrybut STATIC.

Atrybut **NAME** (PROP:NAME) określa nazwę zewnętrzną. Jest on całkowicie niezależny od atrybutu EXTERNAL – nie ma wymagania ich wzajemnego łączenia, jednakże oba atrybuty mogą występować jednocześnie w tej samej deklaracji. Atrybut NAME może być nadawany prototypom procedur, plikom, kluczom KEY i INDEX, polom MEMO, dowolnym polom zadeklarowanym wewnątrz struktury FILE, dowolnym polom zadeklarowanym wewnątrz struktury QUEUE lub dowolnym zmiennym nie zadeklarowanym wewnątrz jakiejś struktury. Atrybut NAME ma różne znaczenie, w zależności od tego, gdzie został użyty.

Zastosowanie w prototypach procedur

Atrybut NAME może zostać użyty w prototypie procedury. Wówczas parametr *name* określa zewnętrzną nazwę wykorzystywaną przez linker do zidentyfikowania procedury bądź funkcji w bibliotece zewnętrznej.

Zastosowanie w odniesieniu zmiennych

Atrybut NAME może być stosowany w deklaracjach zmiennych zdefiniowanych poza strukturami. W ten sposób dostarczamy linkerowi zewnętrzną nazwę identyfikującą zmienną zadeklarowaną w bibliotece zewnętrznej. Jeżeli ta zmienna posiada również atrybut EXTERNAL, jest ona deklarowana jako zmienna publiczna biblioteki zewnętrznej (w niej przydzielana pamięć dla zmiennej). Bez atrybutu EXTERNAL, jest ona deklarowana jako zewnętrzna zmienna zewnętrznej biblioteki (pamięć dla niej jest przydzielana w programie Clariona).

Zastosowanie w odniesieniu do plików

Atrybut NAME użyty w deklaracji pliku (struktura FILE), określa nazwę pliku danych. Jeśli parametr *name* nie zawiera napędu dyskowego i ścieżki dostępu do pliku, to jest przyjmowany bieżący dysk i bieżący katalog. Jeśli zostanie pominięte rozszerzenie nazwy pliku, przyjmowane jest domyślne rozszerzenie skojarzone ze sterownikiem pliku danych wybranym dla naszej struktury.

Niektóre sterowniki plików danych wymagają,, by dane, klucze i pola MEMO były przechowywane w oddzielnych plikach dyskowych. Z tego powodu, atrybutu NAME możemy używać również w deklaracjach kluczy KEY i INDEX i w deklaracjach pól MEMO.

Atrybut NAME pozbawiony parametru *name* powoduje, że nazwą pliku staje się etykieta deklarowanej struktury (włączając w to również prefiks).

Atrybut NAME może być również stosowany w odniesieniu do dowolnego pola zadeklarowanego wewnątrz struktury RECORD definiującej rekord pliku danych (w tym przypadku parametr *name* musi być reprezentowany przez stałą). Dzięki temu możemy wykorzystać możliwości dawane przez sterowniki plików danych pozwalające na operowanie na nazwach pól. Możemy dynamicznie zmieniać nazwę pola w strukturze FILE używając do tego właściwości PROP:NAME występującej w

postaci tablicy. Numer elementu tej tablicy odnosi się do pozycji, na jakiej określone pole zostało umieszczone w definicji rekordu struktury FILE.

Zastosowanie w odniesieniu do kolejek

Atrybut NAME zastosowany przy zmiennej stanowiącej element struktury QUEUE określa zewnętrzną nazwę (*name*) wykorzystywaną przy przetwarzaniu kolejki. Parametr *name* umożliwia zastosowanie alternatywnej metody adresowania zmiennych w kolejce QUEUE, która może być wykorzystana przez instrukcje SORT, GET, PUT i ADD.

Przykład:

```
  PROGRAM
  MAP
    MODULE('External.Obj')
AddCount   PROCEDURE(LONG),LONG,C,NAME('_AddCount')   ! funkcja C o nazwie '_AddCount'
  . .

Cust       FILE,PRE(Cus),NAME(CustName)               ! nazwa pliku w zmiennej CustName
CustKey      KEY('Name'),NAME('c:\data\cust.idx')     ! deklaracja klucza, cust.idx
Record       RECORD
Name           STRING(20)
             . .

SortQue    QUEUE
Field1       STRING(10),NAME('FirstField')    ! nazwa sortowania
Field2       LONG,NAME('SecondField')         ! nazwa sortowania
             END

CurrentCnt LONG,EXTERNAL,NAME('Cur')          ! pole zadeklarowane publicznie w zewnętrznej
                                              ! bibliotece jako 'Cur'
TotalCnt     LONG,NAME('Tot')                 ! pole zadeklarowane w zewnętrznej bibliotece jako 'Tot'
  CODE
    OPEN(Cust)
    Cust{PROP:NAME,1} = 'Fred'                ! pole Cus:Name jest teraz referencjonowane jako 'Fred'
```

Porównaj: Prototypy procedur, QUEUE, SORT, GET, PUT, ADD, FILE, KEY, INDEX, EXTERNAL

NOCASE (klucz niezależny od wielkości liter)

NOCASE

Atrybut **NOCASE** (PROP:NOCASE) użyty w deklaracjach kluczy KEY i INDEX powoduje, że sortowanie jest niezależne od wielkości liter, tzn. małe i duże litery są traktowane jako jednakowe. Wszystkie znaki alfabetu w polach stanowiących składniki klucza są niejawnie konwertowane na odpowiadające im wielkie litery i dopiero wtedy zapisywane do klucza. Nie wpływa to oczywiście na sposób zapisu danych w pliku, a jedynie na sam klucz. Atrybut NOCASE nie daje żadnych efektów w odniesieniu do znaków nie będących znakami alfabetu.

Przykład:

```
Names    FILE,DRIVER('Clarion'),PRE(Nam)
NameKey    KEY(Nam:Name),NOCASE          ! deklaracja klucza, niezależnego od wielkości liter
NbrKey     KEY(Nam:Number)               ! deklaracja klucza
Rec        RECORD
Name         STRING(20)
Number       SHORT
           . .
```

Porównaj: INDEX, KEY

OEM (włączenie obsługi znaków narodowych)

OEM

Atrybut **OEM** (PROP:OEM) określamy dla pliku FILE, który zawiera dane zapisane przez program DOS-owy (lub wymagających odczytu przez program DOS-owy) z użyciem znaków pochodzących z innych alfabetów niż angielski. Łańcuchy tekstowe są wówczas automatycznie konwertowane z zestawu znaków OEM ASCII zastosowanego w pliku na zestaw znaków ANSI wykorzystywany przez Windows. Wszystkie dane tekstowe w rekordzie są automatycznie konwertowane z zestawu znaków ANSI na zestaw znaków OEM ASCII przed zapisaniem go na dysk. Określony zestaw znaków OEM ASCII wykorzystywany przy konwersji pochodzi ze strony kodowej DOS załadowanej poprzez plik COUNTRY.SYS. W ten sposób dane zapisane na dysku stają się specyficzne dla danego języka, co oznacza, że ich odczyt może nie być prawidłowy na komputerach, na których załadowano odmienne strony kodowe.

Omawiany atrybut nie jest obsługiwany przez wszystkie sterowniki plików danych; należy sprawdzić w dokumentacji, czy wybrany sterownik może go wykorzystywać.

Przykład:

```
Cust     FILE,DRIVER('TopSpeed'),PRE(Cus),OEM          ! zwiera łańcuchy ze znakami narodowymi
CustKey     KEY(Cus:Name)
Record      RECORD
Name        STRING(20)
            . .
Screen   WINDOW('Window')
           ENTRY(@S20),USE(Cus:Name)
           BUTTON('&OK'),USE(?Ok),DEFAULT
           BUTTON('&Cancel'),USE(?Cancel)
         END
  CODE
   OPEN(Cust)                                         ! otwarcie pliku Cust
   SET(Cust)
   NEXT(Cust)                                         ! pobranie rekordu, łańcuchy ASCII są
                                                      ! konwertowane do zbioru znaków ANSI
   OPEN(Screen)                                       ! otwarcie okna I wyświetlenie danych ANSI
   ACCEPT
     CASE FIELD()
     OF ?Ok
       CASE EVENT()
       OF EVENT:Accepted
         PUT(Cust)                                    ! aktualizacja rekordu, łańcuchy ANSI są
                                                      ! automatycznie konwertowane do OEM ASCII
                                                      ! zgodnie z załadowaną strona kodową DOS
         BREAK
       END
     END
   END
   CLOSE(Screen)
   CLOSE(Cust)
```

Porównaj: Pliki środowiskowe, LOCALE

OPT (wyłączenie z klucza rekordów z nieokreślonymi wartościami pól)

OPT

Atrybut **OPT** (PROP:OPT) wyłącza z klucza KEY lub INDEX te rekordy, których wszystkie pola stanowiące składniki klucza zawierają wartości „null". Wartością „null" (nieokreśloną) dla pola numerycznego jest wartość 0, dla pola tekstowego – łańcuch pusty lub same spacje (20h).

Przykład:

```
Names      FILE,DRIVER('Clarion'),PRE(Nam)     ! deklaracja struktury pliku
NameKey      KEY(Nam:Name),OPT                 ! deklaracja klucza łańcuchowego, wyłączenie pustych
NbrKey       KEY(Nam:Number),OPT               ! deklaracja klucza numerycznego, wyłączenie zer
Rec          RECORD
Name           STRING(20)
Number         SHORT
             . .
```

Porównaj: INDEX, KEY

ORDER (ustawienie wyrażenia określającego sortowanie widoku)

ORDER(*expression list*)

ORDER Określa listę wyrażeń *expression list* wykorzystywane do określenia sposobu sortowania rekordów w widoku VIEW.

expression list Pojedyncza stała łańcuchowa zawierająca jedno lub więcej wyrażeń. Każde wyrażenie w liście musi być oddzielone od poprzedniego znakiem przecinka.

Atrybut **ORDER** (PROP:ORDER) określa listę wyrażeń *expression list* wykorzystywaną do sortowania rekordów w widoku VIEW. Wyrażenia w liście *expression list* są ewaluowane od lewej do prawej; to, które jest położone najbardziej na lewo jest najbardziej znaczące dla sortowania. Wyrażenia rozpoczynające się znakiem (-) oznaczają sortowanie w porządku malejącym.

Pojedyncze wyrażenie może się odnosić do dowolnego pola struktury VIEW, na wszystkich poziomach struktur JOIN. Wyrażenia umieszczone w liście *expression list* mogą zawierać dowolne, prawidłowe wyrażenie języka programowania Clarion. Lista *expression list* jest ewaluowana w czasie działania programu (podobnie, jak procedura EVALUATE), z tego powodu wszystkie zmienne użyte w wyrażeniach muszą być bindowane (BIND).

Dla systemów plików nie bazujących na SQL, widok VIEW do sortowania używa kluczy, gdzie tylko jest to możliwe. Dołączenie do nich dodatkowych pól sortowania jest w miarę efektywne.

Dla systemów plików bazujących na SQL, właściwość PROP:SQLOrder jest ekwiwalentem właściwości PROP:ORDER. W obu przypadkach, jeśli pierwszym znakiem przypisywanego wyrażenia jest znak plus (+), jest on dołączane do istniejącego wyrażenia określającego prządek sortowania. W przypadku właściwości PROP:SQLOrder, gdy pierwszym znakiem przypisywanego wyrażenia jest znak minus (-), istniejące wyrażenie jest do niego dołączane. Jeżeli pierwszy znak nie jest ani znakiem plus, ani znakiem minus, nowe wyrażenie zastępuje istniejące.

Przykład:

```
! zamówienia sortowane w porządku malejącym wg daty, następnie nazwy klienta
ViewOrder   VIEW(Customer),ORDER('-Hea:OrderDate,Cus:Name')
              PROJECT(Cus:AcctNumber,Cus:Name,Cus:Zip)
              JOIN(Hea:AcctKey,Cus:AcctNumber)                  ! dołączenie pliku Header
                PROJECT(Hea:OrderNumber,Hea:OrderDate)
                JOIN(Dtl:OrderKey,Hea:OrderNumber)              ! dołączenie pliku Detail
                  PROJECT(Det:Item,Det:Quantity)
                  JOIN(Pro:ItemKey,Dtl:Item)                    ! dołączenie pliku Product
                    PROJECT(Pro:Description,Pro:Price)
              ....

  CODE
    ViewOrder{PROP:ORDER} = '-Hea:OrderDate,Pro:Price-Det:DiscountPrice'
    ! zamówienia sortowane wg największej wartości w ramach malejących dat
```

Porównaj: BIND, UNBIND, EVALUATE

OVER (ustawienie wspólnego obszaru pamięci)

OVER(*overvariable*)

OVER Umożliwia odwoływanie się do tego samego adresu pamięci na dwa różne sposoby.

Overvariable Etykieta zmiennej, która aktualnie zajmuje obszar pamięci przeznaczony do współdzielenia.

Atrybut **OVER** umożliwia odwoływanie się do jednego adresu pamięci na dwa różne sposoby. Rozmiar zmiennej zadeklarowanej z atrybutem OVER nie może być większy, niż rozmiar zmiennej wskazywanej przez *overvariable* (może być od niego mniejszy)

Możemy deklarować zmienną „nałożoną" na zmienną *overvariable* stanowiącą część listy parametrów przekazywanych do procedury.

Pole zadeklarowana w strukturze GROUP nie może być deklarowane jako „nałożone" na zmienną zadeklarowaną poza tą strukturą.

Przykład:

```
SomeProc   PROCEDURE(PassedGroup)        ! Proc rejestruje parametr GROUP
NewGroup     GROUP,OVER(PassedGroup)     ! redeklaracja przekazanego parametru GROUP
Field1         STRING(10)                ! kompiler ostrzega, że
Field2         STRING(2)                 ! NewGroup nie może być większa
             END                         ! niż PassedGroup
CustNote   FILE,PRE(Csn)                 ! deklaracja pliku CustNote
Notes        MEMO(2000)                  ! pole memo
Record       RECORD
CustID         LONG
             . .
CsnMemoRow  STRING(10),DIM(200),OVER(Csn:Notes)
! pole memo Csn:Notes może być adresowane jako całość lub w 10-bajtowych fragmentach
```

Porównaj: DIM

OWNER (hasło wykorzystywane do szyfrowania danych)

OWNER(*password*)

OWNER Określa hasło stosowane przy szyfrowaniu pliku danych.

password Stała lub zmienna łańcuchowa.

Atrybut **OWNER** (PROP:OWNER) określa hasło *password* stosowane przy szyfrowaniu pliku danych posiadającego atrybut ENCRYPT. Jeżeli przy dostępie do pliku nie zostanie określone prawidłowe hasło, otrzymamy komunikat o błędzie "Invalid Data File" i odczytanie tego pliku nie będzie możliwe.

Niektóre systemy plików dopuszczają stosowanie atrybutu OWNER bez towarzyszącego mu atrybutu ENCRYPT.

Przykład:

```
Customer FILE,DRIVER('Clarion'),OWNER('abCdeF'),ENCRYPT     ! hasło szyfrowania "abCdeF"
Record     RECORD
Name         STRING(20)
           . .
```

Porównaj: ENCRYPT, EXTERNAL

PRE (ustawienie prefiksu etykiety)

PRE([*prefix*])

PRE Określa prefiks etykiety złożonej struktury danych.

prefix Prefiks. Dopuszczalne znaki prefiksu to litery , cyfry od 0 do 9, znaki podkreślenia. Prefiks musi zaczynać się znakiem alfabetu lub znakiem podkreślenia. W Clarionie przyjęto konwencję, w której prefiks składa się z 1 do 3 znaków, ale nie jest ona obowiązkowa i prefiks może być dłuższy.

Atrybut **PRE** określa prefiks dla struktur FILE, QUEUE, GROUP, REPORT oraz ITEMIZE. Jest on prawidłowy również dla deklaracji LIKE; ma to na celu umożliwienie zdefiniowania oddzielnego prefiksu przy deklarowaniu kolejnych kopii złożonych struktur danych.

Prefiks stosuje się w celu rozróżnienia zmiennych (pól) o identycznych nazwach, ale występujących w różnych strukturach. W momencie, gdy umieszczamy odwołanie do elementu złożonej struktury danych w kodzie programu, musimy poprzedzić jego nazwę prefiksem macierzystej struktury, po którym umieszczamy znak dwukropka (gru:EtykietaPola).

Stosowanie prefiksów jest powoli wypierane przez bardziej elastyczną metodę – kwalifikację pól (Field Qualification). Składnia w niej stosowana polega na poprzedzeniu nazwy pola etykietą struktury, w której zostało ono zadeklarowane i oddzieleniu ich znakiem kropki (NazwaGrupy.EtykietaPola).

Przykład:

```
MasterFile  FILE,DRIVER('Clarion'),PRE(Mst)
Record        RECORD
AcctNumber      LONG                    ! referencjonowana jako Mst:AcctNumber lub MasterFile.AcctNumber
              . .
Detail        FILE,DRIVER('Clarion'),PRE(Dtl)
Record        RECORD
AcctNumber      LONG                    ! referencjonowana jako Dtl:AcctNumber lub Detail.AcctNumber
              . .
SaveQueue     QUEUE,PRE(Sav)
AcctNumber      LONG                    ! referencjonowana jako Sav:AcctNumber lub SaveQueue.AcctNumber
              END
G1            GROUP,PRE(Mem)            ! deklaracje kilku zmiennych
Message         STRING(30)              ! z prefiksem Mem
              END
G2            LIKE(G1),PRE(Me2)                 ! kolejna grupa, jak ta pierwsza zawierająca
  CODE                                          ! pewne zmienne korzystające z prefiksu "Me2"
    IF Dtl:AcctNumber <> Mst:AcctNumber         ! jeśli to jest nowe konto
      Mem:Message = 'New Account'               !  wyświetl komunikat
      Me2:Message = 'Variable in LIKE group'
    END
    IF Detail.AcctNumber <> Masterfile.AcctNumber  ! jakieś wyrażenie
      G1.Message = 'New Account'                   ! wyświetl komunikat
      G2.Message = 'Same Variable in LIKE group'
    END
```

Porównaj: Słowa zastrzeżone, Kwalifikacja pól

PRIMARY (ustawienie podstawowego klucza relacji)

PRIMARY

Atrybut **PRIMARY** (PROP:PRIMARY) określony dla klucza KEY jest unikalny (może wystąpić tylko dla jednego klucza pliku). Powoduje, że do klucza zostają włączone wszystkie rekordy pliku, przy czym nie są dopuszczalne wartości "null" w żadnym z pól stanowiących składnik klucza. Odpowiada to definicji klucza zasadniczego („Primary Key") określonej w teorii relacyjnych baz danych podanej przez E. F. Codd'a.

Niektóre sterowniki plików danych wymagają określenia takiego klucza.

Przykład:

```
Names    FILE,DRIVER('TopSpeed'),PRE(Nam)      ! deklaracja struktury pliku
NameKey    KEY(Nam:Name),OPT                   ! deklaracja klucza łańcuchowego, wykluczenie pustych
NbrKey     KEY(Nam:Number),PRIMARY             ! deklaracja klucza numerycznego jako podstawowego
Rec        RECORD
Name         STRING(20)
Number       SHORT
           . .
```

Porównaj: KEY

PRIVATE (zmienna prywatna dla modułu zawierającego klasę)

PRIVATE

Atrybut **PRIVATE** określa, że zmienna, dla której został użyty, jest widoczna tylko dla procedur zdefiniowanych w tym samym module kodu źródłowego zawierającego metody danej klasy. Tym samym zapewnia się enkapsulację w odniesieniu do innych klas.

Przykład:

```
OneClass    CLASS,MODULE('OneClass.CLW'),TYPE
PublicVar     LONG                          ! deklaracja zmiennej publicznej
PrivateVar    LONG,PRIVATE                  ! deklaracja zmiennej prywatnej
BaseProc      PROCEDURE(REAL Parm)          ! deklaracja metody publicznej
            END
TwoClass    OneClass                        ! egzemplarz OneClass
  CODE
  TwoClass.PublicVar = 1                    ! poprawne przypisanie
  TwoClass.PrivateVar = 1                   ! niepoprawne przypisanie
! OneClass.CLW zawiera:
  MEMBER()
  MAP
SomeLocalProc PROCEDURE
  END
OneClass.BaseProc PROCEDURE(REAL Parm)
  CODE
  SELF.PrivateVar = Parm                    ! poprawne przypisanie
SomeLocalProc PROCEDURE
  CODE
  TwoClass.PrivateVar = 1                   ! poprawne przypisanie
```

Porównaj: CLASS

PROTECTED (zmienna prywatna dla klasy i klas dziedziczących)

PROTECTED

Atrybut **PROTECTED** powoduje, że zmienna, dla której został użyty, jest widoczna tylko dla metod zadeklarowanych w ramach tej samej struktury CLASS oraz dla metod klas dziedziczących. Zapewnia to enkapsulację w odniesieniu do kodu zewnętrznego.

Przykład:

```
OneClass    CLASS,MODULE('OneClass.CLW'),TYPE
PublicVar     LONG                          ! deklaracja zmiennej publicznej
PrivateVar    LONG,PRIVATE                  ! deklaracja zmiennej prywatnej
BaseProc      PROCEDURE(REAL Parm)          ! deklaracja metody publicznej
              END
TwoClass    OneClass                        ! egzemplarz OneClass
  CODE
    TwoClass.PublicVar = 1                  ! poprawne przypisanie
    TwoClass.PrivateVar = 1                 ! niepoprawne przypisanie

! OneClass.CLW zawiera:
  MEMBER()
  MAP
SomeLocalProc   PROCEDURE
  END
OneClass.BaseProc  PROCEDURE(REAL Parm)
  CODE
    SELF.PrivateVar = Parm                  ! poprawne przypisanie
SomeLocalProc PROCEDURE
  CODE
    TwoClass.PrivateVar = 1                 ! poprawne przypisanie
```

Porównaj: CLASS

RECLAIM (odzyskanie miejsca po skasowanych rekordach)

RECLAIM

Atrybut **RECLAIM** (PROP:RECLAIM) powoduje, że sterownik pliku danych umieszcza nowe rekordy w miejscach pozostawionych przez rekordy usunięte (o ile takowe występują). W przeciwnym wypadku nowe rekordy są dopisywane na końcu pliku. Implementacja RECLAIM zależy od konkretnego sterownika pliku danych i nie jest dostępna dla wszystkich z nich.

Przykład:

```
Names   FILE,DRIVER('Clarion'),RECLAIM          ! wykorzystanie miejsca po skasowanych rekordach
Record    RECORD
Name      STRING(20)
        . .
```

STATIC (alokacja pamięci statycznej)

STATIC

Atrybut **STATIC** przydziela pamięć dla zmiennej, grupy GROUP lub kolejki QUEUE zadeklarowanej w procedurze w pamięci statycznej, zamiast na stosie. W przypadku kolejek, w pamięci statycznej jest umieszczany jedynie bufor danych – elementy kolejki zawsze są alokowane dynamicznie na stercie (heap).

Atrybut STATIC powoduje, że wartości zmiennych, czy buforów danych kolejek są stałe dla pierwszego egzemplarza procedury i następnych.

Przykład:

```
SomeProc   PROCEDURE
SaveQueue   QUEUE,STATIC                  ! statyczny bufor danych kolejki
Field1        LONG                        ! wartość pozostaje zachowana
Field2        STRING                      ! pomiędzy wywołaniami prcoedury
            END
AcctFile    STRING(64),STATIC             !STATIC wymagane przy użyciu zmiennej w atrybucie NAME

Transactions  FILE,DRIVER('Clarion'),PRE(TRA),NAME(AcctFile)
AccountKey    KEY(TRA:Account),OPT,DUP
Record        RECORD
Account         SHORT                     ! kod konta
Date            LONG                      ! data transakcji
Amount          DECIMAL(13,2)             ! wielkość transakcji
              . .
```

Porównaj: Deklaracje danych i przydział pamięci

THREAD (alokacja pamięci dla wątku)

THREAD

Atrybut **THREAD** przydziela pamięć dla zmiennej, pliku FILE, grupy GROUP, kolejki QUEUE lub klasy CLASS oddzielnie dla każdego wykonywanego wątku programu. Dzięki temu wartość na przykład określonej zmiennej jest różna dla różnych wątków.

Zmienna wątkowa musi być alokowana w pamięci statycznej, z tego powodu zmienne lokalne z atrybutem THREAD automatycznie otrzymują atrybut STATIC. Ten atrybut powoduje tworzenie niepożądanego „nawisu" w trakcie działania programu, szczególnie w przypadku zmiennych globalnych i zmiennych modułów. Z tego powodu powinien być używany tylko wtedy, gdy jest to niezbędne.

Zastosowanie w odniesieniu do zmiennych i grup

Atrybut THREAD powoduje zadeklarowanie statycznej zmiennej, dla której pamięć jest przydzielana oddzielnie dla każdego wykonywanego wątku programu. Dzięki temu każdy z nich „widzi" swoją własną wartość takiej zmiennej. W momencie uruchomienia nowego wątku, jest tworzony nowy egzemplarz zmiennej i inicjowany, w zależności od jej typu, wartością zerową lub łańcuchem pustym (pod warunkiem, że dla zmiennej nie określono atrybutu AUTO).

Zastosowanie w odniesieniu do plików

Atrybut THREAD (PROP:THREAD – właściwość struktury FILE) zastosowany w deklaracji pliku, przydziela pamięć dla bufora rekordu (oraz bloku kontrolnego pliku - file control block) oddzielnie dla każdego wykonywanego wątku programu. W ten sposób wartości znajdujące się w buforze rekordu będą specyficzne dla każdego wątku. W momencie uruchomienia nowego wątku, plik musi zostać otwarty ponownie w celu pobrania nowego egzemplarza bufora rekordu. Gdy plik zostanie zamknięty, pamięć przydzielona na bufor rekordu dla danego wątku zostaje zwolniona.

Zastosowanie w odniesieniu do kolejek

Atrybut THREAD zastosowany w deklaracji kolejki QUEUE powoduje utworzenie statycznego bufora danych, dla którego pamięć jest przydzielana oddzielnie w każdym wykonywanym wątku programu. Dzięki temu wartości zapisane w kolejce są zależne od wątku. W momencie uruchomienia nowego wątku, tworzony jest nowy egzemplarz kolejki specyficzny dla tego wątku.

Przykład:

```
  PROGRAM
  MAP
Thread1 PROCEDURE
Thread2 PROCEDURE
  END
Names   FILE,DRIVER('Clarion'),PRE(Nam),THREAD    ! plik wątkowy
NbrNdx    INDEX(Nam:Number),OPT
Rec       RECORD
Name        STRING(20)
Number      SHORT
          . .
```

```
GlobalVar LONG,THREAD                    ! każdy wątek ma swoją kopie GlobalVar
  CODE
    START(Thread1)
    START(Thread2)

Thread1 PROCEDURE
LocalVar   LONG,THREAD                   ! lokalna zmienna wątkowa (automatycznie STATIC)
  CODE
    OPEN(Names)                          ! OPEN tworzy nowy egzemplarz bufora rekordu
    SET(Names)                           ! zawierający pierwszy rekord w pliku
    NEXT(Names)

Thread2 PROCEDURE
SaveQueue   QUEUE,THREAD
Name          STRING(20)
Number        SHORT
            END
  CODE
    OPEN(Names)                          ! OPEN tworzy kolejny egzemplarz bufora rekordu
    SET(Names)                           ! zawierający pierwszy rekord w pliku
    PREVIOUS(Names)
```

Porównaj: START, Deklaracje danych i przydział pamięci, STATIC, AUTO

TYPE (definicja typu)

TYPE

Atrybut **TYPE** powoduje utworzenie definicji typu dla grupy GROUP, kolejki QUEUE lub klasy CLASS. Nazwa tego typu może być potem używana podczas definiowania innych grup, kolejek, czy klas o tej samej strukturze (można też do tego użyć LIKE). TYPE może być także stosowane do definiowania nazw struktur przekazywanych w postaci parametrów do procedur. W ten sposób procedura może bezpośrednio adresować składniki definicji typu poprzez składnię kwalifikatora pól (Field Qualification).

Deklaracje grup, kolejek i klas posiadające atrybut TYPE nie są alokowane w pamięci. Podczas gdy nie są zaalokowane w pamięci obiekty klasy z atrybutem TYPE, metody protoypowane w klasie CLASS muszą być zdefiniowane w celu użycia przez dziedziczące obiekty zadeklarowane z danym typem. Nie mają tutaj żadnego związku atrybuty EXTERNAL i DLL.

Gdy definicja typu jest wykorzystywana do przekazania określonej struktury do procedury w postaci parametru , procedura ta może bezpośrednio adresować pola kolejki stosując składnię kwalifikatora pól. Jest to preferowana metoda adresowania składników przekazanej struktury. Istnieje także możliwość zastosowania starszego typu adresowania pól – z użyciem prefiksu. Na przykład, zastosowanie deklaracji PROCEDURE(LOC:PrzekazywanaKolejka) oznacza, że procedura korzysta z prefiksu LOC: (oprócz indywidualnych nazw pól stosowanych w definicji typu) do bezpośredniego adresowania składników kolejki QUEUE przekazanej do niej w postaci parametru, stosując taką samą składnię, jak przy użyciu atrybut PRE. Należy jednak pamiętać, że preferowaną składnią jest kwalifikator pól. Możliwość stosowania prefiksów została zachowana w celu utrzymania kompatybilności z wcześniejszymi wersjami Clariona.

Przykład:

```
  MAP
MyProc1  PROCEDURE(PassQue)          ! przekazuje kolejke zdefiniowaną tak, jak PassGroup
  END

PassQue  QUEUE,TYPE                  ! definicja typu dla przekazywanych parametrów
First      STRING(20)                ! imię
Middle     STRING(1)                 ! inicjał drugiego imienia
Last       STRING(20)                ! nazwisko
         END

NameQue  QUEUE(PassQue)              ! kolejka o takiej strukturze, jak PassQue
         END                         ! koniec deklaracji kolejki
  CODE
    MyProc1(NameQue)                 ! wywołanie procedury przekazujące NameQue jako parametr

MyProc1  PROCEDURE(PassedQue)        ! procedura rejestrująca parametr będący kolejką
LocalVar STRING(20)
  CODE
    LocalVar = PassedQue.First       ! przypisanie NameQue.First do LocalVar z parametru
```

Porównaj: Kwalifikacja pól, Lista parametrów prototypu, CLASS, GROUP

6 - OKNA

Struktury okien

APPLICATION (deklaracja okna sterującego MDI)

```
label   APPLICATION( 'title') [,AT( )] [,CENTER] [,SYSTEM] [,MAX] [,ICON( )] [,STATUS( )] [,HLP( )]
          [,CURSOR( )] [,TIMER( )] [,ALRT( )] [,ICONIZE] [,MAXIMIZE] [,MASK] [,FONT( )]
          [,MSG( )] [,IMM] [,AUTO] [,PALETTE()]
          [,WALLPAPER( )] [,  | TILED    | ] [,  | HSCROLL   | ] [,  | DOUBLE   | ]
                              | CENTERED |       | VSCROLL   |       | NOFRAME  |
                                                  | HVSCROLL  |       | RESIZE   |
        [ MENUBAR
            Deklaracje podmenu i poleceń systemu menu
          END ]
          [ TOOLBAR
            Deklaracje kontrolek paska narzędzi
          END ]
        END
```

APPLICATION Deklaruje okno sterujące (ramkę) aplikacji MDI (Multiple Document Interface).

label Etykieta okna (wymagane).

title Nagłówek okna wyświetlany w pasku tytułowym (PROP:Text).

AT Początkowy rozmiar i położenie okna aplikacji (PROP:AT). Jeśli pominiemy, odpowiednie wartości zostaną nadane przez bibliotekę uruchomieniową.

CENTER Wymusza umieszczenie okna centralnie na pulpicie Windows. (PROP:CENTER). Ten atrybut daje taki efekt tylko wtedy, gdy choć jeden atrybut AT został pominięty.

SYSTEM Umieszcza w oknie systemowe menu sterujące (PROP:SYSTEM).

MAX Umieszcza w oknie przycisk Maksymalizuj (PROP:MAX).

ICON Określa ikonę dla okna i tym samym uaktywnia jego przycisk Minimalizuj. Jeśli nie określimy pliku zawierającego ikonę, wybierana jest domyślna ikona aplikacji (PROP:ICON).

STATUS Umieszcza w oknie pasek stanu znajdujący się w jego dolnej części (PROP:STATUS).

HLP Określa identyfikator kontekstowego systemu pomocy powiązany z oknem aplikacji; staje się on domyślny dla wszystkich okien wewnętrznych (PROP:HLP).

CURSOR Określa kursor wyświetlany, gdy myszka znajdzie się w obszarze okna (PROP:CURSOR). Jeśli parametr ten zostanie pominięty, jest wybierany domyślny kursor Windows.

TIMER	Określa okres czasu, co który jest generowane zdarzenie zegarowe (PROP:TIMER).
ALRT	Określa klawisze skrótów aktywne dla aplikacji (PROP:ALRT).
ICONIZE	Wymusza otwarcie okna aplikacji w postaci zminimalizowanej (PROP:ICONIZE).
MAXIMIZE	Wymusza otwarcie okna aplikacji w trybie, w którym zajmuje ono cały pulpit Windows (PROP:MAXIMIZE).
MASK	Wymusza kontrolę poprawności wprowadzanych danych (zgodnie z określoną maską) dla wszystkich kontrolek znajdujących się w pasku narzędzi (PROP:MASK).
FONT	Określa domyślną czcionkę dla wszystkich kontrolek paska narzędzi (PROP:FONT).
MSG	Określa tekst, domyślny dla wszystkich kontrolek, wyświetlany w pasku stanu okna aplikacji (PROP:MSG).
IMM	Wymusza generowanie zdarzeń przez okno za każdym razem, gdy jest ono przesuwane lub gdy jest zmieniany jego rozmiar (PROP:IMM).
AUTO	Wymusza odświeżanie wszystkich kontrolek paska narzędzi przy każdym wykonaniu pętli ACCEPT (PROP:AUTO).
PALETTE	Określa liczbę kolorów wykorzystywaną przy wyświetlaniu grafiki w oknie (PROP:PALETTE).
WALLPAPER	Wskazuje plik graficzny, którego zawartość ma być wyświetlana jako tapeta okna (PROP:WALLPAPER). Grafika dopasowuje się do rozmiarów okna, chyba, że dodamy atrybut TILED lub CENTERED.
TILED	Powoduje, że grafika WALLPAPER jest wyświetlana w swoim oryginalnym rozmiarze i jest rozmieszczana w tle okna wielokrotnie, tak by całkowicie je wypełnić (PROP:TILED).
CENTERED	Powoduje, że grafika WALLPAPER jest wyświetlana w swoim oryginalnym rozmiarze i jest umieszczana centralnie w oknie (PROP:CENTERED).
HSCROLL	Powoduje, że do okna jest automatycznie dołączany, wtedy, gdy jest to konieczne, poziomy suwak. Taka sytuacja ma miejsce, gdy dowolny fragment okna wewnętrznego znajdzie się poza obszarem widzialnym (PROP:HSCROLL).
VSCROLL	Powoduje, że do okna jest automatycznie dołączany, wtedy, gdy jest to konieczne, pionowy suwak. Taka sytuacja ma miejsce, gdy dowolny fragment okna wewnętrznego znajdzie się poza obszarem widzialnym (PROP:VSCROLL).
HVSCROLL	Powoduje, że do okna są automatycznie dołączane, wtedy, gdy jest to konieczne, suwaki: pionowy i poziomy. Taka sytuacja ma miejsce, gdy dowolny fragment okna wewnętrznego znajdzie się poza obszarem widzialnym (PROP:HVSCROLL).
DOUBLE	Powoduje wyświetlenie podwójnej ramki wokół okna (PROP:DOUBLE).
NOFRAME	Brak ramki wokół okna (PROP:NOFRAME).

RESIZE Powoduje wyświetlenie pogrubionej ramki wokół okna; ramka taka umożliwia zmianę jego rozmiaru (PROP:RESIZE).

MENUBAR Definiuje strukturę systemu menu dla okna (opcjonalnie). Menu zdefiniowane w APPLICATION jest menu globalnym.

TOOLBAR Definiuje strukturę paska narzędzi (opcjonalne). Pasek narzędzi zdefiniowany dla APPLICATION jest paskiem globalnym.

APPLICATION deklaruje okno sterujące MDI (Multiple Document Interface), standardowego interfejsu Windows wykorzystywanego w sytuacjach, gdy dana aplikacja ma otwierać okna wewnętrzne i umożliwiać użytkownikowi jednoczesne ich wykorzystywanie. Okno sterujące MDI (struktura APPLICATION) spełnia rolę okna nadrzędnego dla wszystkich okien wewnętrznych (MDI "childs"). Okna wewnętrzne są ograniczone oknem sterującym (ramką), mogą być wyświetlane tylko w ramach jego obszaru. W programie, w danym czasie może być otwarte tylko jedno okno APPLICATION; przy czym musi ono zostać otwarte **zanim** zostanie otwarte jakiekolwiek jego okno wewnętrzne (posiadające atrybut MDI). Istnieje jednak możliwość otwarcia, przed otwarciem okna APPLICATION, okien nie będących oknami wewnętrznymi (nie posiadających atrybutu MDI). Okno wewnętrzne MDI nie musi się znajdować w tym samym wątku wykonywalnym, co APPLICATION. Jednakże okno wewnętrzne MDI wywoływane bezpośrednio z APPLICATION musi znajdować się w oddzielnej procedurze i musi ona być inicjowana w postaci wątku za pomocą procedury START. Już uruchomione okna wewnętrzne MDI mogą być uruchamiane ponownie w nowych wątkach. W efekcie możemy uzyskać wiele niezależnych egzemplarzy tego samego okna wewnętrznego. Konwencjonalne okno aplikacji Windows powinno posiadać atrybuty ICON, MAX, STATUS, RESIZE oraz SYSTEM. Powodują one utworzenie okna aplikacji z przyciskami Maksymalizuj i Minimalizuj, z paskiem stanu, menu systemowym i ramką pozwalającą na zmianę rozmiaru okna. Oprócz tego okno aplikacji powinno posiadać system menu, zdefiniowany za pomocą struktury MENUBAR oraz pasek narzędzi, który definiujemy w strukturze TOOLBAR. Okno APPLICATION nie powinno zawierać kontrolek, za wyjątkiem tych, które umieścimy w strukturach MENUBAR oraz TOOLBAR. Nie powinno ono również służyć do wyprowadzania danych – do tego służą okna dokumentów i okienka dialogowe (definiowane za pomocą struktury WINDOW). Gdy okno APPLICATION jest otwierane po raz pierwszy, pozostaje ukryte dopóty, dopóki nie napotka pierwszej instrukcji DISPLAY lub, gdy nie zakończy się pętla ACCEPT. Umożliwia to wprowadzenie zmian w wyglądzie okna zanim zostanie ono wyświetlone. Zdarzenia związane z oknem APPLICATION są przetwarzane przez pierwszą pętlę ACCEPT występującą po jego otwarciu.

Generowane zdarzenia:

EVENT:PreAlertKey	Użytkownik nacisnął kombinację klawiszy z atrybutem ALRT.
EVENT:AlertKey	Użytkownik nacisnął kombinację klawiszy z atrybutem ALRT.
EVENT:CloseWindow	Okno jest zamykane.
EVENT:CloseDown	Aplikacja jest zamykana.
EVENT:OpenWindow	Okno jest otwierane.
EVENT:LoseFocus	Okno utraciło sterowanie na rzecz innego wątku.
EVENT:GainFocus	Okno otrzymało sterowanie od innego wątku.
EVENT:Suspend	Okno jest aktywne dla użytkownika, ale przekazuje sterowanie do innego wątku na czas obsługi

	zdarzenia zegarowego.
EVENT:Resume	Okno jest aktywne dla użytkownika i ponownie otrzymało sterowanie po obsłudze zdarzenia EVENT:Suspend.
EVENT:Timer	Zdarzenie zegarowe.
EVENT:Move	Użytkownik przesuwa okno. Instrukcja CYCLE przerywa tę operację.
EVENT:Moved	Użytkownik przesunął okno.
EVENT:Size	Użytkownik zmienia rozmiar okna. Instrukcja CYCLE przerywa tę operację.
EVENT:Sized	Użytkownik zmienił rozmiar okna.
EVENT:Restore	Użytkownik przywraca poprzedni rozmiar okna. Instrukcja CYCLE przerywa tę operację.
EVENT:Restored	Użytkownik przywrócił poprzedni rozmiar okna.
EVENT:Maximize	Użytkownik rozwija okno na cały pulpit. Instrukcja CYCLE przerywa tę operację.
EVENT:Maximized	Użytkownik rozwinął okno na cały pulpit.
EVENT:Iconize	Użytkownik zwija okno do ikony. Instrukcja CYCLE przerywa tę operację.
EVENT:Iconized	Użytkownik zwinął okno do ikony.
EVENT:Completed	Tryb AcceptAll (non-stop) zakończył przetwarzanie wszystkich kontrolek okna.
EVENT:DDErequest	Aplikacja kliencka zażądała danych od serwera DDE, którym jest aplikacja napisana w Clarionie.
EVENT:DDEadvise	Aplikacja kliencka zażądała ciągłej aktualizacji danych od serwera DDE, którym jest aplikacja napisana w Clarionie.
EVENT:DDEexecute	Aplikacja kliencka wykonała instrukcję DDEEXECUTE dla serwera DDE, którym jest aplikacja napisana w Clarionie.
EVENT:DDEpoke	Aplikacja kliencka wysłała żądane dane do serwera DDE, którym jest aplikacja napisana w Clarionie.
EVENT:DDEdata	Serwer DDE dostarczył zaktualizowanych danych dla aplikacji klienckiej napisanej w Clarionie.
EVENT:DDEclosed	Serwer DDE zakończył połączenie z aplikacją kliencką napisaną w Clarionie.

Powiązane procedury: ACCEPT, ALERT, EVENT, POST, REGISTER, UNREGISTER, YIELD, ACCEPTED, CHANGE, CHOICE, CLOSE, CONTENTS, CREATE, DESTROY, DISABLE, DISPLAY, ENABLE, ERASE, FIELD, FIRSTFIELD, FOCUS, GETFONT, GETPOSITION, HELP, HIDE, INCOMPLETE, LASTFIELD, MESSAGE, MOUSEX, MOUSEY, OPEN, POPUP, SELECT, SELECTED, SET3DLOOK, SETCURSOR, SETFONT, SETPOSITION, SETTARGET, UNHIDE, UPDATE

Przykład:

```
! Okno aplikacji MDI zawierające system menu, przyciski Minimalizuj i Maksymalizuj,
! pasek stanu, suwaki, ramkę umożliwiającą zmianę rozmiaru okna, pasek narzędzi

MainWin APPLICATION('My Application'),SYSTEM,MAX,ICON('MyIcon.ICO'),STATUS |
                ,HVSCROLL,RESIZE
        MENUBAR
          MENU('&File'),USE(?FileMenu)
            ITEM('&Open...'),USE(?OpenFile)
            ITEM('&Close'),USE(?CloseFile),DISABLE
            ITEM('E&xit'),USE(?MainExit)
          END
            MENU('&Edit'),USE(?EditMenu)
            ITEM('Cu&t'),USE(?CutText),KEY(CtrlX),STD(STD:Cut),DISABLE
            ITEM('&Copy'),USE(?CopyText),KEY(CtrlC),STD(STD:Copy),DISABLE
            ITEM('&Paste'),USE(?PasteText),KEY(CtrlV),STD(STD:Paste),DISABLE
          END
          MENU('&Window'),STD(STD:WindowList),LAST
            ITEM('&Tile'),STD(STD:TileWindow)
            ITEM('&Cascade'),STD(STD:CascadeWindow)
            ITEM('&Arrange Icons'),STD(STD:ArrangeIcons)
          END
          MENU('&Help'),USE(?HelpMenu)
            ITEM('&Contents'),USE(?HelpContents),STD(STD:HelpIndex)
            ITEM('&Search...'),USE(?HelpSearch),STD(STD:HelpSearch)
            ITEM('&How to Use Help'),USE(?HelpOnHelp),STD(STD:HelpOnHelp)
            ITEM('&About MyApp...'),USE(?HelpAbout)
          END
        END
        TOOLBAR
          BUTTON('E&xit'),USE(?MainExitButton)
          BUTTON('&Open'),USE(?OpenButton),ICON(ICON:Open)
        END
      END

    CODE
      OPEN(MainWin)                          !Otwarcie okna aplikacji
      ACCEPT                                 ! Pętla obsługująca zdarzenia
        CASE ACCEPTED()                      ! Która kontrolka została wybrana?
        OF ?OpenFile                         ! Wybrano polecenie Open
              OROF ?OpenButton               ! Wciśnięto przycisk Open w pasku narzędzi
          START(OpenFileProc)                ! Uruchomienie nowego wątku
        OF ?MainExit                         ! Wybrano polecenie Exit
        OROF ?MainExitButton                 ! Wciśnięto przycisk Exit na pasku narzędzi
          BREAK                              ! Przerwanie i wyjście z pętli ACCEPT
        OF ?HelpAbout                        ! Wybrano polecenie About
          HelpAboutProc                      ! Wywołanie procedury
        END
      END
      CLOSE(MainWin)                         ! Zamknięcie aplikacji
```

Porównaj: ACCEPT, WINDOW, MDI, MENUBAR, TOOLBAR

WINDOW (deklaracja okna)

```
label   WINDOW( 'title') [,AT( )] [,CENTER] [,SYSTEM] [,MAX] [,ICON( )] [,STATUS( )] [,HLP( )]
          [,CURSOR( )] [,MDI] [,MODAL] [,MASK] [,FONT( )] [,GRAY][,TIMER( )] [,ALRT( )]
          [,ICONIZE] [,MAXIMIZE] [,MSG( )] [,PALETTE( )] [,DROPID( )] [,IMM]
          [,AUTO] [,COLOR( )] [,TOOLBOX] [,DOCK( )] [,DOCKED( )]
          [,WALLPAPER( )] [,  | TILED     | ] [,  | HSCROLL   | ] [,  | DOUBLE   | ]
                              | CENTERED  |       | VSCROLL   |       | NOFRAME  |
                                                  | HVSCROLL  |       | RESIZE   |
        [ MENUBAR
            Podmenu i/lub polecenia systemu menu
          END ]
          [ TOOLBAR
            Kontrolki paska narzędzi
          END ]
            Kontrolki
        END
```

WINDOW Deklaruje okno dokumentu lub okno dialogowe.

label Etykieta okna; wymagana.

title Tekst wyświetlany w pasku tytułowym okna (PROP:Text).

AT Początkowy rozmiar i położenie okna (PROP:AT). Jeśli pominiemy, odpowiednie wartości zostaną nadane przez bibliotekę uruchomieniową.

CENTER Wymusza centralne umieszczenie okna (PROP:CENTER). Ten atrybut daje taki efekt tylko wtedy, gdy choć jeden atrybut **AT** został pominięty.

SYSTEM Umieszcza w oknie systemowe menu sterujące (PROP:SYSTEM).

MAX Umieszcza w oknie przycisk Maksymalizuj (PROP:MAX).

ICON Określa ikonę dla okna i tym samym uaktywnia jego przycisk Minimalizuj. Jeśli nie określimy pliku zawierającego ikonę, wybierana jest domyślna ikona aplikacji (PROP:ICON).

STATUS Definiuje pasek stanu dla okna (PROP:STATUS).

HLP Określa identyfikator kontekstowego systemu pomocy powiązany z oknem (PROP:HLP).

CURSOR Wskazuje kursor wyświetlany, gdy myszka znajdzie się w obszarze okna (PROP:CURSOR). Kursor ten jest domyślnie dziedziczony przez wszystkie kontrolki okna, oprócz tych, dla których określimy odmienne kursory.

MDI Określa, że dane okno jest oknem wewnętrznym MDI (PROP:MDI).

MODAL Powoduje, że dane okno jest systemowym oknem modalnym. Oznacza to, że użytkownik nie może wykonać żadnych innych operacji w systemie i w aplikacji dopóty, dopóki okno to nie zostanie zamknięte (PROP:MODAL).

MASK Wymusza kontrolę zgodności danych wprowadzanych w kontrolkach okna ze zdefiniowanymi dla nich maskami (PROP:MASK).

FONT Określa domyślną czcionkę dla wszystkich kontrolek okna (PROP:FONT).

GRAY Powoduje wyświetlanie okna z szarym tłem, jest to wykorzystywane do uzyskania efektu trójwymiarowego wyglądu kontrolek (PROP:GRAY).

TIMER Określa okres czasu, co który jest generowane zdarzenie zegarowe (PROP:TIMER).

ALRT Określa klawisze skrótów aktywne, gdy okno posiada sterowanie (PROP:ALRT).

ICONIZE Wymusza otwarcie okna w postaci zminimalizowanej (PROP:ICONIZE).

MAXIMIZE Wymusza otwarcie okna aplikacji w trybie, w którym zajmuje ono cały dostępny obszar okna nadrzędnego (PROP:MAXIMIZE)

MSG Określa tekst, domyślny dla wszystkich kontrolek okna, wyświetlany w pasku stanu okna aplikacji (PROP:MSG).

PALETTE Określa liczbę kolorów wykorzystywaną przy wyświetlaniu grafiki w oknie (PROP:PALETTE).

DROPID Określa, że okno może służyć jako cel dla operacji drag-and-drop (PROP:DROPID); przeciągane dane mogą być umieszczane w jego kontrolkach.

IMM Wymusza generowanie zdarzeń przez okno za każdym razem, gdy jest ono przesuwane lub gdy jest zmieniany jego rozmiar (PROP:IMM).

AUTO Wymusza odświeżanie wszystkich kontrolek okna przy każdym wykonaniu pętli ACCEPT (PROP:AUTO).

COLOR Określa kolor tła dla okna oraz domyślny kolor tła i zaznaczenia dla kontrolek w oknie (PROP:COLOR).

TOOLBOX Powoduje, że dane okno zawsze znajduje się “na wierzchu”. Przykrywa ono inne okna nawet wtedy, gdy nie jest aktywne (PROP:TOOLBOX).

DOCK Umożliwia zadokowanie okna z atrybutem TOOLBOX przy jednej z krawędzi okna nadrzędnego (PROP:DOCK).

DOCKED Powoduje otwarcie okna z atrybutem DOCK jako już zadokowanego (PROP:DOCKED).

WALLPAPER Wskazuje plik graficzny, którego zawartość ma być wyświetlana jako tapeta okna (PROP:WALLPAPER). Grafika dopasowuje się do rozmiarów okna, chyba, że dodamy atrybut TILED lub CENTERED.

TILED Powoduje, że grafika WALLPAPER jest wyświetlana w swoim oryginalnym rozmiarze i jest rozmieszczana w tle okna wielokrotnie, tak by całkowicie je wypełnić (PROP:TILED).

CENTERED Powoduje, że grafika WALLPAPER jest wyświetlana w swoim oryginalnym rozmiarze i jest umieszczana centralnie w oknie (PROP:CENTERED).

HSCROLL Powoduje, że do okna jest automatycznie dołączany, wtedy, gdy jest to konieczne, poziomy suwak. Taka sytuacja ma miejsce, gdy dowolny przewijalny fragment okna znajdzie się poza obszarem widzialnym (PROP:HSCROLL).

VSCROLL Powoduje, że do okna jest automatycznie dołączany, wtedy, gdy jest to konieczne, pionowy suwak. Taka sytuacja ma miejsce, gdy dowolny przewijalny fragment okna znajdzie się poza obszarem widzialnym (PROP:VSCROLL).

HVSCROLL Powoduje, że do okna są automatycznie dołączane, wtedy, gdy jest to konieczne, suwaki: pionowy i poziomy. Taka sytuacja ma miejsce, gdy dowolny przewijalny fragment okna znajdzie się poza obszarem widzialnym (PROP:HVSCROLL).

DOUBLE Powoduje wyświetlenie podwójnej ramki wokół okna (PROP:DOUBLE).

NOFRAME Brak ramki wokół okna (PROP:NOFRAME).

RESIZE Powoduje wyświetlenie pogrubionej ramki wokół okna; ramka taka umożliwia zmianę jego rozmiaru (PROP:RESIZE).

MENUBAR Definiuje strukturę menu dla okna (opcjonalnie).

Podmenu i/lub polecenia systemu menu

Deklaracje MENU i/lub ITEM definiujące odpowiednio podmenu i polecenia systemu menu.

TOOLBAR Definiuje strukturę paska narzędzi (opcjonalne).

Kontrolki paska narzędzi

Definicje kontrolek umieszczanych w pasku narzędzi.

WINDOW deklaruje okno dokumentu lub okno dialogowe mogące zawierać kontrolki i mogące być wykorzystywane do wyprowadzania danych dla użytkownika. Gdy WINDOW jest otwierane po raz pierwszy, pozostaje ukryte dopóty, dopóki nie zostanie wywołana instrukcja DISPLAY lub gdy nie zakończy się pętla ACCEPT. Umożliwia to wprowadzenie niezbędnych zmian w wyglądzie okna jeszcze zanim zostanie ono wyświetlone. Dowolne, poprzednio otwarte w tym samym wątku okno WINDOW staje się niedostępne. Zdarzenia dla WINDOW są przetwarzane przez pierwszą pętlę ACCEPT występującą po pierwszym otwarciu okna. Okno WINDOW automatycznie otrzymuje pojedynczą ramkę, chyba, że zostanie dla niego określony któryś z atrybutów: DOUBLE, NOFRAME lub RESIZE. Koordynaty ekranowe dla okna są określane w jednostkach dialogowych. Jednostka dialogowa jest zdefiniowana jako jedna czwarta średniej szerokości znaku i jedna ósma średniej wysokości znaku czcionki domyślnie wybranej dla WINDOW (atrybut FONT; w przypadku gdy nie został on określony, automatycznie jest wybierana czcionka systemowa).

Okno z atrybutem MODAL jest systemowym oknem modalnym; przejmuje ono całkowicie sterowanie komputerem. Oznacza to, że nie możemy się przełączyć na inne okno dopóty, dopóki to okno nie zostanie zamknięte. Z tego względu wspomnianego atrybutu MODAL powinniśmy używać tylko wtedy, gdy jest to naprawdę konieczne. Oprócz tego należy pamiętać, że po nadaniu atrybutu MODAL, jest ignorowany atrybut RESIZE, jak również nie jest możliwe przesuwanie okna po pulpicie.

Okno WINDOW nie posiadające atrybutu MDI, a otwarte przez wątek aplikacji MDI, jest oknem modalnym dla tej aplikacji. Oznacza to, że użytkownik może przełączyć się do innego okna tej samej aplikacji dopiero po jego zamknięciu; możliwe jest

natomiast przełączanie się pomiędzy aplikacjami. Okna nie posiadające atrybutu MDI można otwierać przed lub po otwarciu okna sterującego aplikacji, jak również z poziomu okien MDI.

Okno WINDOW z atrybutem MDI jest oknem wewnętrznym aplikacji MDI. Może ono być wyświetlane i przesuwane jedynie w obszarze okna głównego aplikacji MDI. Okna wewnętrzne MDI pozwalają użytkownikowi na przełączanie między nimi. Użytkownik może też uaktywniać inne aplikacje. Okno wewnętrzne MDI nie musi występować w tym samym wątku wykonywalnym co APPLICATION. Jednakże dowolne okno wewnętrzne MDI wywoływane bezpośrednio z APPLICATION musi występować w oddzielnej procedurze. Jest tak dlatego, bo do uruchomienia nowego wątku wykonywalnego jest używana procedura START. Już otwarte okna wewnętrzne MDI mogą być wywoływane w nowych wątkach.

Pasek menu MENUBAR zdefiniowany w oknie WINDOW posiadającym atrybut MDI jest automatycznie "scalany" z globalnym paskiem menu określonym dla aplikacji APPLICATION. Dzieje się to w momencie, gdy dane okno WINDOW otrzyma sterowanie, czyli wtedy, gdy staje się aktywne. Wyjątkiem jest sytuacja, gdy okno WINDOW lub okno APPLICATION posiada atrybut NOMERGE. Pasek menu MENUBAR zdefiniowany w oknie WINDOW, które nie posiada atrybutu MDI, nigdy nie jest scalany z menu globalnym.

Pasek narzędzi TOOLBAR zdefiniowany w oknie WINDOW, które posiada atrybut MDI jest automatycznie scalany z globalnym paskiem narzędzi zdefiniowanym w oknie APPLICATION. Ma to miejsce w momencie otrzymania sterowania przez okno WINDOW. W przypadku, gdy pasek narzędzi okna WINDOW lub APPLICATION posiada atrybut NOMERGE, pasek narzędzi nie jest scalany. Pasek narzędzi okna WINDOW nie posiadającego atrybutu MDI nigdy nie jest scalany z globalnym paskiem narzędzi.

Okno WINDOW posiadające atrybut TOOLBOX jest automatycznie wyświetlane zawsze "na wierzchu". Oznacza to, że jego obszar przykrywa inne okna, nawet wtedy, gdy to one posiadają sterowanie. Okno z takim atrybutem charakteryzuje się również tym, że jego kontrolki nie staja się aktywne – tak, jakby posiadały atrybut SKIP. Atrybut TOOLBOX stosujemy w przypadku okien WINDOW, których kontrolki mają się zachowywać tak, jakby znajdowały się w pasku narzędzi. Dzięki temu możemy tworzyć własne paski narzędzi, dokować je przy różnych krawędziach okna, wyświetlać w samodzielnym okienku itp. Okno WINDOW z atrybutem TOOLBOX uruchamiamy w samodzielnym wątku.

Generowane zdarzenia:

EVENT:PreAlertKey	Użytkownik nacisnął kombinację klawiszy z atrybutem ALRT.
EVENT:AlertKey	Użytkownik nacisnął kombinację klawiszy z atrybutem ALRT.
EVENT:CloseWindow	Okno jest zamykane.
EVENT:CloseDown	Aplikacja jest zamykana.
EVENT:OpenWindow	Okno jest otwierane.
EVENT:LoseFocus	Okno utraciło sterowanie na rzecz innego wątku.
EVENT:GainFocus	Okno otrzymało sterowanie od innego wątku.
EVENT:Suspend	Okno jest aktywne dla użytkownika, ale przekazuje sterowanie do innego wątku na czas obsługi zdarzenia zegarowego.

EVENT:Resume	Okno jest aktywne dla użytkownika i ponownie otrzymało sterowanie po obsłudze zdarzenia EVENT:Suspend.
EVENT:Docked	Okno z atrybutem TOOLBOX zostało zadokowane przy krawędzi okna sterującego MDI.
EVENT:Undocked	Okno z atrybutem TOOLBOX zostało odsunięte od krawędzi okna sterującego MDI.
EVENT:Timer	Zdarzenie zegarowe.
EVENT:Move	Użytkownik przesuwa okno. Instrukcja CYCLE przerywa tę operację.
EVENT:Moved	Użytkownik przesunął okno.
EVENT:Size	Użytkownik zmienia rozmiar okna. Instrukcja CYCLE przerywa tę operację.
EVENT:Sized	Użytkownik zmienił rozmiar okna.
EVENT:Restore	Użytkownik przywraca poprzedni rozmiar okna. Instrukcja CYCLE przerywa tę operację.
EVENT:Restored	Użytkownik przywrócił poprzedni rozmiar okna.
EVENT:Maximize	Użytkownik rozwija okno na cały pulpit. Instrukcja CYCLE przerywa tę operację.
EVENT:Maximized	Użytkownik rozwinął okno na cały pulpit.
EVENT:Iconize	Użytkownik zwija okno do ikony. Instrukcja CYCLE przerywa tę operację.
EVENT:Iconized	Użytkownik zwinął okno do ikony.
EVENT:Completed	Tryb AcceptAll (non-stop) zakończył przetwarzanie wszystkich kontrolek okna.
EVENT:DDErequest	Aplikacja kliencka zażądała danych od serwera DDE, którym jest aplikacja napisana w Clarionie.
EVENT:DDEadvise	Aplikacja kliencka zażądała ciągłej aktualizacji danych od serwera DDE, którym jest aplikacja napisana w Clarionie.
EVENT:DDEexecute	Aplikacja kliencka wykonała instrukcję DDEEXECUTE dla serwera DDE, którym jest aplikacja napisana w Clarionie.
EVENT:DDEpoke	Aplikacja kliencka wysłała żądane dane do serwera DDE, którym jest aplikacja napisana w Clarionie.
EVENT:DDEdata	Serwer DDE dostarczył zaktualizowanych danych dla aplikacji klienckiej napisanej w Clarionie.
EVENT:DDEclosed	Serwer DDE zakończył połączenie z aplikacją kliencką napisaną w Clarionie.

Powiązane procedury: ACCEPT, ALERT, EVENT, POST, REGISTER, UNREGISTER, YIELD, ACCEPTED, CHANGE, CHOICE, CLOSE, CONTENTS, CREATE, DESTROY, DISABLE, DISPLAY, ENABLE, ERASE, FIELD, FIRSTFIELD, FOCUS, GETFONT, GETPOSITION, HELP, HIDE, INCOMPLETE, LASTFIELD, MESSAGE, MOUSEX, MOUSEY, OPEN, POPUP, SELECT, SELECTED, SET3DLOOK, SETCURSOR, SETFONT, SETPOSITION, SETTARGET, UNHIDE, UPDATE

Przykład:

```
! Okno wewnętrzne aplikacji MDI zawierające przyciski Minimalizuj i Maksymalizuj,
! suwaki, ramkę umożliwiającą zmianę rozmiaru okna, pasek stanu oraz menu i pasek narzędzi
! dołączane do menu i paska narzędzi aplikacji

MDIChild WINDOW('Child One'),MDI,SYSTEM,MAX,ICON('Icon.ICO'),STATUS,HVSCROLL,RESIZE
        MENUBAR
          MENU('File'),USE(?FileMenu)
            ITEM('Close'),USE(?CloseFile)
          END
          MENU('Edit'),USE(?EditMenu)
            ITEM('Undo'),USE(?UndoText),KEY(CtrlZ),STD(STD:Undo)
            ITEM('Cu&t'),USE(?CutText),KEY(CtrlX),STD(STD:Cut)
            ITEM('Copy'),USE(?CopyText),KEY(CtrlC),STD(STD:Copy)
            ITEM('Paste'),USE(?PasteText),KEY(CtrlV),STD(STD:Paste)
          END
        END
        TOOLBAR
          BUTTON('Cut'),USE(?CutButton),ICON(ICON:Cut),STD(STD:Cut)
          BUTTON('Copy'),USE(?CopyButton),ICON(ICON:Copy),STD(STD:Copy)
          BUTTON('Paste'),USE(?PasteButton),ICON(ICON:Paste),STD(STD:Paste)
        END
        TEXT,HVSCROLL,USE(Pre:Field)
        BUTTON('&OK'),USE(?Exit),DEFAULT
      END

! Okno wewnętrzne nie będące oknem MDI, zawierające menu systemowe,
! przycisk Maksymalizuj, pasek stanu, ramkę nie pozwalającą na zmianę rozmiaru okna

NonMDI WINDOW('Dialog Window'),SYSTEM,MAX,STATUS
        TEXT,HVSCROLL,USE(Pre:Field),MSG('Enter some text here')
        BUTTON('&OK'),USE(?Exit),DEFAULT
      END

! Okno modalne (dla systemu), zawierające ramkę nie pozwalającą na zmianę rozmiaru, tekst komunikatu i !
przycisk OK
ModalWin WINDOW('Modal Window'),MODAL
        IMAGE(ICON:Exclamation)
        STRING('An ERROR has occurred')
        BUTTON('&OK'),USE(?Exit),DEFAULT
      END
```

Porównaj: ACCEPT, APPLICATION

MENUBAR (deklaracja systemu menu)

```
MENUBAR [, NOMERGE ]
    [ MENU( )
        [ ITEM( ) ]
        [ MENU( )
            [ ITEM( ) ]
        END ]
    END ]
    [ ITEM( ) ]
END
```

MENUBAR	Deklaruje system menu dla okna APPLICATION lub WINDOW.
NOMERGE	Powoduje, że dany pasek nie może być scalany z innymi paskami.
MENU	Element systemu menu stanowiący podmenu zawierające polecenia i kolejne podmenu.
ITEM	Element systemu menu.

MENUBAR deklaruje system menu wyświetlany w oknie APPLICATION lub WINDOW. MENUBAR musi wystąpić w kodzie źródłowym **przed** pierwszą deklaracja paska narzędzi TOOLBAR lub jakąkolwiek kontrolką. MENUBAR użyte w oknie APPLICATION deklaruje globalny system menu dla całej aplikacji. Jest on aktywny i dostępny dla wszystkich okien wewnętrznych MDI. Wyjątkiem są okna posiadające własną strukturę systemu menu z dołączonym atrybutem NOMERGE. Jeśli atrybut NOMERGE określiliśmy dla systemu menu okna APPLICATION, do globalnego systemu menu nie mogą być dołączane systemy menu okien wewnętrznych MDI, o ile oczywiście zostały zdefiniowane. W takim przypadku zastępują one globalny system menu w momencie, gdy okno otrzyma sterowanie.

W oknach MDI, MENUBAR definiuje elementy menu, które są automatycznie dołączane do menu globalnego aplikacji. Zarówno menu globalne, jak i elementy menu są aktywne wtedy, gdy okno wewnętrzne MDI jest oknem aktywnym. W momencie, gdy okno to utraci aktywność, jego elementy menu są usuwane z menu globalnego. Jeżeli dla MENUBAR okna MDI określimy atrybut NOMERGE, menu to zastępuje menu globalne.

W oknach nie posiadających atrybutu MDI, menu nigdy nie jest dołączane do menu globalnego. Menu zdefiniowane za pomocą MENUBAR zawsze pojawia się w oknie, a nie w aplikacji, która była wcześniej otwarta.

Zdarzenia generowane przez elementy menu lokalnego są obsługiwane przez pętlę ACCEPT okna WINDOW. Zdarzenia generowane przez elementy menu globalnego są przekazywane do pętli obsługi zdarzeń wątku, który otworzył aplikację APPLICATION (zazwyczaj, w aplikacji wielowątkowej, jest to pętla ACCEPT aplikacji).

Dynamiczne zmiany elementów menu aktywnego okna dotyczą tylko wyświetlanego w tej chwili menu, nawet wtedy, gdy są to elementy menu globalnego. Zmiany wprowadzone w menu globalnym, gdy aktywne jest okno aplikacji APPLICATION lub gdy odnoszą się do elementów globalnego menu, mają wpływ na wygląd wszystkich menu niezależnie, czy poszczególne okna wewnętrzne są otwarte, czy nie.

W sytuacji, gdy menu okna jest dołączane do menu globalnego aplikacji, w pierwszej kolejności występują elementy tego ostatniego; pod warunkiem jednak, że nie nadaliśmy atrybutów FIRST lub LAST.

Jeżeli chcemy uzyskać dwukolumnowe menu, wystarczy nadać atrybut PROP:Max = 1 temu elementowi, który ma rozpoczynać drugą kolumnę.

Przykład:

```
! Okno aplikacji MDI z systemem menu
MainWin APPLICATION('My Application')
        MENUBAR
          MENU('File'),USE(?FileMenu)
            ITEM('Open...'),USE(?OpenFile)
            ITEM('Close'),USE(?CloseFile),DISABLE
            ITEM('E&xit'),USE(?MainExit),LAST
          END
          MENU('Edit'),USE(?EditMenu)
            ITEM('Cu&t'),USE(?CutText),KEY(CtrlX),STD(STD:Cut),DISABLE
            ITEM('Copy'),USE(?CopyText),KEY(CtrlC),STD(STD:Copy),DISABLE
            ITEM('Paste'),USE(?PasteText),KEY(CtrlV),STD(STD:Paste),DISABLE
          END
          MENU('Window'),STD(STD:WindowList),LAST
            ITEM('Tile'),STD(STD:TileWindow)
            ITEM('Cascade'),STD(STD:CascadeWindow)
          END
          MENU('Help'),USE(?HelpMenu),LAST
            ITEM('Contents'),USE(?HelpContents),STD(STD:HelpIndex)
            ITEM('Search for Help On...'),USE(?HelpSearch),STD(STD:HelpSearch)
            ITEM('How to Use Help'),USE(?HelpOnHelp),STD(STD:HelpOnHelp)
            ITEM('About MyApp...'),USE(?HelpAbout)
          END
        END
      END

! Okno wewnętrzne MDI z menu dołączanym do menu głównego aplikacji

MDIChild WINDOW('Child One'),MDI
        MENUBAR
          MENU('File'),USE(?FileMenu)                    ! scalene z menu File
            ITEM('Pick...'),USE(?PickFile)               ! dołączane do poleceń menu
          END
          MENU('Edit'),USE(?EditMenu)                    ! scalane z menu Edit
            ITEM('Undo'),USE(?UndoText),KEY(CtrlZ),STD(STD:Undo)     ! dołączane do menu
          END
        END
        TEXT,HVSCROLL,USE(Pre:Field)
        BUTTON('&OK'),USE(?Exit),DEFAULT
      END

! Okno wewnętrzne MDI z własnym systemem menu zastępującym główne menu aplikacji
MDIChild2 WINDOW('Dialog Window'),MDI,SYSTEM,MAX,STATUS
        MENUBAR,NOMERGE
          MENU('File'),USE(?FileMenu)
            ITEM('Close'),USE(?CloseFile)
          END
          MENU('Edit'),USE(?EditMenu)
            ITEM('Undo'),USE(?UndoText),KEY(CtrlZ),STD(STD:Undo)
            ITEM('Cu&t'),USE(?CutText),KEY(CtrlX),STD(STD:Cut)
            ITEM('Copy'),USE(?CopyText),KEY(CtrlC),STD(STD:Copy)
            ITEM('Paste'),USE(?PasteText),KEY(CtrlV),STD(STD:Paste)
          END
        END
        TEXT,HVSCROLL,USE(Pre:Field),MSG('Enter some text here')
        BUTTON('&OK'),USE(?Exit),DEFAULT
      END
```

```
! Okno nie będące oknem MDI, z własnym systemem menu
NonMDI WINDOW('Dialog Window'),SYSTEM,MAX,STATUS
        MENUBAR
          MENU('File'),USE(?FileMenu)
            ITEM('Close'),USE(?CloseFile)
          END
          MENU('Edit'),USE(?EditMenu)
            ITEM('Undo'),USE(?UndoText),KEY(CtrlZ),STD(STD:Undo)
            ITEM('Cu&t'),USE(?CutText),KEY(CtrlX),STD(STD:Cut)
            ITEM('Copy'),USE(?CopyText),KEY(CtrlC),STD(STD:Copy)
            ITEM('Paste'),USE(?PasteText),KEY(CtrlV),STD(STD:Paste)
          END
        END
        TEXT,HVSCROLL,USE(Pre:Field),MSG('Enter some text here')
        BUTTON('&OK'),USE(?Exit),DEFAULT
      END
```

TOOLBAR (deklaracja paska narzędzi)

```
TOOLBAR [,AT( )] [,USE( )] [,CURSOR( )] [,FONT( )] [,NOMERGE] [,COLOR]
    [,WALLPAPER( )] [, | TILED | ]
                       | CENTERED |
    controls
END
```

TOOLBAR Deklaruje pasek narzędzi dla aplikacji APPLICATION lub okna WINDOW.

AT Określa początkowy rozmiar paska narzędzi. Jeśli parametr ten zostanie pominięty, domyślne wartości są nadawane przez bibliotekę runtime.

USE Etykieta ekwiwalentu paska narzędzi wykorzystywana w odwołaniach do niego w kodzie wykonywalnym (PROP:USE).

CURSOR Definiuje kursor, który pojawia się w momencie, gdy wskaźnik myszki znajdzie się nad paskiem narzędzi. Jeśli pominiemy ten parametr, wykorzystywany jest kursor zdefiniowany dla okna lub aplikacji bądź domyślny kursor Windows.

FONT Określa domyślną czcionkę dla kontrolek umieszczonych w pasku narzędzi.

NOMERGE Określa, że pasek narzędzi nie ma być łączony z innymi paskami narzędzi

COLOR Określa kolor tła dla paska narzędzi oraz domyślny kolor tła i kolor aktywny dla jego kontrolek.

WALLPAPER Wskazuje grafikę, która będzie wyświetlana w tle paska narzędzi (PROP:WALLPAPER). Rozmiar rysunku jest dopasowywany do rozmiaru paska narzędzi, chyba że nadamy mu atrybut TILED lub CENTERED.

TILED Określa, że grafika stanowiąca WALLPAPER jest wyświetlana w swoim domyślnym rozmiarze i jest powielana tak, by wypełnić cały pasek narzędzi (PROP:TILED).

CENTERED Określa, że grafika stanowiąca WALLPAPER jest wyświetlana w swoim domyślnym rozmiarze i jest wycentrowana w pasku narzędzi (PROP:CENTERED).

Controls Deklaracje kontrolek paska narzędzi.

Struktura **TOOLBAR** deklaruje pasek narzędzi wyświetlany w oknie APPLICATION lub WINDOW. W przypadku okna aplikacji (APPLICATION), TOOLBAR definiuje globalny pasek narzędzi naszego programu. Jeżeli posiada on atrybut NOMERGE, jego kontrolki są lokalne i są wyświetlane tylko wtedy, gdy są otwierane okna inne niż okna wewnętrzne MDI; nie ma w nim żadnych globalnych narzędzi. Globalne narzędzia są aktywne i dostępne dla wszystkich okien wewnętrznych MDI, za wyjątkiem tych, które mają zdefiniowane własne paski narzędzi posiadające atrybut NOMERGE. W takim przypadku "przykrywają" one globalny pasek narzędzi.

W oknie MDI struktura TOOLBAR definiuje narzędzia, które są automatycznie dołączane do globalnego paska narzędzi. Zarówno narzędzia globalne, jak i lokalne narzędzia okna są aktywne wtedy, gdy okno MDI posiada aktywność wprowadzania.

W momencie, gdy ją utraci, specyficzne dla niego narzędzia są usuwane z globalnego paska narzędzi. W przypadku, gdy atrybut NOMERGE został określony dla paska narzędzi okna MDI, narzędzia tego paska zastępują narzędzia globalne. Pasek narzędzi okien innych, niż okna MDIS nigdy nie jest scalana z globalnym paskiem narzędzi. Zawsze pojawia się on w danym oknie, a nie w oknie uprzednio otworzonej aplikacji.

Zdarzenia generowane przez lokalne narzędzia są przekazywane do pętli ACCEPT okna WINDOW. Zdarzenia generowane przez narzędzia globalne są przekazywane do aktywnej pętli zdarzeń wątku, który otworzył okno aplikacji. W typowym programie wielowątkowym jest to pętla ACCEPT okna APPLICATION.

Kontrolki paska narzędzi TOOLBAR generują zdarzenia w typowy sposób. Jednakże nie zachowują aktywności wprowadzania i nie można na nich operować za pomocą klawiatury o ile nie określi się dla nich klawiszy skrótów. Jak tylko użytkownik zakończy działanie na takiej kontrolce, aktywność jest przekazywana do okna – d kontrolki, która wcześniej ją posiadała.

Dynamiczne zmiany narzędzi w aktualnie aktywnym oknie, mają odniesienie tylko dla tego okna, nawet wtedy, gdy dotyczą narzędzi globalnych. Te ostatnie zostają zachowane tylko wtedy, gdy aktywnym oknem jest okno aplikacji APPLICATION lub gdy odnoszą się do globalnych narzędzi wszystkich pasków narzędzi niezależnie od tego, czy są otwarte, czy też nie. Oznacza to, że w momencie gdy jest aktywne okno wewnętrzne MDI, wyświetlany pasek narzędzi aplikacji jest w istocie kopią globalnego paska narzędzi. Umożliwia to każdemu oknu MDI własne modyfikacje paska narzędzi nie wpływając na zawartość paska narzędzi wyświetlanego dla innych okien MDI. Zdarzenia dla poszczególnych kontrolek są nadal przetwarzane przez pętlę ACCEPT aplikacji. Dla przykładu, załóżmy, że przycisk zadeklarowany w globalnym pasku narzędzi aplikacji ma numer pola o wartości 150. Procedura obsługująca okno wewnętrzne MDI może zmodyfikować wygląd tego przycisku bezpośrednio określając właściwości dla kontrolki o numerze 150. Spowoduje to zmiany w wyglądzie kontrolki ale tylko wtedy, gdy dane okno posiada aktywność wprowadzania.

Gdy pasek narzędzi okna jest łączony z globalnym paskiem narzędzi aplikacji, najpierw pojawiają się narzędzia globalne, a za nimi – narzędzia lokalne. Paski narzędzi są łączone tak, że pola z lokalnego paska narzędzi zaczynają się dokładnie w prawej pozycji określonej przez wartość parametru szerokości atrybutu AT globalnego paska narzędzi aplikacji.. Wysokość połączonego paska narzędzi jest wysokością "najwyższego" narzędzia – globalnego lub lokalnego. Jeżeli dowolna część jakiejś kontrolki leży poniżej dolnej krawędzi paska narzędzi, jego wysokość jest odpowiednio zwiększana.

Przykład:

```
! Okno aplikacji MDI zawierające system menu i pasek narzędzi
MainWin APPLICATION('My Application'),SYSTEM,MAX,ICON('MyIcon.ICO'),STATUS |
        ,HVSCROLL,RESIZE
        MENUBAR
          ITEM('E&xit'),USE(?MainExit)
        END
        TOOLBAR
          BUTTON('Exit'),USE(?MainExitButton)
        END
      END

! Okno wewnętrzne MDI zawierające pasek narzędzi dołaczany do paska narzędzi okna głównego
MDIChild WINDOW('Child One'),MDI
        TOOLBAR
          BUTTON('Cut'),USE(?CutButton),STD(STD:Cut)
          BUTTON('Copy'),USE(?CopyButton),STD(STD:Copy)
          BUTTON('Paste'),USE(?PasteButton),STD(STD:Paste)
        END
        TEXT,HVSCROLL,USE(Pre:Field)
        BUTTON('&OK'),USE(?Exit),DEFAULT
      END

! Okno wewnętrzne MDI z własnym paskiem narzędzi zastępującym pasek narzędzi okna głównego
MDIChild2 WINDOW('Dialog Window'),MDI,SYSTEM,MAX,STATUS
        TOOLBAR,NOMERGE
          BUTTON('Cut'),USE(?CutButton),STD(STD:Cut)
          BUTTON('Copy'),USE(?CopyButton),STD(STD:Copy)
          BUTTON('Paste'),USE(?PasteButton),STD(STD:Paste)
        END
        TEXT,HVSCROLL,USE(Pre:Field),MSG('Enter some text here')
        BUTTON('&OK'),USE(?Exit),DEFAULT
      END
```

Okna – ogólne informacje

W większości aplikacji Windows stosuje się trzy typy okienek: okna aplikacji, okna dokumentów i okna dialogowe. Okno aplikacji to główne okno otwierane przez program Windows. Zawiera ono zazwyczaj menu sterujące stanowiące punkt wyjścia dla reszty programu. Wszystkie pozostałe okna programu są oknami dokumentów lub oknami dialogowymi.

Oprócz tych trzech rodzajów okien istnieją jeszcze dwa typy interfejsu użytkownika stosowane w programach Windows: Single Document Interface (SDI) oraz Multiple Document Interface (MDI).

Program typu SDI to program o logice liniowej, która umożliwia użytkownikowi uruchomienie tylko jednego wątku w danym czasie. Użytkownik nie ma możliwości otwierania wielu okien wewnętrznych programu i przełączania się pomiędzy nimi. Jest to analogiczna logika do tej, która obowiązywała w DOS. Program typu SDI nie powinien stosować struktur APPLICATION w roli okna aplikacji. Właściwą jest tutaj struktura WINDOW pozbawiona atrybutu MDI; wszystkie otwierane sekwencyjnie okna dokumentów, czy okna dialogowe "przykrywają" wszystkie poprzednie.

Program typu MDI pozwala użytkownikowi na uruchamianie wielu wątków w ramach programu i swobodne przełączanie między nimi. Jest to najczęściej spotykany typ programu w systemie Windows. Aplikacja służy tutaj jako organizator okien dokumentów, w których użytkownik może przeprowadzać swoje działania. Główne okno programu typu MDI definiuje się w oparciu o strukturę APPLICATION. Okno to pełni rolę okna nadrzędnego dla wszystkich okien dokumentów (okie wewnętrznych MDI). Okna te są wyświetlane w obszarze okna głównego, nie mogą się wysunąć poza jego ramy. Są one również ukrywane w momencie, gdy zwiniemy okno główne do ikony. W programie Clarion może być otwarte w danym czasie tylko jedno okno typu APPLICATION.

Okna dokumentów i okna dialogowe są bardzo podobne do siebie, a to z tego względu, że i jedne i drugie są definiowane w oparciu o tę samą strukturę WINDOW. Różnica między nimi polega na kontekście, w jakim są one zazwyczaj używane; w większości przypadków nie ma to jednak specjalnego znaczenia. Ogólnym terminem odnoszącym się zarówno do okien dokumentów i okien dialogowych jest "okno" i tak jest to dalej przyjęte w tekście.

Okna dokumentów służą zazwyczaj do wyświetlania danych. Standardowo, mogą one być przesuwane, można też zmieniać ich rozmiar. Mają zazwyczaj pasek tytułowy, menu systemowe, przycisk Maksymalizuj.

Okienka dialogowe służą najczęściej do pobierania danych od użytkownika, bądź do wyświetlania dla niego jakichś informacji – na przykład żądania potwierdzenia wykonania określonej operacji. Okienka te mogą posiadać lub nie atrybut pozwalający na ich przesuwanie, jak również mogą posiadać lub nie menu systemowe. Na ogół okienka dialogowe nie pozwalają na zmianę swojego rozmiaru, choć mogą posiadać przycisk Maksymalizuj umożliwiający wybranie jednego z dwóch dostępnych rozmiarów takiego okienka. Okienko dialogowe może być systemowym okienkiem modalnym; oznacza to, że użytkownik musi je zamknąć, zanim będzie mógł zrobić cokolwiek innego w systemie. Może być również okienkiem modalnym dla aplikacji, wtedy użytkownik musi je zamknąć, zanim będzie mógł przełączyć się do innego okna aplikacji, wybrać polecenie z jej systemu menu itp. Dla przykładu, w środowisku okno, które pojawia się po wybraniu polecenia **O**pen z menu **F**ile, jest oknem modalnym aplikacji.

Kontrolki okna i aktywność wprowadzania

Obiekty umieszczone w strukturze APPLICATION lub WINDOW nazywamy kontrolkami (controls). Kontrolka to standardowy termin w Windows określający dowolny obiekt ekranu – przycisk, pole wprowadzania tekstu, przycisk radio, lista, itp. W większości programów DOS, zamiast "kontrolka" stosowało się zazwyczaj pojęcie "pole". W tej dokumentacji terminy "kontrolka" i "pole" używa się zamiennie.

Kontrolki pojawiają się w takich strukturach MENUBAR, TOOLBAR, czy WINDOW. Kontrolki są dostępne dla użytkownika i pozwalają mu na wybieranie i/lub edytowanie danych, ale wtedy, gdy posiadają tzw. "aktywność wprowadzania" (input focus). Ma to miejsce wtedy, gdy użytkownik naciśnie klawisz TAB, kliknie myszką lub wciśnie klawisz skrótu by umieścić kursor w kontrolce. Okno posiada aktywność wprowadzania wtedy, gdy znajduje się na szczycie okien aktywnego wątku wykonywalnego. Dopóki Clarion for Windows dopuszcza wielowątkowość programów, dopóty pojęcie aktywnego okna wprowadzania jest bardzo istotne. Tylko ten wątek, którego okno posiada aktywność wprowadzania jest wątkiem aktywnym. Użytkownik może edytować dane w kontrolkach okna tylko wtedy, gdy posiada ono aktywność wprowadzania.

Etykiety ekwiwalentów pól

Numerowanie kontrolek

W strukturach typu WINDOW każda kontrolka (pole) posiadająca atrybut USE otrzymuje numer przydzielony przez kompilator. Domyślnie numery pól zaczynają się od liczby 1 i są przydzielane kontrolkom w takiej kolejności, w jakiej są one wyszczególnione w strukturze okna (samo okno otrzymuje numer 0). Aktualnie przydzielone numery mogą być zastępowane za pomocą drugiego parametru atrybutu USE danej kontrolki. Kolejność występowania kontrolek w strukturze okna określa „naturalną" kolejność wybierania kontrolek (może ona być zmieniana w trakcie działania programu za pomocą instrukcji SELECT).

Kolejność występowania w strukturze WINDOW jest niezależna od położenia kontrolki na ekranie. Jednakże nie ma konieczności zachowania korelacji między położeniem kontrolki na ekranie, a numerem przydzielonym jej przez kompilator.

W strukturze APPLICATION, każdy element menu w MENUBAR i każda kontrolka z atrybutem USE umieszczona w pasku narzędzi TOOLBAR posiada numer przydzielony przez kompilator. Domyślnie numery te zaczynają się od wartości (-1) i są zmniejszane o jeden (1) w kolejności występowania elementów menu i kontrolek w strukturze APPLICATION.

Ekwiwalenty dla numerów kontrolek

Istnieje kilka instrukcji wykorzystujących numery kontrolek przydzielone przez kompilator do określenia, której właściwie kontrolki taka instrukcja dotyczy. Stosowanie numerów byłoby jednak bardzo uciążliwe i w znaczącym stopniu utrudniałoby programowanie. Przyjęto więc rozwiązanie pomagające w „obejściu" tego problemu. Polega ono na stosowaniu etykiet ekwiwalentów pól (Field Equate Labels).

Etykiety ekwiwalentów pól zawsze zaczynają się znakiem zapytania (?), po którym następuje etykieta kontrolki. Znak zapytania informuje kompilator, że dana operacja nie dotyczy wartości kontrolki (pola, zmiennej), a samej kontrolki, jako elementu systemu Windows. Etykiety ekwiwalentów pól są bardzo zbliżone do typowych

dyrektyw EQUATE kompilatora. Określa on numer pola dla etykiety ekwiwalentu pola w trakcie kompilacji. Dzięki temu nie musimy znać z góry numeru pola, gdy chcemy wykonać na nim jakieś operacje.

Występowanie dwóch lub więcej kontrolek o takim samym atrybucie USE w jednym oknie WINDOW lub aplikacji APPLICATION powinno doprowadzić do utworzenia jednakowych etykiet ekwiwalentów pól dla każdej z nich (każdy odwołujący się do innego numeru pola). Jednakże, gdy kompilator wykryje taką sytuację wszystkie te ekwiwalenty zostaną odrzucone. Z jednej strony uniemożliwi to odwoływanie się do tych kontrolek w kodzie wykonywalnym, z drugiej jednak zapobiegnie działaniom nie na tej kontrolce, na której zamierzaliśmy. Możemy wyeliminować ten problem wyraźnie określając ekwiwalenty nazw pól w trzecim parametrze atrybutu USE.

Ekwiwalenty tablic i struktur złożonych

Etykiety ekwiwalentów zmiennych, które są elementami tablicy zawsze zaczynają się znakiem zapytania, za którym występuje nazwa określona za pomocą atrybutu USE danej zmiennej, po niej z kolei umieszczamy znak podkreślenia i numer danego elementu. Dla przykładu, ekwiwalent zdefiniowany za pomocą atrybutu USE jako USE(Tablica[1]) wygląda następująco: ?Tablica_1. W przypadku tablic wielowymiarowych kolejne indeksy oddzielamy od siebie znakiem podkreślenia, na przykład: ?Tablica_1_1, ?Tablica_1_2, itd. Możemy zastąpić taki sposób własnym, podając ekwiwalent dla poszczególnych kontrolek jako trzeci parametr atrybutu USE zmiennej.

Etykiety ekwiwalentów dla zmiennych będących złożonymi strukturami danych zawsze zaczynają się znakiem zapytania (?), za którym następuje nazwa zmiennej oraz znak dwukropka (:) zastępujący znak kropki (.). Na przykład, pole które posiada atrybut USE(Telefony.Rek.Nazwisko) będzie miało ekwiwalent Telefony:Rek:Nazwisko. Takie zastąpienie jest wykonywane z tego względu, że etykiety Clariona mogą zawierać dwukropki, a nie mogą zawierać kropek, a ekwiwalent jest przecież rodzajem etykiety.

Stosowanie etykiet ekwiwalentów pól

Niektóre kontrolki posiadają atrybut USE stanowiący jedynie etykietę ekwiwalentu pola (unikalną etykietę poprzedzoną znakiem zapytania). W ten prosty sposób dostajemy możliwość odwoływania się do tych pól w instrukcjach kodu lub podczas określania właściwości kontrolek (za pomocą specjalnej składni). W kodzie wykonywalnym istnieje wiele instrukcji, których argumentami są etykiety ekwiwalentów pól, przykładem jednej z nich jest DISPLAY, która odświeża daną kontrolkę na ekranie. We wszystkich tych instrukcjach możemy użyć samego znaku

zapytania (?), bez etykiety ekwiwalentu. W ten sposób określamy, że dana instrukcja odnosi się do aktywnej kontrolki.

Przykład:

```
Window   WINDOW('Dialog Window'),SYSTEM,MAX,STATUS
           TEXT,HVSCROLL,USE(Pre:Field)                  ! FEQ = ?Pre:Field
           ENTRY(@N3),HVSCROLL,USE(Pre:Array[1])         ! FEQ = ?Pre:Array_1
           ENTRY(@N3),HVSCROLL,USE(File.MyField)         ! FEQ = ?File:MyField
           IMAGE(ICON:Exclamation),USE(?Image)           ! atrybut USE jest etykietą ekwiwalentu pola
           BUTTON('&OK'),USE(?Ok)                        ! atrybut USE jest etykietą ekwiwalentu pola
         END
CODE
OPEN(Window)
?Ok{PROP:DEFAULT} = TRUE                  ! ekwiwalenty pól stosowane w przypisaniach właściwości
?Image{PROP:Text} = 'MyImage.GIF'
ACCEPT
  DISPLAY(?)                              ! odświeżenie kontrolki posiadającej aktywność wprowadzania
END
```

Grafika – ogólne informacje

Clarion dostarcza zestaw "graficznych prymitywów", czyli procedur umożliwiających rysowanie w oknach i raportach: ARC, BLANK, BOX, CHORD, ELLIPSE, IMAGE, LINE, PIE, POLYGON, ROUNDBOX, SHOW oraz TYPE. Kontrolki zawsze przykrywają grafikę narysowaną w oknie. Oznacza to, że narysowana grafika nie utrudnia dostępu do kontrolek.

Bieżący kontekst wyświetlania

Grafika jest zawsze rysowana w bieżącym kontekście wyświetlania (okno, raport, sekcja raportu itp.). Jeżeli bieżący kontekst wyświetlania nie zostanie zmieniony za pomocą SETTARGET, jest nim ostatnio otwarte okno (jeszcze nie zamknięte) aktywnego wątku wykonywalnego, które to okno jest aktywnym oknem wprowadzania. Rysunki umieszczone w oknie są trwałe, odświeżanie ich jest automatyczne - zapewnia je biblioteka runtime.

Grafika w raportach

Grafika może być rysowana również w raportach. By to osiągnąć, musimy, za pomocą funkcji SETTARGET wskazać dany raport jako bieżący kontekst wyświetlania. Opcjonalnie, jako bieżący kontekst wyświetlania może być wskazana sekcja raportu (band).

Ustawienia rysowania

Każde okno lub raport posiada własny, bieżący kolor i styl rysowania oraz grubość kreski. Dlatego też, jeśli chcemy we wszystkich oknach używać tej samej kreski (o takim samym stylu, kolorze i grubości), musimy ustawiać ją za pomocą funkcji SETPENWIDTH, SETPENCOLOR i SETPENSTYLE.

Współrzędne graficzne

Współrzędne graficzne x,y zaczynają się w punkcie (0,0) położonym w lewym, górnym rogu okna. Współrzędne te są określane w jednostkach dialogowych, chyba, że zamienimy je na inne za pomocą jednego z atrybutów: THOUS, MM lub POINTS; dotyczy to tylko grafiki w raportach. Jednostka dialogowa jest zdefiniowana jako jedna czwarta średniej szerokości znaku i jedna ósma średniej wysokości znaku dla domyślnej czcionki okna (lub czcionki systemowej, gdy nie określono atrybutu FONT dla okna). Grafika wykraczająca poza widoczne ramy okna będzie wyświetlona, gdy przewiniemy jego zawartość. Wirtualny rozmiar okna (jego zawartość możemy wówczas przewijać) automatycznie zwiększa się tak, by pomieścić całą rysowaną grafikę. Dodatkowo, jeśli dla okna określiliśmy atrybuty HSCROLL, VSCROLL lub HVSCROLL, w oknie pojawią się suwaki umożliwiające jego przewijanie.

7 - RAPORTY

Struktury raportów

REPORT (deklaruje strukturę raportu)

```
label     REPORT([ jobname]), AT( ) [, FONT( )] [, PRE( )] [, LANDSCAPE] [, PREVIEW] [, PAPER]
                                  [,COLOR( )] [   | THOUS    | ]
                                                | MM       |
                                                | POINTS   |
            [FORM
                controls
            END ]
            [HEADER
                controls
            END ]
label       DETAIL
                controls
            END
label       [BREAK( )
                group break structures
            END ]
            [FOOTER
                controls
            END ]
          END
```

REPORT	Deklaruje początek struktury raportu
label	Nazwa, w oparciu o którą struktura REPORT jest adresowana w kodzie programu.
jobname	Nazwa zadania drukowania dla Menedżera druku Windows (PROP:Text). Jeżeli pominiemy ten parametr, nazwą zadania drukowania staje się etykieta raportu (*label*).
AT	Określa rozmiar i położenie obszaru, względem lewego, górnego narożnika strony, w którym jest drukowany detal raportu (PROP:AT).
FONT	Określa domyślną czcionkę dla wszystkich kontrolek raportu (PROP:FONT). Jeśli parametr zostanie pominięty, jest przyjmowana domyślna czcionka drukarki.
PRE	Definiuje prefiks etykiety lub struktury raportu.
LANDSCAPE	Wymusza poziomą orientację strony raportu (PROP:LANDSCAPE). Jeśli parametr ten zostanie pominięty, raport jest drukowany w układzie pionowym.
PREVIEW	Powoduje wygenerowanie raportu do metapliku Windows; jeden plik dla każdej strony raportu (PROP:PREVIEW).

PAPER Określa rozmiar papieru dla raportu. Jeśli parametr ten zostanie pominięty, przyjmowany jest domyślny rozmiar strony drukarki.

COLOR Określa kolor tła raportu REPORT i domyślny kolor tła dla sekcji raportu (PROP:COLOR).

THOUS Powoduje, że wszystkie atrybuty posługujące się współrzędnymi opierają wyliczenia na tysięcznych części cala (PROP:THOUS).

MM Powoduje, że wszystkie atrybuty posługujące się współrzędnymi opierają wyliczenia na milimetrach (PROP:MM).

POINTS Powoduje, że wszystkie atrybuty posługujące się współrzędnymi opierają wyliczenia na punktach (PROP:POINTS). Jeden cal odpowiada 72 punktom, zarówno w poziomie, jak i w pionie.

FORM Struktura układu strony zawierająca elementy nadrukowywane (jako „znak wodny") na każdej stronie raportu.

controls Kontrolki raportu.

HEADER Struktura nagłówka strony; drukowana na początku każdej strony.

DETAIL Struktura detalu strony (w niej umieszczamy zazwyczaj pola rekordu).

BREAK Struktura grupująca rekordy w oparciu o wartość wskazanej zmiennej; gdy wartość ta ulega zmianie, jest tworzona kolejna grupa.

group break structures Nagłówek HEADER, stopka FOOTER oraz detal DETAIL grupy, bądź też kolejne, zagnieżdżone struktury grupujące.

FOOTER Stopka strony raportu; drukowana na końcu każdej strony.

Instrukcja **REPORT** deklaruje początek struktury raportu. Struktura REPORT musi być zakończona instrukcją END (lub równoważną kropką). Składowymi struktury REPORT są struktury FORM, HEADER, DETAIL, FOOTER oraz BREAK; formatują one wygląd raportu.

Raport REPORT musi być otwarty za pomocą instrukcji OPEN. Raport REPORT posiadający atrybut PREVIEW jest „drukowany" do metaplików Windows; każda strona raportu jest przechowywana w oddzielnym metapliku. Atrybut PREVIEW wskazuje kolejkę QUEUE, w której są przechowywane nazwy wygenerowanych metaplików. Możemy utworzyć okno, w którym będą wyświetlane wygenerowane metapliki (w kontrolkach IMAGE). Nazwy tych metaplików pobieramy z pól kolejki QUEUE i przypisujemy je do właściwości {PROP:Text} kontrolki IMAGE. W ten sposób możemy oprogramować okno podglądu raportu przed wydrukiem.

Atrybut AT raportu definiuje obszar każdej strony przeznaczony do drukowania detalu DETAIL. W obszar ten wchodzą również nagłówki i stopki grup. Za pomocą instrukcji PRINT mogą (i muszą) być drukowane jedynie struktury typu detal (DETAIL). Wszystkie pozostałe struktury raportu (HEADER, FOOTER, FORM) są automatycznie drukowane we właściwym miejscu strony.

Struktura FORM jest drukowana na każdej stronie raportu, za wyjątkiem stron zawierających struktury DETAIL posiadające atrybut ALONE. Format struktury FORM jest określany jednorazowo, na początku raportu. Służy ona do tworzenia „podkładu", czy szablonu strony raportu, który jest następnie wypełniany przez struktury HEADER, DETAIL, czy FOOTER. Nagłówek HEADER i stopka FOOTER strony nie są drukowane w ramach grupy BREAK. Są one automatycznie drukowane na każdej kolejnej stronie.

Struktura BREAK definiuje grupę. Może ona zawierać swój własny nagłówek HEADER, stopkę FOOTER oraz detal DETAIL. W ramach jednej grupy mogą być zagnieżdżane kolejne. W ramach grupy może także występować wiele detali DETAIL. Nagłówek HEADER i stopka FOOTER zdefiniowane wewnątrz struktury BREAK stanowią nagłówek i stopkę grupy. Są one automatycznie drukowane w momencie, gdy zmieni się wartość zmiennej, w oparciu o którą są tworzone grupy.

Struktura REPORT nigdy nie staje się domyślnie aktualnym kontekstem; oznacza to, że zmiany właściwości dotyczą ostatnio otwartego okna WINDOW lub APPLICATION. Jeżeli instrukcja lub zmiana właściwości ma dotyczyć raportu, musimy wyraźnie wskazać jego etykietę (*label*) lub też zmienić najpierw kontekst za pomocą instrukcji SETTARGET tak, by stał się nim nasz raport. Jeżeli w odniesieniu do raportu chcemy użyć procedur graficznych (np. rysowania linii), musi on się stać wcześniej aktualnym kontekstem (stosujemy instrukcję SETTARGET), gdyż grafika jest rysowana zawsze w aktualnym kontekście.

Drukowanie oparte na stronach

W raportach Clariona drukowanie jest oparte na całych stronach, a nie na pojedynczych wierszach, jak w przypadku starszych generatorów raportów. Zamiast drukowania każdego wiersza, gdy jego wartości zostaną wygenerowane, na drukarkę jest wysyłana cała strona i to dopiero wtedy, gdy jest gotowa. Oznacza to, że mechanizm drukowania zaszyty w bibliotece runtime wykonuje większość działań, bazując na atrybutach, które zdefiniowaliśmy dla raportu.

Poniżej wymienione zostały niektóre działania realizowane przez mechanizm drukowania biblioteki runtime:

- Drukuje podkład każdej strony, następnie wypełnia ją danymi
- Realizuje wyliczenia (liczniki, sumy, średnie, wartości minimalne i maksymalne)
- Automatycznie zarządza podziałem raportu na strony, włączając w to drukowanie nagłówków i stopek
- Automatycznie zarządza grupowaniami, włączając w to drukowanie nagłówków i stopek grup
- Zapewnia automatyczną kontrolę spójności danych zapobiegając na przykład rozbiciu detalu na dwie strony (widow/orphan control)

Dzięki takiej automatyzacji procesu drukowania, tworzenie nawet bardzo złożonych raportów nie wymaga skomplikowanego kodowania, dzięki czemu życie programisty staje się dużo prostsze. Ponieważ mechanizm drukowania opiera się na stronach, pojęcie nagłówków i stopek traci swój kontekst związany z jednoczesnym rozmieszczeniem na stronie i porządkiem drukowania, a zachowuje jedynie związek z porządkiem drukowania. Nagłówki są zawsze drukowane na początku określonej sekwencji drukowania, a stopki – na jej końcu. Aktualne umiejscowienie danej sekwencji drukowania na stronie nie ma tu znaczenia. Dla przykładu, stopkę strony zawierającej podsumowania możemy umieścić w górnej jej części.

Przepełnienie strony

Przepełnienie strony **(**Page Overflow) następuje w momencie, gdy instrukcja PRINT nie może umieścić detalu DETAIL na stronie. Może to być spowodowane zbyt małą ilością miejsca na stronie, czy też występowaniem atrybutu PAGEBEFORE lub PAGEAFTER w strukturze DETAIL. W momencie wystąpienia przepełnienia strony, wykonywane są następujące kroki (w podanej kolejności):

1. Jeżeli raport REPORT posiada stopkę FOOTER, jest ona drukowana w pozycji określonej przez jej atrybut AT.
2. Jest zwiększany licznik stron.
3. Jeśli raport REPORT posiada strukturę FORM, jest ona drukowana w pozycji określonej przez jej atrybut AT.
4. Jeżeli raport REPORT posiada nagłówek HEADER, jest on drukowany w pozycji określonej przez jego atrybut AT.

Powiązane procedury: CLOSE, OPEN, ENDPAGE, PRINT

Przykład:

```
CustRpt REPORT,AT(1000,1000,6500,9000),THOUS,FONT('Arial',12),PRE(Rpt)
        FORM,AT(1000,1000,6500,9000)
          IMAGE('LOGO.BMP'),AT(0,0,1200,1200),USE(?I1)
        END
        HEADER,AT(1000,1000,6500,1000)
          STRING('ABC Company'),AT(3000,500,1500,500),FONT('Arial',18)
        END
Break1    BREAK(Pre:Key1)
          HEADER,AT(0,0,6500,1000)
            STRING('Group Head'),AT(3000,500,1500,500),FONT('Arial',18)
          END
Detail    DETAIL,AT(0,0,6500,1000)
            STRING(@N$11.2),AT(6000,1500,500,500),USE(Pre:F1)
          END
          FOOTER,AT(0,0,6500,1000)
            STRING('Group Total:'),AT(5500,500,1500,500)
            STRING(@N$11.2),AT(6000,500,500,500),USE(Pre:F1),SUM,RESET(Break1)
          END
        END
        FOOTER,AT(1000,1000,6500,1000)
          STRING('Page Total:'),AT(5500,1500,1500,500)
          STRING(@N$11.2),AT(6000,1500,500,500),USE(Pre:F1),SUM,PAGE
        END
      END                    ! koniec deklaracji raportu
  CODE
    OPEN(CustReport)
    SET(DataFile)
    LOOP
      NEXT(DataFile); IF ERRORCODE() THEN BREAK.
      PRINT(Rpt:Detail)
    END
    CLOSE(CustReport)
```

BREAK (deklaruje strukturę grupującą)

```
label    BREAK( variable) [,USE( )] [,NOCASE]
           group break structures
         END
```

BREAK Deklaruje strukturę grupującą.

label Nazwa struktury grupującej, która umożliwia odwołania do niej w kodzie programu.

variable Zmienna, której zmiana wartości powoduje rozpoczęcie nowej grupy (PROP:BreakVar).

USE Etykieta ekwiwalentu pola pozwalająca na odwołania do struktury BREAK w kodzie wykonywalnym (PROP:USE).

NOCASE Powoduje, że podczas sprawdzania, czy wartość zmiennej uległa zmianie nie są rozróżniane wielkie i małe litery.

group break structures Nagłówek HEADER, stopka FOOTER, detal DETAIL grupy oraz (lub) inne, zagnieżdżone grupy.

Struktura **BREAK** deklaruje zmienną *variable* sygnalizującą tworzenie nowej grupy w momencie, gdy jej wartość ulegnie zmianie. Struktura BREAK musi być zakończona instrukcją END (lub równoważnym znakiem kropki). Może ona zawierać własny nagłówek, HEADER, stopkę FOOTER, detal DETAIL, a także kolejne, zagnieżdżone struktury BREAK. Dla struktury BREAK jest dozwolony tylko jeden nagłówek i jedna kropka; może w niej natomiast występować wiele detali i zagnieżdżonych struktur BREAK.

Przykład:

```
CustRpt REPORT                  ! deklaracja raportu
Break1    BREAK(SomeVariable)
          HEADER                ! deklaracja nagłówka grupy
            ! kontrolki raportu
          END                   ! koniec deklaracji nagłówka grupy
GroupDet    DETAIL
            ! kontrolki raportu
          END                   ! koniec deklaracji detalu
          FOOTER                ! początek deklaracji stopki grupy
            ! kontrolki raportu
          END                   ! koniec deklaracji stopki grupy
        END                     ! koniec deklaracji grupy
      END                       ! koniec deklaracji raportu
```

DETAIL (detal raportu)

```
label    DETAIL ,AT( ) [,FONT( )] [,ALONE] [,ABSOLUTE] [,PAGEBEFORE( )] [,PAGEAFTER( )]
             [,WITHPRIOR( )] [,WITHNEXT( )] [,USE( )] [,COLOR( )]
           controls
         END
```

DETAIL	Deklaruje elementy stanowiące treść raportu.
label	Etykieta, poprzez którą struktura może być adresowana w kodzie wykonywalnym.
AT	Określa przesunięcie detalu względem obszaru określonego przez atrybut AT raportu oraz minimalną jego szerokość i wysokość (PROP:AT).
FONT	Określa domyślną czcionkę dla wszystkich kontrolek struktury DETAIL (PROP:FONT). Jeśli parametr zostanie pominięty, jest przyjmowana domyślna czcionka raportu, jeżeli nie występuje – domyślna czcionka drukarki.
ALONE	Wymusza drukowanie struktury DETAIL na stronie pozbawionej podkładu strony (FORM), nagłówka strony i stopki strony (PROP:ALONE).
ABSOLUTE	Powoduje, że struktura DETAIL jest drukowana zawsze w stałej pozycji względem strony (PROP:ABSOLUTE).
PAGEBEFORE	Powoduje drukowanie struktury DETAIL na początku strony, po uaktywnieniu standardowych akcji związanych z rozpoczęciem nowej strony (PROP:PAGEBEFORE).
PAGEAFTER	Powoduje, po wydrukowaniu DETAIL, wymuszenie rozpoczęcia nowej strony w oparciu o standardowe akcje związane z tą operacją (PROP:PAGEAFTER).
WITHPRIOR	Powoduje wymuszenie drukowania DETAIL na tej samej stronie, co bezpośrednio poprzedzający go detal, nagłówek lub stopka grupy (PROP:WITHPRIOR).
WITHNEXT	Powoduje wymuszenie drukowania DETAIL na tej samej stronie, co bezpośrednio po nim następujący detal, nagłówek lub stopka grupy (PROP:WITHNEXT).
USE	Etykieta ekwiwalentu pola służącego jako odnośnik struktury DETAIL w kodzie wykonywalnym (PROP:USE).
COLOR	Określa kolor tła detalu DETAIL i domyślny kolor tła dla kontrolek detalu (PROP:COLOR).
controls	Kontrolki służące do wyprowadzania danych.

Struktura **DETAIL** deklaruje elementy stanowiące główną treść raportu. Struktura DETAIL musi być zakończona instrukcją END (lub równoważną jej kropką). W raporcie może występować wiele struktur DETAIL.

Struktury DETAIL nigdy nie są drukowane automatycznie, musi być w odniesieniu do nich zastosowana instrukcja PRINT. Oznacza to, że dla każdej deklaracji DETAIL wymagana jest etykieta *label*, która jest później stosowana w wywołaniu PRINT. Struktura DETAIL może być drukowana wtedy, gdy jest to konieczne. Ze względu na to, że możemy zadeklarować wiele struktur DETAIL, łatwo jest oprogramować warunkowe ich drukowanie. Określa się to w kodzie raportu.

Detale są drukowane w obrębie obszaru drukowania określonego przez atrybut AT raportu. Atrybut AT struktury DETAIL określa pozycję względem tego obszaru oraz szerokość i wysokość detalu. Jeżeli w poziomie jest dosyć miejsca na kilka detali, są one drukowane jeden po drugim.

Przykład:

```
CustRpt REPORT              ! deklaracja raportu
        HEADER              ! początek deklaracji nagłówka strony
          ! elementy struktury
        END                 ! koniec deklaracji nagłówka strony
CustDetail1 DETAIL          ! początek deklaracji detalu
          ! elementy struktury
        END                 ! koniec deklaracji detalu
CustDetail2 DETAIL          ! początek deklaracji detalu
          ! elementy struktury
        END                 ! koniec deklaracji detalu
      END                   ! koniec deklaracji raportu

  CODE
    OPEN(CustRpt)
    SET(SomeFile)
    LOOP
      NEXT(SomeFile)
      IF ERRORCODE() THEN BREAK.
      IF SomeCondition
        PRINT(CustDetail1)
      ELSE
        PRINT(CustDetail2)
      END
    END
    CLOSE(CustRpt)
```

Porównaj: PRINT, AT

FOOTER (stopka strony lub grupy)

```
FOOTER ,AT( ) [,FONT( )] [,ABSOLUTE] [,PAGEBEFORE( )] [,PAGEAFTER( )]
    [,WITHPRIOR( )] [,WITHNEXT( )] [,ALONE] [,USE( )] [,COLOR( )]
  controls
END
```

FOOTER	Deklaruje stopkę strony lub grupy.
AT	Określa rozmiar i położenie stopki FOOTER (PROP:AT).
FONT	Określa domyślną czcionkę dla wszystkich kontrolek struktury (PROP:FONT). Jeśli parametr zostanie pominięty, jest przyjmowana domyślna czcionka raportu, a jeżeli nie występuje – domyślna czcionka drukarki.
ABSOLUTE	Powoduje, że struktura FOOTER jest drukowana zawsze w stałej pozycji względem strony (PROP:ABSOLUTE). Właściwe tylko dla stopek grup BREAK.
PAGEBEFORE	Powoduje drukowanie stopki FOOTER na początku strony, po uaktywnieniu standardowych akcji związanych z rozpoczęciem nowej strony (PROP:PAGEBEFORE). Właściwe tylko dla stopek grup BREAK.
PAGEAFTER	Powoduje, po wydrukowaniu stopki FOOTER, wymuszenie rozpoczęcia nowej strony w oparciu o standardowe akcje związane z tą operacją (PROP:PAGEAFTER). Właściwe tylko dla stopek grup BREAK.
WITHPRIOR	Powoduje wymuszenie drukowania stopki na tej samej stronie, co bezpośrednio poprzedzający ją detal, nagłówek lub stopka grupy (PROP:WITHPRIOR). Właściwe tylko dla stopek grup BREAK.
WITHNEXT	Powoduje wymuszenie drukowania stopki na tej samej stronie, co bezpośrednio po niej następujący detal, nagłówek lub stopka grupy (PROP:WITHNEXT). Właściwe tylko dla stopek grup BREAK.
ALONE	Wymusza drukowanie stopki FOOTER grupy na stronie pozbawionej podkładu strony (FORM), nagłówka strony i stopki strony (PROP:ALONE).
USE	Etykieta ekwiwalentu pola służącego jako odnośnik struktury FOOTER w kodzie wykonywalnym (PROP:USE).
COLOR	Określa kolor tła stopki FOOTER i domyślny kolor tła dla kontrolek stopki (PROP:COLOR).
controls	Kontrolki służące do wyprowadzania danych.

Struktura **FOOTER** deklaruje stopkę drukowaną na końcu strony bądź grupy.

Struktura FOOTER musi być zakończona instrukcją END (lub odpowiadającym jej znakiem kropki). Struktura FOOTER nie znajdująca się wewnątrz struktury BREAK jest stopką strony. W raporcie może wystąpić tylko jedna stopka strony. Stopka strony jest automatycznie drukowana w momencie, gdy kończy się strona, w pozycji wyznaczonej przez jej atrybut AT określonej względem strony.

Struktura **BREAK** definiuje grupę Może ona zawierać własny nagłówek HEADER, stopkę FOOTER, detal DETAIL, a także kolejne, zagnieżdżone struktury BREAK. W strukturze BREAK może występować wiele detali i zagnieżdżonych struktur BREAK. Nagłówek HEADER i stopka FOOTER zadeklarowane wewnątrz struktury BREAK są odpowiednio nagłówkiem i stopką grupy. Są one automatycznie drukowane wtedy, gdy zmienna grupująca zmieni swoją wartość, w następnej pozycji dostępnej w obszarze drukowania (obszar ten jest wyznaczony przez atrybut AT raportu). W strukturze BREAK może wystąpić tylko jeden FOOTER.

Przykład:

```
CustRpt REPORT                  ! deklaracja raportu
        FOOTER                  ! początek deklaracji stopki strony
          ! kontrolki raportu
        END                     ! koniec deklaracji stopki strony
Break1      BREAK(SomeVariable)
GroupDet      DETAIL
              ! kontrolki raportu
            END                 ! koniec deklaracji detalu
            FOOTER              ! początek deklaracji stopki grupy
              ! kontrolki raportu
            END                 ! koniec deklaracji stopki
          END                   ! koniec deklaracji grupy
        END                     ! koniec deklaracji raportu
```

FORM (podkład strony)

```
FORM ,AT( ) [,FONT( )] [,USE( )] [,COLOR( )]
   controls
END
```

FORM Deklaruje strukturę raportu drukowaną na każdej jego stronie.

AT Określa rozmiar i położenie podkładu FORM względem lewego, górnego rogu strony (PROP:AT).

FONT Określa domyślną czcionkę dla wszystkich kontrolek danej struktury raportu (PROP:FONT). Jeśli parametr zostanie pominięty, jest przyjmowana domyślna czcionka raportu, a jeżeli nie występuje – domyślna czcionka drukarki.

USE Etykieta ekwiwalentu pola służącego jako odnośnik struktury FORM w kodzie wykonywalnym (PROP:USE).

COLOR Określa kolor tła podkładu FORM i domyślny kolor tła dla kontrolek podkładu (PROP:COLOR).

controls Kontrolki służące do wyprowadzania danych.

FORM deklaruje strukturę raportu, która jest drukowana na każdej jego stronie (za wyjątkiem stron, na których są drukowane detale DETAIL posiadające atrybut ALONE). Struktura FORM musi być zakończona instrukcją END (lub odpowiadającym jej znakiem kropki).

W raporcie może wystąpić tylko jedna struktura FORM. Struktura FORM jest automatycznie drukowana w momencie przepełnienia strony. Wygląd drukowanego podkładu FORM jest określany tylko raz, na początku raportu. Umiejscowienie podkładu na stronie nie ma żadnego wpływu na umiejscowienie na niej innych drukowanych struktur. Wszystkie one są „nadrukowywane" na podkład FORM. Z tego względu FORM jest najczęściej stosowany do definiowania szablonów strony, na które są nanoszone dane raportu, jego nagłówki i stopki. Innym zastosowaniem jest drukowanie tzw. znaków wodnych, stanowiących tło strony.

Przykład:

```
CustRpt REPORT                  ! deklaracja raportu
        FORM
          IMAGE('LOGO.BMP'),AT(0,0,1200,1200),USE(?I1)
          STRING(@N3),AT(6000,500,500,500),PAGENO
        END
GroupDet  DETAIL
          ! kontrolki raportu
        END
      END                       ! koniec deklaracji raportu
```

HEADER (nagłówek strony lub grupy)

```
HEADER ,AT( ) [,FONT( )] [,ABSOLUTE] [,PAGEBEFORE( )] [,PAGEAFTER( )]
       [,WITHPRIOR( )] [,WITHNEXT( )] [,ALONE] [,USE( )] [,COLOR( )]
   controls
END
```

HEADER	Deklaruje nagłówek strony bądź grupy.
AT	Określa rozmiar i położenie nagłówka HEADER (PROP:AT).
FONT	Określa domyślną czcionkę dla wszystkich kontrolek struktury (PROP:FONT). Jeśli parametr zostanie pominięty, jest przyjmowana domyślna czcionka raportu, a jeżeli nie występuje – domyślna czcionka drukarki.
ABSOLUTE	Powoduje, że struktura HEADER jest drukowana zawsze w stałej pozycji względem strony (PROP:ABSOLUTE). Właściwe tylko dla nagłówków grup BREAK.
PAGEBEFORE	Powoduje drukowanie nagłówka HEADER na początku strony, po uaktywnieniu standardowych akcji związanych z rozpoczęciem nowej strony (PROP:PAGEBEFORE). Właściwe tylko dla nagłówków grup BREAK.
PAGEAFTER	Powoduje, po wydrukowaniu nagłówka HEADER, wymuszenie rozpoczęcia nowej strony w oparciu o standardowe akcje związane z tą operacją (PROP:PAGEAFTER). Właściwe tylko dla nagłówków grup BREAK.
WITHPRIOR	Powoduje wymuszenie drukowania nagłówka na tej samej stronie, co bezpośrednio poprzedzający go detal, nagłówek lub stopka grupy (PROP:WITHPRIOR). Właściwe tylko dla nagłówków grup BREAK.
WITHNEXT	Powoduje wymuszenie drukowania nagłówka na tej samej stronie, co bezpośrednio za nim następujący detal, nagłówek lub stopka grupy (PROP:WITHNEXT). Właściwe tylko dla nagłówków grup BREAK.
ALONE	Wymusza drukowanie nagłówka HEADER grupy na stronie pozbawionej podkładu strony (FORM), nagłówka strony i stopki strony (PROP:ALONE).
USE	Etykieta ekwiwalentu pola służącego jako odnośnik struktury HEADER w kodzie wykonywalnym (PROP:USE).
COLOR	Określa kolor tła nagłówka HEADER i domyślny kolor tła dla kontrolek nagłówka (PROP:COLOR).
controls	Kontrolki służące do wyprowadzania danych.

Struktura **HEADER** deklaruje nagłówek drukowany na początku strony bądź grupy.

Struktura HEADER musi być zakończona instrukcją END (lub odpowiadającym jej znakiem kropki). Struktura HEADER nie znajdująca się wewnątrz struktury BREAK jest nagłówkiem strony. W raporcie może wystąpić tylko jeden nagłówek strony. Nagłówek strony jest automatycznie drukowany w momencie, gdy zaczyna się strona, w pozycji wyznaczonej przez jego atrybut AT określonej względem strony.

Struktura **BREAK** definiuje grupę Może ona zawierać własny nagłówek HEADER, stopkę FOOTER, detal DETAIL, a także kolejne, zagnieżdżone struktury BREAK. W strukturze BREAK może występować wiele detali i zagnieżdżonych struktur BREAK. Nagłówek HEADER i stopka FOOTER zadeklarowane wewnątrz struktury BREAK są odpowiednio nagłówkiem i stopką grupy. Są one automatycznie drukowane wtedy, gdy zmienna grupująca zmieni swoją wartość, w następnej pozycji dostępnej w obszarze drukowania (obszar ten jest wyznaczony przez atrybut AT raportu). W strukturze BREAK może wystąpić tylko jeden HEADER.

Przykład:

```
CustRpt REPORT                  ! deklaracja raportu
        HEADER                  ! początek deklaracji nagłówka strony
          ! kontrolki raportu
        END                     ! koniec deklaracji nagłówka
Break1    BREAK(SomeVariable)
          HEADER                ! początek deklaracji nagłówka grupy
            ! kontrolki raportu
          END                   ! koniec deklaracji nagłówka
GroupDet    DETAIL
            ! kontrolki raportu
          END                   ! koniec deklaracji detalu
        END                     ! koniec deklaracji grupy
      END                       ! koniec deklaracji raportu
```

Właściwości sterujące drukarki

Tego typu właściwości sterują zachowaniem raportu i drukarki. Wszystkie mogą być stosowane w odniesieniu od wbudowanej zmiennej PRINTER odwołującej się bezpośrednio do drukarki bądź też w odniesieniu do etykiety raportu służącego jako aktualny kontekst. Z drugiej strony, nie zawsze mają takie same znaczenie w odniesieniu do drukarki i raportu. Omawiane właściwości są zdefiniowane w pliku PRNPROP.CLW, który musimy dołączyć, za pomocą instrukcji INCLUDE, do naszej aplikacji.

PROPPRINT:DevMode

Kompletna struktura trybu urządzenia (devmode) zdefiniowana w Windows Software Development Kit. Umożliwia ona funkcjom API dostęp do wszystkich właściwości drukarki (porównaj podręcznik Windows API).

```
DevMode        GROUP          !16-bit DevMode
DeviceName       STRING(32)   ! PROPPRINT:Device
SpecVersion      USHORT
DriverVersion    USHORT
Size             USHORT
DriverExtra      USHORT
Fields           ULONG
Orientation      SHORT
PaperSize        SHORT        ! PROPPRINT:Paper
PaperLength      SHORT        ! PROPPRINT:PaperHeight
PaperWidth       SHORT        ! PROPPRINT:PaperWidth
Scale            SHORT        ! PROPPRINT:Percent
Copies           SHORT        ! PROPPRINT:Copies
DefaultSource    SHORT        ! PROPPRINT:PaperBin
PrintQuality     SHORT        ! PROPPRINT:Resolution
Color            SHORT        ! PROPPRINT:Color
Duplex           SHORT        ! PROPPRINT:Duplex
                 END
```

Struktura devmode wykazuje różnice w trybie 16-to i 32-bitowym (porównaj Windows API), jednakże opisane właściwości są wspólne dla obu trybów:

PROPPRINT:Collate

Decyduje o sortowaniu kopii wydruku: 0=wyłączone, 1=włączone (funkcja nie jest obsługiwana przez każdą drukarkę).

PROPPRINT:Color

Wydruk kolorowy lub monochromatyczny: 1=mono, 2=kolor (funkcja nie jest obsługiwana przez każdą drukarkę).

PROPPRINT:Context

Daje w rezultacie uchwyt (handle) kontekstu drukarki po pierwszej instrukcji PRINT odnoszącej się do raportu lub informację kontekstu przed pierwszym wywołaniem instrukcji PRINT. Właściwość ta nie może być ustawiana dla wbudowanej zmiennej globalnej PRINTER, możemy ja jedynie odczytywać.

PROPPRINT:Copies

Liczba drukowanych kopii (funkcja nie jest obsługiwana przez każdą drukarkę).

PROPPRINT:Device

Nazwa drukarki w takiej postaci, w jakiej pojawia się w okienku drukowania Windows. Jeżeli wiele nazw drukarek zaczyna się tymi samymi znakami, brana jest pierwsza z brzegu (bez rozróżniania wielkich i małych liter). Może być ustawiona dla zmiennej PRINTER, ale tylko przed otwarciem raportu.

PROPPRINT:Driver

Nazwa pliku sterownika drukarki (bez rozszerzenia .DLL).

PROPPRINT:Duplex

Dupleksowy tryb drukowania (funkcja nie jest obsługiwana przez każdą drukarkę). Ekwiwalenty DUPLEX::xxx dla standardowych wyborów są wymienione w pliku PRNPROP.CLW.

PROPPRINT:FontMode

Tryb cziconek TrueType. Ekwiwalenty FONTMODE:xxx są wymienione w pliku PRNPROP.CLW.

PROPPRINT:FromMin

Po ustawieniu dla zmiennej PRINTER, wstawia wartość w polu „Strony od:" okienka dialogowego PRINTERDIALOG. Jeżeli chcemy wyłączyć zakres drukowania, podstawiamy wartość –1.

PROPPRINT:FromPage

Numer strony, od której ma się zacząć drukowanie. Jeżeli wydruk ma być wykonany od początku, wstawiamy wartość -1.

PROPPRINT:Paper

Standardowy rozmiar papieru. Ekwiwalenty PAPER:xxx dla standardowych rozmiarów papieru zostały wymienione w pliku PRNPROP.CLW. W ten sposób definiujemy rozmiar plików .WMF tworzonych przez mechanizm drukowania biblioteki runtime.

PROPPRINT:PaperBin

Źródło papieru. Ekwiwalenty PAPERBIN:xxx dla standardowych lokalizacji papieru w drukarce zostały wymienione w pliku PRNPROP.CLW.

PROPPRINT:PaperHeight

Wysokość papieru określona w dziesiątych milimetra (mm/10). Jeden cal liczy 25.4 mm. Właściwość wykorzystywana wtedy, gdy właściwości PROPPRINT:Paper nadajemy wartość PAPER:Custom (nie stosuje się zazwyczaj dla drukarek laserowych).

PROPPRINT:PaperWidth

Szerokość papieru określona w dziesiątych milimetra (mm/10). Jeden cal liczy 25.4 mm. Właściwość wykorzystywana wtedy, gdy właściwości PROPPRINT:Paper nadajemy wartość PAPER:Custom (nie stosuje się zazwyczaj dla drukarek laserowych).

PROPPRINT:Percent

Współczynnik skalowania, określany w procentach, stosowany przy powiększaniu lub pomniejszaniu wydruku (funkcja nie jest obsługiwana przez każdą drukarkę). Domyślną wartością jest 100 procent. Ustawiamy tę wartość w celu uzyskania wydruku w pożądanej skali (o ile drukarka i jej sterownik obsługują skalowanie). Dla przykładu, ustawienie omawianej właściwości na 200 powoduje dwukrotne powiększenie wydruku, na 50 – dwukrotne jego zmniejszenie.

PROPPRINT:Port

Port, do którego jest kierowany wydruk (LPT1, COM1, itp.).

PROPPRINT:PrintToFile

Wydruk do pliku: 0=wyłączony, 1=włączony.

PROPPRINT:PrintToName

Nazwa pliku, do którego jest kierowany wydruk.

PROPPRINT:Resolution

Rozdzielczość wydruku określona w punktach na cal (dots per inch (DPI)). Ekwiwalenty RESOLUTION:xxx dla standardowych rozdzielczości zostały wymienione w pliku PRNPROP.CLW. Rozdzielczość musi zostać określona przed otwarciem raportu.

PROPPRINT:ToMax

Po ustawieniu dla zmiennej PRINTER, wstawia wartość w polu „Strony do:" okienka dialogowego PRINTERDIALOG. Jeżeli chcemy wyłączyć zakres drukowania, podstawiamy wartość –1.

PROPPRINT:ToPage

Numer strony, na której kończy się wydruk. Po podstawieniu wartości -1 wydruk jest wykonywany do końca.

PROPPRINT:Yresolution

Pionowa rozdzielczość wydruku określona w punktach na cal (dots per inch (DPI)). Ekwiwalenty RESOLUTION:xxx dla standardowych rozdzielczości zostały wymienione w pliku PRNPROP.CLW.

Przykład:

```
SomeReport  REPORT
            END
  CODE
    PRINTER{PROPPRINT:Device} = 'Epson'         ! pobiera 1-szy Epson z listy
    PRINTER{PROPPRINT:Port} = 'LPT2:'           ! wysyła raport do LPT2
    PRINTER{PROPPRINT:Percent} = 250            ! strona drukowana z powiększeniem 2.5 raza
    PRINTER{PROPPRINT:Copies} = 3               ! drukowanie 3 kopii każdej strony
    PRINTER{PROPPRINT:Collate} = False          ! drukowanie 1,1,1,2,2,2,3,3,3,...
    PRINTER{PROPPRINT:Collate} = True           ! drukowanie 1,2,3..., 1,2,3...,
    PRINTER{PROPPRINT:PrintToFile} = True       ! drukowanie do pliku
    PRINTER{PROPPRINT:PrintToName} = 'OUTPUT.RPT'      ! nazwa pliku do drukowania
    OPEN(SomeReport)                            ! otwarcie raportu po ustawieniu właściwości PRINTER
    SomeReport{PROPPRINT:Paper} = PAPER:User        ! własny rozmiar papieru
    SomeReport{PROPPRINT:PAPERHeight} = 6 * 254     ! formularz o wysokości 6"
    SomeReport{PROPPRINT:PAPERWidth} = 3.5 * 254    ! formularz o szerokości 3.5"
```

8 - KONTROLKI

Deklaracje kontrolek

BOX (rysuje prostokąt)

BOX ,**AT**( ) [,**USE**( )] [,**DISABLE**] [,**COLOR**( )] [,**FILL**( )] [,**ROUND**] [,**FULL**] [,**SCROLL**] [,**HIDE**] [,**LINEWIDTH**( )]

BOX	Umieszcza prostokąt w oknie lub w raporcie.
AT	Określa początkowy rozmiar i położenie kontrolki (PROP:AT). Jeśli jest pominięty, domyślne wartości są przyjmowane przez bibliotekę runtime.
USE	Etykieta ekwiwalentu pola umożliwiająca odwoływanie się do kontrolki w kodzie wykonywalnym (PROP:USE).
DISABLE	Powoduje, że kontrolka po pierwszym otwarciu okna WINDOW lub APPLICATION jest niedostępna, wyszarzona (PROP:DISABLE).
COLOR	Określa kolor obramowania kontrolki (PROP:COLOR). Jeśli atrybut ten zostanie pominięty, kontrolka nie będzie miała obramowania.
FILL	Określa kolor wypełnienia dla kontrolki (PROP:FILL). Jeśli atrybut ten zostanie pominięty, kontrolka nie będzie wypełniania żadnym kolorem.
ROUND	Powoduje zaokrąglenie narożników prostokąta (PROP:ROUND).
FULL	Powoduje, że rozmiar dla każdego brakującego atrybutu AT (długość i wysokość) kontrolki zwiększa się tak, by obejmował cały dostępny obszar okna WINDOW (PROP:FULL). Nie dotyczy raportu.
SCROLL	Nadanie tego parametru powoduje, że kontrolka jest przewijana wraz z oknem (PROP:SCROLL). Nie dotyczy raportów.
HIDE	Powoduje, że po pierwszym otwarciu okna WINDOW lub APPLICATION kontrolka pozostaje ukryta (PROP:HIDE). Zostanie ona wyświetlona dopiero po użyciu procedury UNHIDE. W raportach atrybut ten oznacza, że kontrolka nie będzie drukowana dopóty, dopóki nie użyjemy w stosunku do niej procedury UNHIDE.
LINEWIDTH	Określa długość obramowania kontrolki BOX (PROP:LINEWIDTH).

Kontrolka **BOX** umieszcza prostokąt w oknie WINDOW, pasku narzędzi TOOLBAR, lub w raporcie REPORT w pozycji określonej za pomocą atrybut AT. Kontrolka ta nie może otrzymać aktywności wprowadzania i nie generuje żadnych zdarzeń.

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
        BOX,AT(0,0,20,20)                                       ! nie wypełniona, czarna ramka
        BOX,AT(0,20,20,20),USE(?Box1),DISABLE                   ! nie wypełniona, czarna ramka, wyszarzona
        BOX,AT(20,20,20,20),ROUND                               ! nie wypełniona, zaokrąglona, czarna ramka
        BOX,AT(40,40,20,20),FILL(COLOR:ACTIVEBORDER)            ! wypełniona, czarna ramka
        BOX,AT(60,60,20,20),COLOR(COLOR:ACTIVEBORDER)           ! nie wypełniona, o kolorze aktywnej ramki
        BOX,AT(480,180,20,20),SCROLL                            ! przewijane wraz z ekranem
      END

CustRpt REPORT,AT(1000,1000,6500,9000),THOUS
CustDetail   DETAIL,AT(0,0,6500,1000)
          BOX,AT(0,0,20,20),USE(?B1)                            ! nie wypełniona, czarna ramka
          BOX,AT(20,20,20,20),ROUND                             ! nie wypełniona, zaokrąglona, czarna ramka
          BOX,AT(40,40,20,20),FILL(COLOR:ACTIVEBORDER)          ! wypełniona, czarna ramka
          BOX,AT(60,60,20,20),COLOR(COLOR:ACTIVEBORDER)         ! nie wypełniona, o kolorze aktywnej ramki
        END
      END
```

Porównaj: PANEL

BUTTON (deklaruje przycisk)

BUTTON(tekst) ,**AT**( ) [,**CURSOR**( )] [,**USE**( )] [,**DISABLE**] [,**KEY**( )] [,**MSG**( )] [,**HLP**( )] [,**SKIP**] [,**STD**( )] [,**FONT**( )] [,**ICON**( )] [,**DEFAULT**] [,**IMM**][,**REQ**] [,**FULL**] [,**SCROLL**] [,**ALRT**( )] [,**HIDE**] [,**DROPID**( )] [,**TIP**( )] [,**FLAT**] [,**REPEAT**( )] [,**DELAY**( )] [, | **LEFT** |] | **RIGHT** |

BUTTON Umieszcza przycisk polecenia w oknie WINDOW lub w pasku narzędzi TOOLBAR.

Tekst Stała łańcuchowa zawierająca tekst wyświetlany na przycisku (PROP:Text). Może ona zawierać znak ampersand (&) w celu wskazania (podkreślenia) klawisza skrótu przypisanego do danego przycisku.

AT Określa początkowy rozmiar i położenie kontrolki (PROP:AT). Jeśli jest pominięty, domyślne wartości są przyjmowane przez bibliotekę runtime.

CURSOR Określa kursor wyświetlany w momencie, gdy wskaźnik myszki zatrzyma się na kontrolce (PROP:CURSOR). Jeśli ominiemy ten atrybut, jest stosowany kursor wskazany dla okna lub też domyślny kursor systemu Windows.

USE Etykieta ekwiwalentu pola umożliwiająca odwoływanie się do kontrolki w kodzie wykonywalnym (PROP:USE).

DISABLE Powoduje, że kontrolka po pierwszym otwarciu okna WINDOW lub APPLICATION jest niedostępna, wyszarzona (PROP:DISABLE).

KEY Określa stałą całkowitą lub ekwiwalent kodu klawisza, którego wciśnięcie powoduje bezpośrednie przekazanie aktywności i jest równoznaczne z wciśnięciem przycisku (PROP:KEY).

MSG Określa tekst wyświetlany w pasku stanu aplikacji wtedy, gdy dana kontrolka jest aktywna (PROP:MSG).

HLP Określa łańcuch stanowiący identyfikator sekcji systemu pomocy związanej z daną kontrolką (PROP:HLP).

SKIP Powoduje, że kontrolka nie otrzymuje aktywności (jest pomijana podczas poruszania się między kontrolkami za pomocą klawisza TAB). Dostęp do niej jest możliwy po jej kliknięciu myszką lub po wciśnięciu klawisza skrótu z nią związanego (PROP:SKIP).

STD Określa stałą całkowitą lub ekwiwalent dla „standardowej akcji Windows", która jest wykonywana przez daną kontrolkę (PROP:STD).

FONT Określa czcionkę dla danej kontrolki (PROP:FONT).

ICON Wskazuje standardową ikonę lub plik z grafiką, która będzie wyświetlana na przycisku (PROP:ICON). Nie dotyczy raportu.

DEFAULT Powoduje, że wciśnięcie klawisza ENTER jest równoznaczne z naciśnięciem danego przycisku (PROP:DEFAULT).

IMM Powoduje, że kontrolka generuje zdarzenie w momencie, gdy zostanie na niej wciśnięty i przytrzymany lewy przycisk myszki i trwa to aż do momentu, gdy ten przycisk zostanie zwolniony (PROP:IMM). Jeśli atrybut ten pominiemy, zdarzenie jest generowane tylko wtedy, gdy lewy przycisk myszki na kontrolce zostanie wciśnięty i zwolniony.

REQ Powoduje, że po wciśnięciu danego przycisku, biblioteka runtime automatycznie sprawdza, czy wszystkie kontrolki ENTRY tego samego okna WINDOW posiadające również atrybut REQ zawierają dane różne od zera lub pustego łańcucha (PROP:REQ).

FULL Powoduje, że rozmiar dla każdego brakującego atrybutu AT kontrolki zwiększa się tak, by obejmował cały dostępny obszar okna WINDOW (PROP:FULL). Nie dotyczy raportu.

SCROLL Nadanie tego parametru powoduje, że kontrolka jest przewijana wraz z oknem (PROP:SCROLL).

ALRT Określa klawisze skrótu aktywne dla kontrolki (PROP:ALRT). Nie dotyczy raportu.

HIDE Powoduje, że po pierwszym otwarciu okna WINDOW lub APPLICATION kontrolka pozostaje ukryta (PROP:HIDE). Zostanie ona wyświetlona dopiero po użyciu procedury UNHIDE.

DROPID Określa, że kontrolka może służyć jako cel dla operacji „przenieś i upuść” (PROP:DROPID).

TIP Definiuje tekst wyświetlany „w dymkach” po zatrzymaniu kursora myszki na kontrolce (PROP:ToolTip).

FLAT Powoduje, że dany przycisk jest wyświetlany jako przycisk płaski, którego ramki pojawiają się dopiero wtedy, gdy znajdzie się nad nim kursor myszki (PROP:FLAT). Wymaga atrybutu ICON.

REPEAT Określa częstotliwość, z jaką jest generowane zdarzenie EVENT:Accepted, gdy użytkownik wciśnie i „przytrzyma” przycisk posiadający atrybut IMM (PROP:REPEAT). Wymaga atrybutu IMM.

DELAY Określa odstęp czasu pomiędzy pierwszym i kolejnym zdarzeniem EVENT:Accepted wygenerowanym dla przytrzymanego przycisku z atrybutem IMM (PROP:DELAY). Wymaga atrybutu IMM.

LEFT Powoduje, że ikona jest umieszczana po lewej stronie *tekstu* (PROP:LEFT).

RIGHT Powoduje, że ikona jest umieszczana po lewej stronie *tekstu* (PROP:RIGHT).

Kontrolka **BUTTON** umieszcza przycisk w oknie WINDOW lub pasku narzędzi TOOLBAR (nie jest prawidłowa dla raportu) w pozycji określonej przez atrybut AT.

Przycisk BUTTON z atrybutem IMM generuje zdarzenie EVENT:Accepted gdy tylko lewy przycisk myszki zostanie wciśnięty na kontrolce i kontynuuje to aż do momentu jego zwolnienia. Umożliwia to ciągłe wykonywanie kodu przypisanego do kontrolki, aż do momentu zwolnienia przycisku myszki. Częstotliwość generowania zdarzenia i odstęp pomiędzy kolejnymi wygenerowanymi zdarzeniami mogą być ustawione odpowiednio za pomocą atrybutów REPEAT i DELAY. Przycisk BUTTON bez atrybutu IMM generuje zdarzenie EVENT:Accepted tylko wtedy, gdy lewy przycisk myszki został wciśnięty i zwolniony na kontrolce.

Przycisk BUTTON z atrybutem REQ spełnia specjalną rolę. Otóż kontrolki ENTRY i TEXT z atrybutem REQ nie zostaną sprawdzone dopóty, dopóki nie zostanie wciśnięty ten przycisk lub nie zostanie wywołana procedura INCOMPLETE. Aktywność wprowadzania jest wówczas przekazywana do pierwszej kontrolki, której wypełnienie jest wymagane (atrybut REQ), a której wartość jest zerowa lub jest łańcuchem pustym.

Na przycisku BUTTON posiadającym atrybut ICON jest wyświetlany wskazany rysunek (ikona) – obok jego tekstu (parametr *text*); domyślnie tekst pojawia się pod rysunkiem. Parametr *text* służy również do wskazania klawisza skrótu dla przycisku.

Generowane zdarzenia:

EVENT:Selected	Przycisk stał się aktywny.
EVENT:Accepted	Przycisk został naciśnięty przez użytkownika.
EVENT:PreAlertKey	Użytkownik nacisnął klawisz wskazany atrybutem ALRT.
EVENT:AlertKey	Użytkownik nacisnął klawisz wskazany atrybutem ALRT.
EVENT:Drop	Operacja drag-and-drop dla przycisku zakończyła się pomyślnie.

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            BUTTON('1'),AT(0,0,20,20),USE(?B1)
            BUTTON('2'),AT(20,0,20,20),USE(?B2),KEY(F10Key)
            BUTTON('3'),AT(40,0,20,20),USE(?B3),MSG('Button 3')
            BUTTON('4'),AT(60,0,20,20),USE(?B4),HLP('Button4Help')
            BUTTON('5'),AT(80,0,20,20),USE(?B5),STD(STD:Cut)
            BUTTON('6'),AT(100,0,20,20),USE(?B6),FONT('Arial',12)
            BUTTON('7'),AT(120,0,20,20),USE(?B7),ICON(ICON:Question)
            BUTTON('8'),AT(140,0,20,20),USE(?B8),DEFAULT
            BUTTON('9'),AT(160,0,20,20),USE(?B9),IMM
            BUTTON('10'),AT(180,0,20,20),USE(?B10),CURSOR(CURSOR:Wait)
            BUTTON('11'),AT(200,0,20,20),USE(?B11),REQ
            BUTTON('12'),AT(220,0,20,20),USE(?B12),ALRT(F10Key)
            BUTTON('13'),AT(240,0,20,20),USE(?B13),SCROLL
          END
  CODE
    OPEN(MDIChild)
    ACCEPT
      CASE ACCEPTED()
      OF ?B1
        ! wykonanie jakiejś akcji
      END
    END
```

Porównaj: CHECK, OPTION, RADIO

CHECK (deklaruje pole wyboru)

```
CHECK(text) ,AT( ) [,CURSOR( )] [,USE( )] [,DISABLE] [,KEY( )] [,MSG( )] [,HLP( )] [,SKIP]
      [,FONT( )] [,ICON( )] [,FULL] [,SCROLL] [,ALRT( )] [,HIDE] [,DROPID( )] [,TIP( )]
      [,  | LEFT  | ] [,VALUE( )] [,TRN] [,COLOR( )] [,FLAT]
          | RIGHT |
```

CHECK Umieszcza pole wyboru w oknie WINDOW, w pasku narzędzi TOOLBAR lub w raporcie REPORT.

tekst Stała łańcuchowa zawierająca tekst wyświetlany obok pola wyboru (PROP:Text). Może ona zawierać znak ampersand (&) w celu wskazania (podkreślenia) klawisza skrótu przypisanego do danego pola wyboru.

AT Określa początkowy rozmiar i położenie kontrolki (PROP:AT). Jeśli jest pominięty, domyślne wartości są przyjmowane przez bibliotekę runtime.

CURSOR Określa kursor wyświetlany w momencie, gdy wskaźnik myszki zatrzyma się na kontrolce (PROP:CURSOR). Jeśli ominiemy ten atrybut, jest stosowany kursor wskazany dla okna lub też domyślny kursor systemu Windows. Nie dotyczy raportu.

USE Etykieta zmiennej, w której jest zapamiętywana wartość pola wyboru (PROP:USE). Zero (0) oznacza, że pole wyboru nie jest zaznaczone, jeden (1) oznacza, że pole wyboru jest zaznaczone; pod warunkiem, że atrybut VALUE nie określa innych wartości.

DISABLE Powoduje, że kontrolka po pierwszym otwarciu okna WINDOW lub APPLICATION jest wyszarzona (PROP:DISABLE).

KEY Określa stałą całkowitą lub ekwiwalent kodu klawisza, którego wciśnięcie powoduje bezpośrednie przekazanie aktywności i jest równoznaczne z zaznaczeniem pola wyboru (PROP:KEY). Nie dotyczy raportu.

MSG Określa tekst wyświetlany w pasku stanu aplikacji wtedy, gdy dana kontrolka jest aktywna (PROP:MSG). Nie dotyczy raportu.

HLP Określa łańcuch stanowiący identyfikator sekcji systemu pomocy związanej z daną kontrolką (PROP:HLP). Nie dotyczy raportu.

SKIP Powoduje, że kontrolka nie otrzymuje aktywności (jest pomijana podczas poruszania się między kontrolkami za pomocą klawisza TAB). Dostęp do niej jest możliwy po jej kliknięciu myszką lub po wciśnięciu klawisza skrótu z nią związanego (PROP:SKIP). Nie dotyczy raportu.

FONT Określa czcionkę dla danej kontrolki (PROP:FONT).

ICON Wskazuje standardową ikonę lub plik z grafiką, która będzie wyświetlana przy polu wyboru (PROP:ICON). Nie dotyczy raportu.

FULL Powoduje, że rozmiar dla każdego brakującego atrybutu AT (długość i wysokość) kontrolki zwiększa się tak, by obejmował cały dostępny obszar okna WINDOW (PROP:FULL). Nie dotyczy raportu.

SCROLL Nadanie tego parametru powoduje, że kontrolka jest przewijana wraz z oknem (PROP:SCROLL). Nie dotyczy raportów.

ALRT Określa klawisze skrótu aktywne dla kontrolki (PROP:ALRT).

HIDE Powoduje, że po pierwszym otwarciu okna WINDOW lub APPLICATION kontrolka pozostaje ukryta lub też kontrolka nie jest drukowana w raporcie. (PROP:HIDE). Zostanie ona wyświetlona lub wydrukowana dopiero po użyciu procedury UNHIDE.

DROPID Określa, że kontrolka może służyć jako cel dla operacji „przenieś i upuść" (PROP:DROPID). Nie dotyczy raportu.

TIP Definiuje tekst wyświetlany „w dymkach" po zatrzymaniu kursora myszki na kontrolce (PROP:ToolTip). Nie dotyczy raportu.

LEFT Powoduje, że *tekst* jest umieszczany po lewej stronie pola zaznaczania (PROP:LEFT).

RIGHT Powoduje, że *tekst* jest umieszczany po prawej stronie pola zaznaczania (PROP:RIGHT). Ustawienie domyślne.

VALUE Określa wartości prawda i fałsz dla zmiennej USE ustawiane po zaznaczeniu (lub nie) pola przez użytkownika (PROP:Value).

TRN Powoduje, że kontrolka jest wyświetlana jako przeźroczysta, nie przykrywa innych kontrolek ani tła (PROP:TRN).

COLOR Określa kolor tła dla tekstu wyświetlanego w kontrolce (PROP:COLOR).

FLAT Powoduje, że dany przycisk jest wyświetlany jako przycisk płaski, którego ramki pojawiają się dopiero wtedy, gdy znajdzie się nad nim kursor myszki (PROP:FLAT). Wymaga atrybutu ICON, nie dotyczy raportu.

Kontrolka **CHECK** umieszcza w oknie WINDOW, w pasku narzędzi TOOLBAR lub w raporcie REPORT pole wyboru w pozycji określonej za pomocą atrybutu AT.

Pole wyboru CHECK umieszczone w oknie i posiadające atrybut ICON ma wygląd specjalnego przycisku, który może być (i pozostawać) wciśnięty lub nie. Na tym przycisku jest wyświetlana wybrana grafika. W momencie, gdy przycisk nie jest wciśnięty, pole wyboru nie jest zaznaczone, gdy jest wciśnięty – pole wyboru jest zaznaczone.

Standardowo, gdy pole wyboru CHECK nie jest zaznaczone, zmienna wskazana poprzez atrybut USE otrzymuje wartość zero (0); gdy jest zaznaczone – wartość jeden (1). Możemy to zmienić za pomocą atrybutu VALUE lub, w czasie działania aplikacji, za pomocą właściwości PROP:TrueValue oraz PROP:FalseValue.

Generowane zdarzenia:

EVENT:Selected	Pole wyboru stało się aktywne.
EVENT:Accepted	Pole wyboru zostało przełączone przez użytkownika.
EVENT:PreAlertKey	Użytkownik nacisnął klawisz wskazany atrybutem ALRT.
EVENT:AlertKey	Użytkownik nacisnął klawisz wskazany atrybutem ALRT.
EVENT:Drop	Operacja drag-and-drop dla pola wyboru zakończyła się pomyślnie.

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            CHECK('1'),AT(0,0,20,20),USE(C1)
            CHECK('2'),AT(0,20,20,20),USE(C2),VALUE('T','F')
          END
CustRpt REPORT,AT(1000,1000,6500,9000),THOUS
CustDetail      DETAIL,AT(0,0,6500,1000)
              CHECK('1'),AT(0,0,20,20),USE(C1)
              CHECK('2'),AT(20,80,20,20),USE(C2),LEFT
              CHECK('3'),AT(0,100,20,20),USE(C3),FONT('Arial',12)
            END
          END
  CODE
    OPEN(MDIChild)
    ACCEPT
      CASE ACCEPTED()
      OF ?C1
        IF C1 = 1 THEN DO C1Routine.
      OF ?C2
        IF C2 = 'T' THEN DO C2Routine.
      END
    END
```

Porównaj: BUTTON, OPTION, RADIO

COMBO (deklaruje listę kombinowaną)

```
COMBO( picture) ,FROM( ) ,AT( ) [,CURSOR( )] [,USE( )] [,DISABLE] [,KEY( )] [,MSG( )] [,HLP( )]
     [,SKIP][,FONT( )][,FORMAT( )][,DROP][,COLUMN][,VCR][,FULL][,GRID( )][,SCROLL]
     [,ALRT( )][,HIDE][,READONLY][,REQ][,NOBAR][DROPID( )][,TIP( )][,TRN][,COLOR()]
     [,  | MARK()  |] [,  | HSCROLL   |][,  | LEFT    |][,  | INS  |][,  | UPR   |] [,MASK]
         | IMM     |      | VSCROLL   |      | RIGHT   |      | OVR  |      | CAP   |
                          | HVSCROLL  |      | CENTER  |
                                             | DECIMAL |
```

COMBO Umieszcza w oknie WINDOW lub na pasku narzędzi TOOLBAR listę kombinowaną (okienko z listą elementów wzbogacone o pole edycyjne).

picture Wzorzec wyświetlania określający format wprowadzanych do kontrolki danych (PROP:Text).

FROM Wskazuje źródło danych wyświetlanych w liście (PROP:FROM).

AT Określa początkowy rozmiar i położenie kontrolki (PROP:AT). Jeśli jest pominięty, domyślne wartości są przyjmowane przez bibliotekę runtime.

CURSOR Określa kursor wyświetlany w momencie, gdy wskaźnik myszki zatrzyma się na kontrolce (PROP:CURSOR). Jeśli ominiemy ten atrybut, jest stosowany kursor wskazany dla okna lub też domyślny kursor systemu Windows.

USE Etykieta ekwiwalentu pola umożliwiająca odwoływanie się do kontrolki w kodzie wykonywalnym (PROP:USE).

DISABLE Powoduje, że kontrolka po pierwszym otwarciu okna WINDOW lub APPLICATION jest wyszarzona (PROP:DISABLE).

KEY Określa stałą całkowitą lub ekwiwalent kodu klawisza, którego wciśnięcie powoduje bezpośrednie przekazanie aktywności kontrolce (PROP:KEY).

MSG Określa tekst wyświetlany w pasku stanu aplikacji wtedy, gdy dana kontrolka jest aktywna (PROP:MSG).

HLP Określa łańcuch stanowiący identyfikator sekcji systemu pomocy związanej z daną kontrolką (PROP:HLP).

SKIP Powoduje, że kontrolka może otrzymać aktywność wprowadzania tylko wtedy, gdy zostanie kliknięta myszką lub zostanie wciśnięty przypisany do niej klawisz skrótu. Kontrolka nie zachowuje aktywności wprowadzania (PROP:SKIP).

FONT Określa czcionkę dla danej kontrolki (PROP:FONT).

FORMAT Określa format wyświetlania danych (PROP:FORMAT).

DROP Określa, że kontrolka jest listą rozwijalną; jednocześnie określa liczbę elementów w niej widocznych (PROP:DROP).

COLUMN Powoduje, że podświetlane, w liście wielokolumnowej, będzie tylko pole jednej kolumny, a nie cały wiersz (PROP:COLUMN).

VCR Powoduje umieszczenie przycisków typu VCR (takich jak na magnetowidzie) w lewej części poziomego suwaka okienka z listą (PROP:VCR).

FULL Powoduje, że rozmiar dla każdego brakującego atrybutu AT (długość i wysokość) kontrolki zwiększa się tak, by obejmował cały dostępny obszar okna WINDOW (PROP:FULL). Nie dotyczy raportu.

GRID Określa kolor linii oddzielających kolumny w liście (PROP:GRID).

SCROLL Nadanie tego parametru powoduje, że kontrolka jest przewijana wraz z oknem (PROP:SCROLL).

ALRT Określa klawisze skrótu aktywne dla kontrolki (PROP:ALRT).

HIDE Powoduje, że po pierwszym otwarciu okna WINDOW lub APPLICATION kontrolka pozostaje ukryta (PROP:HIDE). Zostanie ona wyświetlona dopiero po użyciu procedury UNHIDE.

READONLY Nadaje kontrolce atrybut tylko-do-odczytu co uniemożliwia wprowadzanie w niej danych (PROP:READONLY).

NOBAR Powoduje, że podświetlenie jest wyświetlane w okienku tylko wtedy, gdy posiada ono aktywność wprowadzania (PROP:NOBAR).

DROPID Określa, że kontrolka może służyć jako cel dla operacji „przenieś i upuść" (PROP:DROPID).

TIP Definiuje tekst wyświetlany „w dymkach" po zatrzymaniu kursora myszki na kontrolce (PROP:ToolTip).

TRN Powoduje, że kontrolka jest wyświetlana jako przeźroczysta, nie przykrywa innych kontrolek ani tła (PROP:TRN).

COLOR Określa kolor tła i kolor podświetlenia dla kontrolki (PROP:COLOR).

REQ Określa, że kontrolka nie może pozostać nie wypełniona i nie może zawierać wartości zerowej (PROP:REQ).

MARK Umożliwia zaznaczanie wielu elementów listy jednocześnie (PROP:MARK).

IMM Wymusza generowanie zdarzenia za każdym razem, gdy użytkownik naciśnie jakiś klawisz (PROP:IMM).

HSCROLL Powoduje, że w okienku pojawia się automatycznie poziomy pasek przewijania, gdy jego poziomy rozmiar nie pozwala na wyświetlenie wszystkich danych (PROP:HSCROLL).

VSCROLL Powoduje, że w okienku pojawia się automatycznie pionowy pasek przewijania, gdy jego pionowy rozmiar nie pozwala na wyświetlenie wszystkich danych (PROP:VSCROLL).

HVSCROLL Powoduje, że w okienku pojawiają się odpowiednie paski przewijania, gdy jego rozmiar nie pozwala na wyświetlenie wszystkich danych (PROP:HVSCROLL).

LEFT Powoduje, że dane w kontrolce są wyrównywane do lewej (PROP:LEFT).

RIGHT Powoduje, że dane w kontrolce są wyrównywane do prawej (PROP:RIGHT).

CENTER Powoduje, że dane w kontrolce są centrowane (PROP:CENTER).

DECIMAL Powoduje, że dane w kontrolce są wyrównywane do kropki dziesiętnej (PROP:DECIMAL).

INS/OVR Wprowadza tryb wstawiania (Insert) lub zastępowania (Overwrite) podczas wprowadzania danych (PROP:INS i PROP:OVR). Daje to efekt tylko w przypadku okien posiadających atrybut MASK.

UPR/CAP Wprowadza tryb zastępowania wszystkich liter na wielkie lub na kapitaliki (Każda Pierwsza Litera Wyrazu W Zdaniu Jest Wielka); odpowiednie właściwości to PROP:UPR i PROP:CAP.

MASK Wymusza wprowadzanie według wzorca w polu tekstowym kontrolki (PROP:MASK).

Kontrolka **COMBO** umieszcza pole wprowadzania danych wraz z powiązaną z nim listą elementów w oknie WINDOW lub w pasku narzędzi TOOLBAR w pozycji określonej przez atrybut AT. COMBO stanowi kombinację kontrolek ENTRY i LIST. Użytkownik może wpisywać dane lub wybierać je z listy.

Wprowadzona dana jest automatycznie porównywana z elementami listy; musi w niej występować. Część kontrolki COMBO przeznaczona do wprowadzania danych funkcjonuje jak lokator inkrementalny (incremental locator) dla listy. Gdy tylko użytkownik wpisze znak, pasek podświetlenia ustawia się na najbardziej pasującym do wpisanej wartości elemencie listy.

Kontrolka COMBO posiadająca atrybut DROP wyświetla tylko aktualnie wybrany element danych. Gdy posiada ona aktywność wprowadzania i użytkownik naciśnie klawisz strzałki w dół lub kliknie ikonę znajdującą się w prawej części kontrolki, zostanie rozwinięta lista z elementami. Użytkownik ma wówczas możliwość wybrania jednego z nich; zostaje on automatycznie przeniesiony do pola edycyjnego.

Kontrolka COMBO z atrybutem IMM generuje zdarzenie EVENT:NewSelection za każdym razem, gdy użytkownik przesuwa podświetlenie w liście do innej pozycji lub naciska klawisz skrótu (wszystkie klawisze zarejestrowane za pomocą ALRT). Umożliwia to wstawienie kodu źródłowego odświeżającego kolejkę QUEUE zawierającą elementy listy, czy też pobranie odpowiedniego rekordu w celu wyświetlenia innych jego pól.

Kontrolka COMBO z atrybutem VCR posiada specjalne przyciski przewijania z piktogramami analogicznymi do tych, które spotykamy w urządzeniach video – stąd zresztą nazwa atrybutu (**V**ideo **C**assette **R**ecorder). Przyciski te są umieszczane w lewej części poziomego paska przewijania (jeśli występuje). Przyciski przewijania umożliwiają przeglądanie zawartości listy za pomocą myszki.

Generowane zdarzenia:

EVENT:Selected	Kontrolka otrzymała aktywność wprowadzania.
EVENT:Accepted	Użytkownik wybrał daną z listy lub wpisał ją bezpośrednio do kontrolki, następnie przekazał aktywność wprowadzania do innej kontrolki.
EVENT:Rejected	Użytkownik wprowadził niewłaściwą wartość, niezgodną ze wzorcem wprowadzania.
EVENT:NewSelection	Zmienił się aktualnie wybrany element listy (podświetlenie zostało przesunięte w górę lub w dół) lub też użytkownik nacisnął dowolny klawisz (tylko z atrybutem IMM).
EVENT:PreAlertKey	Użytkownik nacisnął klawisz wskazany atrybutem ALRT.
EVENT:AlertKey	Użytkownik nacisnął klawisz wskazany atrybutem ALRT.
EVENT:Drop	Operacja drag-and-drop dla przycisku zakończyła się pomyślnie.

	pomyślnie.
EVENT:ScrollUp	Użytkownik wcisnął klawisz strzałki w górę (tylko z atrybutem IMM).
EVENT:ScrollDown	Użytkownik wcisnął klawisz strzałki w dół (tylko z atrybutem IMM).
EVENT:PageUp	Użytkownik wcisnął klawisz PageUp (tylko z atrybutem IMM).
EVENT:PageDown	Użytkownik wcisnął klawisz PageDown (tylko z atrybutem IMM).
EVENT:ScrollTop	Użytkownik wcisnął klawisz Ctrl+PageUp (tylko z atrybutem IMM).
EVENT:ScrollBottom	Użytkownik wcisnął klawisz Ctrl+PageDown (tylko z atrybutem IMM).
EVENT:PreAlertKey	Użytkownik nacisnął klawisz ze znakiem (tylko z atrybutem IMM) lub wcisnął klawisz skrótu (zdefiniowany za pomocą atrybutu ALRT).
EVENT:AlertKey	Użytkownik nacisnął klawisz ze znakiem (tylko z atrybutem IMM) lub wcisnął klawisz skrótu (zdefiniowany za pomocą atrybutu ALRT).
EVENT:Locate	Użytkownik wcisnął przycisk lokatora w pasku przycisków VCR (tylko z atrybutem IMM).
EVENT:ScrollDrag	Użytkownik przesunął suwak na pasku przewijania; jego aktualną pozycję wskazuje PROP:VScrollPos (tylko z atrybutem IMM).
EVENT:ScrollTrack	Użytkownik przesuwa suwak na pasku przewijania; jego aktualną pozycję wskazuje PROP:VScrollPos (tylko z atrybutem IMM).
EVENT:DroppingDown	Użytkownik wcisnął przycisk z ikoną strzałki w dół (tylko z atrybutem DROP).
EVENT:DroppedDown	Lista została rozwinięta (tylko z atrybutem DROP).
EVENT:ColumnResize	Kolumna w liście zmieniła swój rozmiar.

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            COMBO(@S8),AT(0,0,20,20),USE(C1),FROM(Que)
            COMBO(@S8),AT(20,0,20,20),USE(C2),FROM(Que),KEY(F10Key)
            COMBO(@S8),AT(40,0,20,20),USE(C3),FROM(Que),MSG('Button 3')
            COMBO(@S8),AT(60,0,20,20),USE(C4),FROM(Que),HLP('Check4Help')
            COMBO(@S8),AT(80,0,20,20),USE(C5),FROM(Q) |
              ,FORMAT('5C~List~15L~Box~'),COLUMN
            COMBO(@S8),AT(100,0,20,20),USE(C6),FROM(Que),FONT('Arial',12)
            COMBO(@S8),AT(120,0,20,20),USE(C7),FROM(Que),DROP(8)
            COMBO(@S8),AT(140,0,20,20),USE(C8),FROM(Que),HVSCROLL,VCR
            COMBO(@S8),AT(160,0,20,20),USE(C9),FROM(Que),IMM
            COMBO(@S8),AT(180,0,20,20),USE(C10),FROM(Que),CURSOR(CURSOR:Wait)
            COMBO(@S8),AT(200,0,20,20),USE(C11),FROM(Que),ALRT(F10Key)
            COMBO(@S8),AT(220,0,20,20),USE(C12),FROM(Que),LEFT
            COMBO(@S8),AT(240,0,20,20),USE(C13),FROM(Que),RIGHT
            COMBO(@S8),AT(260,0,20,20),USE(C14),FROM(Que),CENTER
            COMBO(@N8.2),AT(280,0,20,20),USE(C15),FROM(Que),DECIMAL
            COMBO(@S8),AT(300,0,20,20),USE(C16),FROM('Apples|Peaches|Pumpkin|Pie')
            COMBO(@S8),AT(320,0,20,20),USE(C17),FROM('TBA')
          END
  CODE
    OPEN(MDIChild)
    ?C17{PROP:From} = 'Live|Long|And|Prosper'        ! przypisanie atrybutu FROM w czasie działania
    ACCEPT
      CASE ACCEPTED()
      OF ?C1
        LOOP X# = 1 to RECORDS(Que)                  ! sprawdzenie elementów kolejki
          GET(Que,X#)
          IF C1 = Que THEN BREAK.                    ! przerwanie pętli, jeśli występuje
        END
        IF X# > RECORDS(Que)                         ! kontrola BREAK
          Que = C1                                   ! i dodanie elementu
          ADD(Que)
    ...
```

Porównaj: LIST, ENTRY

ELLIPSE (deklaruje elipsę)

ELLIPSE ,**AT**( ) [,**USE**( )] [,**DISABLE**] [,**COLOR**( )] [,**FILL**( )] [,**FULL**] [,**SCROLL**] [,**HIDE**] [,**LINEWIDTH**( )]

ELLIPSE Umieszcza elipsę w oknie WINDOW, na pasku narzędzi TOOLBAR, lub w raporcie REPORT.

AT Określa początkowy rozmiar i położenie kontrolki (PROP:AT). Jeśli jest pominięty, domyślne wartości są przyjmowane przez bibliotekę runtime.

USE Etykieta ekwiwalentu pola umożliwiająca odwoływanie się do kontrolki w kodzie wykonywalnym (PROP:USE).

DISABLE Powoduje, że kontrolka po pierwszym otwarciu okna WINDOW lub APPLICATION jest niedostępna, wyszarzona (PROP:DISABLE).

COLOR Określa kolor obramowania elipsy (PROP:COLOR). Jeśli atrybut ten zostanie pominięty, elipsa nie będzie miała obramowania.

FILL Określa kolor wypełnienia dla kontrolki (PROP:FILL). Jeśli atrybut ten zostanie pominięty, kontrolka nie będzie wypełniania żadnym kolorem.

FULL Powoduje, że rozmiar dla każdego brakującego atrybutu AT (długość i wysokość) kontrolki zwiększa się tak, by obejmował cały dostępny obszar okna WINDOW (PROP:FULL). Nie dotyczy raportu.

SCROLL Nadanie tego parametru powoduje, że kontrolka jest przewijana wraz z oknem (PROP:SCROLL). Nie dotyczy raportów.

HIDE Powoduje, że po pierwszym otwarciu okna WINDOW lub APPLICATION kontrolka pozostaje ukryta (PROP:HIDE). Zostanie ona wyświetlona dopiero po użyciu procedury UNHIDE.

LINEWIDTH Określa szerokość linii, którą jest rysowana elipsa (PROP:LINEWIDTH).

Kontrolka **ELLIPSE** umieszcza „okrągłą” figurę w oknie WINDOW, pasku narzędzi TOOLBAR lub raporcie REPORT w pozycji i o rozmiarze określonym przez atrybut AT. Elipsa jest rysowana w granicach zdefiniowanych parametrami *x, y, width* oraz *height* atrybutu AT. Parametry *x* i *y* stanowią współrzędne punktu startowego, a parametry *width* i *height* określają rozmiar w poziomie i w pionie “obszaru ograniczającego”. Ta kontrolka nie może otrzymywać aktywności wprowadzania i nie generuje żadnych zdarzeń.

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            ELLIPSE,FILL(COLOR:MENU),FULL                          ! wypełniona, pełny ekran, czarna ramka
            ELLIPSE,AT(0,0,20,20)                                  ! nie wypełniona, czarna ramka
            ELLIPSE,AT(0,20,20,20),USE(?Box1),DISABLE              ! wyszarzona
            ELLIPSE,AT(20,20,20,20),ROUND                          ! nie wypełniona, zaokrąglona, czarna ramka
            ELLIPSE,AT(40,40,20,20),FILL(COLOR:ACTIVEBORDER)       ! wypełniona, czarna ramka
            ELLIPSE,AT(60,60,20,20),COLOR(COLOR:ACTIVEBORDeR)      ! nie wypełniona, kolor ramki aktywnej
            ELLIPSE,AT(480,180,20,20),SCROLL                       ! przewijane wraz z ekranem
          END
CustRpt REPORT,AT(1000,1000,6500,9000),THOUS
CustDetail      DETAIL,AT(0,0,6500,1000)
              ELLIPSE,AT(0,0,20,20)                                ! nie wypełniona, czarna ramka
              ELLIPSE,AT(0,20,20,20),USE(?Ellipse1),DISABLE        ! nie wypełniona, czarna ramka, wyszarzona
              ELLIPSE,AT(20,20,20,20),ROUND                        ! nie wypełniona, zaokrąglona, czarna ramka
              ELLIPSE,AT(40,40,20,20),FILL(COLOR:ACTIVEBORDER)     ! wypełniona, czarna ramka
              ELLIPSE,AT(60,60,20,20),COLOR(COLOR:ACTIVEBORDER)    ! nie wypełniona, kolor ramki aktywnej
            END
          END
```

ENTRY (deklaruje pole wprowadzania)

```
ENTRY( picture) ,AT() [,CURSOR()] [,USE()] [,DISABLE] [,KEY()] [,MSG()] [,HLP()] [,SKIP][,FONT()]
    [,IMM][,PASSWORD][,REQ][,FULL][,SCROLL][,ALRT()][,HIDE][,TIP( )][,TRN][,READONLY]
    [DROPID( )] [,  | INS  | ] [,  | CAP  | ] [,  | LEFT    | ] [,COLOR( )] [,MASK]
                    | OVR  |      | UPR  |      | RIGHT   |
                                               | CENTER  |
                                               | DECIMAL |
```

ENTRY Umieszcza pole wprowadzania danych w oknie WINDOW lub na pasku narzędzi TOOLBAR.

picture Wzorzec wyświetlania określający format wprowadzanych do kontrolki danych (PROP:Text).

AT Określa początkowy rozmiar i położenie kontrolki (PROP:AT). Jeśli jest pominięty, domyślne wartości są przyjmowane przez bibliotekę runtime.

CURSOR Określa kursor wyświetlany w momencie, gdy wskaźnik myszki zatrzyma się na kontrolce (PROP:CURSOR). Jeśli ominiemy ten atrybut, jest stosowany kursor wskazany dla okna lub też domyślny kursor systemu Windows.

USE Etykieta zmiennej, w której jest zapisywana wartość wpisana w kontrolce przez użytkownika (PROP:USE).

DISABLE Powoduje, że kontrolka po pierwszym otwarciu okna WINDOW lub APPLICATION jest niedostępna, wyszarzona (PROP:DISABLE).

KEY Określa stałą całkowitą lub ekwiwalent kodu klawisza, którego wciśnięcie powoduje bezpośrednie przekazanie aktywności kontrolce (PROP:KEY).

MSG Określa tekst wyświetlany w pasku stanu aplikacji wtedy, gdy dana kontrolka jest aktywna (PROP:MSG).

HLP Określa łańcuch stanowiący identyfikator sekcji systemu pomocy związanej z daną kontrolką (PROP:HLP).

SKIP Powoduje, że kontrolka może otrzymać aktywność wprowadzania tylko wtedy, gdy zostanie kliknięta myszką lub zostanie wciśnięty przypisany do niej klawisz skrótu. Kontrolka nie zachowuje aktywności wprowadzania (PROP:SKIP).

FONT Określa czcionkę dla danej kontrolki (PROP:FONT).

IMM Wymusza generowanie zdarzenia za każdym razem, gdy użytkownik naciśnie jakiś klawisz (PROP:IMM).

PASSWORD Powoduje, że zamiast wprowadzanych przez użytkownika znaków, w kontrolce będą wyświetlane znaki gwiazdki (*); jest to przydatne na przykład wtedy, gdy kontrolka służy do wprowadzania hasła (PROP:PASSWORD).

REQ Określa, że kontrolka nie może pozostać nie wypełniona i nie może zawierać wartości zerowej (PROP:REQ).

FULL Powoduje, że rozmiar dla każdego brakującego atrybutu AT (długość i wysokość) kontrolki zwiększa się tak, by obejmował cały dostępny obszar okna WINDOW (PROP:FULL). Nie dotyczy raportu.

SCROLL Nadanie tego parametru powoduje, że kontrolka jest przewijana wraz z oknem (PROP:SCROLL).

ALRT Określa klawisze skrótu aktywne dla kontrolki (PROP:ALRT).

HIDE Powoduje, że po pierwszym otwarciu okna WINDOW lub APPLICATION kontrolka pozostaje ukryta (PROP:HIDE). Zostanie ona wyświetlona dopiero po użyciu procedury UNHIDE.

TIP Definiuje tekst wyświetlany „w dymkach" po zatrzymaniu kursora myszki na kontrolce (PROP:ToolTip).

TRN Powoduje, że kontrolka jest wyświetlana jako przeźroczysta, nie przykrywa innych kontrolek ani tła (PROP:TRN).

READONLY Powoduje, że nie jest możliwe wprowadzanie danych do kontrolki; służy ona wtedy tylko i wyłącznie do wyświetlania danych (PROP:READONLY).

DROPID Określa, że kontrolka może służyć jako cel dla operacji „przenieś i upuść" (PROP:DROPID).

INS/OVR Wprowadza tryb wstawiania (Insert) lub zastępowania (Overwrite) podczas wprowadzania danych (PROP:INS i PROP:OVR). Daje to efekt tylko w przypadku okien posiadających atrybut MASK.

UPR/CAP Wprowadza tryb zastępowania wszystkich liter na wielkie lub na kapitaliki (Każda Pierwsza Litera Wyrazu W Zdaniu Jest Wielka); odpowiednie właściwości to PROP:UPR i PROP:CAP.

LEFT Powoduje, że dane są wyrównywane do lewej w obszarze określonym przez atrybut AT (PROP:LEFT).

RIGHT Powoduje, że dane w kontrolce są wyrównywane do prawej w obszarze określonym przez atrybut AT (PROP:RIGHT).

CENTER Powoduje, że dane w kontrolce są centrowane w obszarze określonym przez atrybut AT (PROP:CENTER).

DECIMAL Powoduje, że dane w kontrolce są wyrównywane do kropki dziesiętnej w obszarze określonym przez atrybut AT (PROP:DECIMAL).

MASK Wymusza wprowadzanie według wzorca w polu tekstowym kontrolki (PROP:MASK).

COLOR Określa kolor tła i kolor podświetlenia dla kontrolki (PROP:COLOR).

MASK Wymusza wprowadzanie według wzorca w polu tekstowym (PROP:MASK).

Kontrolka **ENTRY** umieszcza pole wprowadzania danych w oknie WINDOW lub w pasku narzędzi TOOLBAR w pozycji określonej przez atrybut AT.

Dane wprowadzone do kontrolki są formatowane zgodnie ze wzorcem *picture*; są one zapamiętywane w zmiennej wskazanej za pomocą atrybutu USE w momencie, gdy użytkownik zatwierdzi kontrolkę i przeniesie aktywność wprowadzania do innej kontrolki.

Dane są przewijane w poziomie, tak by umożliwić użytkownikowi wprowadzenie takiej ilości znaków, jaką zadeklarowano dla zmiennej. Do przesuwania zawartości kontrolki ENTRY służą klawisze sterowania kursorem.

Standardowe funkcje Windows, takie jak kopiowanie, wycinanie i wklejanie tekstu są automatycznie dostępne dla kontrolki; stosuje się tu odpowiednie kombinacje klawiszy: Ctrl+C, Ctrl+X, Ctrl+V. Zaimplementowana została również operacja „Cofnij", jest ona wykonywana po wciśnięciu kombinacji Ctrl+Z (zanim użytkownik przeniesie aktywność wprowadzania do innej kontrolki).

Kontrolka ENTRY posiadająca atrybut PASSWORD wyświetla znaki gwiazdki (*) zamiast wprowadzanych znaków. Operacje „Wytnij" i „Kopiuj" dla takiej kontrolki są niedostępne. Jest to bardzo przydatne w sytuacjach, gdy pole tekstowe ma służyć na przykład do wprowadzenia hasła.

Kontrolka ENTRY z atrybutem SKIP służyć powinna do wprowadzania danych do rzadko wypełnianych pól. Dane, które mają być tylko wyświetlane, bez możliwości aktualizacji, powinny znaleźć się w kontrolce posiadającej atrybut READONLY.

Atrybut MASK wprowadza kontrolowany tryb edycji. Oznacza to, że każdy wpisany do pola znak jest automatycznie weryfikowany pod kątem zgodności ze wzorcem wprowadzania kontrolki (np. czy wprowadzony znak jest cyfrą, gdy kontrolka ma zawierać dane numeryczne itp.). Zmusza to użytkownika do wprowadzania danych w formacie określonym przez wzorzec przypisany do kontrolki. Jeśli atrybut zostanie pominięty, użytkownik może wpisać do kontrolki dowolne dane; jest to domyślny tryb edycji. Dane są formatowane zgodnie ze wzorcem kontrolki dopiero po jej zaakceptowaniu przez użytkownika (EVENT:Accepted). Dzięki temu użytkownik może wprowadzać dane tak, jak mu się podoba, a one i tak, na koniec, zostaną prawidłowo sformatowane. W przypadku, gdy użytkownik wprowadzi dane w formacie niezgodnym ze wzorcem określonym dla kontrolki, biblioteka runtime dokona próby rozpoznania formatu zastosowanego przez użytkownika i przekonwertowania go na format stosowany w kontrolce. Dla przykładu, jeśli użytkownik wpisze datę w postaci „January 1, 1995" w kontrolce, dla której został określony wzorzec @D1, biblioteka runtime sformatuje datę do postaci „1/1/95." Operacja ta zostanie wykonana wtedy, gdy użytkownik przeniesie aktywność wprowadzania do innej kontrolki. Jeśli biblioteka runtime nie będzie w stanie rozpoznać zastosowanego przez użytkownika formatu, nie zaktualizuje ona zawartości zmiennej wskazanej przez atrybut USE kontrolki i wygeneruje zdarzenie EVENT:Rejected.

Generowane zdarzenia:

EVENT:Selected	Kontrolka otrzymała aktywność wprowadzania.
EVENT:Accepted	Użytkownik zaakceptował kontrolkę; utraciła ona aktywność wprowadzania.
EVENT:Rejected	Użytkownik wprowadził niewłaściwą wartość, niezgodną ze wzorcem wprowadzania.
EVENT:PreAlertKey	Użytkownik nacisnął klawisz wskazany atrybutem ALRT.
EVENT:AlertKey	Użytkownik nacisnął klawisz wskazany atrybutem ALRT.
EVENT:Drop	Operacja drag-and-drop dla przycisku zakończyła się pomyślnie.
EVENT:NewSelection	Użytkownik wprowadził znak (tylko z atrybutem IMM).

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            ENTRY(@S8),AT(0,0,20,20),USE(E1)
            ENTRY(@S8),AT(20,0,20,20),USE(E2),KEY(F10Key)
            ENTRY(@S8),AT(40,0,20,20),USE(E3),MSG('Button 3')
            ENTRY(@S8),AT(60,0,20,20),USE(E4),HLP('Entry4Help')
            ENTRY(@S8),AT(80,0,20,20),USE(E5),DISABLE
            ENTRY(@S8),AT(100,0,20,20),USE(E6),FONT('Arial',12)
            ENTRY(@S8),AT(120,0,20,20),USE(E7),REQ,INS,CAP
            ENTRY(@S8),AT(140,0,20,20),USE(E8),SCROLL,OVR,UPR
            ENTRY(@S8),AT(180,0,20,20),USE(E9),CURSOR(CURSOR:Wait),IMM
            ENTRY(@S8),AT(200,0,20,20),USE(E10),ALRT(F10Key)
            ENTRY(@N8.2),AT(280,0,20,20),USE(E11),DECIMAL(10)
          END
```

Porównaj: TEXT, PROMPT

GROUP (deklaruje grupę kontrolek)

```
GROUP( tekst) ,AT() [,CURSOR()][,USE()][,DISABLE][,KEY()][,MSG()][,HLP()][,FONT()][,TIP()]
    [,BOXED][,FULL][,SCROLL][,HIDE][,ALRT()][,SKIP][,DROPID()][,COLOR()][,BEVEL()]
  controls
END
```

GROUP Deklaruje grupę kontrolek, do której można się odwoływać, jak do jednej całości.

tekst Stała łańcuchowa zawierająca tekst opisujący grupę GROUP (PROP:Text). Może on zawierać znak ampersand (&) w celu wskazania (podkreślenia) klawisza skrótu przypisanego do okienka grupy. *tekst* jest wyświetlany tylko wtedy, gdy grupa posiada atrybut BOXED.

AT Określa początkowy rozmiar i położenie kontrolki (PROP:AT). Jeśli jest pominięty, domyślne wartości są przyjmowane przez bibliotekę runtime.

CURSOR Określa kursor wyświetlany w momencie, gdy wskaźnik myszki zatrzyma się na kontrolce GROUP lub dowolnej kontrolce w niej się znajdującej (PROP:CURSOR). Jeśli ominiemy ten atrybut, jest stosowany kursor wskazany dla okna lub też domyślny kursor systemu Windows. Nie dotyczy raportu.

USE Ekwiwalent nazwy pola umożliwiający odwoływanie się do kontrolki w kodzie wykonywalnym (PROP:USE).

DISABLE Powoduje, że kontrolka GROUP i wszystkie kontrolki znajdujące się w grupie, po pierwszym otwarciu okna WINDOW lub APPLICATION, są niedostępne, wyszarzone (PROP:DISABLE). Nie dotyczy to raportów.

KEY Określa stałą całkowitą lub ekwiwalent kodu klawisza, którego wciśnięcie powoduje bezpośrednie przekazanie aktywności pierwszej kontrolce grupy GROUP (PROP:KEY). Nie dotyczy raportu.

MSG Określa tekst wyświetlany w pasku stanu aplikacji wtedy, gdy dowolna kontrolka grupy GROUP jest aktywna (PROP:MSG). Nie dotyczy raportu.

HLP Określa łańcuch stanowiący domyślny identyfikator sekcji systemu pomocy związanej z dowolną kontrolką grupy GROUP (PROP:HLP). Nie dotyczy raportu.

FONT Określa czcionkę dla danej kontrolki, stanowi ona domyślną czcionkę dla wszystkich kontrolek w grupie GROUP (PROP:FONT).

TIP Definiuje tekst wyświetlany „w dymkach” po zatrzymaniu kursora myszki na kontrolce (PROP:ToolTip). Nie dotyczy raportu.

BOXED Powoduje rysowanie, pojedynczą linią, ramki wokół grupy; w lewym górnym rogu tej ramki jest wyświetlany tekst zdefiniowany za pomocą parametru *text* (PROP:BOXED).

FULL Powoduje, że rozmiar dla każdego brakującego atrybutu AT (długość i wysokość) kontrolki zwiększa się tak, by obejmował cały dostępny obszar okna WINDOW (PROP:FULL). Nie dotyczy raportu.

SCROLL Nadanie tego atrybutu powoduje, że kontrolka GROUP i znajdujące się w niej kontrolki są przewijane wraz z oknem (PROP:SCROLL).

HIDE Powoduje, że po pierwszym otwarciu okna WINDOW lub APPLICATION kontrolka GROUP i znajdujące się w niej kontroli pozostają ukryte (PROP:HIDE). Zostaną one wyświetlone dopiero po użyciu procedury UNHIDE.

ALRT Określa klawisze skrótu aktywne dla kontrolek grupy GROUP (PROP:ALRT).

SKIP Powoduje, że kontrolki w grupie GROUP nie otrzymują aktywności (są pomijane podczas poruszania się między kontrolkami za pomocą klawisza TAB). Dostęp do nich jest możliwy po kliknięciu myszką lub po wciśnięciu klawisza skrótu związanego z daną kontrolką (PROP:SKIP). Nie dotyczy raportu.

DROPID Określa, że kontrolka może służyć jako cel dla operacji „przenieś i upuść" (PROP:DROPID). Nie dotyczy raportu.

COLOR Określa domyślny kolor tła i kolor podświetlenia dla kontrolek znajdujących się w grupie GROUP (PROP:COLOR).

BEVEL Umożliwia wprowadzenie efektu trójwymiarowości dla ramki grupy (PROP:BEVEL). Nie dotyczy raportu.

controls Deklaracje kontrolek tworzących grupę.

GROUP deklaruje grupę kontrolek, do której można się odwoływać, jak do pojedynczego egzemplarza.

Grupa GROUP umożliwia użytkownikowi stosowanie klawiszy strzałek, zamiast klawisza Tab, do przemieszczania się pomiędzy kontrolkami *controls* w grupie GROUP. Dla grupy można zadeklarować wspólny atrybut MSG i HLP. Kontrolka GROUP nie otrzymuje aktywności wprowadzania.

Generowane zdarzenia:

EVENT:Drop	Operacja drag-and-drop dla kontrolki zakończyła się pomyślnie.

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            GROUP('Group 1'),USE(?G1),KEY(F10Key)
              ENTRY(@S8),AT(0,0,20,20),USE(?E1)
              ENTRY(@S8),AT(20,0,20,20),USE(?E2)
            END
            GROUP('Group 2'),USE(?G2),MSG('Group 2'),CURSOR(CURSOR:Wait)
              ENTRY(@S8),AT(40,0,20,20),USE(?E3)
              ENTRY(@S8),AT(60,0,20,20),USE(?E4)
            END
            GROUP('Group 3'),USE(?G3),AT(80,0,20,20),BOXED
              ENTRY(@S8),AT(80,0,20,20),USE(?E5)
              ENTRY(@S8),AT(100,0,20,20),USE(?E6)
            END
          END
CustRpt REPORT,AT(1000,1000,6500,9000),THOUS
CustDetail     DETAIL,AT(0,0,6500,1000)
             GROUP('Group 1'),USE(!G1),AT(80,0,20,20),BOXED
               STRING(@S8),AT(80,0,20,20),USE(E5)
               STRING(@S8),AT(100,0,20,20),USE(E6)
             END
             GROUP('Group 2'),USE(?G2),FONT('Arial',12)
               STRING(@S8),AT(120,0,20,20),USE(E7)
               STRING(@S8),AT(140,0,20,20),USE(E8)
             END
           END
          END
```

Porównaj: PANEL

IMAGE (deklaruje grafikę)

```
IMAGE( file) ,AT( ) [,USE( )] [,DISABLE] [,FULL] [,SCROLL] [,HIDE]
                [,  | TILED      | ] [,  | HSCROLL   | ]
                    | CENTERED   |       | VSCROLL   |
                                          | HVSCROLL  |
```

IMAGE Umieszcza grafikę w oknie WINDOW, pasku narzędzi TOOLBAR lub w raporcie REPORT.

file Ścieżka dostępu do pliku zawierającego wyświetlaną grafikę (PROP:Text). Plik o określonej z góry nazwie jest na stałe dołączany do pliku .EXE jako zasób aplikacji.

AT Określa początkowy rozmiar i położenie kontrolki (PROP:AT). Jeśli jest pominięty, domyślne wartości są przyjmowane przez bibliotekę runtime.

USE Ekwiwalent nazwy pola umożliwiający odwoływanie się do kontrolki w kodzie wykonywalnym (PROP:USE).

DISABLE Powoduje, że kontrolka po pierwszym otwarciu okna WINDOW lub APPLICATION jest niedostępna, wyszarzona (PROP:DISABLE).

FULL Powoduje, że rozmiar dla każdego brakującego atrybutu AT (długość i wysokość) kontrolki zwiększa się tak, by obejmował cały dostępny obszar okna WINDOW (PROP:FULL). Nie dotyczy raportu.

SCROLL Nadanie tego parametru powoduje, że kontrolka jest przewijana wraz z oknem (PROP:SCROLL). Nie dotyczy raportów.

HIDE Powoduje, że po pierwszym otwarciu okna WINDOW lub APPLICATION kontrolka pozostaje ukryta (PROP:HIDE). Zostanie ona wyświetlona dopiero po użyciu procedury UNHIDE.

TILED Powoduje, że grafika jest wyświetlana w swoim oryginalnym rozmiarze i jest powielana tak, by wypełnić cały obszar kontrolki (PROP:TILED).

CENTERED Powoduje, że grafika jest wyświetlana w swoim oryginalnym rozmiarze i jest umieszczana centralnie w kontrolce (PROP:CENTERED).

HSCROLL Powoduje, że w do rysunku jest automatycznie dołączany poziomy pasek przewijania, gdy jego poziomy rozmiar nie pozwala na wyświetlenie całej zawartości (PROP:HSCROLL). Nie dotyczy raportów.

VSCROLL Powoduje, że w do rysunku jest automatycznie dołączany pionowy pasek przewijania, gdy jego pionowy rozmiar nie pozwala na wyświetlenie całej zawartości (PROP:VSCROLL). Nie dotyczy raportów.

HVSCROLL Powoduje, że do rysunku są dołączane poziomy i/lub pionowy pasek przewijania, gdy rozmiar kontrolki nie pozwala na wyświetlenie go w całości (PROP:HVSCROLL). Nie dotyczy raportu.

Kontrolka **IMAGE** pozwala na umieszczenie grafiki w oknie WINDOW (lub w pasku narzędzi TOOLBAR) w pozycji określonej przez atrybut AT. Jeżeli nie został

określony atrybut TILED lub CENTERED, rozmiar grafiki jest dopasowywany tak, by całkowicie wypełniła ona obszar kontrolki określony przez AT.

Wyświetlany plik *file* może występować w formacie mapy bitowej (.BMP), PaintBrush (.PCX), Graphic Interchange Format (.GIF), JPEG (.JPG) lub jako metaplik Windows (.WMF). Plik *file* może również być plikiem ikony (.ICO), ale tylko w oknie WINDOW – nigdy w raporcie REPORT. Jest tak dlatego, bo Windows nie obsługuje standardowo druku ikon. Typ pliku *file* jest określany na podstawie jego rozszerzenia.

Ta kontrolka nie może otrzymywać aktywności wprowadzania, nie generuje również żadnych zdarzeń.

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            IMAGE('PIC.BMP'),AT(0,0,20,20),USE(?I1)
            IMAGE('PIC.WMF'),AT(40,0,20,20),USE(?I3),SCROLL
          END
CustRpt REPORT,AT(1000,1000,6500,9000),THOUS
CustDetail      DETAIL,AT(0,0,6500,1000)
                  IMAGE('PIC.BMP'),AT(0,0,20,20),USE(?I1)
                  IMAGE('PIC.WMF'),AT(40,0,20,20),USE(?I2)
                  IMAGE('PIC.JPG'),AT(60,0,20,20),USE(?I3)
                END
              END
```

Porównaj: PALETTE

ITEM (deklaruje element menu)

ITEM(*text*) [,**USE**( )] [,**KEY**( )] [,**MSG**( )] [,**HLP**( )] [,**STD**( )] [,**CHECK**] [,**DISABLE**] [,**SEPARATOR**]
[,**ICON**( )] [,**FONT**( )] [, | **FIRST** |]
| **LAST** |

ITEM Deklaruje element systemu menu w ramach struktury MENUBAR lub MENU.

Tekst Stała łańcuchowa zawierająca tekst elementu menu (PROP:Text).

USE Ekwiwalent nazwy pola umożliwiający odwoływanie się do elementu menu w kodzie wykonywalnym (PROP:USE) lub zmienna wykorzystywana przez CHECK (PROP:USE).

KEY Określa stałą całkowitą lub ekwiwalent kodu klawisza, którego wciśnięcie powoduje bezpośrednie wykonanie akcji związanej z danym elementem menu (PROP:KEY).

MSG Określa tekst wyświetlany w pasku stanu aplikacji wtedy, gdy menu jest podświetlone (PROP:MSG).

HLP Określa łańcuch stanowiący identyfikator sekcji systemu pomocy związanej z danym elementem menu (PROP:HLP).

STD Określa stałą całkowitą lub ekwiwalent dla „standardowej akcji Windows", która jest wykonywana przez dany element menu (PROP:STD).

CHECK Definiuje dany element, jako element przełącznikowy - mogący być włączony (on) lub wyłączony (off) – (PROP:CHECK).

DISABLE Powoduje, że kontrolka po pierwszym otwarciu okna WINDOW lub APPLICATION jest niedostępna, wyszarzona (PROP:DISABLE).

SEPARATOR Powoduje, że dany element jest wyświetlany jako pozioma linia oddzielająca od siebie inne elementy menu. Jeżeli dany element posiada taki atrybut, nie określamy żadnych innych.

ICON Wskazuje standardową ikonę lub plik z grafiką, która będzie wyświetlana w elemencie menu (PROP:ICON).

FONT Określa czcionkę dla danej kontrolki (PROP:FONT).

FIRST Powoduje, że dany element jest umieszczany jako pierwszy w sytuacji gdy są łączone różne menu (PROP:FIRST).

LAST Powoduje, że dany element jest umieszczany jako ostatni w sytuacji gdy są łączone różne menu (PROP:LAST).

Kontrolka **ITEM** deklaruje element dla struktury MENUBAR lub MENU.

Łańcuch *text* może zawierać znak ampersand (&) powodujący podkreślenie (w menu) występującego po nim znaku; znak ten staje się również automatycznie klawiszem skrótu dla elementu menu. Jeżeli element menu ITEM znajduje się w pasku menu, wciśnięcie klawisza Alt łącznie z klawiszem skrótu jest równoznaczne z jego wybraniem. Jeśli element menu ITEM znajduje się w MENU, wciśnięcie samego klawisza skrótu, gdy dane menu jest wyświetlane, jest równoznaczne z wybraniem elementu. Jeśli w tekście *text* nie znajdzie się znak ampersand, klawiszem skrótu staje się pierwszy niepusty jego znak; nie jest on jednak podkreślany. Jeżeli w tekście *text*

chcemy wyświetlić znak ampersand, musimy go tam umieścić w postaci dwóch znaków ampersand położonych obok siebie (&&). Atrybut KEY pozwala na zdefiniowanie oddzielnego klawisza skrótu dla elementu menu. Może to być dowolna kombinacja klawiszy, której wciśnięcie będzie równoznaczne z wybraniem elementu menu.

Z każdym elementem ITEM jest zazwyczaj powiązany fragment kodu, który jest wykonywany w odpowiedzi na jego wybranie; wyjątkiem jest sytuacja, gdy dla elementu określony został atrybut STD. Atrybut ten przypisuje elementowi standardową akcję Windows, taką jak na przykład kaskadowe rozmieszczenie okien, wywołanie okienka konfiguracji drukarki itp.

Atrybut SEPARATOR tworzy element ITEM służący jako separator oddzielający od siebie inne elementy. Taki element nie posiada ani parametru *text*, ani żadnych innych atrybutów. Występuje on w postaci poziomej linii w menu.

Element ITEM nie występujący w ramach struktury MENU jest umieszczany w pasku menu (menu głównym). Powoduje to utworzenie elementu paska menu, dla którego nie jest rozwijane podmenu.

Generowane zdarzenia:

EVENT:Accepted	Element menu został wybrany przez użytkownika.

Przykład:

```
MainWinAPPLICATION('My Application'),SYSTEM,MAX,ICON('MyIcon.ICO'),STATUS |
        ,HVSCROLL,RESIZE
      MENUBAR
        ITEM('E&xit!'),USE(?MainExit),FIRST
        MENU('File'),USE(?FileMenu),FIRST
          ITEM('Open...'),USE(?OpenFile) ,HLP('OpenFileHelp') ,FIRST
          ITEM('Close'),USE(?CloseFile),HLP('CloseFileHelp'),DISABLE
          ITEM('Auto Increment'),USE(ToggleVar),CHECK
        END
        MENU('Edit'),USE(?EditMenu),KEY(CtrlE),HLP('EditMenuHelp')
          ITEM('Undo'),USE(?UndoText),KEY(CtrlZ),STD(STD:Undo),DISABLE
          ITEM,SEPARATOR
          ITEM('Cu&t'),USE(?CutText),KEY(CtrlX),STD(STD:Cut),DISABLE
          ITEM('Copy'),USE(?CopyText),KEY(CtrlC),STD(STD:Copy),DISABLE
          ITEM('Paste'),USE(?PasteText),KEY(CtrlV),STD(STD:Paste),DISABLE
        END
        MENU('Window'),STD(STD:WindowList),MSG('Arrange or Select Window'),LAST
          ITEM('Tile'),STD(STD:TileWindow)
          ITEM('Cascade'),STD(STD:CascadeWindow)
          ITEM('Arrange Icons'),STD(STD:ArrangeIcons)
          ITEM,SEPARATOR
        END
        MENU('Help'),USE(?HelpMenu),LAST,RIGHT
          ITEM('Contents'),USE(?HelpContents),STD(STD:HelpIndex)
          ITEM('Search for Help On...'),USE(?HelpSearch),STD(STD:HelpSearch)
          ITEM('How to Use Help'),USE(?HelpOnHelp),STD(STD:HelpOnHelp)
          ITEM('About MyApp...'),USE(?HelpAbout),MSG('Copyright Info'),LAST
        END
      END
    END
```

LINE (deklaruje linię)

LINE ,**AT**( ) [,**USE**( )] [,**DISABLE**] [,**COLOR**( )] [,**FULL**] [,**SCROLL**] [,**HIDE**] [,**LINEWIDTH**( )]

LINE Umieszcza linię w oknie WINDOW, pasku narzędzi TOOLBAR lub w raporcie REPORT.

AT Określa początkowy rozmiar i położenie kontrolki (PROP:AT). Jeśli jest pominięty, domyślne wartości są przyjmowane przez bibliotekę runtime.

USE Ekwiwalent nazwy pola umożliwiający odwoływanie się do kontrolki w kodzie wykonywalnym (PROP:USE).

DISABLE Powoduje, że kontrolka po pierwszym otwarciu okna WINDOW lub APPLICATION jest niedostępna, wyszarzona (PROP:DISABLE).

COLOR Określa kolor linii (PROP:COLOR). Jeśli atrybut ten zostanie pominięty, domyślnie jest wybierany kolor czarny.

FULL Powoduje, że rozmiar dla każdego brakującego atrybutu AT (długość i wysokość) kontrolki zwiększa się tak, by obejmował cały dostępny obszar okna WINDOW (PROP:FULL). Nie dotyczy raportu.

SCROLL Nadanie tego parametru powoduje, że kontrolka jest przewijana wraz z oknem (PROP:SCROLL). Nie dotyczy raportów.

HIDE Powoduje, że po pierwszym otwarciu okna WINDOW lub APPLICATION kontrolka pozostaje ukryta (PROP:HIDE). Zostanie ona wyświetlona dopiero po użyciu procedury UNHIDE.

LINEWIDTH Definiuje grubość linii LINE (PROP:LINEWIDTH).

Kontrolka **LINE** umieszcza linię prostą w oknie WINDOW, pasku narzędzi TOOLBAR lub raporcie REPORT w pozycji i o rozmiarach określonych przez atrybut AT. Parametry *x* i *y* atrybutu AT określają punkt początkowy linii. Parametry *width* i *height* atrybutu AT określają poziomą i pionową odległość do końcowego punktu linii. Jeśli obie te wartości są dodatnie, to linia biegnie w prawo i w dół w stosunku do punktu początkowego. Jeśli *width* jest wartością ujemną, linia biegnie w lewo; jeśli *height* ma wartość ujemną – linia biegnie w górę. W przypadku, gdy *width* lub *height* ma wartość zerową, linia jest odpowiednio linią pionową lub poziomą.

Ta kontrolka nie otrzymuje aktywności wprowadzania i nie generuje zdarzeń.

Szerokość	Wysokość	Rezultat
Dodatnia	Dodatnia	W prawo i w dół od punktu początkowego
Ujemna	Dodatnia	W lewo i w dół od punktu początkowego
Dodatnia	Ujemna	W prawo i w górę od punktu początkowego
Ujemna	Ujemna	W lewo i w górę od punktu początkowego
Zerowa	Dodatnia	Pionowo, w dół od punktu początkowego
Zerowa	Ujemna	Pionowo, w górę od punktu początkowego
Dodatnia	Zerowa	Poziomo, w prawo od punktu początkowego
Ujemna	Zerowa	Poziomo, w prawo od punktu początkowego

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            LINE,AT(60,60,20,20),COLOR(COLOR:ACTIVEBORDER)          ! kolor ramki
            LINE,AT(480,180,20,20),SCROLL                           ! przewijane wraz z ekranem
          END
CustRpt REPORT,AT(1000,1000,6500,9000),THOUS
CustDetail      DETAIL,AT(0,0,6500,1000)
              LINE,AT(60,60,20,20),COLOR(COLOR:ACTIVEBORDER)        ! kolor ramki
              LINE,AT(480,180,20,20),USE(?L2)
            END
          END
```

LIST (deklaruje okienko z listą elementów)

```
LIST ,FROM( ) ,AT( ) [,CURSOR( )] [,USE( )] [,DISABLE] [,KEY( )] [,MSG( )] [,HLP( )] [,SKIP]
     [,FONT( )] [,FORMAT( )] [,DROP] [,COLUMN] [,VCR] [,FULL] [,SCROLL] [,NOBAR]
     [,ALRT( )] [,HIDE] [,DRAGID( )] [,DROPID( )] [,TIP( )] [,GRID( )] [,TRN] [,COLOR( )]
     [,  | MARK()  |] [,  | HSCROLL    |][,  | LEFT    |]
         | IMM     |      | VSCROLL    |      | RIGHT   |
                          | HVSCROLL   |      | CENTER  |
                                              | DECIMAL |
```

LIST Umieszcza okienko z przewijalną listą elementów w oknie WINDOW, w pasku narzędzi TOOLBAR lub w raporcie REPORT.

FROM Określa źródło pochodzenia danych wyświetlanych w liście (PROP:FROM).

AT Określa początkowy rozmiar i położenie kontrolki (PROP:AT). Jeśli jest pominięty, domyślne wartości są przyjmowane przez bibliotekę runtime.

CURSOR Określa kursor wyświetlany w momencie, gdy wskaźnik myszki zatrzyma się na kontrolce (PROP:CURSOR). Jeśli ominiemy ten atrybut, jest stosowany kursor wskazany dla okna lub też domyślny kursor systemu Windows. Nie dotyczy raportu.

USE Ekwiwalent nazwy pola umożliwiający odwoływanie się do kontrolki w kodzie wykonywalnym lub etykieta zmiennej, w której jest zapisywana wartość wybrana przez użytkownika (PROP:USE).

DISABLE Powoduje, że kontrolka po pierwszym otwarciu okna WINDOW lub APPLICATION jest niedostępna, wyszarzona (PROP:DISABLE).

KEY Określa stałą całkowitą lub ekwiwalent kodu klawisza, którego wciśnięcie powoduje bezpośrednie przekazanie aktywności kontrolce (PROP:KEY). Nie dotyczy raportu.

MSG Określa tekst wyświetlany w pasku stanu aplikacji wtedy, gdy dana kontrolka jest aktywna (PROP:MSG). Nie dotyczy raportu.

HLP Określa łańcuch stanowiący identyfikator sekcji systemu pomocy związanej z daną kontrolką (PROP:HLP). Nie dotyczy raportu.

SKIP Powoduje, że kontrolka nie otrzymuje aktywności (jest pomijana podczas poruszania się między kontrolkami za pomocą klawisza TAB). Dostęp do niej jest możliwy po jej kliknięciu myszką lub po wciśnięciu klawisza skrótu z nią związanego (PROP:SKIP). Nie dotyczy raportu.

FONT Określa czcionkę dla danej kontrolki (PROP:FONT).

FORMAT Definiuje format wyświetlania danych w liście (PROP:FORMAT). Definicja ta obejmuje ikony, kolory, kontrolki rozwijania poziomów.

DROP Powoduje, że lista jest wyświetlana jako lista rozwijalna. Określa również maksymalną ilość wierszy, które będą widoczne po jej rozwinięciu (PROP:DROP). Nie dotyczy raportu.

COLUMN Powoduje, że podświetlany jest nie cały wiersz, a tylko pole w jednej kolumnie (PROP:COLUMN). Nie dotyczy raportu.

VCR Powoduje umieszczenie przycisków typu VCR (takich jak na magnetowidzie) w lewej części poziomego suwaka okienka (PROP:VCR).

FULL Powoduje, że rozmiar dla każdego brakującego atrybutu AT (długość i wysokość) kontrolki zwiększa się tak, by obejmował cały dostępny obszar okna WINDOW (PROP:FULL). Nie dotyczy raportu.

SCROLL Nadanie tego parametru powoduje, że kontrolka jest przewijana wraz z oknem (PROP:SCROLL). Nie dotyczy raportów.

NOBAR Powoduje, że podświetlenie jest wyświetlane tylko wtedy, gdy kontrolka LIST posiada aktywność wprowadzania (PROP:NOBAR). Nie dotyczy raportu.

ALRT Określa klawisze skrótu aktywne dla kontrolki (PROP:ALRT). Nie dotyczy raportu.

HIDE Powoduje, że po pierwszym otwarciu okna WINDOW lub APPLICATION kontrolka pozostaje ukryta (PROP:HIDE). Zostanie ona wyświetlona dopiero po użyciu procedury UNHIDE.

DRAGID Powoduje, że lista LIST może służyć jako źródło dla operacji „przenieś i upuść" (PROP:DRAGID). Nie dotyczy raportu.

DROPID Określa, że kontrolka może służyć jako cel dla operacji „przenieś i upuść" (PROP:DROPID). Nie dotyczy raportu.

TIP Definiuje tekst wyświetlany „w dymkach" po zatrzymaniu kursora myszki na kontrolce (PROP:ToolTip). Nie dotyczy raportu.

GRID Określa kolor dla linii rozdzielających poszczególne kolumny listy (PROP:GRID).

TRN Powoduje, że kontrolka jest wyświetlana jako przeźroczysta, nie przykrywa innych kontrolek ani tła (PROP:TRN).

COLOR Określa kolor tła i kolor podświetlenia dla kontrolki (PROP:COLOR).

MARK Umożliwia zaznaczanie w liście wielu elementów; zgodnie z ogólnymi zasadami stosowanymi w Windows – z użyciem klawiszy Shift i Ctrl (PROP:MARK). Nie dotyczy raportu.

IMM Wymusza generowanie zdarzenia za każdym razem, gdy użytkownik naciśnie jakiś klawisz (PROP:IMM). Nie dotyczy raportu.

HSCROLL Powoduje, że w okienku listy pojawia się automatycznie poziomy pasek przewijania, gdy jego poziomy rozmiar nie pozwala na wyświetlenie wszystkich danych (PROP:HSCROLL). Nie dotyczy raportów.

VSCROLL Powoduje, że w okienku listy pojawia się automatycznie pionowy pasek przewijania, gdy jego pionowy rozmiar nie pozwala na wyświetlenie wszystkich danych (PROP:VSCROLL). Nie dotyczy raportów.

HVSCROLL Powoduje, że w okienku listy pojawia się automatycznie pionowy lub poziomy pasek przewijania, gdy jego rozmiar nie pozwala na wyświetlenie wszystkich danych (PROP:HVSCROLL). Nie dotyczy raportów.

LEFT Powoduje, że dane w liście są wyrównywane do lewej (PROP:LEFT).

RIGHT Powoduje, że dane w liście są wyrównywane do prawej (PROP:RIGHT).

CENTER Powoduje, że dane w liście są centrowane (PROP:CENTER).

DECIMAL Powoduje, że dane w liście są wyrównywane do kropki dziesiętnej (PROP:DECIMAL).

Kontrolka **LIST** umieszcza przewijalną listę elementów w oknie WINDOW, pasku narzędzi TOOLBAR lub w raporcie REPORT w pozycji i o rozmiarze określonym przez atrybut AT.

Elementy wyświetlane w kontrolce LIST są pobierane albo z kolejki QUEUE albo z łańcucha STRING, które wskazujemy za pomocą atrybutu FROM i formatujemy poprzez parametry określone w atrybucie FORMAT (włączając w to kolory, ikony, kontrolki umożliwiające budowę listy drzewiastej).

Funkcja CHOICE daje w rezultacie numer aktualnie wybranego elementu kolejki QUEUE (wartość ta jest określana przez POINTER(kolejka)) w momencie, gdy dla kontrolki LIST zostaje wygenerowane zdarzenie EVENT:Accepted. Dane wyświetlane w liście LIST są automatycznie odświeżane za każdym razem, gdy zaczyna się pętla ACCEPT, niezależnie od tego, czy dla listy został określony atrybut AUTO, czy tez nie.

Lista LIST z atrybutem DROP wyświetla tylko aktualnie wybrany element. W momencie, gdy lista otrzyma aktywność wprowadzania i użytkownik naciśnie klawisz strzałki w dół bądź kliknie ikonę znajdującą się w jej prawej części, zostanie ona rozwinięta. Użytkownik ma wówczas możliwość podświetlenia wybranego jej elementu.

Kontrolka LIST z atrybutem IMM generuje zdarzenie za każdym razem, gdy użytkownik przesuwa podświetlenie w liście do innej pozycji lub naciska klawisz skrótu (wszystkie klawisze zarejestrowane za pomocą ALRT). Umożliwia to wstawienie kodu źródłowego odświeżającego kolejkę QUEUE zawierającą elementy listy, czy też pobranie odpowiedniego rekordu w celu wyświetlenia innych jego pól. Jeśli dla listy zadeklarowano atrybut VSCROLL, pojawia się w niej pionowy pasek przewijania. Gdy użytkownik klika jego obszar są generowane zdarzenia, ale lista nie jest przewijana. Jeśli chcemy uzyskać taki efekt, musimy zdefiniować odpowiednie wstawki kodu źródłowego. Do określenia aktualnej pozycji suwaka na pasku przewijania wykorzystujemy właściwość PROP:VscrollPos; pozycja ta znajduje się w zakresie od 0 (położenie górne) do 255 (położenie dolne).

Kontrolka LIST z atrybutem VCR posiada specjalne przyciski przewijania z piktogramami analogicznymi do tych, które spotykamy w urządzeniach video – stąd zresztą nazwa atrybutu (**V**ideo **C**assette **R**ecorder). Przyciski te są umieszczane w lewej części poziomego paska przewijania (jeśli występuje). Przyciski przewijania umożliwiają przeglądanie zawartości listy za pomocą myszki.

Lista LIST posiadająca atrybut DRAGID służy jako źródło operacji „przenieś i upuść”. Dzięki temu, za pomocą tej techniki, można „przeciągać” dane z listy do innych kontrolek. Lista z atrybutem LIST może służyć jako cel operacji „przenieś i upuść”, możemy w niej umieszczać dane „przeciągane” z innych kontrolek. Oba atrybuty są wzajemnie powiązane i określają specjalne sygnatury identyfikujące kontrolki, pomiędzy którymi mogą być realizowane operacje drag-and-drop. Procedury DRAGID() i DROPID(), w połączeniu z procedurą SETDROPID, są wykorzystywane do oprogramowywania operacji wymiany danych między kontrolkami w oparciu o technikę drag-and-drop.

Zastosowanie w raportach

Kontrolka LIST jest prawidłowa tylko w strukturach DETAIL. Z założenia ma ona umożliwiać kopiowanie w raporcie ustawień takich, jakie określa parametr FORMAT dla list okienkowych. Gdy jest drukowany pierwszy egzemplarz struktury DETAIL zawierającej kontrolkę LIST, wszystkie nagłówki zdefiniowane w atrybucie FORMAT są drukowane wraz z bieżącą pozycją atrybutu FROM. Gdy jest drukowany ostatni egzemplarz struktury DETAIL zawierającej kontrolkę LIST, stopki kontrolki LIST są drukowane wraz z pozycją atrybutu FROM.

Generowane zdarzenia:

EVENT:Selected	Kontrolka otrzymała aktywność wprowadzania.
EVENT:Accepted	Użytkownik zakończył wprowadzanie danych do kontrolki; utraciła ona aktywność wprowadzania.
EVENT:NewSelection	Zmienił się aktualnie wybrany element listy (podświetlenie zostało przesunięte w górę lub w dół).
EVENT:ScrollUp	Użytkownik wcisnął klawisz strzałki w górę (tylko z atrybutem IMM).
EVENT:ScrollDown	Użytkownik wcisnął klawisz strzałki w dół (tylko z atrybutem IMM).
EVENT:PageUp	Użytkownik wcisnął klawisz PageUp (tylko z atrybutem IMM).
EVENT:PageDown	Użytkownik wcisnął klawisz PageDown (tylko z atrybutem IMM).
EVENT:ScrollTop	Użytkownik wcisnął klawisz Ctrl+PageUp (tylko z atrybutem IMM).
EVENT:ScrollBottom	Użytkownik wcisnął klawisz Ctrl+PageDown (tylko z atrybutem IMM).
EVENT:Locate	Użytkownik wcisnął przycisk lokatora w pasku przycisków VCR (tylko z atrybutem IMM).
EVENT:ScrollDrag	Użytkownik przesunął suwak na pasku przewijania; jego aktualną pozycję wskazuje PROP:VScrollPos (tylko z atrybutem IMM).
EVENT:ScrollTrack	Użytkownik przesuwa suwak na pasku przewijania; jego aktualną pozycję wskazuje PROP:VScrollPos (tylko z atrybutem IMM).
EVENT:PreAlertKey	Użytkownik nacisnął klawisz ze znakiem (tylko z atrybutem IMM) lub klawisz skrótu zdefiniowany przez ALRT.
EVENT:AlertKey	Użytkownik nacisnął klawisz ze znakiem (tylko z atrybutem IMM) lub klawisz skrótu zdefiniowany przez ALRT.
EVENT:Dragging	Wskaźnik myszki, przy wciśniętym jej lewym przycisku, znajduje się nad potencjalnym celem operacji drag-and-drop (tylko z atrybutem DRAGID).
EVENT:Drag	Przycisk myszki został zwolniony w momencie, gdy jej wskaźnik znajdował się nad celem operacji drag-and-drop

	(tylko z atrybutem DRAGID).
EVENT:Drop	Przycisk myszki został zwolniony w momencie, gdy jej wskaźnik znajdował się nad celem operacji drag-and-drop (tylko z atrybutem DROPID).
EVENT:DroppingDown	Użytkownik rozwinął listę opadającą (tylko z atrybutem DROP). CYCLE przerywa rozwijanie.
EVENT:DroppedDown	Użytkownik rozwinął listę opadającą (tylko z atrybutem DROP).
EVENT:Expanding	Użytkownik kliknął ikonę rozwijającą gałąź drzewa (tylko ze znakiem T w łańcuchu atrybutu FORMAT). Instrukcja CYCLE anuluje tę operację.
EVENT:Expanded	Użytkownik kliknął ikonę rozwijającą gałąź drzewa (tylko ze znakiem T w łańcuchu atrybutu FORMAT).
EVENT:Contracting	Użytkownik kliknął ikonę zwijającą gałąź drzewa (tylko ze znakiem T w łańcuchu atrybutu FORMAT). Instrukcja CYCLE anuluje tę operację.
EVENT:Contracted	Użytkownik kliknął ikonę zwijającą gałąź drzewa (tylko ze znakiem T w łańcuchu atrybutu FORMAT).
EVENT:ColumnResize	Kolumna listy zmieniła swój rozmiar.

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
          LIST,AT(0,0,20,20),USE(?L1),FROM(Que),IMM
          LIST,AT(20,0,20,20),USE(?L2),FROM(Que),KEY(F10Key)
          LIST,AT(40,0,20,20),USE(?L3),FROM(Que),MSG('Button 3')
          LIST,AT(60,0,20,20),USE(?L4),FROM(Que),HLP('Check4Help')
          LIST,AT(80,0,20,20),USE(?L5),FROM(Q),FORMAT('5C~List~15L~Box~'),COLUMN
          LIST,AT(100,0,20,20),USE(?L6),FROM(Que),FONT('Arial',12)
          LIST,AT(120,0,20,20),USE(?L7),FROM(Que),DROP(6)
          LIST,AT(140,0,20,20),USE(?L8),FROM(Que),HVSCROLL,VCR
          LIST,AT(180,0,20,20),USE(?L10),FROM(Que),CURSOR(CURSOR:Wait)
          LIST,AT(200,0,20,20),USE(?L11),FROM(Que),ALRT(F10Key)
          LIST,AT(220,0,20,20),USE(?L12),FROM(Que),LEFT
          LIST,AT(240,0,20,20),USE(?L13),FROM(Que),RIGHT
          LIST,AT(260,0,20,20),USE(?L14),FROM(Que),CENTER
          LIST,AT(280,0,20,20),USE(?L15),FROM(Que),DECIMAL
          LIST,AT(300,0,20,20),USE(?L16),FROM('Apples|Peaches|Pumpkin|Pie')
          LIST,AT(320,0,20,20),USE(?L17),FROM('TBA')
        END
  CODE
    OPEN(MDIChild)
    ?L17{PROP:From} = 'Live|Long|And|Prosper'                ! przypisanie atrybutu FROM w czasie działania

QQUEUE
F1    STRING(1)
F2    STRING(4)
    END

CustRpt REPORT,AT(1000,1000,6500,9000),THOUS
CustDetail     DETAIL,AT(0,0,6500,1000)
            LIST,AT(80,0,20,20),USE(?L1),FROM(Q),FORMAT('5C~List~15L~Box~')
          END
        END
```

Porównaj: COMBO, DRAGID, DROPID, SETDROPID

MENU (deklaruje menu)

```
MENU( text) [,USE( )] [,KEY( )] [,MSG( )] [,HLP( )] [,STD( )] [,RIGHT] [,DISABLE]
     [,ICON( )] [,FONT( )] [,  | FIRST  | ]
                               | LAST   |
     END
```

MENU Deklaruje menu w ramach struktury MENUBAR.

Tekst Stała łańcuchowa zawierająca tekst menu (PROP:Text).

USE Ekwiwalent nazwy pola umożliwiający odwoływanie się do kontrolki w kodzie wykonywalnym (PROP:USE).

KEY Określa stałą całkowitą lub ekwiwalent kodu klawisza, którego wciśnięcie powoduje bezpośrednie otwarcie menu (PROP:KEY).

MSG Określa tekst wyświetlany w pasku stanu aplikacji wtedy, gdy menu jest rozwinięte (PROP:MSG).

HLP Określa łańcuch stanowiący identyfikator sekcji systemu pomocy związanej z danym menu (PROP:HLP).

STD Określa stałą całkowitą lub ekwiwalent dla „standardowej akcji Windows", która jest wykonywana przez dane menu (PROP:STD).

RIGHT Powoduje, że menu MENU pojawia się na skrajnej prawej pozycji paska menu (PROP:RIGHT).

DISABLE Powoduje, że menu po pierwszym otwarciu okna WINDOW lub APPLICATION jest niedostępne, wyszarzone (PROP:DISABLE).

ICON Wskazuje standardową ikonę lub plik z grafiką, która będzie wyświetlana w menu (PROP:ICON).

FONT Określa czcionkę dla danej kontrolki (PROP:FONT).

FIRST Powoduje, że menu, po scaleniu z innym systemem menu, występuje w lewej lub górnej pozycji (PROP:FIRST).

LAST Powoduje, że menu, po scaleniu z innym systemem menu, występuje w prawej lub dolnej pozycji (PROP:LAST).

MENU deklaruje opadające (drop-down) lub kaskadowe (cascading) menu w ramach struktury MENUBAR. Gdy MENU jest podświetlone, w okienku menu pojawiają się zadeklarowane w jego ramach podmenu (kolejne struktury MENU) i (lub) polecenia - ITEM. Okienko menu pojawia się zazwyczaj (jest rozwijane) bezpośrednio pod tekstem *text* (lub też nad nim, jeśli pod spodem nie ma wystarczająco dużo miejsca). Gdy wybierzemy je za pomocą klawisza ENTER lub po wciśnięciu klawisza strzałki w prawo, zostanie kaskadowo wyświetlone po prawej stronie (lub też po lewej, jeśli nie ma wystarczająco dużo miejsca) kolejne okienko menu zawierające elementy zdefiniowane w ramach danej struktury MENU. Wciśnięcie klawisza strzałki w lewo powoduje powrót do poprzedniego menu.

Atrybut KEY pozwala na określenie oddzielnego klawisza skrótu dla menu. Może to być dowolny, poprawny kod klawisza, którego wciśnięcie pociągnie za sobą natychmiastowe rozwinięcie MENU.

Łańcuch *text* może zawierać znak ampersand (&), który powoduje wskazanie następującego po nim znaku, jako klawisza skrótu (znak ten jest podkreślany podczas wyświetlania menu). Jeśli MENU jest zdefiniowane na najwyższym poziomie, w

pasku menu, wciśnięcie klawisza Alt w połączeniu z omawianym znakiem powoduje wyświetlenie MENU. Jeżeli MENU znajduje się wewnątrz innej struktury MENU, wciśnięcie klawisza skrótu, podświetla i rozwija odpowiadające mu menu. Jeśli w tekście *text* nie zastosowano znaku ampersand, klawiszem skrótu staje się jego pierwszy znak różny od spacji (nie jest on jednak podkreślany). W przypadku, gdy chcemy, by znak ampersand stanowił część tekstu *text*, musimy w nim umieścić dwa takie znaki występujące wspólnie (&&); wyświetlany będzie oczywiście tylko jeden.

Przykład:

```
! okno sterujące MDI z menu głównym dla aplikacji:
MainWin  APPLICATION('My Application'),SYSTEM,MAX,ICON('MyIcon.ICO'),STATUS |
           ,HVSCROLL,RESIZE
         MENUBAR
           MENU('File'),USE(?FileMenu),FIRST
             ITEM('Open...'),USE(?OpenFile)
             ITEM('Close'),USE(?CloseFile),DISABLE
             ITEM('E&xit'),USE(?MainExit)
           END
           MENU('Edit'),USE(?EditMenu),KEY(CtrlE),HLP('EditMenuHelp')
             ITEM('Undo'),USE(?UndoText),KEY(CtrlZ),STD(STD:Undo),DISABLE
             ITEM('Cu&t'),USE(?CutText),KEY(CtrlX),STD(STD:Cut),DISABLE
             ITEM('Copy'),USE(?CopyText),KEY(CtrlC),STD(STD:Copy),DISABLE
             ITEM('Paste'),USE(?PasteText),KEY(CtrlV),STD(STD:Paste),DISABLE
           END
           MENU('Window'),STD(STD:WindowList),MSG('Arrange or Select Window'),LAST
             ITEM('Tile'),STD(STD:TileWindow)
             ITEM('Cascade'),STD(STD:CascadeWindow)
             ITEM('Arrange Icons'),STD(STD:ArrangeIcons)
           END
           MENU('Help'),USE(?HelpMenu),LAST,RIGHT
             ITEM('Contents'),USE(?HelpContents),STD(STD:HelpIndex)
             ITEM('Search for Help On...'),USE(?HelpSearch),STD(STD:HelpSearch)
             ITEM('How to Use Help'),USE(?HelpOnHelp),STD(STD:HelpOnHelp)
             ITEM('About MyApp...'),USE(?HelpAbout)
           END
         END
       END
```

OLE (deklaruje kontrolkę OLE lub .OCX)

```
OLE ,AT( ) [,CURSOR( )] [,USE()] [,DISABLE] [,KEY()] [,MSG()] [,HLP()] [,SKIP] [,FULL] [,TIP()]
      [,SCROLL] [,ALRT( )] [,HIDE] [,FONT( )] [,DROPID( )] [,COMPATIBILITY( )]
      [,   | CREATE( )       | ] [,   | CLIP          | ] [, property( value )]
           | OPEN( )         |        | AUTOSIZE      |
           | LINK( )         |        | STRETCH       |
           | DOCUMENT( )     |        | ZOOM          |
   [ MENUBAR
      multiple menu and/or item declarations
   END ]
END
```

OLE Umieszcza w oknie WINDOW lub w pasku narzędzi TOOLBAR kontrolkę OLE (Object Linking and Embedding) lub kontrolkę .OCX.

AT Określa początkowy rozmiar i położenie kontrolki (PROP:AT). Jeśli jest pominięty, domyślne wartości są przyjmowane przez bibliotekę runtime.

CURSOR Określa kursor wyświetlany w momencie, gdy wskaźnik myszki zatrzyma się na kontrolce (PROP:CURSOR). Jeśli ominiemy ten atrybut, jest stosowany kursor wskazany dla okna lub też domyślny kursor systemu Windows.

USE Etykieta ekwiwalentu pola lub etykieta zmiennej, w której jest zapamiętywana „wartość" kontrolki (PROP:USE).

DISABLE Powoduje, że kontrolka po pierwszym otwarciu okna WINDOW lub APPLICATION jest niedostępna, wyszarzona (PROP:DISABLE).

KEY Określa stałą całkowitą lub ekwiwalent kodu klawisza, którego wciśnięcie powoduje bezpośrednie przekazanie aktywności kontrolce (PROP:KEY).

MSG Określa tekst wyświetlany w pasku stanu aplikacji wtedy, gdy dana kontrolka jest aktywna (PROP:MSG).

HLP Określa łańcuch stanowiący identyfikator sekcji systemu pomocy związanej z daną kontrolką (PROP:HLP).

SKIP Powoduje, że kontrolka nie otrzymuje aktywności (jest pomijana podczas poruszania się między kontrolkami za pomocą klawisza TAB). Dostęp do niej jest możliwy po jej kliknięciu myszką lub po wciśnięciu klawisza skrótu z nią związanego (PROP:SKIP).

FULL Powoduje, że rozmiar dla każdego brakującego atrybutu AT (długość i wysokość) kontrolki zwiększa się tak, by obejmował cały dostępny obszar okna WINDOW (PROP:FULL).

TIP Definiuje tekst wyświetlany „w dymkach" po zatrzymaniu kursora myszki na kontrolce (PROP:ToolTip).

SCROLL Nadanie tego parametru powoduje, że kontrolka jest przewijana wraz z oknem (PROP:SCROLL).

ALRT Określa klawisze skrótu aktywne dla kontrolki (PROP:ALRT).

HIDE Powoduje, że po pierwszym otwarciu okna WINDOW lub APPLICATION kontrolka pozostaje ukryta (PROP:HIDE). Zostanie ona wyświetlona dopiero po użyciu procedury UNHIDE.

FONT Określa czcionkę dla danej kontrolki (PROP:FONT).

DROPID Określa, że kontrolka może służyć jako cel dla operacji „przenieś i upuść" (PROP:DROPID).

COMPATIBILITY Określa tryb zgodności dla pewnych kontrolek OLE lub .OCX, które tego wymagają (PROP:COMPATIBILITY).

CREATE Określa, że kontrolka tworzy nowy obiekt OLE lub .OCX (PROP:CREATE).

OPEN Określa, że kontrolka otwiera obiekt zapisany w pliku OLE Compound Storage (PROP:AT). Gdy obiekt jest otwarty, zostają załadowane zachowane wersje właściwości kontenera, tak, że nie trzeba ich określać na nowo.

LINK Określa, że dany obiekt OLE jest powiązany z obiektem zapisanym w pliku, na przykład z arkuszem Excela (PROP:LINK).

DOCUMENT Określa, że dany obiekt OLE jest obiektem pochodzącym z pliku, na przykład arkuszem Excela (PROP:DOCUMENT).

CLIP Powoduje, że obiekt OLE wyświetla tylko to, co zmieści się w obszarze określonym przez atrybut AT kontrolki będącej kontenerem OLE (PROP:CLIP). Jeżeli obiekt jest większy, niż kontener OLE, wyświetlany jest tylko lewy, górny narożnik.

AUTOSIZE Powoduje, że obiekt OLE automatycznie zmienia swój rozmiar wtedy, gdy parametry atrybutu AT kontenera OLE są zmieniane w trakcie działania aplikacji za pomocą właściwości PROP:AT (PROP:AUTOSIZE).

STRETCH Powoduje takie dobranie rozmiaru obiektu OLE, by całkowicie wypełniał on obszar wytyczony poprzez parametry atrybutu AT kontenera OLE (PROP:STRETCH).

ZOOM Powoduje dobranie rozmiaru obiektu OLE, tak by wypełniał on obszar wytyczony poprzez parametry atrybutu AT kontenera OLE jednakże z zachowaniem jego właściwych proporcji (PROP:ZOOM).

property Stała łańcuchowa zawierająca nazwę parametru kontrolki OLE lub .OCX.

value Stała łańcuchowa zawierająca wartość lub ekwiwalent EQUATE dla parametru *property*.

MENUBAR Definiuje strukturę menu dla kontrolki. Jest to dokładnie ten sam typ struktury, co w przypadku MENUBAR w oknie APPLICATION lub WINDOW. Struktura ta jest scalana z menu aplikacji.

menus and/or items Deklaracje menu i/lub elementów menu (ITEM) definiujące system menu dla kontrolki.

Kontrolka **OLE** powoduje umieszczenie w oknie WINDOW lub pasku narzędzi TOOLBAR kontrolki OLE lub .OCX (nie dotyczy raportów) w pozycji i o rozmiarze określonym przez atrybut AT. Atrybut *property* umożliwia określenie dodatkowych właściwości związanych z parametrami konkretnej kontrolki OLE lub .OCX. Są to właściwości, których ustawienie umożliwia poprawne funkcjonowanie kontrolki OLE

lub .OCX; nie mają one nic wspólnego ze standardowymi właściwościami stosowanymi w Clarionie, takimi jak AT, CURSOR, czy USE. Typowa kontrolka będzie jedynie rejestrowała wartości określone dla jej właściwości. Informacji na temat, jakie to są właściwości i jakie są prawidłowe dla nich wartości, należy szukać w dokumentacji kontrolki. Dla pojedynczej kontrolki OLE możemy określić wiele atrybutów *property*.

Generowane zdarzenia:

EVENT:Selected	Kontrolka otrzymała aktywność wprowadzania.
EVENT:Accepted	Użytkownik zaakceptował kontrolkę; utraciła ona aktywność wprowadzania.
EVENT:PreAlertKey	Użytkownik nacisnął klawisz wskazany atrybutem ALRT.
EVENT:AlertKey	Użytkownik nacisnął klawisz wskazany atrybutem ALRT.
EVENT:Drop	Operacja drag-and-drop dla kontrolki zakończyła się pomyślnie.

Przykład:

```
PROGRAM
MAP
  INCLUDE('OCX.CLW')
END
W  WINDOW('OCX Controls'),AT(,,200,200),RESIZE,STATUS(-1,-1),SYSTEM
     MENUBAR
       ITEM ('E&xit!'),USE(?Exit)
       ITEM('&About!'),USE(?About)
       ITEM('&Properties!'),USE(?Property)
     END
     OLE,AT(0,0,0,0),USE(?oc1),HIDE,CREATE('COMCTL.ImagelistCtrl.1').
     OLE,AT(0,0,150,20),USE(?oc2),CREATE('TOOLBAR.ToolbarCtrl.1').
   END

  CODE
    OPEN(W)
    ?OC1{'ListImages.Add(1,xyz,' & ocxloadimage('IRCLOCK.BMP') & ')'}
    ?OC1{'ListImages.Add(2,abc,' & ocxloadimage('IRCLOCK2.BMP') & ')'}
    ?oc2{'ImageList'} = ?oc1{PROP:Object}
    LOOP X# = 1 TO 3
      ?oc2{'Buttons.Add(,,,,1)'}; ?oc2{'Buttons.Add(,,,,2)'}
    END
    ACCEPT
      CASE EVENT()
      OF EVENT:Accepted
        CASE FIELD()
        OF ?Exit
          BREAK
        OF ?About
          ?oc1{'AboutBox'}                    ! wyświetla About Box kontrolki OCX
        OF ?Property
          ?oc1{PROP:DoVerb} = -7              ! wyświetla okienko właściwości kontrolki OCX
...
```

Porównaj: Osadzanie i łączenie obiektów, Kontrolki OLE (.OCX), Procedury biblioteki OCX

OPTION (deklaruje zbiór kontrolek RADIO)

```
OPTION( text) ,AT( ) [,CURSOR( )] [,USE( )] [,DISABLE] [,KEY( )] [,MSG( )] [,HLP( )] [,BOXED]
        [,FULL] [,SCROLL] [,HIDE] [,FONT( )] [,ALRT( )] [,SKIP] [DROPID( )] [,TIP( )] [,TRN]
        [,COLOR( )] [,BEVEL( )]
   radios
END
```

OPTION Deklaruje zbiór kontrolek RADIO.

text Stała łańcuchowa zawierająca tekst przypisany do zbioru kontrolek (PROP:Text). Może on zawierać znak ampersand (&) w celu wskazania (podkreślenia) przypisanego klawisza skrótu. *tekst* jest wyświetlany tylko wtedy, gdy grupa opcji posiada atrybut BOXED.

AT Określa początkowy rozmiar i położenie kontrolki (PROP:AT). Jeśli jest pominięty, domyślne wartości są przyjmowane przez bibliotekę runtime.

CURSOR Określa kursor wyświetlany w momencie, gdy wskaźnik myszki zatrzyma się na kontrolce (PROP:CURSOR). Jeśli ominiemy ten atrybut, jest stosowany kursor wskazany dla okna lub też domyślny kursor systemu Windows. Nie dotyczy raportu.

USE Etykieta zmiennej, w której jest zapamiętywany wybór (PROP:USE). Jeśli jest to zmienna łańcuchowa, otrzymuje wartość łańcucha kontrolki RADIO (łącznie ze znakiem & identyfikującym klawisz skrótu) wybranej przez użytkownika. Jeśli jest to wartość numeryczna, otrzymuje wartość odpowiadającą pozycji wybranej kontrolki RADIO w grupie opcji OPTION (jest to wartość zwracana przez procedurę CHOICE()).

DISABLE Powoduje, że kontrolka po pierwszym otwarciu okna WINDOW lub APPLICATION jest niedostępna, wyszarzona (PROP:DISABLE).

KEY Określa stałą całkowitą lub ekwiwalent kodu klawisza, którego wciśnięcie powoduje bezpośrednie przekazanie aktywności do aktualnie wybranego elementu RADIO w kontrolce OPTION (PROP:KEY). Nie dotyczy raportu.

MSG Określa tekst wyświetlany w pasku stanu aplikacji wtedy, gdy dowolna kontrolka grupy OPTION jest aktywna (PROP:MSG). Nie dotyczy raportu.

HLP Określa łańcuch stanowiący domyślny identyfikator sekcji systemu pomocy związanej z dowolną kontrolką grupy OPTION (PROP:HLP). Nie dotyczy raportu.

BOXED Powoduje rysowanie, pojedynczą linią, ramki wokół grupy kontrolek RADIO; w lewym górnym rogu tej ramki jest wyświetlany tekst zdefiniowany za pomocą parametru *text* (PROP:BOXED).

FULL Powoduje, że rozmiar dla każdego brakującego atrybutu AT (długość i wysokość) kontrolki zwiększa się tak, by obejmował cały dostępny obszar okna WINDOW (PROP:FULL). Nie dotyczy raportu.

SCROLL Nadanie tego parametru powoduje, że kontrolka jest przewijana wraz z oknem (PROP:SCROLL). Nie dotyczy raportów.

HIDE Powoduje, że po pierwszym otwarciu okna WINDOW lub APPLICATION kontrolka pozostaje ukryta (PROP:HIDE). Zostanie ona wyświetlona dopiero po użyciu procedury UNHIDE.

FONT Określa czcionkę dla danej kontrolki, stanowi ona domyślną czcionkę dla wszystkich kontrolek w grupie OPTION (PROP:FONT).

ALRT Określa klawisze skrótu aktywne dla kontrolek w grupie OPTIONS (PROP:ALRT). Nie dotyczy raportu.

SKIP Powoduje, że kontrolka nie otrzymuje aktywności (jest pomijana podczas poruszania się między kontrolkami za pomocą klawisza TAB). Dostęp do niej jest możliwy po jej kliknięciu myszką lub po wciśnięciu klawisza skrótu z nią związanego (PROP:SKIP). Nie dotyczy raportu.

DROPID Określa, że kontrolka może służyć jako cel dla operacji „przenieś i upuść" (PROP:DROPID). Nie dotyczy raportu.

TIP Definiuje tekst wyświetlany „w dymkach" po zatrzymaniu kursora myszki na kontrolce (PROP:ToolTip). Nie dotyczy raportu.

COLOR Określa kolor tła dla kontrolki (PROP:COLOR).

BEVEL Pozwala na wprowadzanie efektu trójwymiarowości w ramce grupy (PROP:BEVEL). Nie dotyczy raportu.

radios Deklaracje kontrolek RADIO.

Kontrolka **OPTION** deklaruje zestaw kontrolek RADIO udostępniając użytkownikowi listę opcji do wyboru. Każdy możliwy wybór jest zdefiniowany za pomocą odpowiadającej mu kontrolki RADIO struktury OPTION. W raporcie REPORT kontrolka OPTION powoduje wydrukowanie grupy kontrolek RADIO wyświetlanych jako lista opcji do wyboru. Wybrana opcja jest identyfikowana w ten sposób, że jej przycisk RADIO jest wypełniony. Zmiana aktywności wprowadzania pomiędzy kontrolkami RADIO należącymi do grupy OPTION jest sygnalizowana tylko tym kontrolkom RADIO, których dotyczy. Oznacza to, że zdarzenia EVENT:Selected generowane w momencie przeniesienia aktywności wprowadzania wewnątrz grupy OPTION są zależne od pola dla kontrolek RADIO, których dotyczą, a nie dla struktury OPTION zawierającej te kontrolki. Dla struktury OPTION nie jest generowane zdarzenie EVENT:Selected. Kontrolka RADIO nie rejestruje zdarzenia EVENT:Accepted, rejestruje je natomiast struktura OPTION w momencie, gdy użytkownik wybierze którąś z opcji reprezentowanych prze z kontrolki RADIO.

Zmienna łańcuchowa wskazana w atrybucie USE struktury OPTION rejestruje tekst wybranej przez użytkownika kontrolki RADIO. Funkcja CHOICE(*?Option*) daje w rezultacie numer wybranego przycisku RADIO. Jeśli zmienna wskazana przez atrybut USE struktury OPTION jest zmienną numeryczną, jest w niej zapisywany numer przycisku RADIO wybranego przez użytkownika (jest to właśnie wartość zwracana przez funkcję CHOICE).

Brak wyboru jakiegokolwiek przycisku RADIO jest również prawidłowy. Zdarza się to tylko wtedy, gdy zmienna wskazana przez atrybut USE struktury OPTION nie zawiera wartości powiązanej z którąkolwiek kontrolką RADIO. Takie zdarzenie może zachodzić tylko do momentu, gdy użytkownik nie wybrał jeszcze żadnego przycisku RADIO.

Generowane zdarzenia:

EVENT:Accepted	Jedna z opcji OPTION kontrolki RADIO została wybrana przez użytkownika.
EVENT:PreAlertKey	Użytkownik nacisnął klawisz wskazany atrybutem ALRT.
EVENT:AlertKey	Użytkownik nacisnął klawisz wskazany atrybutem ALRT.
EVENT:Drop	Operacja drag-and-drop dla kontrolki zakończyła się pomyślnie.

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            OPTION('Option 1'),USE(OptVar1),KEY(F10Key),HLP('Option1Help')
              RADIO('Radio 1'),AT(0,0,20,20),USE(?R1)
              RADIO('Radio 2'),AT(20,0,20,20),USE(?R2)
            END
            OPTION('Option 2'),USE(OptVar2),MSG('Option 2'),SCROLL
              RADIO('Radio 3'),AT(40,0,20,20),USE(?R3)
              RADIO('Radio 4'),AT(60,0,20,20),USE(?R4)
            END
            OPTION('Option 3'),USE(OptVar3),AT(80,0,20,20),BOXED
              RADIO('Radio 5'),AT(80,0,20,20),USE(?R5)
              RADIO('Radio 6'),AT(100,0,20,20),USE(?R6)
            END
            OPTION('Option 4'),USE(OptVar4),FONT('Arial',12),CURSOR(CURSOR:Wait)
              RADIO('Radio 7'),AT(120,0,20,20),USE(?R7)
              RADIO('Radio 8'),AT(140,0,20,20),USE(?R8)
            END
          END
CustRpt REPORT,AT(1000,1000,6500,9000),THOUS
CustDetail      DETAIL,AT(0,0,6500,1000)
              OPTION('Option'),USE(OptVar),AT(80,0,20,20),BOXED
                RADIO('Radio 1'),AT(80,0,20,20),USE(?R1)
                RADIO('Radio 2'),AT(100,0,20,20),USE(?R2)
              END
            END
          END
```

Porównaj: RADIO, BUTTON, CHECK

PANEL (deklaruje panel)

PANEL ,**AT**( ) [,**USE**( )] [,**DISABLE**] [,**FULL**] [,**FILL**( )] [,**SCROLL**] [,**HIDE**] [,**BEVEL**( )]

PANEL Definiuje obszar w oknie WINDOW lub pasku narzędzi TOOLBAR.

AT Określa początkowy rozmiar i położenie kontrolki (PROP:AT). Jeśli jest pominięty, domyślne wartości są przyjmowane przez bibliotekę runtime.

USE Etykieta ekwiwalentu pola umożliwiająca odwoływanie się do kontrolki w kodzie wykonywalnym (PROP:USE).

DISABLE Powoduje, że kontrolka po pierwszym otwarciu okna WINDOW lub APPLICATION jest niedostępna, wyszarzona (PROP:DISABLE).

FULL Powoduje, że rozmiar dla każdego brakującego atrybutu AT (długość i wysokość) kontrolki zwiększa się tak, by obejmował cały dostępny obszar okna WINDOW (PROP:FULL).

FILL Określa kolor wypełnienia dla kontrolki (PROP:FILL). Jeśli atrybut ten zostanie pominięty, kontrolka nie będzie wypełniania żadnym kolorem.

SCROLL Nadanie tego parametru powoduje, że kontrolka jest przewijana wraz z oknem (PROP:SCROLL).

HIDE Powoduje, że po pierwszym otwarciu okna WINDOW lub APPLICATION kontrolka pozostaje ukryta (PROP:HIDE). Zostanie ona wyświetlona dopiero po użyciu procedury UNHIDE.

BEVEL Nadaje ramce panelu efekt trójwymiarowości (PROP:BEVEL).

Kontrolka **PANEL** definiuje obszar w oknie WINDOW lub w pasku narzędzi TOOLBAR (nie dotyczy raportu) w pozycji i o rozmiarze określonym przez atrybut AT. Typowym zastosowaniem tej kontrolki jest narysowanie obszaru, dla którego można dodatkowo zdefiniować trójwymiarowe cieniowanie (BEVEL).

Ta kontrolka nie otrzymuje aktywności wprowadzania i nie generuje zdarzeń.

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            PANEL,AT(10,100,20,20),USE(?P1),BEVEL(-2,2)
          END
```

Porównaj: BOX, GROUP

PROMPT (deklaruje opis pola)

```
PROMPT( text) ,AT( ) [,CURSOR( )] [,USE( )] [,DISABLE] [,FONT( )] [,FULL] [,SCROLL] [,TRN]
    [,HIDE] [,DROPID( )]  [,  | LEFT   | ] [,COLOR( )]
                              | RIGHT  |
                              | CENTER |
```

PROMPT Umieszcza opis pola (następnej aktywnej kontrolki) w oknie WINDOW lub pasku narzędzi TOOLBAR.

tekst Stała łańcuchowa zawierająca tekst przeznaczony do wyświetlenia (PROP:Text). Może on zawierać znak ampersand (&) w celu wskazania (podkreślenia) przypisanego klawisza skrótu.

AT Określa początkowy rozmiar i położenie kontrolki (PROP:AT). Jeśli jest pominięty, domyślne wartości są przyjmowane przez bibliotekę runtime.

CURSOR Określa kursor wyświetlany w momencie, gdy wskaźnik myszki zatrzyma się na kontrolce (PROP:CURSOR). Jeśli ominiemy ten atrybut, jest stosowany kursor wskazany dla okna lub też domyślny kursor systemu Windows.

USE Etykieta ekwiwalentu pola umożliwiająca odwoływanie się do kontrolki w kodzie wykonywalnym (PROP:USE).

DISABLE Powoduje, że kontrolka po pierwszym otwarciu okna WINDOW lub APPLICATION jest niedostępna, wyszarzona (PROP:DISABLE).

FONT Określa czcionkę dla danej kontrolki (PROP:FONT).

FULL Powoduje, że rozmiar dla każdego brakującego atrybutu AT (długość i wysokość) kontrolki zwiększa się tak, by obejmował cały dostępny obszar okna WINDOW (PROP:FULL).

SCROLL Nadanie tego parametru powoduje, że kontrolka jest przewijana wraz z oknem (PROP:SCROLL).

TRN Powoduje, że kontrolka jest wyświetlana jako przeźroczysta, nie przykrywa innych kontrolek ani tła (PROP:TRN).

HIDE Powoduje, że po pierwszym otwarciu okna WINDOW lub APPLICATION kontrolka pozostaje ukryta (PROP:HIDE). Zostanie ona wyświetlona dopiero po użyciu procedury UNHIDE.

DROPID Określa, że kontrolka może służyć jako cel dla operacji „przenieś i upuść” (PROP:DROPID).

LEFT Powoduje, że zawartość kontrolki jest wyrównywana do lewej (PROP:LEFT).

RIGHT Powoduje, że zawartość kontrolki jest wyrównywana do prawej (PROP:RIGHT).

CENTER Powoduje, że zawartość kontrolki jest centrowana (PROP:CENTER).

COLOR Określa kolor tła dla kontrolki (PROP:COLOR).

Kontrolka **PROMPT** umieszcza w oknie WINDOW lub w pasku narzędzi TOOLBAR (nie dotyczy raportu) podpis dla następnej, występującej po PROMPT, kontrolki. Tekst *text* jest umieszczany w pozycji i w rozmiarze określonym przez atrybut AT.

Tekst *text* może zawierać znak ampersand (&), który identyfikuje znak występujący bezpośrednio za nim, jak klawisz skrótu. Znak ten jest w tekście *text* podkreślany. Wciśnięcie klawisza skrótu w połączeniu z klawiszem ALT powoduje przekazanie aktywności wprowadzania do kontrolki występującej po kontrolce PROMPT, pod warunkiem oczywiście, że może ona przyjmować aktywność wprowadzania.

Wyłączanie lub ukrywanie kontrolki występującej w powiązaniu z kontrolką PROMPT nie pociąga za sobą automatycznego wyłączenia lub ukrycia kontrolki PROMPT. Musimy bezpośrednio, za pomocą właściwych funkcji, wyłączyć lub ukryć tę kontrolkę, w przeciwnym razie będzie się ona odnosiła do następnej kontrolki, która nie została wyłączona lub ukryta. Pozwala to na umieszczanie w oknie jednej kontrolki PROMPT, która może być łączona z wieloma kontrolkami, pod warunkiem, że w danym czasie tylko jedna z nich będzie włączona lub widoczna. Jeśli następną aktywną kontrolką jest przycisk BUTTON, wciśnięcie klawisza skrótu kontrolki PROMPT jest równoznaczne z wciśnięciem przycisku.

Jeśli w tekście *text* chcemy wyświetlić znak ampersand, musimy w nim umieścić dwa znaki ampersands występujące obok siebie (&&).

Ta kontrolka nie może otrzymywać aktywności wprowadzania.

Generowane zdarzenia:

EVENT:Drop	Operacja drag-and-drop dla kontrolki zakończyła się pomyślnie.

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
          PROMPT('Enter Data:'),AT(10,100,20,20),USE(?P1),CURSOR(CURSOR:Wait)
          ENTRY(@S8),AT(100,100,20,20),USE(E1)
          PROMPT('Enter More Data:'),AT(10,200,20,20),USE(?P2),CURSOR(CURSOR:Wait)
          ENTRY(@S8),AT(100,200,20,20),USE(E2)
          ENTRY(@D1),AT(100,200,20,20),USE(E3)
        END
  CODE
    OPEN(MDIChild)
    IF SomeCondition
      HIDE(?E2)                          ! Prompt powinien dotyczyć E3
    ELSE
      HIDE(?E3)                          ! Prompt powinien dotyczyć E2
    END
```

Porównaj: ENTRY, TEXT

PROGRESS (deklaruje pasek postępu)

PROGRESS, **AT**( ) [,**CURSOR**( )] [,**USE**( )] [,**DISABLE**] [,**FULL**] [,**SCROLL**] [,**HIDE**] [,**DROPID**( )] [,**RANGE**( )]

PROGRESS Umieszcza w oknie WINDOW lub pasku narzędzi TOOLBAR kontrolkę ilustrującą postęp wykonania operacji przetwarzania wsadowego.

AT Określa początkowy rozmiar i położenie kontrolki (PROP:AT). Jeśli jest pominięty, domyślne wartości są przyjmowane przez bibliotekę runtime.

CURSOR Określa kursor wyświetlany w momencie, gdy wskaźnik myszki zatrzyma się na kontrolce (PROP:CURSOR). Jeśli ominiemy ten atrybut, jest stosowany kursor wskazany dla okna lub też domyślny kursor systemu Windows.

USE Etykieta zmiennej zawierającej aktualną wartość procentową postępu lub etykieta ekwiwalentu pola umożliwiająca odwoływanie się do kontrolki w kodzie wykonywalnym (PROP:USE).

DISABLE Powoduje, że kontrolka po pierwszym otwarciu okna WINDOW lub APPLICATION jest niedostępna, wyszarzona (PROP:DISABLE).

FULL Powoduje, że rozmiar dla każdego brakującego atrybutu AT (długość i wysokość) kontrolki zwiększa się tak, by obejmował cały dostępny obszar okna WINDOW (PROP:FULL).

SCROLL Nadanie tego parametru powoduje, że kontrolka jest przewijana wraz z oknem (PROP:SCROLL).

HIDE Powoduje, że po pierwszym otwarciu okna WINDOW lub APPLICATION kontrolka pozostaje ukryta (PROP:HIDE). Zostanie ona wyświetlona dopiero po użyciu procedury UNHIDE.

DROPID Określa, że kontrolka może służyć jako cel dla operacji „przenieś i upuść” (PROP:DROPID).

RANGE Określa zakres wartości opisujących postęp operacji i mających wpływ na wyświetlanie paska postępu (PROP:RANGE). Jeśli atrybut ten zostanie pominięty domyślnym zakresem jest (0 – 100).

Kontrolka **PROGRESS** deklaruje kontrokę okna WINDOW lub paska narzędzi TOOLBAR (nie dotyczy raportu), w której jest wyświetlany postęp wykonania jakiejś operacji. Jest to zazwyczaj procent wykonania procesu przetwarzania wsadowego.

Jeśli w atrybucie USE została wskazana zmienna, pasek postępu jest automatycznie aktualizowany w momencie, gdy zmieni się jej wartość. Jeśli w atrybucie USE wskazano etykietę ekwiwalentu pola, musimy bezpośrednio aktualizować wyświetlanie paska postępu poprzez przypisywanie wartości (w ramach zakresu zdefiniowanego za pomocą atrybutu RANGE) właściwości PROP:progress kontrolki (patrz *Undeclared Properties*).

Ta kontrolka nie otrzymuje aktywności wprowadzania.

Generowane zdarzenia:

EVENT:Drop	Operacja drag-and-drop dla kontrolki zakończyła się pomyślnie.

Przykład:

```
BackgroundProcess  PROCEDURE                    ! proces wsadowego przetwarzania w tle

ProgressVariable     LONG

Win     WINDOW('Batch Processing...'),AT(,,400,400),TIMER(1),MDI,CENTER
          PROGRESS,AT(100,100,200,20),USE(ProgressVariable),RANGE(0,200)
          PROGRESS,AT(100,140,200,20),USE(?ProgressBar),RANGE(0,200)
          BUTTON('Cancel'),AT(190,300,20,20),STD(STD:Close)
        END
  CODE
    OPEN(Win)
    OPEN(File)
    ?ProgressVariable{PROP:rangehigh} = RECORDS(File)
    ?ProgressBar{PROP:rangehigh} = RECORDS(File)
    SET(File)                                   ! konfiguracja procesu
    ACCEPT
      CASE EVENT()
      OF EVENT:CloseWindow
        BREAK
      OF EVENT:Timer                            ! przetwarzanie rekordów, gdy pozwala na to zegar
        ProgressVariable += 3                   ! automatyczna aktualizacja pierwszego paska progresu
        LOOP 3 TIMES
          NEXT(File)
          IF ERRORCODE() THEN BREAK.
            ?ProgressBar{PROP:progress} = ?ProgressBar{PROP:progress} + 1
            ! ręczna aktualizacja drugiego paska progresu
            ! wykonaj kod przetwarzania wsadowego
  . . .
    CLOSE(File)
```

RADIO (deklaruje przycisk radio)

```
RADIO( text) ,AT( ) [,CURSOR( )] [,USE( )] [,DISABLE] [,KEY( )] [,MSG( )] [,HLP( )] [,SKIP]
    [,FONT( )] [,ICON( )] [,FULL] [,SCROLL] [,HIDE] [,ALRT( )] [DROPID( )] [VALUE( )]
    [,TIP( )] [,TRN] [,COLOR( )] [,FLAT] [,   | LEFT  | ]
                                              | RIGHT |
```

RADIO Umieszcza przycisk radio w oknie WINDOW lub w pasku narzędzi TOOLBAR.

tekst Stała łańcuchowa zawierająca tekst przycisku RADIO (PROP:Text). Może on zawierać znak ampersand (&) w celu wskazania (podkreślenia) klawisza skrótu przypisanego do danego przycisku.

AT Określa początkowy rozmiar i położenie kontrolki (PROP:AT). Jeśli jest pominięty, domyślne wartości są przyjmowane przez bibliotekę runtime.

CURSOR Określa kursor wyświetlany w momencie, gdy wskaźnik myszki zatrzyma się na kontrolce (PROP:CURSOR). Jeśli ominiemy ten atrybut, jest stosowany kursor wskazany dla okna lub też domyślny kursor systemu Windows. Nie dotyczy raportu.

USE Etykieta ekwiwalentu pola umożliwiająca odwoływanie się do kontrolki w kodzie wykonywalnym (PROP:USE).

DISABLE Powoduje, że kontrolka po pierwszym otwarciu okna WINDOW lub APPLICATION jest niedostępna, wyszarzona (PROP:DISABLE).

KEY Określa stałą całkowitą lub ekwiwalent kodu klawisza, którego wciśnięcie powoduje bezpośrednie wybranie danego elementu RADIO (PROP:KEY).

MSG Określa tekst wyświetlany w pasku stanu aplikacji wtedy, gdy dana kontrolka jest aktywna (PROP:MSG). Nie dotyczy raportu.

HLP Określa łańcuch stanowiący identyfikator sekcji systemu pomocy związanej z daną kontrolką (PROP:HLP). Nie dotyczy raportu.

SKIP Powoduje, że kontrolka nie otrzymuje aktywności (jest pomijana podczas poruszania się między kontrolkami za pomocą klawisza TAB). Dostęp do niej jest możliwy po jej kliknięciu myszką lub po wciśnięciu klawisza skrótu z nią związanego (PROP:SKIP). Nie dotyczy raportu.

FONT Określa czcionkę dla danej kontrolki (PROP:FONT).

ICON Wskazuje plik grafiki lub standardową ikonę przeznaczoną do wyświetlania na przycisku radio (PROP:ICON). Nie dotyczy raportu.

FULL Powoduje, że rozmiar dla każdego brakującego atrybutu AT (długość i wysokość) kontrolki zwiększa się tak, by obejmował cały dostępny obszar okna WINDOW (PROP:FULL). Nie dotyczy raportu.

SCROLL Nadanie tego parametru powoduje, że kontrolka jest przewijana wraz z oknem (PROP:SCROLL). Nie dotyczy raportów.

HIDE Powoduje, że po pierwszym otwarciu okna WINDOW lub APPLICATION kontrolka pozostaje ukryta (PROP:HIDE). Zostanie ona wyświetlona dopiero po użyciu procedury UNHIDE.

ALRT Określa klawisze skrótu aktywne dla kontrolki (PROP:ALRT). Nie dotyczy raportu.

DROPID Określa, że kontrolka może służyć jako cel dla operacji „przenieś i upuść" (PROP:DROPID). Nie dotyczy raportu.

VALUE Określa wartość, która jest przypisywana zmiennej wskazywanej przez atrybut USE grupy OPTION w momencie, gdy zostanie wciśnięty dany przycisk RADIO (PROP:VALUE).

TIP Definiuje tekst wyświetlany „w dymkach" po zatrzymaniu kursora myszki na kontrolce (PROP:ToolTip). Nie dotyczy raportu.

TRN Powoduje, że kontrolka jest wyświetlana jako przeźroczysta, nie przykrywa innych kontrolek ani tła (PROP:TRN).

COLOR Określa kolor tła dla kontrolki (PROP:COLOR).

FLAT Powoduje, że dany przycisk jest wyświetlany jako przycisk płaski, którego ramki pojawiają się dopiero wtedy, gdy znajdzie się nad nim kursor myszki (PROP:FLAT). Wymaga atrybutu ICON, nie dotyczy raportu.

LEFT Powoduje, że tekst pojawia się po lewej stronie przycisku radio (PROP:LEFT).

RIGHT Powoduje, że tekst pojawia się po prawej stronie przycisku radio (PROP:RIGHT). Jest to pozycja domyślna.

Kontrolka **RADIO** umieszcza w oknie WINDOW, pasku narzędzi TOOLBAR lub w raporcie REPORT przycisk radio w pozycji i o rozmiarze określonym przez atrybut AT. Kontrolka RADIO może być umieszczana tylko w ramach kontrolki OPTION. Gdy zostaje ona wybrana przez użytkownika, tekst *text* (pozbawiony znaku ampersand) jest przypisywany zmiennej wskazanej przez atrybut USE kontrolki OPTION; pod warunkiem, że nie został zastosowany atrybut VALUE.

W raporcie kontrolka RADIO wybrana przez użytkownika (wartość zmiennej wskazywanej przez atrybut USE kontrolki OPTION) jest wyświetlana jako wypełniony przycisk RADIO.

Kontrolka RADIO, dla której określono atrybut ICON jest wyświetlana jako przycisk, który może pozostawać wciśnięty (kontrolka wybrana) lub nie wciśnięty (kontrolka nie wybrana).

Dla kontrolki RADIO jest generowane zdarzenie EVENT:Selected, natomiast struktura OPTION zawierająca tę kontrolkę rejestruje zdarzenie EVENT:Accepted.

Generowane zdarzenia:

EVENT:Selected	Kontrolka otrzymała aktywność wprowadzania.
EVENT:PreAlertKey	Użytkownik nacisnął klawisz wskazany atrybutem ALRT.
EVENT:AlertKey	Użytkownik nacisnął klawisz wskazany atrybutem ALRT.
EVENT:Drop	Operacja drag-and-drop dla kontrolki zakończyła się pomyślnie.

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            OPTION('Option 1'),USE(OptVar1)
              RADIO('Radio 1'),AT(0,0,20,20),USE(?R1),KEY(F10Key)
              RADIO('Radio 2'),AT(20,0,20,20),USE(?R2),MSG('Radio 2')
            END
            OPTION('Option 2'),USE(OptVar2)
              RADIO('Radio 3'),AT(40,0,20,20),USE(?R3),FONT('Arial',12)
              RADIO('Radio 4'),AT(60,0,20,20),USE(?R4),CURSOR(CURSOR:Wait)
            END
            OPTION('Option 3'),USE(OptVar3)
              RADIO('Radio 5'),AT(80,0,20,20),USE(?R5),HLP('Radio5Help')
              RADIO('Radio 6'),AT(100,0,20,20),USE(?R6)
            END
            OPTION('Option 4'),USE(OptVar4)
              RADIO('Radio 7'),AT(120,0,20,20),USE(?R7),ICON('Radio1.ICO')
              RADIO('Radio 8'),AT(140,0,20,20),USE(?R8),ICON('Radio2.ICO')
            END
            OPTION('Option 5'),USE(OptVar5)
              RADIO('Radio 9'),AT(100,20,20,20),USE(?R9),LEFT
              RADIO('Radio 10'),AT(120,20,20,20),USE(?R10),LEFT
            END
            OPTION('Option 6'),USE(OptVar6),SCROLL
              RADIO('Radio 11'),AT(200,0,20,20),USE(?R11),SCROLL
              RADIO('Radio 12'),AT(220,0,20,20),USE(?R12),SCROLL
            END
          END
CustRpt REPORT,AT(1000,1000,6500,9000),THOUS
CustDetail  DETAIL,AT(0,0,6500,1000)
              OPTION('Option'),USE(OptVar),AT(80,0,20,20),BOXED
                RADIO('Radio 1'),AT(80,0,20,20),USE(?R1)
                RADIO('Radio 2'),AT(100,0,20,20),USE(?R2)
                RADIO('Radio 3'),AT(100,0,20,20),USE(?R2),LEFT
              END
            END
          END
```

Porównaj: OPTION, CHECK, BUTTON

REGION (deklaruje region)

REGION ,**AT**( ) [,**CURSOR**( )] [,**USE**( )] [,**DISABLE**] [,**FILL**] [,**COLOR**( )] [,**IMM**] [,**FULL**] [,**TRN**] [,**SCROLL**] [,**HIDE**] [,**DRAGID**( )] [,**DROPID**( )] [,**BEVEL**( )]

REGION Definiuje obszar w oknie WINDOW lub pasku narzędzi TOOLBAR.

AT Określa początkowy rozmiar i położenie kontrolki (PROP:AT). Jeśli jest pominięty, domyślne wartości są przyjmowane przez bibliotekę runtime.

CURSOR Określa kursor wyświetlany w momencie, gdy wskaźnik myszki zatrzyma się na kontrolce (PROP:CURSOR). Jeśli ominiemy ten atrybut, jest stosowany kursor wskazany dla okna lub też domyślny kursor systemu Windows.

USE Ekwiwalent nazwy pola umożliwiający odwoływanie się do kontrolki w kodzie wykonywalnym (PROP:USE).

DISABLE Powoduje, że kontrolka po pierwszym otwarciu okna WINDOW lub APPLICATION jest niedostępna, wyszarzona (PROP:DISABLE).

FILL Określa wartości składników kolorów czerwonego, zielonego i niebieskiego tworzące kolor wypełnienia dla kontrolki (PROP:FILL). Jeśli atrybut ten zostanie pominięty, region nie będzie wypełniany żadnym kolorem.

COLOR Określa kolor obramowania dla kontrolki (PROP:COLOR). Jeśli atrybut ten zostanie pominięty, kontrolka nie będzie posiadała obramowania.

IMM Wymusza generowanie zdarzenia za każdym razem, gdy myszka porusza się w obrębie regionu (PROP:IMM).

FULL Powoduje, że rozmiar dla każdego brakującego atrybutu AT (długość i wysokość) kontrolki zwiększa się tak, by obejmował cały dostępny obszar okna WINDOW (PROP:FULL).

SCROLL Nadanie tego parametru powoduje, że kontrolka jest przewijana wraz z oknem (PROP:SCROLL).

HIDE Powoduje, że po pierwszym otwarciu okna WINDOW lub APPLICATION kontrolka pozostaje ukryta (PROP:HIDE). Zostanie ona wyświetlona dopiero po użyciu procedury UNHIDE.

DRAGID Określa, że kontrolka może służyć jako źródło dla operacji „przenieś i upuść” (PROP:DRAGID).

DROPID Określa, że kontrolka może służyć jako cel dla operacji „przenieś i upuść” (PROP:DROPID).

BEVEL Nadaje efekt trójwymiarowości ramce kontrolki (PROP:BEVEL).

Kontrolka **REGION** definiuje obszar w oknie WINDOW lub pasku narzędzi TOOLBAR (nie dotyczy raportu) w pozycji i o rozmiarze określonym przez atrybut AT.

Region stosujemy na ogół wtedy, gdy chcemy mieć możliwość śledzenia i odczytywania położenia wskaźnika myszki. Funkcje MOUSEX i MOUSEY stosuje się wówczas do odczytania dokładnego położenia tego wskaźnika w momencie, gdy zachodzi zdarzenie związane z myszką. Zastosowanie atrybutu IMM pociąga za sobą wygenerowanie dodatkowego kodu i obniżenie szybkości działania, z tego względu powinno być ograniczane tylko do tych sytuacji, gdy jest konieczne.

Ta kontrolka nie otrzymuje aktywności wprowadzania.

REGION z atrybutem DRAGID może służyć jako źródło operacji drag-and-drop. Dostarcza on wówczas informacje, które mogą być przenoszone bądź kopiowane do innych kontrolek. REGION z atrybutem DROPID może służyć jako cel operacji drag-and-drop. Może on wtedy otrzymywać informacje z innych kontrolek. Oba atrybuty służą do zdefiniowania sygnatur drag-and-drop definiujących prawidłowe źródło i cel. Procedury DRAGID() oraz DROPID(), w powiązaniu z procedurą SETDROPID, są stosowane do przeprowadzenia operacji wymiany danych. Ponieważ REGION można definiować jako „nałożony" na inne kontrolki, możemy zdefiniować kod drag-and-drop realizujący wymianę pomiędzy dwoma dowolnymi kontrolkami. Wystarczy po prostu zdefiniować kontrolki REGION do obsługi wymaganych funkcji drag-and drop.

Generowane zdarzenia:

EVENT:Accepted	Użytkownik kliknął myszką w obrębie regionu.
EVENT:MouseIn	Myszka weszła w obręb regionu (tylko z atrybutem IMM).
EVENT:MouseOut	Myszka opuściła region (tylko z atrybutem IMM).
EVENT:MouseMove	Myszka jest przesuwana w obrębie regionu (tylko z atrybutem IMM).
EVENT:Dragging	Wskaźnik myszki, przy wciśniętym jej lewym przycisku, znajduje się nad potencjalnym celem operacji drag-and-drop (tylko z atrybutem DRAGID).
EVENT:Drag	Przycisk myszki został zwolniony w momencie, gdy jej wskaźnik znajdował się nad celem operacji drag-and-drop (tylko z atrybutem DRAGID).
EVENT:Drop	Przycisk myszki został zwolniony w momencie, gdy jej wskaźnik znajdował się nad celem operacji drag-and-drop (tylko z atrybutem DROPID).

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            REGION,AT(10,100,20,20),USE(?R1),BEVEL(-2,2)
            REGION,AT(100,100,20,20),USE(?R2),CURSOR(CURSOR:Wait)
            REGION,AT(10,200,20,20),USE(?R3),IMM
            REGION,AT(100,200,20,20),USE(?R4),COLOR(COLOR:ACTIVEBORDER)
            REGION,AT(10,300,20,20),USE(?R4),FILL(COLOR:ACTIVEBORDER)
          END
```

Porównaj: PANEL

SHEET (deklaruje grupę zakładek)

```
SHEET ,AT( ) [,CURSOR( )][,USE( )][,DISABLE][,KEY( )][,FULL][,SCROLL][,HIDE][,FONT()]
      [,DROPID()][,WIZARD][,SPREAD][,HSCROLL][,JOIN][,NOSHEET][,COLOR()]
      [,UP ] [,DOWN ] [,   | LEFT       | ] [,IMM ]
                           | RIGHT      |
                           | ABOVE      |
                           | BELOW      |
  tabs
END
```

SHEET Deklaruje grupę zakładek TAB.

AT Określa początkowy rozmiar i położenie kontrolki (PROP:AT). Jeśli jest pominięty, domyślne wartości są przyjmowane przez bibliotekę runtime.

CURSOR Określa kursor wyświetlany w momencie, gdy wskaźnik myszki zatrzyma się na kontrolce (PROP:CURSOR). Jeśli ominiemy ten atrybut, jest stosowany kursor wskazany dla okna lub też domyślny kursor systemu Windows.

USE Etykieta zmiennej, w której jest zapamiętywana wybrana zakładka (PROP:USE). Jeśli jest to zmienna łańcuchowa, jest w niej umieszczany łańcuch (tytuł) zakładki TAB (łącznie ze znakiem & identyfikującym klawisz skrótu). Jeśli jest to wartość numeryczna, jest w niej zapamiętywany numer zakładki TAB wybranej przez użytkownika (wartość zwracana przez procedurę CHOICE()).

DISABLE Powoduje, że kontrolka po pierwszym otwarciu okna WINDOW lub APPLICATION jest niedostępna, wyszarzona (PROP:DISABLE).

KEY Określa stałą całkowitą lub ekwiwalent kodu klawisza, którego wciśnięcie powoduje bezpośrednie przekazanie aktywności aktualnie wybranej zakładce TAB kontrolki SHEET (PROP:KEY).

FULL Powoduje, że rozmiar dla każdego brakującego atrybutu AT (długość i wysokość) kontrolki zwiększa się tak, by obejmował cały dostępny obszar okna WINDOW (PROP:FULL).

SCROLL Nadanie tego parametru powoduje, że kontrolka jest przewijana wraz z oknem (PROP:SCROLL).

HIDE Powoduje, że po pierwszym otwarciu okna WINDOW lub APPLICATION kontrolka pozostaje ukryta (PROP:HIDE). Zostanie ona wyświetlona dopiero po użyciu procedury UNHIDE.

FONT Określa czcionkę dla danej kontrolki, stanowi ona domyślną czcionkę dla wszystkich kontrolek arkusza SHEET (PROP:FONT).

DROPID Określa, że kontrolka może służyć jako cel dla operacji „przenieś i upuść” (PROP:DROPID).

WIZARD Powoduje, że tytuły zakładek nie pojawiają się w arkuszu (PROP:WIZARD); jest to przydatne przy tworzeniu kreatorów. Przejścia między zakładkami muszą być oprogramowane, na przykład za pomocą przycisków.

SPREAD Powoduje, że tytuły zakładek mają tak dobierane rozmiary, by tworzyły jedną linię wzdłuż całej krawędzi arkusza (PROP:SPREAD).

HSCROLL Powoduje, że tytuły zakładek są wyświetlane w jednym wierszu, niezależnie od tego, czy się w nim mieszczą, czy nie (PROP:HSCROLL). Po lewej i prawej stronie pojawiają się wówczas strzałki umożliwiające przewijanie zakładek.

JOIN Powoduje, że tytuły zakładek są wyświetlane w jednym wierszu, niezależnie od tego, czy się w nim mieszczą, czy nie (PROP:JOIN). Po prawej stronie (lub w dolnej części) arkusza pojawiają się wówczas strzałki umożliwiające przewijanie zakładek.

NOSHEET Powoduje, że ramka arkusza staje się niewidoczna, wyświetlane są tylko tytuły zakładek (PROP:NOSHEET).

COLOR Określa kolor tła dla kontrolki (PROP:COLOR).

UP Powoduje, że teksty tytułów zakładek są wyświetlane w pionie z dołu do góry (PROP:UP).

DOWN Powoduje, że teksty tytułów zakładek są wyświetlane w pionie z góry na dół (PROP:DOWN).

LEFT Powoduje, że teksty tytułów zakładek są wyświetlane z lewej strony arkusza (PROP:LEFT).

RIGHT Powoduje, że teksty tytułów zakładek są wyświetlane z prawej strony arkusza (PROP:RIGHT).

ABOVE Powoduje, że teksty tytułów zakładek są wyświetlane u góry arkusza (PROP:ABOVE). Jest to położenie domyślne.

BELOW Powoduje, że teksty tytułów zakładek są wyświetlane u dołu arkusza (PROP:BELOW).

IMM Wymusza generowanie zdarzenia EVENT:NewSelection za każdym razem, gdy użytkownik kliknie zakładkę TAB (PROP:IMM).

tabs Deklaracje zakładek arkusza.

Kontrolka **SHEET** deklaruje grupę kontrolek TAB pozwalając na rozmieszczenie innych kontrolek okna na przełączalnych stronach. Każda kontrolka TAB arkusza SHEET definiuje oddzielną „stronę”.

Zmiana aktywności wprowadzania pomiędzy zakładkami TAB arkusza SHEET jest sygnalizowana tylko do kontrolki SHEET. Oznacza to, że zdarzenia generowane w momencie, gdy użytkownik przekazuje aktywność wprowadzania w ramach struktury SHEET są zależne od pola i dotyczą struktury SHEET, a nie indywidualnych kontrolek TAB. Zmienna łańcuchowa wskazana przez atrybut USE struktury SHEET rejestruje tekst kontrolki TAB wybranej przez użytkownika. Funkcja CHOICE(*?Sheet*) daje w rezultacie numer wybranej zakładki. Jeśli zmienna wskazywana przez atrybut USE struktury SHEET jest zmienną numeryczną, rejestrują ona numer wybranej zakładki TAB (ten sam, który jest rezultatem funkcji CHOICE).

Do wymuszenia uaktywnienia wybranej zakładki można stosować instrukcję SELECT(?Sheet,TabNumber), gdzie TabNumber jest numerem zakładki, którą chcemy uaktywnić.

Generowane zdarzenia:

EVENT:TabChanging	Aktywność wprowadzania została przekazana do innej zakładki.
EVENT:NewSelection	Aktywność wprowadzania została przekazana do innej zakładki lub użytkownik kliknął zakładkę posiadającą atrybut IMM.
EVENT:Drop	Operacja drag-and-drop dla kontrolki zakończyła się pomyślnie.

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            SHEET,AT(0,0,320,175),USE(SelectedTab)
              TAB('Tab One'),USE(?TabOne)
                OPTION('Option 1'),USE(OptVar1),KEY(F10Key),HLP('Option1Help')
                  RADIO('Radio 1'),AT(20,0,20,20),USE(?R1)
                  RADIO('Radio 2'),AT(40,0,20,20),USE(?R2)
                END
                OPTION('Option 2'),USE(OptVar2),MSG('Option 2')
                  RADIO('Radio 3'),AT(60,0,20,20),USE(?R3)
                  RADIO('Radio 4'),AT(80,0,20,20),USE(?R4)
                END
                PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P1)
                ENTRY(@S8),AT(100,140,32,20),USE(E1)
                PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P2)
                ENTRY(@S8),AT(100,240,32,20),USE(E2)
              END
              TAB('Tab Two'),USE(?TabTwo)
                OPTION('Option 3'),USE(OptVar3)
                  RADIO('Radio 1'),AT(20,0,20,20),USE(?R5)
                  RADIO('Radio 2'),AT(40,0,20,20),USE(?R6)
                END
                PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P3)
                ENTRY(@S8),AT(100,140,32,20),USE(E3)
                PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P4)
                ENTRY(@S8),AT(100,240,32,20),USE(E4)
              END
            END
            BUTTON('Ok'),AT(100,180,20,20),USE(?Ok)
            BUTTON('Cancel'),AT(200,180,20,20),USE(?Cancel)
          END
```

Porównaj: TAB

SPIN (deklaruje pole spin)

```
SPIN( picture) ,AT( ) [,CURSOR()][,USE()][,DISABLE][,KEY()][,MSG()][,HLP()][,SKIP][,FONT()][,FULL]
      [,SCROLL][,ALRT()][,HIDE][,READONLY][,REQ] [,IMM][,TIP()][,TRN][DROPID()][,COLOR()]
      [,REPEAT( )] [,DELAY( )] [,MASK]
      [,  | UPR    |] [,  | LEFT    | ] [,  | INS    | ] , | RANGE()[,STEP]| [,  | HSCROLL   | ]
          | CAP    |      | RIGHT   |       | OVR    |     | FROM( ) |             | VSCROLL   |
                          | CENTER  |                                               | HVSCROLL  |
                          | DECIMAL |
```

SPIN Umieszcza pole spin w oknie WINDOW lub w pasku narzędzi TOOLBAR.

picture Wzorzec wyświetlania określający format wprowadzanych do kontrolki danych (PROP:Text).

AT Określa początkowy rozmiar i położenie kontrolki (PROP:AT). Jeśli jest pominięty, domyślne wartości są przyjmowane przez bibliotekę runtime.

CURSOR Określa kursor wyświetlany w momencie, gdy wskaźnik myszki zatrzyma się na kontrolce (PROP:CURSOR). Jeśli ominiemy ten atrybut, jest stosowany kursor wskazany dla okna lub też domyślny kursor systemu Windows.

USE Etykieta ekwiwalentu pola umożliwiająca odwoływanie się do kontrolki w kodzie wykonywalnym lub etykieta zmiennej, w której jest zapamiętywana wartość wybrana przez użytkownika (PROP:USE).

DISABLE Powoduje, że kontrolka po pierwszym otwarciu okna WINDOW lub APPLICATION jest niedostępna, wyszarzona (PROP:DISABLE).

KEY Określa stałą całkowitą lub ekwiwalent kodu klawisza, którego wciśnięcie powoduje bezpośrednie przekazanie aktywności kontrolce (PROP:KEY).

MSG Określa tekst wyświetlany w pasku stanu aplikacji wtedy, gdy dana kontrolka jest aktywna (PROP:MSG). Nie dotyczy raportu.

HLP Określa łańcuch stanowiący identyfikator sekcji systemu pomocy związanej z daną kontrolką (PROP:HLP).

SKIP Powoduje, że kontrolka nie otrzymuje aktywności (jest pomijana podczas poruszania się między kontrolkami za pomocą klawisza TAB). Dostęp do niej jest możliwy po jej kliknięciu myszką lub po wciśnięciu klawisza skrótu z nią związanego (PROP:SKIP).

FONT Określa czcionkę dla danej kontrolki (PROP:FONT).

FULL Powoduje, że rozmiar dla każdego brakującego atrybutu AT (długość i wysokość) kontrolki zwiększa się tak, by obejmował cały dostępny obszar okna WINDOW (PROP:FULL).

SCROLL Nadanie tego parametru powoduje, że kontrolka jest przewijana wraz z oknem (PROP:SCROLL).

ALRT Określa klawisze skrótu aktywne dla kontrolki (PROP:ALRT).

HIDE Powoduje, że po pierwszym otwarciu okna WINDOW lub APPLICATION kontrolka pozostaje ukryta (PROP:HIDE). Zostanie ona wyświetlona dopiero po użyciu procedury UNHIDE.

READONLY Kontrolka nie pozwala na wprowadzanie danych, a jedynie na ich odczytanie (PROP:READONLY).

REQ Określa, że kontrolka nie może pozostać nie wypełniona i nie może zawierać wartości zerowej (PROP:REQ).

IMM Wymusza generowanie zdarzenia za każdym razem, gdy użytkownik naciśnie jakiś klawisz (PROP:IMM).

TIP Definiuje tekst wyświetlany „w dymkach" po zatrzymaniu kursora myszki na kontrolce (PROP:ToolTip).

TRN Powoduje, że kontrolka jest wyświetlana jako przeźroczysta, nie przykrywa innych kontrolek ani tła (PROP:TRN).

DROPID Określa, że kontrolka może służyć jako cel dla operacji „przenieś i upuść" (PROP:DROPID).

COLOR Określa kolor tła i kolor podświetlenia dla kontrolki (PROP:COLOR).

REPEAT Określa częstotliwość, z jaką jest generowane zdarzenie EVENT:NewSelection, gdy użytkownik wciśnie i „przytrzyma" jeden z przycisków kontrolki SPIN (PROP:REPEAT).

DELAY Określa odstęp czasu pomiędzy pierwszym i drugim zdarzeniem EVENT:NewSelection wygenerowanym dla przytrzymanego przycisku kontrolki SPIN (PROP:DELAY).

MASK Wymusza wprowadzanie według wzorca w polu tekstowym kontrolki (PROP:MASK).

UPR/CAP Wprowadza tryb zastępowania wszystkich liter na wielkie lub na kapitaliki (Każda Pierwsza Litera Wyrazu W Zdaniu Jest Wielka); odpowiednie właściwości to PROP:UPR i PROP:CAP.

LEFT Powoduje, że dane są wyrównywane do lewej w obszarze określonym przez atrybut AT (PROP:LEFT).

RIGHT Powoduje, że dane są wyrównywane do prawej w obszarze określonym przez atrybut AT (PROP:RIGHT).

CENTER Powoduje, że dane są centrowane w obszarze określonym przez atrybut AT (PROP:CENTER).

DECIMAL Powoduje, że dane są wyrównywane do kropki dziesiętnej w obszarze określonym przez atrybut AT (PROP:DECIMAL).

INS/OVR Wprowadza tryb wstawiania (Insert) lub zastępowania (Overwrite) podczas wprowadzania danych (PROP:INS i PROP:OVR). Daje to efekt tylko w przypadku okien posiadających atrybut MASK.

RANGE Definiuje zakres wartości, które można wprowadzić do kontrolki (PROP:RANGE).

STEP Definiuje wartość o jaką zwiększa się lub zmniejsza wartość znajdująca się w kontrolce w wyniku wciśnięcia odpowiedniego przycisku; wartość kontrolki musi dodatkowo mieścić się w zakresie określonym przez atrybut RANGE (PROP:STEP). Jeśli pominiemy atrybut STEP, to jest dla niego przyjmowana domyślna wartość 1.0.

FROM Wskazuje źródło opcji udostępnianych użytkownikowi do wyboru (PROP:FROM).

HSCROLL Powoduje, że przyciski spin znajdują się jeden obok drugiego; przycisk zmniejszający wartość to strzałka skierowana w lewo, przycisk zwiększający wartość – strzałka skierowana w prawo (PROP:HSCROLL).

VSCROLL Powoduje, że przyciski spin znajdują się jeden pod drugim; przycisk zmniejszający wartość to strzałka skierowana w lewo, przycisk zwiększający wartość – strzałka skierowana w prawo (PROP:VSCROLL).

HVSCROLL Powoduje, że przyciski spin znajdują się jeden obok drugiego; przycisk zmniejszający wartość to strzałka skierowana w dół, przycisk zwiększający wartość – strzałka skierowana w górę (PROP:HVSCROLL).

Kontrolka **SPIN** umieszcza listę elementów danych w oknie WINDOW lub w pasku narzędzi TOOLBAR (nie dotyczy raportu) w pozycji i o rozmiarze określonym przez atrybut AT. W kontrolce wyświetlany jest tylko aktualnie wybrany element oraz, po prawej stronie, para przycisków ze strzałkami. Przyciski te pozwalają na wybieranie kolejnych elementów listy, w porządku rosnącym lub malejącym.

Jeśli kontrolka SPIN pozwala użytkownikowi na wybieranie wartości numerycznych zwiększających się lub zmniejszających o stałą wartość, atrybut RANGE określa wartość minimalną i maksymalną. Atrybut STEP działa w połączeniu z atrybutem RANGE określając wartość o jaką jest zmniejszana bądź zwiększana wartość znajdująca się w kontrolce.

Jeśli elementy nie są regularne, bądź są łańcuchami tekstowymi, zamiast atrybutów RANGE i STEP jest stosowany atrybut FROM. Atrybut FROM pozwala na pobranie listy elementów z kolejki QUEUE lub z łańcucha. Stosując atrybut FROM mamy możliwość umieszczania w kontrolce SPIN dowolnych typów elementów. Użytkownik może wybrać element z listy lub wpisać wartość bezpośrednio, ponieważ ta kontrolka może działać analogicznie jak kontrolka ENTRY.

Generowane zdarzenia:

EVENT:Selected	Kontrolka otrzymała aktywność wprowadzania.
EVENT:Accepted	Użytkownik wybrał wartość lub wprowadził dane bezpośrednio do kontrolki. Kontrolka utraciła aktywność wprowadzania.
EVENT:Rejected	Użytkownik wprowadził niewłaściwą wartość, niezgodną ze wzorcem wprowadzania.
EVENT:NewSelection	Użytkownik wybrał nową wartość.
EVENT:PreAlertKey	Użytkownik nacisnął klawisz wskazany atrybutem ALRT.
EVENT:AlertKey	Użytkownik nacisnął klawisz wskazany atrybutem ALRT.
EVENT:Drop	Operacja drag-and-drop dla kontrolki zakończyła się pomyślnie.

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            SPIN(@S8),AT(0,0,20,20),USE(SpinVar1),FROM(Que)
            SPIN(@N3),AT(20,0,20,20),USE(SpinVar2),RANGE(1,999),KEY(F10Key)
            SPIN(@N3),AT(40,0,20,20),USE(SpinVar3),RANGE(5,995),STEP(5)
            SPIN(@S8),AT(60,0,20,20),USE(SpinVar4),FROM(Que),HLP('Check4Help')
            SPIN(@S8),AT(80,0,20,20),USE(SpinVar5),FROM(Que),MSG('Button 3')
            SPIN(@S8),AT(100,0,20,20),USE(SpinVar6),FROM(Que),FONT('Arial',12)
            SPIN(@S8),AT(120,0,20,20),USE(SpinVar7),FROM(Que),DROP
            SPIN(@S8),AT(160,0,20,20),USE(SpinVar8),FROM(Que),IMM
            SPIN(@S8),AT(220,0,20,20),USE(SpinVar9),FROM('Mr|Mrs|Ms'),LEFT
          END
```

STRING (deklaruje łańcuch tekstowy)

```
STRING( text) ,AT( ) [,CURSOR( )] [,USE( )] [,DISABLE] [,FONT( )] [,FULL] [,SCROLL] [,HIDE]
      [,TRN] [,DROPID( )] [,COLOR( )] [,ANGLE( )] [,SKIP]
      [,  | LEFT      | ] [,  | PAGENO                                          | ]
          | RIGHT     |       | CNT( )     [, RESET( ) / PAGE ] [, TALLY( ) ]   |
          | CENTER    |       | SUM( )     [, RESET( ) / PAGE ] [, TALLY( ) ]   |
          | DECIMAL   |       | AVE( )     [, RESET( ) / PAGE ] [, TALLY( ) ]   |
                              | MIN( )     [, RESET( ) / PAGE ] [, TALLY( ) ]   |
                              | MAX( )     [, RESET( ) / PAGE ] [, TALLY( ) ]   |
```

STRING Umieszcza tekst reprezentowany przez *text* w oknie WINDOW, pasku narzedzi TOOLBAR lub w raporcie REPORT.

text Łańcuch zawierający tekst przeznaczony do wyświetlenia lub wzorzec wyświetlania dla zmiennej wskazanej za pomocą atrybutu USE (PROP:Text).

AT Określa początkowy rozmiar i położenie kontrolki (PROP:AT). Jeśli jest pominięty, domyślne wartości są przyjmowane przez bibliotekę runtime.

CURSOR Określa kursor wyświetlany w momencie, gdy wskaźnik myszki zatrzyma się na kontrolce (PROP:CURSOR). Jeśli ominiemy ten atrybut, jest stosowany kursor wskazany dla okna lub też domyślny kursor systemu Windows. Nie dotyczy raportu.

USE Etykieta ekwiwalentu pola umożliwiająca odwoływanie się do kontrolki w kodzie wykonywalnym lub zmienna, której wartość jest wyświetlana wg wzorca zadeklarowanego zamiast tekstu (PROP:USE).

DISABLE Powoduje, że kontrolka po pierwszym otwarciu okna WINDOW lub APPLICATION jest niedostępna, wyszarzona (PROP:DISABLE).

FONT Określa czcionkę dla danej kontrolki (PROP:FONT).

FULL Powoduje, że rozmiar dla każdego brakującego atrybutu AT (długość i wysokość) kontrolki zwiększa się tak, by obejmował cały dostępny obszar okna WINDOW (PROP:FULL). Nie dotyczy raportu.

SCROLL Nadanie tego parametru powoduje, że kontrolka jest przewijana wraz z oknem (PROP:SCROLL). Nie dotyczy raportów.

HIDE Powoduje, że po pierwszym otwarciu okna WINDOW lub APPLICATION kontrolka pozostaje ukryta (PROP:HIDE). Zostanie ona wyświetlona dopiero po użyciu procedury UNHIDE.

TRN Powoduje, że kontrolka jest wyświetlana jako przeźroczysta, nie przykrywa innych kontrolek ani tła (PROP:TRN).

DROPID Określa, że kontrolka może służyć jako cel dla operacji „przenieś i upuść” (PROP:DROPID). Nie dotyczy raportu.

SKIP Powoduje, że kontrolka nie jest drukowana, gdy jej zawartość jest łańcuchem pustym. Wszystkie występujące po niej kontrolki są przesuwane w górę w ramach danej sekcji raportu, by wypełnić wolne miejsce (PROP:SKIP). Dotyczy tylko raportów.

LEFT Powoduje, że tekst jest wyrównywany do lewej w obszarze określonym przez atrybut AT (PROP:LEFT).

RIGHT Powoduje, że tekst jest wyrównywany do prawej w obszarze określonym przez atrybut AT (PROP:RIGHT).

CENTER Powoduje, że tekst jest centrowany w obszarze określonym przez atrybut AT (PROP:CENTER).

DECIMAL Powoduje, że tekst jest wyrównywany do kropki dziesiętnej w obszarze określonym przez atrybut AT (PROP:DECIMAL).

COLOR Określa kolor tła dla kontrolki (PROP:COLOR).

ANGLE Powoduje wyświetlanie lub drukowanie tekstu pod określonym kątem liczonym w kierunku przeciwnym do ruchu wskazówek zegara od poziomu lub od orientacji raportu (PROP:ANGLE).

PAGENO Powoduje, że w kontrolce jest drukowany numer bieżącej strony raportu; w formacie określonym przez parametr *text* (PROP:PAGENO). Dotyczy tylko raportów.

CNT Powoduje, że w kontrolce jest drukowana liczba elementów (details) raportu; w formacie określonym przez parametr *text* (PROP:CNT). Dotyczy tylko raportów.

SUM Powoduje, że w kontrolce jest drukowana suma wartości zmiennej wskazanej przez atrybut USE; w formacie określonym przez parametr *text* (PROP:SUM). Dotyczy tylko raportów.

AVE Powoduje, że w kontrolce jest drukowana średnia wartości zmiennej wskazanej przez atrybut USE; w formacie określonym przez parametr *text* (PROP:AVE). Dotyczy tylko raportów.

MIN Powoduje, że w kontrolce jest drukowana wartość minimalna wartości zmiennej wskazanej przez atrybut USE; w formacie określonym przez parametr *text* (PROP:MIN). Dotyczy tylko raportów.

MAX Powoduje, że w kontrolce jest drukowana wartość maksymalna wartości zmiennej wskazanej przez atrybut USE; w formacie określonym przez parametr *text* (PROP:MAX). Dotyczy tylko raportów.

RESET Wymusza zerowanie wartości kontrolki z atrybutem CNT, SUM, AVE, MIN lub MAX w momencie, gdy zaczyna się nowa grupa rekordów w raporcie (PROP:RESET). Dotyczy tylko raportów.

PAGE Wymusza zerowanie wartości kontrolki z atrybutem CNT, SUM, AVE, MIN lub MAX w momencie, gdy zaczyna się nowa strona raportu (PROP:PAGE). Dotyczy tylko raportów.

TALLY Określa, kiedy mają być wyliczane wartości kontrolki z atrybutem CNT, SUM, AVE, MIN lub MAX (PROP:TALLY). Dotyczy tylko raportów.

Kontrolka **STRING** umieszcza *text* w oknie WINDOW, pasku narzędzi TOOLBAR lub w raporcie REPORT w pozycji i o rozmiarze określonym przez atrybut AT.

Jeśli parametr *text* jest wzorcem wprowadzania (picture token), a nie stałym łańcuchem, zawartość zmiennej wskazywanej przez atrybut USE jest formatowana w oparciu o ten właśnie wzorzec. Otrzymujemy wówczas kontrolkę, w której jest wyświetlana wartość zmiennej w trybie "tylko-do-odczytu", bez możliwości aktualizacji. Dane wyświetlane w kontrolce STRING są automatycznie odświeżane za każdym razem, gdy zaczyna się pętla ACCEPT, niezależnie od tego, czy występuje atrybut AUTO, czy też nie.

Występuje pewna różnica pomiędzy znakiem ampersand (&) stosowanym w kontrolkach STRING i PROMPT. Znak ampersand w kontrolce STRING jest wyświetlany jako część tekstu *text*, podczas gdy znak ampersand w kontrolce PROMPT definiuje klawisz skrótu.

Kontrolka STRING posiadająca atrybut TRN jest wyświetlana jako przeźroczysta (bez tła). W tej sytuacji pojawiają się tylko piksele tworzące napis, tło nie przykrywa tego, co znajduje się pod kontrolką. Pozwala to na przykład na eleganckie umieszczanie napisów na rysunkach.

Kontrolka nie otrzymuje aktywności wprowadzania.

Generowane zdarzenia:

EVENT:Drop	Operacja drag-and-drop dla kontrolki zakończyła się pomyślnie.

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            STRING('String Constant'),AT(10,0,20,20),USE(?S1)
            STRING(@S30),AT(10,20,20,20),USE(StringVar1)
            STRING(@S30),AT(10,20,20,20),USE(StringVar2),CURSOR(CURSOR:Wait)
            STRING(@S30),AT(10,20,20,20),USE(StringVar3),FONT('Arial',12)
          END
CustRpt REPORT,AT(1000,1000,6500,9000),THOUS
Break1      BREAK(Pre:Key1)
              HEADER,AT(0,0,6500,1000)
                STRING('Group Head'),AT(3000,500,1500,500),FONT('Arial',18)
              END
Detail        DETAIL,AT(0,0,6500,1000)
                STRING(@N$11.2),AT(6000,1500,500,500),USE(Pre:F1)
              END
              FOOTER,AT(0,0,6500,1000)
                STRING('Group Total:'),AT(5500,500,1500,500)
                STRING(@N$11.2),AT(6000,500,500,500),USE(Pre:F1),SUM,RESET(Pre:Key1)
              END
            END
          END
```

TAB (deklaruje zakładkę)

```
TAB( text ) [,USE()] [,KEY()] [,MSG()] [,HLP()] [,REQ] [DROPID()] [,TIP()]
        [,COLOR()] [,FONT()]
   controls
END
```

TAB Deklaruje zakładkę grupującą kontrolki w ramach arkusza zakładek (SHEET).

tekst Stała łańcuchowa zawierająca tekst wyświetlany na zakładce (PROP:Text).

USE Etykieta ekwiwalentu pola umożliwiająca odwoływanie się do kontrolki w kodzie wykonywalnym (PROP:USE).

KEY Określa stałą całkowitą lub ekwiwalent kodu klawisza, którego wciśnięcie powoduje bezpośrednie przekazanie aktywności kontrolce (PROP:KEY).

MSG Określa tekst wyświetlany w pasku stanu aplikacji wtedy, gdy dowolna kontrolka zakładki TAB jest aktywna (PROP:MSG). Nie dotyczy raportu.

HLP Określa łańcuch stanowiący domyślny identyfikator sekcji systemu pomocy związanej z dowolną kontrolką zakładki TAB (PROP:HLP).

REQ Powoduje, że w momencie wybrania innej zakładki TAB przez użytkownika, biblioteka runtime automatycznie sprawdza, czy wszystkie kontrolki ENTRY danej zakładki posiadające atrybut REQ zawierają dane różne od 0 lub łańcuchy niepuste (PROP:REQ).

DROPID Określa, że kontrolka może służyć jako cel dla operacji „przenieś i upuść” (PROP:DROPID).

TIP Definiuje tekst wyświetlany „w dymkach” po zatrzymaniu kursora myszki na kontrolce (PROP:ToolTip).

COLOR Określa kolor tła dla kontrolki oraz domyślny kolor dla wszystkich kontrolek znajdujących się w zakładce TAB (PROP:COLOR).

FONT Określa czcionkę dla tekstu wyświetlanego na zakładce TAB (PROP:FONT). Nie ma to wpływu na kontrolki znajdujące się tej zakładce.

controls Deklaracje kontrolek.

Struktura **TAB** deklaruje zakładkę arkusza zakładek SHEET, w której są zgrupowane kontrolki (nie dotyczy raportu). Kontrolki TAB występujące w strukturze SHEET definiują „strony” wyświetlane użytkownikowi. Atrybut USE struktury SHEET rejestruje *text* kontrolki TAB wybranej przez użytkownika.

Zmiana aktywności wprowadzania kontrolek zgrupowanych w zakładce TAB jest sygnalizowana tylko strukturze SHEET, do której ta zakładka należy. Oznacza to, że zdarzenia generowane w momencie, gdy użytkownik przekazuje aktywność wprowadzania w ramach struktury SHEET są zdarzeniami zależnymi od pól dla kontrolki SHEET, natomiast indywidualne zakładki TAB nie generują zdarzeń.

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            SHEET,AT(0,0,320,175),USE(SelectedTab)
              TAB('Tab One'),USE(?TabOne)
                OPTION('Option 1'),USE(OptVar1),KEY(F10Key),HLP('Option1Help')
                  RADIO('Radio 1'),AT(20,0,20,20),USE(?R1)
                  RADIO('Radio 2'),AT(40,0,20,20),USE(?R2)
                END
                OPTION('Option 2'),USE(OptVar2),MSG('Option 2')
                  RADIO('Radio 3'),AT(60,0,20,20),USE(?R3)
                  RADIO('Radio 4'),AT(80,0,20,20),USE(?R4)
                END
                PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P1)
                ENTRY(@S8),AT(100,140,32,20),USE(E1)
                PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P2)
                ENTRY(@S8),AT(100,240,32,20),USE(E2)
              END
              TAB('Tab Two'),USE(?TabTwo)
                OPTION('Option 3'),USE(OptVar3)
                  RADIO('Radio 1'),AT(20,0,20,20),USE(?R5)
                  RADIO('Radio 2'),AT(40,0,20,20),USE(?R6)
                END
                OPTION('Option 4'),USE(OptVar4)
                  RADIO('Radio 3'),AT(60,0,20,20),USE(?R7)
                  RADIO('Radio 4'),AT(80,0,20,20),USE(?R8)
                END
                PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P3)
                ENTRY(@S8),AT(100,140,32,20),USE(E3)
                PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P4)
                ENTRY(@S8),AT(100,240,32,20),USE(E4)
              END
            END
            BUTTON('Ok'),AT(100,180,20,20),USE(?Ok)
            BUTTON('Cancel'),AT(200,180,20,20),USE(?Cancel)
          END
```

Porównaj: SHEET

TEXT (deklaruje wielowierszowe pole wprowadzania)

```
TEXT ,AT( ) [,CURSOR( )] [,USE( )] [,DISABLE] [,KEY( )] [,MSG( )] [,HLP( )] [,SKIP] [,FONT( )]
     [,REQ] [,FULL] [,SCROLL] [,ALRT( )] [,HIDE] [,READONLY] [,DROPID( )] [,UPR] [,TRN]
     [,TIP( )] [,  | INS    |] [,  | HSCROLL    |] [,  | LEFT       | ][,COLOR( )][,SINGLE][,RESIZE]
                   | OVR    |      | VSCROLL    |      | RIGHT      |
                                   | HVSCROLL   |      | CENTER     |
```

TEXT Umieszcza wielowierszowe pole wprowadzania w oknie WINDOW, pasku narzędzi TOOLBAR lub w raporcie REPORT.

AT Określa początkowy rozmiar i położenie kontrolki (PROP:AT). Jeśli jest pominięty, domyślne wartości są przyjmowane przez bibliotekę runtime.

CURSOR Określa kursor wyświetlany w momencie, gdy wskaźnik myszki zatrzyma się na kontrolce (PROP:CURSOR). Jeśli ominiemy ten atrybut, jest stosowany kursor wskazany dla okna lub też domyślny kursor systemu Windows. Nie dotyczy raportu.

USE Etykieta zmiennej, w której są zapamiętywane wartości wprowadzone do kontrolki przez użytkownika (PROP:USE).

DISABLE Powoduje, że kontrolka po pierwszym otwarciu okna WINDOW lub APPLICATION jest niedostępna, wyszarzona (PROP:DISABLE).

KEY Określa stałą całkowitą lub ekwiwalent kodu klawisza, którego wciśnięcie powoduje bezpośrednie przekazanie aktywności kontrolce (PROP:KEY). Nie dotyczy raportu.

MSG Określa tekst wyświetlany w pasku stanu aplikacji wtedy, gdy dana kontrolka jest aktywna (PROP:MSG). Nie dotyczy raportu.

HLP Określa łańcuch stanowiący identyfikator sekcji systemu pomocy związanej z daną kontrolką (PROP:HLP). Nie dotyczy raportu.

SKIP Powoduje, że kontrolka nie otrzymuje aktywności (jest pomijana podczas poruszania się między kontrolkami za pomocą klawisza TAB). Dostęp do niej jest możliwy po jej kliknięciu myszką lub po wciśnięciu klawisza skrótu z nią związanego (PROP:SKIP). W raportach SKIP powoduje, że dana kontrolka nie będzie drukowana, gdy jej wartość jest pusta. Dodatkowo, wszystkie następujące po niej kontrolki w danej sekcji raportu są przesuwane do góry, by zapełnić puste miejsce.

FONT Określa czcionkę dla danej kontrolki (PROP:FONT).

REQ Określa, że kontrolka nie może pozostać nie wypełniona i nie może zawierać wartości zerowej (PROP:REQ). Nie dotyczy raportu.

FULL Powoduje, że rozmiar dla każdego brakującego atrybutu AT (długość i wysokość) kontrolki zwiększa się tak, by obejmował cały dostępny obszar okna WINDOW (PROP:FULL). Nie dotyczy raportu.

SCROLL Nadanie tego parametru powoduje, że kontrolka jest przewijana wraz z oknem (PROP:SCROLL). Nie dotyczy raportów.

ALRT Określa klawisze skrótu aktywne dla kontrolki (PROP:ALRT). Nie dotyczy raportu.

HIDE Powoduje, że po pierwszym otwarciu okna WINDOW lub APPLICATION kontrolka pozostaje ukryta (PROP:HIDE). Zostanie ona wyświetlona dopiero po użyciu procedury UNHIDE.

READONLY Powoduje, że kontrolka nie umożliwia edytowania wyświetlanych w niej danych; są one udostępniane tylko do odczytu (PROP:READONLY). Nie dotyczy raportu.

DROPID Określa, że kontrolka może służyć jako cel dla operacji „przenieś i upuść" (PROP:DROPID). Nie dotyczy raportu.

UPR Wprowadza tryb edycji, w którym wszystkie wpisywane litery są zastępowane odpowiadającymi im wielkimi literami (PROP:UPR).

TIP Definiuje tekst wyświetlany „w dymkach" po zatrzymaniu kursora myszki na kontrolce (PROP:ToolTip). Nie dotyczy raportu.

INS/OVR Wprowadza tryb wstawiania (Insert) lub zastępowania (Overwrite) podczas wprowadzania danych (PROP:INS i PROP:OVR). Daje to efekt tylko w przypadku okien posiadających atrybut MASK. Nie dotyczy raportu.

HSCROLL Powoduje, że w polu tekstowym pojawia się automatycznie poziomy pasek przewijania, gdy jego poziomy rozmiar nie pozwala na wyświetlenie wszystkich danych (PROP:HSCROLL). Nie dotyczy raportów.

VSCROLL Powoduje, że w polu tekstowym pojawia się automatycznie pionowy pasek przewijania, gdy jego pionowy rozmiar nie pozwala na wyświetlenie wszystkich danych (PROP:VSCROLL). Nie dotyczy raportów.

HVSCROLL Powoduje, że w polu tekstowym pojawiają się automatycznie poziomy i pionowy pasek przewijania, gdy rozmiar pola nie pozwala na wyświetlenie wszystkich danych (PROP:HVSCROLL). Nie dotyczy raportów.

LEFT Powoduje, że dane są wyrównywane do lewej w obszarze określonym przez atrybut AT (PROP:LEFT).

RIGHT Powoduje, że dane są wyrównywane do prawej w obszarze określonym przez atrybut AT (PROP:RIGHT).

CENTER Powoduje, że dane są środkowane w obszarze określonym przez atrybut AT (PROP:CENTER).

COLOR Określa kolor tła dla kontrolki (PROP:COLOR).

SINGLE Powoduje, że w kontrolce można umieścić tylko pojedynczy wiersz danych (PROP:SINGLE). Jest to stosowane na przykład w celu umożliwienia stosowania kontrolki TEXT zamiast ENTRY dla alfabetów hebrajskich lub arabskich. Nie dotyczy raportu.

RESIZE Powoduje dobieranie wysokości drukowanej czcionki dla kontrolki w zależności od jej aktualnej zawartości (PROP:RESIZE). Dotyczy tylko raportu.

Kontrolka **TEXT** umieszcza w oknie WINDOW (lub w pasku narzędzi TOOLBAR) wielowierszowe pole wprowadzania danych w pozycji i o rozmiarze określonym przez atrybut AT. Zmienna określona atrybutem USE rejestruje dane wprowadzone przez użytkownika w momencie, gdy kontrolka zostaje zatwierdzona. Wprowadzane wiersze są automatycznie zawijane (word-wrap).

Pojemność kontrolki TEXT zależy od systemu operacyjnego. W systemach 16-bitowych pojemność ta zależy od ilości wolnego miejsca w domyślnym segmencie danych. W systemach 32-bitowych pojemność jest o wiele większa.

Generowane zdarzenia:

EVENT:Selected	Kontrolka otrzymała aktywność wprowadzania.
EVENT:Accepted	Użytkownik zakończył wprowadzanie danych i zaakceptował kontrolkę.
EVENT:PreAlertKey	Użytkownik nacisnął klawisz wskazany atrybutem ALRT.
EVENT:AlertKey	Użytkownik nacisnął klawisz wskazany atrybutem ALRT.
EVENT:Drop	Operacja drag-and-drop dla kontrolki zakończyła się pomyślnie.

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            TEXT,AT(0,0,40,40),USE(E1),ALRT(F10Key),CENTER
            TEXT,AT(20,0,40,40),USE(E2),KEY(F10Key),HLP('Text4Help')
            TEXT,AT(40,0,40,40),USE(E3),SCROLL,OVR,UPR
            TEXT,AT(60,0,40,40),USE(E4),CURSOR(CURSOR:Wait),RIGHT
            TEXT,AT(80,0,40,40),USE(E5),DISABLE,FONT('Arial',12)
            TEXT,AT(100,0,40,40),USE(E6),HVSCROLL,LEFT
            TEXT,AT(120,0,40,40),USE(E7),REQ,INS,CAP,MSG('Text Field 7')
          END
CustRpt REPORT,AT(1000,1000,6500,9000),THOUS
Detail    DETAIL,AT(0,0,6500,1000)
            TEXT,AT(0,0,40,40),USE(E1)
            TEXT,AT(100,0,40,40),USE(E6),FONT('Arial',12)
            TEXT,AT(120,0,40,40),USE(E7),CAP
            TEXT,AT(140,0,40,40),USE(E8),UPR
            TEXT,AT(160,0,40,40),USE(E9),LEFT
            TEXT,AT(180,0,40,40),USE(E10),RIGHT
            TEXT,AT(200,0,40,40),USE(E11),CENTER
          END
        END
```

Porównaj: ENTRY

VBX (deklaruje kontrolkę .VBX)

VBX(*text*) ,**AT**( ) [,**CLASS**( )] [,**CURSOR**( )] [,**USE**( )] [,**DISABLE**] [,**KEY**( )] [,**MSG**( )]
[,**HLP**( )] [,**SKIP**] [,**FULL**] [,**SCROLL**] [,**ALRT**( )] [,**HIDE**] [,**FONT**( )] [**DROPID**( )]
[,**TIP**( )] [,**META**] [, *property*(*value*)]

VBX Umieszcza kontrolkę .VBX języka Visual Basic w oknie WINDOW, pasku narzędzi TOOLBAR lub w raporcie REPORT.

tekst Stała łańcuchowa zawierająca tekst stanowiący tytuł kontrolki (PROP:Text).

AT Określa początkowy rozmiar i położenie kontrolki (PROP:AT). Jeśli jest pominięty, domyślne wartości są przyjmowane przez bibliotekę runtime.

CLASS Określa nazwę pliku .VBX oraz typ kontrolki (PROP:CLASS).

CURSOR Określa kursor wyświetlany w momencie, gdy wskaźnik myszki zatrzyma się na kontrolce (PROP:CURSOR). Jeśli ominiemy ten atrybut, jest stosowany kursor wskazany dla okna lub też domyślny kursor systemu Windows. Nie dotyczy raportu.

USE Etykieta zmiennej, w której jest zapamiętywana wartość kontrolki (PROP:USE).

DISABLE Powoduje, że kontrolka po pierwszym otwarciu okna WINDOW lub APPLICATION jest niedostępna, wyszarzona (PROP:DISABLE).

KEY Określa stałą całkowitą lub ekwiwalent kodu klawisza, którego wciśnięcie powoduje bezpośrednie przekazanie aktywności kontrolce (PROP:KEY). Nie dotyczy raportu.

MSG Określa tekst wyświetlany w pasku stanu aplikacji wtedy, gdy dana kontrolka jest aktywna (PROP:MSG). Nie dotyczy raportu.

HLP Określa łańcuch stanowiący identyfikator sekcji systemu pomocy związanej z daną kontrolką (PROP:HLP). Nie dotyczy raportu.

SKIP Powoduje, że kontrolka nie otrzymuje aktywności (jest pomijana podczas poruszania się między kontrolkami za pomocą klawisza TAB). Dostęp do niej jest możliwy po jej kliknięciu myszką lub po wciśnięciu klawisza skrótu z nią związanego (PROP:SKIP). Nie dotyczy raportu.

FULL Powoduje, że rozmiar dla każdego brakującego atrybutu AT (długość i wysokość) kontrolki zwiększa się tak, by obejmował cały dostępny obszar okna WINDOW (PROP:FULL). Nie dotyczy raportu.

SCROLL Nadanie tego parametru powoduje, że kontrolka jest przewijana wraz z oknem (PROP:SCROLL). Nie dotyczy raportów.

ALRT Określa klawisze skrótu aktywne dla kontrolki (PROP:ALRT). Nie dotyczy raportu.

HIDE Powoduje, że po pierwszym otwarciu okna WINDOW lub APPLICATION kontrolka pozostaje ukryta (PROP:HIDE). Zostanie ona wyświetlona dopiero po użyciu procedury UNHIDE.

FONT Określa czcionkę dla danej kontrolki (PROP:FONT).

DROPID Określa, że kontrolka może służyć jako cel dla operacji „przenieś i upuść” (PROP:DROPID). Nie dotyczy raportu.

TIP Definiuje tekst wyświetlany „w dymkach” po zatrzymaniu kursora myszki na kontrolce (PROP:ToolTip). Nie dotyczy raportu.

META Powoduje drukowanie kontrolki jako meta-pliku (metafile - .WMF) Windows (PROP:META). Dotyczy tylko raportu.

property Stała łańcuchowa zawierająca nazwę właściwości kontrolki.

value Stała łańcuchowa zawierająca wartość (lub etykietę ekwiwalentu EQUATE) określonej właściwości (*property*) kontrolki.

Kontrolka **VBX** umieszcza kontrolkę Visual Basic .VBX w oknie WINDOW, w pasku narzędzi TOOLBAR lub w raporcie REPORT w pozycji i o rozmiarze określonym przez atrybut AT.

Parametr *property* pozwala na określenie dodatkowych ustawień właściwości wymaganych dla konkretnej kontrolki .VBX. Są to właściwości, które muszą być ustawione, by kontrolka działała prawidłowo, nie należy ich mylić ze standardowymi właściwościami Clariona, takimi jak AT, CURSOR, czy USE. Typowa kontrolka rejestruje jedynie wartości nadane określonym dla niej właściwościom. Prawidłowe właściwości dla kontrolki i ich wartości powinny zostać opisane w dokumentacji kontrolki. Możemy określić wiele parametrów *property* dla pojedynczej kontrolki VBX.

VBX dostarcza mechanizm analogiczny do instrukcji ON ERROR języka Visual Basic. Gdy tylko .VBX wygeneruje błąd wewnętrzny, program zarejestruje zdarzenie EVENT:VBXevent, przy którym właściwość PROP:VBXEvent będzie ustawiona na wartość ‘&OnError’, a właściwość PROP:VBXEventArg będzie zawierała numer błędu (pierwszy argument) i jego opis (drugi argument).

Generowane zdarzenia:

EVENT:Selected	Kontrolka otrzymała aktywność wprowadzania.
EVENT:VBXevent	Zaszło zdarzenie związane z kontrolką VBX. Należy sprawdzić wartości właściwości PROP:VBXEvent oraz PROP:VBXEventArg.
EVENT:Accepted	Użytkownik zaakceptował kontrolkę; utraciła ona aktywność wprowadzania.
EVENT:PreAlertKey	Użytkownik nacisnął klawisz wskazany atrybutem ALRT.
EVENT:AlertKey	Użytkownik nacisnął klawisz wskazany atrybutem ALRT.
EVENT:Drop	Operacja drag-and-drop dla kontrolki zakończyła się pomyślnie.

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
          VBX,AT(0,0,120,320),USE(C1), |
            CLASS('graph.vbx','graph'),'graphstyle'('2')
          END

MsgNum    LONG
MsgTxt    STRING

  CODE
    OPEN(MDIChild)
    ACCEPT
      CASE EVENT()
      OF ?EVENT:VBXevent
        IF ?C1{PROP:VBXEvent} = '&OnError'              ! wykrycie warunku ON ERROR
          MsgNum = ?C1{PROP:VBXArg,1}
          MsgTxt = ?C1{PROP:VBXArg,2}
          MESSAGE('VBX Error ' & MsgNum & ' ' & MsgTxt)
        END
      END
      CASE ACCEPTED()
      OF ?C1
        ?C1{'graphstyle'} = '3'                         ! zmiana właściwości graphstyle "w locie"
                                                        ! za pomoca składni przypisania właściwości
      END
    END

Report    REPORT
DetailOne   DETAIL
            VBX,AT(0,0,120,320),CLASS('graph.vbx','graph'),'graphstyle'('2')
          END
        END
```

Porównaj: OLE

9 – ATRYBUTY OKIEN I RAPORTÓW

Ekwiwalenty właściwości atrybutów

Każdy atrybut posiada odpowiadającą mu właściwość runtime wymienioną w jego opisie (PROP:*atrybut*). Ekwiwalenty (equates) dla wszystkich właściwości runtime są zadeklarowane w pliku PROPERTY.CLW. Plik ten zawiera ponadto ekwiwalenty dla standardowych wartości wykorzystywanych przez niektóre z tych właściwości. Część właściwości jest przeznaczona jedynie do odczytu, część jedynie do zapisu (ich wartości nie da się odczytać), a część zarówno do odczytu, jak i do zapisu.

PROP:Text

PROP:Text odpowiada parametrowi *text* w deklaracjach APPLICATION(*text*), WINDOW(*text*) i dowolnych kontrolek *control*(*text*). Ta właściwość reprezentuje tekst w dowolnej deklaracji kontrolki lub okna i może zawierać wartość prawidłową dla określonej deklaracji.

Przykład:

```
?Image{PROP:Text} = 'Obrazek.BMP'              ! nowa mapa bitowa dla kontrolki IMAGE
?Prompt{PROP:Text} = 'Nowy tekst dla prompta'  ! nowy tekst dla kontrolki PROMPT
?Entry{PROP:Text} = '@N03'                     ! nowy wzorzec dla kontrolki ENTRY
```

Parametry właściwości atrybutów

Wiele atrybutów nie wymaga parametrów – mogą one występować bądź nie. Wtedy odpowiadające im właściwości runtime po prostu włączają (on) lub wyłączają (off) dany atrybut. Przypisanie właściwości runtime pustego łańcucha ('') lub wartości zerowej (0) wyłącza atrybut, przypisanie '1' lub 1 – włącza go. Zazwyczaj stosuje się tu standardowe ekwiwalenty dla wartości PRAWDA (TRUE) i FAŁSZ (FALSE). Sprawdzanie wartości tego typu właściwości runtime daje w rezultacie łańcuch pusty, gdy atrybut jest wyłączony dla danego okna, raportu, czy kontrolki. Jako przykład omawianego typu atrybutów mogą służyć: PROP:ABOVE, PROP:ABSOLUTE, czy PROP:ALONE.

Przykład:

```
?MyControl{PROP:DISABLE} = TRUE    ! wyłącza kontrolkę
```

Wiele atrybutów posiada pojedynczy parametr, którego występowanie oznacza zarówno aktywność atrybutu, jak również jego określoną wartość. Przypisanie łańcucha pustego ('') lub wartości zerowej (0) wyłącza atrybut. Przypisanie dowolnej innej, prawidłowej wartości – włącza go. Przykładem tego typu atrybutów mogą być PROP:TIMER i PROP:DROP.

Przykład:

```
MyWindow{PROP:TIMER} = 100    ! ustawia obsługę zdarzenia zegarowego dla okna co 1 sekundę
```

Właściwości tablicowe

Wiele właściwości atrybutów występuje w postaci tablic zawierających wiele wartości, takich jak PROP:ALRT, który może zawierać do 255 kodów klawiszy skrótów. Istnieją również takie, do których możemy się odwoływać tak, jak do tablic, zamiast stosować oddzielnie zadeklarowane ekwiwalenty. Przykładem takiej właściwości jest PROP:AT, dla której możemy stosować odwołania w postaci {PROP:AT,n}, gdzie n=1,2,3,4, zamiast odpowiadających im właściwości PROP:Xpos, PROP:Ypos, PROP:Width i PROP:Height.

Przykład:

```
PoleWyboru     STRING(1)
Screen   WINDOW
           ENTRY(@N3),USE(Kli:Kod)
           ENTRY(@S30),USE(Kli:Nazwa),REQ
           CHECK('Prawda lub Fałsz'),USE(PoleWyboru)
           IMAGE('Obrazek.BMP'),USE(?Image)
           BUTTON('OK'),USE(?OkButton),KEY(EnterKey)
           BUTTON('Anuluj'),USE(?CanxButton),KEY(EscKey)
         END

  CODE
  OPEN(Screen)
  Screen{PROP:AT,1} = 0                                  ! umiejscowienie okna w lewym, górnym rogu
  Screen{PROP:AT,2} = 0
  Screen{PROP:GRAY} = 1                                  ! włączenie wyglądu 3D
  Screen{PROP:STATUS,1} = -1                             ! utworzenie paska stanu z dwiema sekcjami
  Screen{PROP:STATUS,2} = 180
  Screen{PROP:STATUS,3} = 0                              ! Zakończenie tablicy paska stanu
  Screen{PROP:StatusText,2} = FORMAT(TODAY(),@D2)        ! umieszczenie daty w 2 sekcji paska stanu
  ?CtlCode{PROP:ALRT,1} = F10Key                         ! zgłoszenie klawisza F10 dla kontrolki Kli:kod
  ?CtlCode{PROP:Text} = '@N4'                            ! zmiana wzorca wprowadzania
  ?Image{PROP:Text} = 'MojObrazek.BMP'                   ! zmiana pliku wyświetlanego w kontrolce
  ?OkButton{PROP:DEFAULT} = '1'                          ! nadania atrybutu DEFAULT dla przycisku OK
  ?MyButton{PROP:ICON} = 'C:\Windows\MORICONS.DLL[10]'
                                                         ! wyświetlenie 11-tej ikony z pliku MORICONS.DLL
                                                         ! (licząc od zera)
  ?MyButton{PROP:ICON} = 'C:\Windows\MORICONS.DLL[0]'
                                                         ! wyświetlenie 1-ej ikony z pliku MORICONS.DLL
                                                         ! (licząc od zera)
  ?PoleWyboru{PROP:TrueValue} = 'T'                      ! ustawienei wartości dla stanów zaznaczenia (lub
                                                         ! jego braku) dla kontrolki CHECK
  ?PoleWyboru{PROP:FalseValue} = 'F'
  ACCEPT
  END
```

Atrybuty okna i raportu

ABSOLUTE (drukowanie w stałej pozycji)

ABSOLUTE

Atrybut **ABSOLUTE** (PROP:ABSOLUTE) zapewnia drukowanie sekcji DETAIL, nagłówka HEADER lub stopki FOOTER grupy (zawartej w strukturze BREAK), zawsze w tej samej, stałej pozycji strony. Gdy atrybut ABSOLUTE występuje, pozycja określona parametrami *x* i *y* atrybutu AT struktury jest względna wobec lewego, górnego narożnika strony. Atrybut ABSOLUTE nie wpływa na kolejne drukowane struktury nie posiadające takiego atrybutu.

Przykład:

```
MojRaport REPORT,AT(1000,2000,6500,9000),THOUS
          HEADER
            ! elementy struktury
          END
DetalPierwszy   DETAIL,AT(0,0,6500,1000)
            ! elementy struktury
          END
DetalDrugi   DETAIL,AT(1000,1000,6500,1000),ABSOLUTE   ! stała pozycja detalu
            ! elementy struktury
          END
        END
```

ALONE (drukowanie bez nagłówka, stopki i podkładu strony)

ALONE

Atrybut **ALONE** (PROP:ALONE) powoduje, że sekcja DETAIL, nagłówek HEADER lub stopka FOOTER grupy (znajdującej się wewnątrz struktury BREAK), są drukowane na stronie pozbawionej sekcji FORM, nagłówka HEADER bądź stopki FOOTER strony (nie znajdujących się wewnątrz struktury BREAK). Jest to przydatne przy drukowaniu stron tytułowych, czy też stron podsumowań.

Przykład:

```
MojRaport REPORT
TitlePage    DETAIL,ALONE        ! detal strony tytułowej
            ! elementy struktury
          END
CustDetail     DETAIL
            ! elementy struktury
          END
          FOOTER
            ! elementy struktury
          END
        END
```

ALRT (ustawienie klawiszy skrótów dla okna)

ALRT(*keycode*)

ALRT Określa klawisz skrótu (“gorący klawisz”) aktywny wtedy, gdy aplikacja APPLICATION, okno WINDOW lub kontrolka, dla której został zdefiniowany, posiada aktywność wprowadzania.

keycode Numeryczny kod klawisza lub jego ekwiwalent EQUATE.

Atrybut **ALRT** (PROP:ALRT) określa klawisz skrótu “gorący klawisz” aktywny wtedy, gdy aplikacja APPLICATION, okno WINDOW lub kontrolka, dla której został zdefiniowany, posiada aktywność wprowadzania.

Gdy użytkownik naciśnie „gorącą” kombinację klawiszy wskazaną za pomocą ALRT, generowane są dwa zdarzenia (niezależne od pola: jeśli ALRT został określony dla APPLICATION lub WINDOW; zależne od pola: jeśli ALRT został określony dla kontrolki): EVENT:PreAlertKey i EVENT:AlertKey; zdarzenia te występują w podanej kolejności.

Jeśli w kodzie nie jest wykonywana instrukcja CYCLE podczas przetwarzania zdarzenia EVENT:PreAlertKey, przerywamy wykonanie przez bibliotekę runtime domyślnej akcji przypisanej do określonego klawisza skrótu. Jeśli w kodzie jest wykonywana instrukcja CYCLE podczas przetwarzania EVENT:PreAlertKey, biblioteka runtime wykonuje domyślną akcję dla wciśniętego klawisza skrótu. W obu przypadkach po zdarzeniu EVENT:PreAlertKey jest generowane zdarzenie EVENT:AlertKey. Gdy zdarzenie EVENT:AlertKey jest generowane, zmienna wskazana atrybutem USE kontrolki posiadającej aktywność wprowadzania nie jest automatycznie aktualizowana; jeśli akcja taka jest wymagana, musimy użyć instrukcji UPDATE.

W oknie APPLICATION lub WINDOW bądź dla kontrolki możemy zdefiniować wiele atrybutów ALRT (do 255). Instrukcja ALERT oraz atrybut ALRT okna lub kontrolki są wzajemnie odseparowane. Oznacza to, że usunięcie klawiszy skrótów za pomocą ALERT nie ma żadnego wpływu na klawisze skrótów „zgłoszone” poprzez użycie atrybutu ALRT.

PROP:ALRT jest tablicą zawierająca do 255 kodów klawiszy. Aktualnie wykorzystywany numer elementu tablicy jest wewnętrznie przypisywany do pierwszego wolnego elementu, jeśli wskazany numer elementu jest większy niż bieżąca liczba przypisanych kodów klawiszy. Na przykład, przyjmując, że w ogóle nie ma klawiszy alertów, jeśli określimy przypisanie do elementu numer 255, to tak naprawdę będzie to dotyczyło elementu numer 1. Kolejne przypisanie następnego kodu klawisza do elementu numer 255 (nadal wolnego), spowoduje w rzeczywistości

przypisanie do elementu numer 2. Bezpośrednie przypisanie kodu klawisza do elementu numer 1 oczywiście zadziała, ale usunie poprzednie istniejące tam przypisanie.

Przykład:

```
OknoPierwsze  WINDOW,AT(0,0,160,400)
            ENTRY,AT(6,40),USE(PewnaWartosc1),ALRT(MouseLeft)    ! kontrolka śledzi kliknięcia myszką
            ENTRY,AT(60,40),USE(PewnaWartosc2),ALRT(F10Key)      ! kontrolka śledzi wciśnięcie klawisza F10
          END
  CODE
  OPEN(OknoPierwsze)
  ACCEPT
    CASE FIELD()
    OF ?PewnaWartosc1
      CASE EVENT()
      OF EVENT:PreAlertKey                                       ! wstępne sprawdzenie zdarzeń alertu
        CYCLE                                                    ! umożliwia przeprowadzenie
                                                                 ! standardowej akcji dla MouseLeft2
      OF EVENT:AlertKey                                          ! przetwarzanie alertu
        DO ObslugaKliknieca
      END
    OF ?PewnaWartosc2
      CASE EVENT()
      OF EVENT:AlertKey                                          ! przetwarzanie alertu
        DO ObslugaKlawiszaF10
      END
    END
  END
```

ANGLE (ustawienie kąta wyświetlania lub drukowania kontrolki)

ANGLE(*size*)

ANGLE Definiuje kąt położenia (orientację) kontrolki STRING.

size Stała całkowita lub wyrażenie stałe określające wielkość kąta w dziesiątych stopnia. Jeżeli wartość ta jest dodatnia, kontrolka jest obracana, w stosunku do poziomu, przeciwnie do ruchu wskazówek zegara, jeśli ujemna – zgodnie z ruchami wskazówek zegara. Prawidłowe wartości leżą w zakresie od 3600 do -3600.

Atrybut **ANGLE** (PROP:ANGLE) powoduje, że kontrolka STRING jest wyświetlana lub drukowana pod określonym kątem, liczonym w kierunku przeciwnym do ruchu wskazówek zegara od poziomu okna lub poziomej orientacji raportu (zarówno dla raportu Portrait, jak i Landscape). Umożliwia to wyświetlanie lub drukowanie tekstów pod dowolnym kątem względem standardowego poziomu. Czcionka użyta do wyświetlenia lub wydrukowania takiego tekstu musi być czcionką TrueType.

Przykład:

```
OknoPierwsze  WINDOW,AT(0,0,160,400),FONT('Arial')
          STRING('jakiś napis'),AT(6,40),USE(?String1)                    ! wyświetla tekst w poziomie
          STRING('jakiś napis'),AT(6,40),USE(?String2),ANGLE(900)         ! wyświetla tekst w pionie
          STRING('jakiś napis'),AT(6,40),USE(?String3),ANGLE(1800)        ! wyświetla tekst w układzie góra-dół
        END
MojRaport   REPORT,AT(1000,1000,6500,9000),THOUS,FONT('Arial',10)
Detal         DETAIL,AT(0,0,6500,1000)
            STRING('jakiś napis'),AT(500,500,1500,500)                    ! drukuje tekst w poziomie
            STRING('jakiś napis'),AT(500,500,1500,500),ANGLE(900)         ! drukuje tekst w pionie
            STRING('jakiś napis'),AT(500,500,1500,500),ANGLE(1800)        ! drukuje tekst w układzie góra-dół
          END
        END
```

AT (ustawienie pozycji i rozmiaru)

AT([*x*] [, *y*] [, *width*] [, *height*])

AT Definiuje pozycję i rozmiar danej struktury lub kontrolki.

x Stała całkowita lub wyrażenie stałe określające poziomą pozycję lewego, górnego narożnika (PROP:Xpos, równoważne z {PROP:At,1}). Jeśli parametr ten zostanie pominięty, biblioteka runtime przydziela domyślną wartość.

y Stała całkowita lub wyrażenie stałe określające pionową pozycje lewego, górnego narożnika (PROP:Ypos, równoważne z {PROP:At,2}). Jeśli parametr ten zostanie pominięty, biblioteka runtime przydziela domyślną wartość.

width Stała całkowita lub wyrażenie stałe określające szerokość (PROP:Width, równoważne z {PROP:At,3}). Jeśli parametr ten zostanie pominięty, biblioteka runtime przydziela domyślną wartość.

height Stała całkowita lub wyrażenie stałe określające wysokość (PROP:Height, równoważne z {PROP:At,4}). Jeśli parametr ten zostanie pominięty, biblioteka runtime przydziela domyślną wartość.

Atrybut **AT** (PROP:AT) definiuje pozycję i rozmiar określonej struktury bądź kontrolki. Pozycje *x,y* są względne i zależą od instrukcji, dla której został użyty atrybut AT.

Wartości umieszczone w parametrach *x, y, width* i *height* są określane w jednostkach dialogowych okna APPLICATION lub WINDOW. Wartości parametrów *x, y, width* i *height* w raportach REPORT pozbawionych atrybutów THOUS, MM lub POINTS również są określane w jednostkach dialogowych. Jednostka dialogowa jest zdefiniowana jako jedna czwarta średniej szerokości znaku i jedna ósma średniej wysokości znaku. Aktualny rozmiar jednostki dialogowej zależy od rozmiaru domyślnej czcionki okna lub raportu. Taki sposób wymiarowania bazuje na czcionce określonej przez atrybut FONT okna lub raportu bądź też, jeśli taka nie została zdefiniowana, na domyślnej czcionce systemu Windows.

Zastosowanie w oknach

Parametry *x* i *y* stanowią przesunięcie względem lewego górnego rogu okna, gdy atrybut AT jest włączony dla struktury okna aplikacji APPLICATION lub okna WINDOW (bez atrybutu MDI) – jeżeli to okno zostało otwarte przed otwarciem struktury APPLICATION programu.

Parametry *x* i *y* stanowią przesunięcie względem lewego górnego rogu obszaru roboczego okna APPLICATION, gdy atrybut AT jest umieszczony w definicji struktury WINDOW z atrybutem MDI lub struktury okna WINDOW bez atrybutu MDI otwartego po otwarciu okna aplikacji APPLICATION.

Parametry *width* oraz *height* definiują rozmiar obszaru roboczego okna aplikacji APPLICATION. Jest to obszar leżący pod paskiem menu MENUBAR i nad paskiem stanu. W obszarze tym jest umieszczany pasek narzędzi TOOLBAR oraz są otwierane okna wewnętrzne MDI. W oknie WINDOW obszar roboczy wyznacza obszar, w którym są umieszczane kontrolki.

Zastosowanie w kontrolkach okna

Parametry *x* i *y* oznaczają przesunięcie względem lewego górnego rogu obszaru roboczego okna aplikacji APPLICATION lub okna WINDOW.

Zastosowanie w strukturze raportu

Atrybut AT w strukturze REPORT definiuje pozycję i rozmiar obszaru, w którym będzie drukowany pasek (detail) zawierający dane rekordu. Jest to obszar, w którym mogą być drukowane wszystkie struktury DETAIL i dowolne struktury HEADER oraz FOOTER zawarte wewnątrz struktur BREAK.

Zastosowanie w strukturach raportu

Atrybut AT w strukturach raportu posiada dwa różne znaczenia, zależne od tego, przy jakiej strukturze został umieszczony. Gdy jest to struktura FORM lub też nagłówek HEADER lub stopka FOOTER strony (nie znajdujące się wewnątrz struktury BREAK), atrybut AT określa pozycję i rozmiar na stronie, na której dana struktura jest drukowana. Pozycja określona przez parametry *x* i *y* jest liczona względem górnego, lewego rogu strony. W sytuacjach, gdy atrybut AT jest umieszczany przy strukturze DETAIL lub przy nagłówku HEADER bądź stopce FOOTER będącymi elementem struktury BREAK, drukowanie danej struktury odbywa się zgodnie z przedstawionymi poniżej regułami (za wyjątkiem sytuacji, gdy występuje atrybut ABSOLUTE):

- Parametry *width* i *height* atrybutu AT określają minimalny rozmiar drukowanej struktury.
- Struktura jest drukowana w następnej dostępnej pozycji w ramach obszaru przeznaczonego dla detalu (określonego przez atrybut AT raportu REPORT).
- Pozycja określona przez parametry *x* i *y* atrybutu AT jest przesunięciem względem następnej dostępnej pozycji drukowania w ramach obszaru przeznaczonego na drukowanie detalu.
- Pierwsza struktura raportu jest drukowana na stronie w lewym, górnym rogu obszaru przeznaczonego na drukowanie detalu (z przesunięciem określonym przez atrybut AT).
- Następna i kolejne struktury raportu są drukowane w pozycjach uwzględniających końcową pozycję ostatnio wydrukowanej struktury raportu: jeśli jest miejsce – obok, jeśli nie – poniżej.

Zastosowanie w kontrolkach raportu

Parametry *x* i *y* stanowią relatywne przesunięcie względem lewego, górnego rogu struktury raportu zawierającej daną kontrolkę.

Przykład:

```
OknoPierwsze  WINDOW,AT(0,0,380,200),MDI     ! lewy górny róg, względem ramki aplikacji frame
        END
OknoDrugie    WINDOW,AT(0,0,380,200)          ! lewy górny róg, względem ekranu
        END

! układ miar w jednostkach dialogowych
OknoPierwsze  WINDOW,AT(0,0,160,400)
          ENTRY,AT(8,40,80,8)                 ! w przybliżeniu 2 w bok, 5 w dół, 20 szerokości, 1 wysokości
        END
MojRaport REPORT,AT(1000,1000,6500,9000),THOUS          ! AT określa obszar drukowania detalu
Detal       DETAIL,AT(0,0,6500,1000)                    ! AT określa rozmiar wstęgi i relatywne
                                                        ! przesunięcie pozycji względem ostatnio
                                                        ! drukowanego detalu
            STRING('jakiś napis'),AT(500,500,1500,500)
            ! AT określa rozmiar kontrolki i jej przesunięcie względem detalu
          END
        END
```

```
MojRaport REPORT,AT(1000,2000,6500,7000),THOUS          ! 1" marginesy wokół
          HEADER,AT(1000,1000,6500,1000)                ! względna pozycja strony
            ! elementy struktury
                                                        ! 1" od górnej krawędzi strony
          END
DetalPierwszy  DETAIL,AT(0,0,6500,1000)                 ! względna pozycja detalu
            ! elementy struktury
                                                        ! 1" wokół strony
          END
DetalDrugi  DETAIL,ABSOLUTE,AT(1000,8000,6500,1000)     ! względna pozycja strony
            ! elementy struktury
                                                        ! 1" wstęga przy dolnej krawędzi strony
          END
          FOOTER,AT(1000,9000,6500,1000)                ! względna pozycja strony
            ! elementy struktury
                                                        ! 1" wstęga przy dolnej krawędzi strony
          END
        END

Raport1 REPORT,AT(1000,1000,6500,9000),THOUS            ! 1" marginesy wokół strony
          ! obszar detalu w 8.5" x 11"
          ! deklaracje raportu
        END
Raport2 REPORT,AT(72,72,468,648),POINTS                 ! 1" marginesy wokół strony
          ! obszar detalu w 8.5" x 11"
          ! deklaracje raportu
        END
```

Porównaj: SETPOSITION, GETPOSITION

AUTO (automatyczne odświeżanie kontrolek)

AUTO

Atrybut **AUTO** (PROP:AUTO) powoduje, że zmienne występujące w polach USE wszystkich kontrolek okna i paska narzędzi są odświeżane na ekranie przy każdym przejściu pętli ACCEPT. Pociąga to za sobą nieznaczne spowolnienie, ale otrzymujemy pewność, że zawsze są wyświetlane aktualne wartości zmiennych i nie musimy dodatkowo pamiętać o stosowaniu instrukcji DISPLAY odświeżającej zawartość kontrolek.

Przykład:

```
OknoPierwsze  WINDOW,AT(,,380,200),MDI,CENTER,AUTO      ! wartości kontrolek są zawsze wyświetlane
                ! kontrolki
              END
  CODE
  ACCEPT                                    ! ACCEPT automatycznie odświeża zmienione zmienne USE
  END
```

AUTOSIZE (automatyczna zmiana rozmiaru obiektu OLE)

AUTOSIZE

Atrybut **AUTOSIZE** (PROP:AUTOSIZE, tylko-do-zapisu) powoduje automatyczną zmianę rozmiarów obiektu OLE w momencie, gdy atrybut AT kontenera OLE zmieni swoje parametry (w czasie działania programu, na przykład przez zmianę właściwości PROP:AT).

AVE (wyliczanie wartości średniej w raporcie)

AVE([*variable*])

AVE Wylicza średnią arytmetyczną dla kontrolek STRING w oparciu o wartości zmiennej występującej w polu USE tej kontrolki.

variable Etykieta zmiennej numerycznej, w której będą przechowywane wartości pośrednie wyliczane przez AVE. Umożliwia to tworzenie podliczeń i podliczeń częściowych. Wartość w zmiennej *variable* jest wewnętrznie aktualizowana przez mechanizm drukowania, z tego względu można stosować ten sposób tylko w strukturach raportu REPORT.

Atrybut **AVE** (PROP:AVE) powoduje drukowanie średniej arytmetycznej wartości zmiennej występującej w polu USE kontrolki typu STRING raportu. O ile nie występuje atrybut TALLY, rezultat jest wyliczany zgodnie z poniższymi zasadami:

- Pole AVE znajdujące się w strukturze DETAIL jest wyliczane za każdym razem, gdy struktura DETAIL zawierająca kontrolkę jest drukowana.
- Pole AVE znajdujące się w stopce FOOTER grupy jest wyliczane za każdym razem, gdy jest drukowana dowolna struktura DETAIL zawarta w strukturze BREAK zawierającej kontrolkę.
- Pole AVE znajdujące się w stopce FOOTER strony jest wyliczane za każdym razem, gdy jest drukowana dowolna struktura DETAIL w dowolnej strukturze BREAK.
- Pole AVE znajdujące się w nagłówku HEADER nie ma sensu, gdyż żaden detal nie jest drukowany w czasie drukowania nagłówka.

Średnia jest resetowana wtedy, gdy są określone dla niej atrybuty RESET lub PAGE. Kontrolka STRING z omawianym atrybutem powinna się zazwyczaj znajdować w stopce FOOTER grupy lub strony.

Przykład:

```
MojRaport  REPORT,AT(1000,1000,6500,9000),THOUS
Grupa1       BREAK(LocalVar),USE(?BreakOne)
Grupa2         BREAK(Pre:Key1),USE(?BreakTwo)
Detal            DETAIL,AT(0,0,6500,1000),USE(?DetailOne)
                   STRING(@N$11.2),AT(6000,1500,500,500),USE(Pre:F1)
                 END
                 FOOTER,AT(0,0,6500,1000),USE(?BreakOneGroupFoot)
                   STRING('Group Average:'),AT(5500,500)
                   STRING(@N$11.2),AT(6000,500),USE(Pre:F1),AVE(LocalVar),RESET(Grupa2)
                 END
               END
               FOOTER,AT(0,0,6500,1000),USE(?BreakOneGroupFoot)
                 STRING('Grand Average:'),AT(5500,500)
                 STRING(@N$11.2),AT(6000,500),USE(LocalVar),AVE,TALLY(?BreakTwo)
               END
             END
           END
```

BEVEL (włączenie efektów 3-D dla ramki kontrolki)

BEVEL(*outer* [, *inner*] [, *style*])

BEVEL Określa efekt 3- D dla ramki kontrolki.

outer Stała całkowita lub wyrażenie stałe określające szerokość zewnętrznej krawędzi obramowania (PROP:BevelOuter, równoważne z {PROP:Bevel,1}). Jeśli jest to wartość ujemna, zewnętrzna krawędź pojawia się jako obniżona, jeśli dodatnia- jako podniesiona.

inner Stała całkowita lub wyrażenie stałe określające szerokość wewnętrznej krawędzi obramowania (PROP:BevelInner, równoważne z {PROP:Bevel,2}). Jeśli jest to wartość ujemna, wewnętrzna krawędź pojawia się jako obniżona, jeśli dodatnia- jako podniesiona. Pominięcie tego parametru powoduje, że wewnętrzna krawędź nie występuje.

style Stała całkowita lub wyrażenie stałe określające wygląd ramki kontrolki, przykrywając ustawienia wynikające z parametrów *outer* i *inner* (PROP:BevelStyle, równoważne z {PROP:Bevel,3}).

Atrybut **BEVEL** (PROP:BEVEL) określony dla kontrolki PANEL, OPTION, GROUP lub REGION nadaje jej ramce efekt trójwymiarowości (3-D). Znak wartości parametrów *outer* oraz *inner* pociąga za sobą odpowiednie podniesienie lub obniżenie krawędzi ramki. Parametr *style* umożliwia wybranie stylu dla ramki Parametr ten jest mapą bitową, w której poszczególne grupy bitów odnoszą się do krawędzi ramki w sposób następujący:

Bity:	15 – 12	11 - 08	07 - 04	03 - 00
Krawędzie:	lewa	górna	prawa	dolna

Każda grupa czterobitowa jest następnie dzielona dwie sekcje dwubitowe odpowiadające części zewnętrznej i wewnętrznej odpowiedniej krawędzi. Mniej znaczące bity dotyczą części zewnętrznej, bardziej znaczące – zewnętrznej.

Wartość binarna:	00b	01b	10b	11b
Rezultat:	brak krawędzi	podniesiona	obniżona	szara

Łącząc ze sobą odpowiednie wartości w grupy czterobitowe otrzymujemy odpowiedni efekt, np.

0110b = podniesiona wewnętrzna, obniżona zewnętrzna

1001b = obniżona wewnętrzna, podniesiona zewnętrzna

Przykład:

```
Win1    WINDOW,AT(0,0,160,400)
          PANEL,AT(25,15,50,50),USE(?Panel1),BEVEL(5,-5) ! podniesiona zewnętrzna, obniżona wewnętrzna
          PANEL,AT(0,0,,),USE(?Panel2),FULL,BEVEL(2,2,1111010110101001b)
            ! left = wszystko szare
            ! top = wewnętrzna podniesiona, zewnętrzna podniesiona
            ! right = wewnętrzna obniżona, zewnętrzna obniżona
            ! bottom = wewnętrzna obniżona, zewnętrzna podniesiona
          REGION,AT(0,80,5,),USE(?ResizeBar),FULL,IMM,BEVEL(2,2,0101000010100000b)
            ! pionowy pasek zmiany rozmiaru
        END
```

BOXED (ustawienie ramki dla grupy kontrolek)

BOXED

Atrybut **BOXED** (PROP:BOXED) powoduje rysowanie pojedynczej ramki wokół kontrolki GROUP lub OPTION. Parametr *text* kontrolki GROUP lub OPTION pojawia się w górnej krawędzi ramki . Jeśli atrybut BOXED jest pominięty, parametr *text* nie jest wyświetlany (drukowany).

CAP, UPR (ustawienie wielkości liter)

CAP
UPR

Atrybuty **CAP** oraz **UPR** powodują automatyczną zmianę wielkości liter przy wprowadzaniu ich do pola tekstowego (kontrolki ENTRY i TEXT); warunkiem jest posiadanie atrybutu MASK przez okno lub kontrolkę TEXT.

Atrybut UPR (PROP:UPR) powoduje, że wszystkie litery są zamieniane na wielkie. Atrybut CAP (PROP:CAP) powoduje, że teksty są pisane kapitalikami; pierwsza litera każdego wyrazu jest automatycznie zamieniana na wielką, a wszystkie pozostałe są małe.

Użytkownik może wymusić zmianę powyższych ustawień wciskając klawisz SHIFT wraz z odpowiednim klawiszem literowym, by móc wprowadzić np. „McDonald's" lub klawisz SHIFT przy aktywnym klawiszu CAPS-LOCK, by wymusić wprowadzenie pierwszej małej litery, np. dla „von Richtofen".

CENTER (centrowanie pozycji okna)

CENTER

Atrybut **CENTER** (PROP:CENTER) powoduje, że domyślna pozycja okna jest centrowana. Okno WINDOW z atrybutem MDI jest centrowane względem okna aplikacji APPLICATION. Okno aplikacji APPLICATION jest centrowane względem pulpitu Windows. Okna WINDOW bez atrybutu MDI są centrowane względem ich okien nadrzędnych (z poziomu których zostały otwarte).

Atrybut ten nie ma żadnego znaczenia dopóty, dopóki nie zostanie pominięty chociaż jeden parametr atrybutu AT. Oznacza to, że atrybut CENTER określa domyślne wartości dla pominiętych parametrów atrybutu AT.

Przykład:

```
OknoPierwsze  WINDOW,AT(,,380,200),MDI,CENTER   ! okno centrowane względem ramki aplikacji
              END
OknoDrugie    WINDOW,AT(,,380,200),CENTER       ! ! okno centrowane względem okna nadrzędnego
              END
```

CENTERED (ustawienie centrowania grafiki)

CENTERED

Atrybut **CENTERED** (PROP:CENTERED) powoduje, że grafika jest wyświetlana w swoim oryginalnym rozmiarze i jest środkowana w obszarze wyświetlania:

- W kontrolce IMAGE grafika jest centrowana w obszarze wytyczonym przez parametry atrybutu AT.
- W pasku narzędzi TOOLBAR posiadającym atrybut WALLPAPER, grafika stanowiąca tło paska narzędzi jest w nim centrowana.
- W oknie APPLICATION lub WINDOW posiadającym atrybut WALLPAPER, grafika stanowiąca tło okna jest centrowana w jego obszarze.

Przykład:

```
MDIChild    WINDOW('Child One'),MDI,SYSTEM,MAX
              MENUBAR
                MENU('Edit'),USE(?EditMenu)
                  ITEM('Undo'),USE(?UndoText),KEY(CtrlZ),STD(STD:Undo)
                  ITEM('Cu&t'),USE(?CutText),KEY(CtrlX),STD(STD:Cut)
                  ITEM('Copy'),USE(?CopyText),KEY(CtrlC),STD(STD:Copy)
                  ITEM('Paste'),USE(?PasteText),KEY(CtrlV),STD(STD:Paste)
                END
              END
              TOOLBAR,USE(?Toolbar),WALLPAPER('MyWall.GIF'),CENTERED
                BUTTON('Cut'),USE(?CutButton),ICON(ICON:Cut),STD(STD:Cut),FLAT
                BUTTON('Copy'),USE(?CopyButton),ICON(ICON:Copy),STD(STD:Copy),FLAT
                BUTTON('Paste'),USE(?PasteButton),ICON(ICON:Paste),STD(STD:Paste),FLAT
              END
            END

OknoPierwsze  WINDOW,AT(,,380,200),MDI,WALLPAPER('MyWall.GIF'),CENTERED
              END

OknoPierwsze  WINDOW,AT(,,380,200),MDI
                IMAGE('MyWall.GIF'),AT(0,0,380,200),CENTERED
              END
```

Porównaj: WALLPAPER, TILED

CHECK (ustawienie elementu przełącznikowego)

CHECK

Atrybut **CHECK** (PROP:CHECK) oznacza, że element ITEM może być włączony (ON) lub wyłączony (OFF). Gdy jest włączony, w menu, po lewej stronie elementu pojawia się znak zaznaczenia, a zmienna wskazywana przez atrybut USE otrzymuje wartość (1). Gdy jest wyłączony, znak zaznaczenia znika, a zmienna wskazywana przez atrybut USE otrzymuje wartość (0).

CLASS (ustawienie własnej klasy kontrolki .VBX)

CLASS(*file* [, *name*])

CLASS Wskazuje nazwę pliku i typ kontrolki .VBX.

file Stała łańcuchowa zawierająca nazwę pliku .VBX (włączając w to rozszerzenie .VBX) zawierającego implementację kontrolki (PROP:VbxFile, równoważny z {PROP:Class,1}).

name Stała łańcuchowa zawierająca nazwę typu kontrolki z pliku .VBX (PROP:VbxName, równoważny z {PROP:Class,2}). Jeśli parametr ten zostanie pominięty, wykorzystywany jest pierwszy typ kontrolki zdefiniowany w pliku .VBX.

Atrybut **CLASS** (PROP:CLASS) określa nazwę pliku i typ kontrolki .VBX. Parametr *name* identyfikuje konkretną kontrolkę w pliku .VBX zawierającym definicje wielu kontrolek.

Przykład:

```
OknoPierwsze  WINDOW,AT(0,0,160,400)
                CUSTOM,AT(0,0,120,320),CLASS('graph.vbx','graph'),'graphstyle'('2')
              END
```

CLIP (ustawienie trybu obcinania obiektu OLE)

CLIP

Atrybut **CLIP** (PROP:CLIP, tylko-do-zapisu) powoduje, że w kontenerze OLE, w obszarze wytyczonym przez jego atrybut AT jest wyświetlana tylko ta część obiektu OLE, która się w nim mieści. Jeśli rozmiar obiektu jest większy, niż rozmiar kontenera, wyświetlana jest jego lewa, górna część.

CNT (ustawienie licznika w raporcie)

CNT([*variable*])

CNT Oblicza, ile razy była drukowana struktura DETAIL.

variable Etykieta zmiennej numerycznej, w której będą przechowywane wartości pośrednie wyliczane przez CNT. Umożliwia to tworzenie podliczeń i podliczeń częściowych. Wartość w zmiennej *variable* jest wewnętrznie aktualizowana przez mechanizm drukowania, z tego względu można stosować ten sposób tylko w strukturach raportu REPORT.

Atrybut **CNT** (PROP:CNT) pozwala na automatyczne zliczanie, ile razy była drukowana określona struktura DETAIL. O ile nie występuje atrybut TALLY, obowiązują następujące zasady zliczania:

- Pole CNT w strukturze DETAIL zwiększa o 1 swoją wartość, gdy jest drukowana struktura DETAIL, w której znajduje się kontrolka reprezentująca zmienną *variable*. W ten sposób możemy tworzyć liczniki.
- Pole CNT w stopce FOOTER grupy zwiększa o 1 swoją wartość, gdy jest drukowana dowolna struktura DETAIL struktury BREAK zawierającej kontrolkę reprezentującą zmienną *variable*. W ten sposób możemy podliczać (podsumowywać) ilość detali w grupie.
- Pole CNT w stopce FOOTER strony zwiększa o 1 swoją wartość, gdy jest drukowana dowolna struktura DETAIL zawarta w strukturze BREAK. W ten sposób możemy podliczać liczbę detali drukowanych na stronie.
- Pole CNT w nagłówku HEADER nie ma sensu, gdyż żaden detal nie jest drukowany w czasie drukowania nagłówka.

Atrybut CNT jest resetowany wtedy, gdy występują z nim dodatkowo atrybuty RESET lub PAGE.

Przykład:

```
MojRaport  REPORT,AT(1000,1000,6500,9000),THOUS
Grupa1       BREAK(LocalVar),USE(?BreakOne)
Grupa2         BREAK(Pre:Key1),USE(?BreakTwo)
Detail           DETAIL,AT(0,0,6500,1000),USE(?DetailOne)
                   STRING(@N$11.2),AT(6000,1500,500,500),USE(Pre:F1)
                 END
                 FOOTER,AT(0,0,6500,1000),USE(?BreakOneGroupFoot)
                   STRING('Group Count:'),AT(5500,500)
                   STRING(@N$11.2),AT(6000,500),USE(Pre:F1),CNT(LocalVar),RESET(Grupa2)
                 END
               END
               FOOTER,AT(0,0,6500,1000),USE(?BreakOneGroupFoot)
                 STRING('Grand Count:'),AT(5500,500)
                 STRING(@N$11.2),AT(6000,500),USE(LocalVar),CNT,TALLY(?BreakTwo)
               END
             END
           END
```

COLOR (ustawienie koloru)

COLOR(*color* [, *selected fore*] [, *selected back*])

COLOR	Definiuje kolor wyświetlania lub drukowania.
color	Określa kolor tła (PROP:Background lub PROP:FillColor, równoważne z {PROP:Color,1}). Kolor pisania jest określany przez atrybut FONT.
selected fore	Określa domyślny kolor pisania dla wyróżnionego tekstu kontrolki, która może otrzymywać aktywność wprowadzania (PROP:SelectedColor, równoważne z {PROP:Color,2}). Nie dotyczy raportu.
selected back	Określa domyślny kolor tła dla wyróżnionego tekstu kontrolki, która może otrzymywać aktywność wprowadzania (PROP:SelectedFillColor, równoważne z {PROP:Color,3}). Nie dotyczy raportu.

Atrybut **COLOR** (PROP:COLOR) określa domyślny kolor tła oraz kolor tła i pisania wyróżnionego tekstu. Wartości kolorów każdego z trzech atrybutów są stałymi zbudowanymi ze składników kolorów: czerwonego, zielonego i niebieskiego. Kolor jest przechowywany w trzech mniej znaczących bajtach wartości typu LONG (bajty 0, 1 oraz 2, gdzie Czerwony (Red) = 000000FFh, Zielony (Green) = 0000FF00h i Niebieski (Blue) = 00FF0000h). Kolor może też być zdefiniowany w oparciu ekwiwalenty EQUATE dla standardowych wartości kolorów Windows (wszystkie są liczbami ujemnymi). Ekwiwalenty EQUATE dla standardowych kolorów Windows są zapisane w pliku EQUATES.CLW. Każda z właściwości daje w rezultacie COLOR:None jeżeli przypisany je parametr nie występuje.

Windows automatycznie znajduje najbardziej zbliżony kolor do określonego, wynika to zawsze z konfiguracji sprzętu, na którym pracuje program. Standardowe kolory Windows mogą być rekonfigurowane przez użytkownika za pośrednictwem Panelu sterowania. Dowolna kontrolka korzystająca ze standardowych kolorów Windows jest automatycznie odświeżana w oparciu o nowy kolor, gdy taka zmiana nastąpi.

Zastosowanie w oknach i paskach narzędzi

W oknie WINDOW lub w pasku narzędzi TOOLBAR atrybut COLOR oznacza kolor tła okna (paska narzędzi) oraz domyślne kolory tła, rysowania i wyróżnienia dla wszystkich ich kontrolek; za wyjątkiem tych, dla których zostały określone atrybuty COLOR.

Zastosowanie w kontrolkach okna

Atrybut COLOR określa kolor wyświetlania kontrolki LINE. W kontrolkach BOX, ELLIPSE oraz REGION parametr *color* określa kolor wykorzystywany przy rysowaniu ich obramowania. We wszystkich pozostałych kontrolkach parametr *color* definiuje kolor tła „przykrywając" ustawienia wynikające z wybranego przez użytkownika schematu kolorów Windows. W przypadku większości z tych kontrolek, które mogą posiadać aktywność wprowadzania, parametry *selected fore* oraz *selected back* oznaczają odpowiednio kolor rysowania i kolor tła dla zaznaczonych (wyróżnionych) tekstów lub elementów.

Zastosowanie w raportach

W instrukcji REPORT atrybut COLOR określa kolor wypełnienia tła raportu oraz domyślny kolor tła dla wszystkich elementów raportu: DETAIL, HEADER, FOOTER, FORM nie posiadających atrybutu COLOR.

Atrybut COLOR określa kolor wypełnienia tła elementów DETAIL, HEADER, FOOTER, FORM, przy których jest umieszczony oraz domyślny kolor tła dla wszystkich kontrolek wymienionych struktur nie posiadających atrybutu COLOR.

Atrybut kolor określa kolor drukowania kontrolki LINE, kolor obramowania kontrolek BOX oraz ELLIPSE lub kolor tła w przypadku kontrolek pozostałych typów.

Przykład:

```
OknoPierwsze  WINDOW,AT(0,0,160,400),COLOR(00FF0000h,0000FF00h,000000FFh)
                TOOLBAR,COLOR(00FF0000h,0000FF00h,000000FFh)
                  BOX,AT(20,20,20,20),COLOR(COLOR:ACTIVEBORDER)
                END
                BOX,AT(100,100,20,20),COLOR(00FF0000h)                  ! niebieski
                BOX,AT(140,140,20,20),COLOR(0000FF00h)                  ! zielony
                BOX,AT(180,180,20,20),COLOR(000000FFh)                  ! czerwony
              END

MojRaport     REPORT,AT(1000,1000,6500,9000),THOUS,COLOR(00FF0000h)     ! niebieskie tło
CustDetail      DETAIL,AT(0,0,6500,1000)
                  ELLIPSE,AT(60,60,200,200),COLOR(COLOR:ACTIVEBORDER)   ! EQUATE koloru
                  BOX,AT(360,60,200,200),COLOR(00FF0000h)
                END
              END

RptOne        REPORT,AT(0,0,160,400),COLOR(00FF0000h)
                HEADER,COLOR(0000FF00h)
                  ! elementy struktury
                END
DetalPierwszy   DETAIL                                                  ! domyślny kolor tła
                  ! elementy struktury
                END
                FOOTER,COLOR(000000FFh)
                  ! elementy struktury
                END
              END
```

Porównaj: FONT

COLUMN (ustawienie podświetlania w liście)

COLUMN

Atrybut **COLUMN** (PROP:COLUMN) powoduje, że w wielokolumnowej liście LIST lub COMBO podświetlany jest nie cały wiersza, a pojedyncza komórka. PROP:COLUMN daje w rezultacie (0) jeśli atrybut ten jest wyłączony, jeśli nie – rezultatem jest numer kolumny, w której znajduje się podświetlona aktualnie komórka.

COMPATIBILITY (ustawienie kompatybilności kontrolki OLE)

COMPATIBILITY(*mode*)

COMPATIBILITY Określa ustawienia kompatybilności kontrolki OLE.

mode Stała całkowita wykorzystywana do ustawienia kompatybilności.

Atrybut **COMPATIBILITY** (PROP:COMPATIBILITY, tylko-do-zapisu) określa tryb kompatybilności dla obiektów OLE lub .OCX tego wymagających. Parametr *mode* generalnie powinien mieć wartość (0), jednakże niektóre obiekty OLE (na przykład edytor map bitowych Windows) nie będą pracowały, jeśli nie będzie on ustawiony na wartość (1).

Przykład:

```
OknoPierwsze  WINDOW,AT(0,0,200,200)
                OLE,AT(10,10,160,100),USE(?OLEObject),CREATE('Excel.Sheet.5'),COMPATIBILITY(0)
                END
              END
```

CREATE (utworzenie obiektu kontrolki OLE)

CREATE(*server* [, *object*])

CREATE Powoduje utworzenie nowego obiektu dla kontrolki OLE.

server Stała łańcuchowa zawierająca nazwę aplikacji-serwera OLE, w takiej postaci, w jakiej jest wyszczególniona w rejestrze systemowym.

object Stała łańcuchowa zawierająca nazwę pliku OLE Compound Storage i otwieranego z niego obiektu.

Atrybut **CREATE** (PROP:CREATE, tylko-do-zapisu) powoduje utworzenie nowego obiektu OLE lub .OCX dla kontrolki OLE. Wartość *server* jest nazwą obiektu w takiej postaci, w jakiej jest ona zapisana w rejestrze systemu. W Windows 95/98/NT rejestr możemy edytować za pomocą programu REGEDIT.EXE. Nazwy znajdziemy w gałęzi HKEY_CLASSES_ROOT. Wpisy rejestru możemy także sprawdzić za pomocą programu Microsoft System Information (MSINFO32.EXE) wchodzącego w skład pakietu Microsoft Office.

Gdy parametr *object* występuje, CREATE wykonuje działania podobne, jak w przypadku atrybutu OPEN, otwierając w kontrolce OLE *object* zapisany w pliku OLE Compound Storage (ignorując parametr *server*). Gdy obiekt jest już otwarty, jest ładowana zachowana wersja właściwości kontenera, z tego względu nie musimy wcześniej ich określać. Składnia parametru *object* musi zachowywać formę: *NazwaPliku\!NazwaObiektu*.

Przykład:

```
OknoPierwsze  WINDOW,AT(0,0,200,200)
                OLE,AT(10,10,160,100),USE(?OLEObject),CREATE('Excel.Sheet.5')
              END
            END
```

CURSOR (ustawienie typu kursora myszki)

CURSOR(*file*)

CURSOR Określa wyświetlany kursor.

file Stała łańcuchowa zawierająca nazwę pliku .CUR (z definicją kursora) lub ekwiwalent wskazujący jeden ze standardowych kursorów Windows.

Plik .CUR jest linkowany do pliku .EXE jako zasób.

Atrybut CURSOR (PROP:CURSOR) wskazuje kursor wyświetlany w momencie, gdy wskaźnik myszki znajdzie się w obszarze aplikacji APPLICATION, okna WINDOW, paska narzędzi TOOLBAR lub kontrolki. Ten kursor jest dziedziczony przez wszystkie kontrolki w aplikacji APPLICATION, oknie WINDOW lub w pasku narzędzi TOOLBAR, chyba że definiują one własne za pomocą omawianego atrybutu. Windows 3.1 potrafi obsługiwać tylko kursory monochromatyczne (326-bajtowe pliki .CUR).

Ekwiwalenty EQUATE dla standardowych kursorów Windows są umieszczone w pliku EQUATES.CLW. Poniższa lista prezentuje przykładowe z nich:

CURSOR:None	Brak kursora
CURSOR:Arrow	Typowy kursor Windows w postaci strzałki
CURSOR:IBeam	Kursor w postaci dużej litery I
CURSOR:Wait	Kursor w kształcie klepsydry
CURSOR:Cross	Kursor w kształcie dużego plusa

Przykład:

```
! okno z własnym kursorem
OknoDrugie   WINDOW,CURSOR('CUSTOM.CUR')
               TOOLBAR,CURSOR('CURSOR:Cross')                ! pasek narzędzi z dużym
                                                             ! kursorem w kształcie znaku +

               BUTTON('Cut'),USE(?CutButton),STD(STD:Cut)
               BUTTON('Copy'),USE(?CopyButton),STD(STD:Copy)
               BUTTON('Paste'),USE(?PasteButton),STD(STD:Paste)
             END
             REGION,AT(20,20,20,20),CUSOR(CURSOR:IBeam)      ! region z kursorem I-beam
             REGION,AT(100,100,20,20)
           END
```

DEFAULT (wskazanie przycisku związanego z klawiszem Enter)

DEFAULT

Atrybut **DEFAULT** (PROP:DEFAULT) określa przycisk jako domyślnie wciskany w momencie, gdy użytkownik wciśnie klawisz ENTER. W oknie tylko jeden z aktywnych przycisków może posiadać ten atrybut.

DELAY (opóźnienie powtórzenia przycisku)

DELAY(*time*)

DELAY Określa czas pomiędzy pierwszym i drugim wygenerowaniem zdarzenia.

time Stała całkowita oznaczająca odstęp czasowy określony w setnych sekundy.

Atrybut **DELAY** (PROP:DELAY) oznacza odstęp czasowy pomiędzy pierwszym i drugim wygenerowanym zdarzeniem dla automatycznie powtarzanych przycisków. W przypadku przycisku BUTTON z atrybutem IMM, jest to czas pomiędzy pierwszym i drugim zdarzeniem EVENT:Accepted. W przypadku kontrolek typu SPIN, jest to czas pomiędzy pierwszym i drugim zdarzeniem EVENT:NewSelection wygenerowanym przez przycisk typu SPIN.

Cel stosowania atrybutu DELAY polega na zmianie czasu opóźnienia z wartości domyślnej, tak by użytkownik nie wykonał, w sposób dla siebie nieoczekiwany, akcji która nie leżała w jego zamiarze.

Przypisanie wartości zerowej do właściwości PROP:DELAY przywraca ustawienia domyślne.

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            BUTTON('Press Me'),AT(10,10,40,20),USE(?PressMe),IMM,DELAY(100) !1 second
            SPIN(@n3),AT(60,10,40,10),USE(SpinVar),RANGE(0,999),DELAY(100) !1 second
          END
  CODE
  OPEN(MDIChild)
  ?PressMe{PROP:Delay} = 50       ! ustawienie opóźnienia na 1/2 sekundy
  ?SpinVar{PROP:Delay} = 50       ! ustawienie opóźnienia na 1/2 sekundy
  ?PressMe{PROP:Repeat} = 5       ! ustawienie powtórzenia na 5 setnych sekundy
  ?SpinVar{PROP:Repeat} = 5       ! ustawienie powtórzenia na 5 setnych sekundy
```

Porównaj: IMM, REPEAT

DISABLE (ustawienie kontrolki jako niedostępnej)

DISABLE

Atrybut **DISABLE** (PROP:DISABLE) powoduje, że dana kontrolka staje się niedostępna (wyszarzona), gdy zostanie otwarte okno WINDOW lub okno aplikacji APPLICATION.

Niedostępne kontrolki możemy uaktywniać za pomocą instrukcji ENABLE.

DOCK (umożliwienie dokowania okna)

DOCK(*positions*)

DOCK Powoduje, że okienko staje się okienkiem narzędziowym, które może być dokowane do krawędzi okna głównego aplikacji.

positions Maska bitowa określająca, do których krawędzi okna głównego może być dokowane dane okno.

Atrybut **DOCK** (PROP:DOCK) umieszczony w definicji okna WINDOW posiadającego atrybut TOOLBOX powoduje, że może ono być dokowane do wybranych krawędzi głównego okna aplikacji.

Poniżej zostały przedstawione ekwiwalenty (EQUATE) dla standardowych wartości *positions* (zostały one zdefiniowane w pliku EQUATES.CLW):

DOCK:Left	EQUATE(1)
DOCK:Top	EQUATE(2)
DOCK:Right	EQUATE(4)
DOCK:Bottom	EQUATE(8)
DOCK:Float	EQUATE(16)
DOCK:All	EQUATE(31)

Przykład:

```
Win1  WINDOW('Tools'),TOOLBOX,DOCK(DOCK:Left+DOCK:Right)    ! dokowalne tylko do lewej
                                                            ! i prawej krawędzi
        BUTTON('Date'),USE(?Button1)
        BUTTON('Time'),USE(?Button2)
      END
```

Porównaj: DOCKED,TOOLBOX

DOCKED (dokowanie okna typu toolbox przy otwarciu)

DOCKED(*position*)

DOCKED Powoduje, że dokowalne okno z atrybutem TOOLBOX jest dokowane od razu po otwarciu.

position Maska bitowa określająca, do której krawędzi okna głównego ma być zadokowane dane okno.

Atrybut **DOCKED** (PROP:DOCKED) powoduje, że okno WINDOW posiadające atrybut DOCK jest dokowane po jego otwarciu.

Poniżej zostały przedstawione ekwiwalenty (EQUATE) dla standardowych wartości *positions* (zostały one zdefiniowane w pliku EQUATES.CLW):

DOCK:Left	EQUATE(1)
DOCK:Top	EQUATE(2)
DOCK:Right	EQUATE(4)
DOCK:Bottom	EQUATE(8)
DOCK:Float	EQUATE(16)
DOCK:All	EQUATE(31)

Przykład:

```
Win1  WINDOW('Tools'),TOOLBOX,DOCK(DOCK:All),DOCKED(DOCK:Top)  ! dokowalne wszędzie,
                                                               ! zadokowane
        BUTTON('Date'),USE(?Button1)                           ! do góry przy otwarciu
        BUTTON('Time'),USE(?Button2)
      END
```

Porównaj: DOCK, TOOLBOX

DOCUMENT (tworzenie kontrolki OLE z pliku)

DOCUMENT(*filename*)

DOCUMENT Określa tworzenie obiektu dla kontrolki OLE w oparciu o plik danych specyficzny dla aplikacji będącej serwerem OLE.

filename Stała łańcuchowa zawierająca nazwę pliku.

Atrybut **DOCUMENT** (PROP:DOCUMENT, tylko-do-odczytu) powoduje utworzenie obiektu dla kontrolki OLE w oparciu o plik danych specyficzny dla aplikacji będącej serwerem OLE. Parametr *filename* musi być pełną ścieżką dostępu do tego pliku, o ile plik ten nie występuje w tym samym katalogu, co aplikacja sterująca OLE.

Przykład:

```
OknoPierwsze  WINDOW,AT(0,0,200,200)
                OLE,AT(10,10,160,100),USE(?OLEObject),DOCUMENT('Book1.XLS')  ! arkusz Excel
                  MENUBAR
                    MENU('&Clarion App')
                      ITEM('&Deactivate Object'),USE(?DeactOLE)
                    END
                  END
                END
              END
```

DOUBLE, NOFRAME, RESIZE (określenie ramki okna)

DOUBLE
NOFRAME
RESIZE

Atrybuty DOUBLE, NOFRAME oraz RESIZE określają wygląd i zachowanie ramki okna WINDOW lub APPLICATION, inne niż w przypadku domyślnej ramki pojedynczej (Single).

Atrybut DOUBLE (PROP:DOUBLE) powoduje utworzenie ramki podwójnej, atrybut NOFRAME w ogóle usuwa ramkę. Okno posiadające ramki tych typów nie może zmieniać swego rozmiaru.

Atrybut RESIZE (PROP:RESIZE) powoduje umieszczenie wokół okna ramki pogrubionej. Jest to jedyny typ ramki umożliwiający użytkownikowi dynamiczną zmianę rozmiaru okna. Atrybut RESIZE jest ignorowany w oknach WINDOW posiadających atrybut MODAL. Typ ramki RESIZE jest zazwyczaj stosowany w oknach aplikacji APPLICATION oraz w oknach WINDOW przeznaczonych na przykład do wyświetlania dokumentów. Nie stosuje się go w okienkach dialogowych.

Atrybut NOFRAME jest na ogół stosowany w „ukrytych" oknach służących jedynie do aktywacji pętli ACCEPT.

DOUBLE jest standardowym typem ramki dla okienek dialogowych

Przykład:

```
! okno z ramką pojedynczą:
Win1   WINDOW
       END

! okno o zmiennym rozmiarze:
Win2   WINDOW,RESIZE
       END

! okno z ramką podwójną:
Win3   WINDOW,DOUBLE
       END

! okno bez ramki:
Win4   WINDOW,NOFRAME
       END
```

DRAGID (ustawienie sygnatur hosta drag-and-drop)

DRAGID(*signature* [, *signature*])

DRAGID Określa kontrolkę typu LIST lub REGION służącą za źródło (host) operacji drag-and-drop.

signature Stała łańcuchowa zawierająca identyfikator służący do określania, czy kontrolka może być celem (target) operacji drag-and-drop. Dowolna sygnatura *signature* zaczynająca się znakiem tyldy (~) oznacza, że informacje mogą być przenoszone do programów zewnętrznych (nie napisanych w Clarionie). Pojedynczy DRAGID może zawierać do 16 sygnatur *signatures*.

Atrybut **DRAGID** (PROP:DRAGID, tablica) określa kontrolkę typu LIST lub REGION służącą za źródło (host) operacji drag-and-drop. Atrybut DRAGID działa w powiązaniu z atrybutem DROPID. Sygnatury *signature* atrybutu DRAGID (maksymalnie 16) definiują klucze walidacyjne porównywane z sygnaturami DROPID kontrolki, która ma być celem operacji drag-and-drop. W ten sposób możemy kontrolować, czy przenoszenie informacji pomiędzy określonymi kontrolkami jest dozwolone.

Operacja drag-and-drop występuje wtedy, gdy użytkownik „przeciąga" myszką informacje z kontrolki posiadającej atrybut DRAGID do kontrolki posiadającej atrybut DROPID. By operacja ta zakończyła się powodzeniem, obie kontrolki muszą posiadać przynajmniej jedną identyczną sygnaturę określoną odpowiednio w atrybutach DRAGID (host) i DROPID (target).

Przykład:

```
OknoPierwsze  WINDOW,AT(0,0,160,400)
              LIST,AT(120,0,20,20),USE(?List1),FROM(Que1),DRAGID('FromList1')
              ! umożliwia przeciąganie, ale nie upuszczanie
              LIST,AT(120,120,20,20),USE(?List2),FROM(Que2),DROPID('FromList1')
              ! umożliwia upuszczanie z List1, ale nie przeciąganie
              END
  CODE
  OPEN(OknoPierwsze)
  ACCEPT
    CASE EVENT()
    OF EVENT:Drag          ! zgłoszone zdarzenie związane z przeciąganiem
      IF DRAGID()          ! sprawdzenie, czy pomyślne
      SETDROPID(Que1)      ! przygotowanie informacji do przekazania
    END
    OF EVENT:Drop          ! gdy zdarzenie związane z upuszczeniem jest pomyślne
      Que2 = DROPID()      ! pobranie informacji
      ADD(Que2)            ! i dodanie jej do kolejki
    END
  END
```

Porównaj: DROPID

DROP (zdefiniowanie działania listy elementów)

DROP(*count* [, *width*])

DROP Powoduje, że lista pojawia się tylko wtedy, gdy użytkownik wciśnie klawisz strzałki w dół lub gdy kliknie ikonę strzałki umieszczoną w prawej części pola.

count Stała całkowita określająca liczbę wyświetlanych elementów.

width Stała całkowita określająca szerokość listy rozwijalnej, w jednostkach dialogowych (PROP:DropWidth, analogiczne do {PROP:DROP,2}).

Atrybut **DROP** (PROP:DROP) określa, że lista elementów pojawia się tylko wtedy, gdy użytkownik naciśnie klawisz strzałki w dół lub kliknie myszką ikonę strzałki znajdującą się w prawej części listy rozwijalnej. Gdy lista zostanie rozwinięta, pojawia się w niej *count* elementów. W przypadku, gdy atrybut DROP jest pominięty, kontrolka LIST lub COMBO zawsze wyświetla listę elementów, ich liczba jest wówczas określona przez parametr *height* atrybutu AT danej kontrolki.

Atrybut DROP nie działa w połączeniu z oknem WINDOW posiadającym atrybut MODAL i nie należy go w takich przypadkach stosować. Kontrolka posiada właściwość PROP:Icon, której możemy przypisać nazwę pliku zawierającego ikonę. Ikona ta będzie wyświetlana zamiast standardowej strzałki skierowanej w dół.

Przykład:

```
OknoPierwsze  WINDOW,AT(0,0,160,400)
                LIST,AT(120,0,20,20),USE(?L7),FROM(Que1),DROP(6)
                COMBO(@S8),AT(120,120,20,20),USE(?C7),FROM(Que2),DROP(8)
              END
  CODE
  OPEN(OknoPierwsze)
  ?C7{PROP:Icon} = 'MyDrop.ICO'              ! zmiana ikony drop w kontrolce COMBO
```

DROPID (ustawienie sygnatur celu drag-and-drop)

DROPID(*signature* [, *signature*])

DROPID Określa, że kontrolka może występować w roli celu dla operacji „przenieś i upuść”.

signature Stała łańcuchowa zawierająca identyfikator wskazujący prawidłowy host operacji drag-and-drop. Pojedynczy DROPID może zawierać do 16 sygnatur *signatures*. Dowolna sygnatura *signature* zaczynająca się znakiem tyldy (~) oznacza, że może być również wstawiana z zewnętrznych programów. Sygnatura *signature* DROPID w postaci ‘~FILE’ oznacza, że cel akceptuje listę nazw plików (oddzielonych od siebie przecinkami) przeciągniętych z okna Menedżera plików Windows.

Atrybut **DROPID** (PROP:DROPID, tablica) wskazuje kontrolkę, która może służyć jako cel operacji drag-and-drop. Atrybut DROPID pracuje w połączeniu z atrybutem DRAGID. Łańcuch sygnatur *signature* DROPID (do 16) definiuje klucze walidacyjne, z których jeden musi być zgodny z kluczem kontrolki pełniącej rolę hosta DRAGID. Dzięki temu możemy kontrolować, czy jest dopuszczalna operacja drag-and-drop pomiędzy dwiema różnymi kontrolkami.

Operacja drag-and-drop zachodzi wtedy, gdy użytkownik „przeciąga” myszką informacje z kontrolki posiadającej atrybut DRAGID do kontrolki posiadającej atrybut DROPID. By operacja ta zakończyła się powodzeniem, obie kontrolki muszą posiadać przynajmniej jedną identyczną sygnaturę określoną odpowiednio w atrybutach DRAGID (host) i DROPID (target).

Przykład:

```
OknoPierwsze  WINDOW,AT(0,0,160,400)
                LIST,AT(120,0,20,20),USE(?List1),FROM(Que1),DRAGID('FromList1')
                ! umożliwia przeciąganie, ale nie upuszczanie
                LIST,AT(120,120,20,20),USE(?List2),FROM(Que2),DROPID('FromList1','~FILE')
                ! umożliwia upuszczanie z List1 lub z Menedżera plików Windows, ale nie przeciąganie
              END
  CODE
  OPEN(OknoPierwsze)
  ACCEPT
    CASE EVENT()
    OF EVENT:Drag                ! zgłoszone zdarzenie związane z przeciąganiem
      IF DRAGID()                ! sprawdzenie, czy pomyślne
      SETDROPID(Que1)            ! przygotowanie informacji do przekazania
    END
    OF EVENT:Drop                ! gdy zdarzenie związane z upuszczeniem jest pomyślne
      Que2 = DROPID()            ! pobranie informacji
      ADD(Que2)                  ! i dodanie jej do kolejki
    END
  END
```

Porównaj: DRAGID

FILL (określenie koloru wypełnienia)

FILL(*rgb*)

FILL Określa kolor wypełnienia dla kontrolki BOX lub ELLIPSE.

rgb Stała całkowita typu LONG lub ULONG zawierająca komponenty czerwieni, zieleni i błękitu tworzące kolor w trzech mniej znaczących bajtach (0, 1, 2) lub ekwiwalent standardowego koloru Windows.

Atrybut **FILL** (PROP:FILL) określa kolor wypełniania dla kontrolki BOX lub ELLIPSE. Jeśli jest pominięty, kontrolka nie jest wypełniana kolorem. Właściwość PROP:FILL daje w rezultacie COLOR:None jeśli atrybut FILL nie występuje.

Przykład:

```
MojRaport   REPORT,AT(1000,1000,6500,9000),THOUS
CustDetail    DETAIL,AT(0,0,6500,1000)
                ELLIPSE,AT(60,60,200,200),FILL(COLOR:ACTIVEBORDER)   ! EQUATE koloru
                BOX,AT(360,60,200,200),FILL(00FF0000h)
              END
            END

OknoPierwsze  WINDOW,AT(0,0,160,400)
                BOX,AT(20,20,20,20),FILL(COLOR:ACTIVEBORDER)
                ! kolor aktywnej ramki ustawiony w Windows
                BOX,AT(100,100,20,20),FILL(00FF0000h)                ! Niebieski
                BOX,AT(140,140,20,20),FILL(0000FF00h)                ! Zielony
                BOX,AT(180,180,20,20),FILL(000000FFh)                ! Czerwony
              END
```

FIRST, LAST (określenie pozycji menu lub elementu menu)

FIRST
LAST

Atrybuty **FIRST** i **LAST** (PROP:FIRST oraz PROP:LAST) określają sposób umiejscawiania elementów menu w sytuacji, gdy do globalnego menu aplikacji są dołączane elementy menu okna WINDOW (zdefiniowane w strukturze MENUBAR). Kolejność priorytetów przedstawia się następująco:

1. Globalne elementy z atrybutem FIRST
2. Lokalne elementy z atrybutem FIRST
3. Globalne elementy bez atrybutu FIRST lub LAST
4. Lokalne elementy bez atrybutu FIRST lub LAST
5. Globalne elementy z atrybutem LAST
6. Lokalne elementy z atrybutem LAST

FLAT (ustawienie przeźroczystego przycisku)

FLAT

Atrybut **FLAT** (PROP:FLAT) powoduje, że kontrolka BUTTON, CHECK lub RADIO posiadająca atrybut ICON staje się przeźroczysta dopóty, dopóki nie znajdzie się nad nią wskaźnik myszki. Atrybut ten jest zazwyczaj stosowany w przypadku kontrolek umieszczanych w pasku narzędzi i działa najlepiej w połączeniu z ikonami reprezentowanymi przez pliki formatu .GIF, a to z tego względu, że grafika jest automatycznie „wyszarzana" gdy kontrolka nie jest aktywna (nie znajduje się nad nią wskaźnik myszki).

Przykład:

```
OknoPierwsze  WINDOW,AT(0,0,160,400)
                TOOLBAR
                  CHECK('1'),AT(0,0,20,20),USE(C1),ICON('Check1.GIF'),FLAT
                  BUTTON,AT(120,0,20,20),USE(?B7),ICON('Button1.GIF'),FLAT
                  OPTION('Option 4'),USE(OptVar4)
                    RADIO('Radio 7'),AT(120,0,20,20),USE(?R7),ICON('Radio1.GIF'),FLAT
                    RADIO('Radio 8'),AT(140,0,20,20),USE(?R8),ICON('Radio2.GIF').FLAT
                  END
                END
              END
```

FONT (ustawienie domyślnej czcionki)

FONT([*typeface*] [, *size*] [, *color*] [, *style*])

FONT Określa domyślną czcionkę dla paska narzędzi TOOLBAR .

typeface Stała łańcuchowa zawierająca nazwę czcionki (PROP:FontName, równoważna z {PROP:Font,1}). Jeśli pominiemy, stosowana jest czcionka systemowa.

size Stała całkowita zawierająca rozmiar (w punktach) czcionki (PROP:FontSize, równoważna z {PROP:Font,2}). Jeśli pominiemy, stosowany jest domyślny systemowy rozmiar czcionki.

color Stała całkowita LONG zawierająca komponenty czerwieni, zieleni i błękitu tworzące kolor w trzech mniej znaczących bajtach (0, 1, 2) lub ekwiwalent standardowego koloru Windows (PROP:FontColor, równoważne z {PROP:Font,3}). Jeśli pominiemy, domyślnym kolorem jest czarny.

style Stała całkowita, wyrażenie stałe lub ekwiwalent EQUATE określający wagę i styl czcionki (PROP:FontStyle, równoważna z {PROP:Font,4}). Jeśli pominiemy, przyjmowana jest waga normalna.

Atrybut **FONT** (PROP:FONT) określa czcionkę dla kontrolki. Jeśli źródłem *target* właściwości kontrolki jest wbudowana zmienna systemowa SYSTEM, właściwość PROP:FONT określa czcionkę dla procedury MESSAGE.

Parametr *typeface* może zawierać nazwę dowolnej czcionki zarejestrowanej w systemie Windows. W przypadku raportów, daną czcionkę *typeface* musi obsługiwać sterownik drukarki (jest to spełnione w odniesieniu do czcionki TrueType dla większości drukarek).

Plik EQUATES.CLW zawiera ekwiwalenty EQUATE dla standardowych stylów *style* czcionek. Wartość *style* w zakresie od 0 do 1000 określa wagę czcionki. Możemy do tego dodawać wartości odpowiadające kursywie, podkreśleniu lub podkreśleniu tekstu. Następujące ekwiwalenty można znaleźć w pliku EQUATES.CLW:

FONT:thin	EQUATE (100)
FONT:regular	EQUATE (400)
FONT:bold	EQUATE (700)
FONT:italic	EQUATE (01000H)
FONT:underline	EQUATE (02000H)
FONT:strikeout	EQUATE (04000H)

Zastosowanie w oknie

Atrybut FONT w oknie WINDOW lub APPLICATION określa domyślną czcionkę wyświetlania dla wszystkich ich kontrolek nie posiadających atrybutu FONT. Jest to również domyślna czcionka dla nowo tworzonych kontrolek w oknie, jak również czcionka stosowana przez instrukcje SHOW i TYPE rysujące w oknie.

Atrybut FONT w pasku narzędzi TOOLBAR określa domyślną czcionkę wyświetlania dla wszystkich jego kontrolek nie posiadających atrybutu FONT

Ustawienie jednej z właściwości (PROP:*property*) atrybutu FONT dla okna WINDOW, APPLICATION lub paska narzędzi TOOLBAR nie ma wpływu na czcionkę stosowaną w już wyświetlanych kontrolkach. Dotyczy to dopiero kontrolek utworzonych (za pomocą CREATE) już po określeniu danej właściwości.

Atrybut FONT określony w deklaracji kontrolki przykrywa ustawienie wynikające z atrybutu FONT umieszczonego w deklaracji okna WINDOW, APPLICATION lub paska narzędzi TOOLBAR.

Zastosowanie w raporcie

Atrybut FONT umieszczony w strukturze REPORT określa domyślną czcionkę drukowania dla wszystkich kontrolek raportu REPORT. Czcionka ta jest stosowana wtedy, gdy dana kontrolka nie posiada swojego własnego atrybutu FONT i gdy atrybutu takiego nie posiada również struktura raportu, w której została ona umieszczona.

Atrybut FONT w strukturach FORM, DETAIL, HEADER oraz FOOTER określa domyślną czcionkę drukowania dla wszystkich kontrolek w nich zawartych, które nie posiadają atrybutu FONT.

Atrybut FONT określony w deklaracji kontrolki przykrywa ustawienie wynikające z atrybutu FONT umieszczonego w deklaracji raportu REPORT lub danej jego struktury.

Przykład:

```
OknoPierwsze  WINDOW,AT(0,0,160,400)
                LIST,AT(120,0,20,20),USE(?L7),FROM(Que1),FONT('Arial',14,0FFh)
                ! 14-punktowy Arial, czerwony, normalny
                LIST,AT(120,120,20,20),USE(?C7),FROM(Que2),FONT('Arial',14,0,700)
                ! 14-punktowy Arial, czarny, pogrubiony
                LIST,AT(120,240,20,20),USE(?C7),FROM(Que2),FONT('Arial',14,0,700+01000h)
                ! 14-punktowy Arial, czarny, pogrubiona kursywa
              END

MojRaport     REPORT,AT(1000,1000,6500,9000),THOUS, |
                FONT('Arial',12,,FONT:Bold+FONT:Italic)
                ! deklaracje raportu
              END

! okno stosujące 14-punktowy Times New Roman, pogrubiona kursywa
Win           WINDOW,FONT('Times New Roman',14,00H,FONT:italic+FONT:bold)
                STRING('This is Times 14 pt Bold Italic'),AT(42,14),USE(?String1)
              END
  CODE
  OPEN(Win)
  Win{PROP:FontSize} = 20              ! ustawia domyślny rozmiar czcionki dla tworzonych kontrolek
  CREATE(100,CREATE:string)            ! tworzy kontrolkę
  100{PROP:Text} = 'This is 20 point'
  SETPOSITION(100,82,24)
  UNHIDE(100)
  ACCEPT
  END
```

Porównaj: SETFONT, GETFONT, FONTDIALOG, COLOR, CREATE

FORMAT (określenie układu kontrolki LIST lub COMBO)

FORMAT(*format string*)

FORMAT Określa format wyświetlania lub drukowania danych kontrolki LIST lub COMBO.

format string Stała łańcuchowa określająca format.

Atrybut **FORMAT** (PROP:FORMAT) określa sposób wyświetlania danych w kontrolkach LIST oraz COMBO. Łańcuch *format string* zawiera informacje o formatowaniu danych. Właściwość PROP:FORMAT jest aktualizowana, gdy użytkownik dynamicznie zmienia format listy LIST lub COMBO w czasie działania programu.

Łańcuch *format string* zawiera „specyfikatory pól" mapujące pola kolejki przeznaczone do wyświetlania. Specyfikatory pól mogą być łączone w grupy – ujmuje się je w nawiasach kwadratowych (**[ ]**) – wyświetlane jako pojedyncza jednostka. Do wyświetlania są pobierane z kolejki QUEUE tylko te pola, dla których określono specyfikatory. Oznacza to, że jeśli w łańcuchu *format string* określono dwa specyfikatory pól, a element kolejki QUEUE składa się z trzech pól, to będą wyświetlane tylko te dwa wyspecyfikowane.

Format opisujący pola

Każda z kolumn listy LIST jest formatowana w oparciu o przedstawione poniżej składniki. Format określonej kolumny jest ustawiany albo określany przez właściwość PROPLIST:Format.

width justification [**(***indent***)**] [*modifiers*]

width Wymagana liczba całkowita definiująca szerokość pola (PROPLIST:Width). Określa się ją w jednostkach dialogowych

justification Pojedyncza, wielka litera (**L**, **R**, **C** lub **D**) określająca wyrównywanie do lewej – **L**eft (PROPLIST:Left), prawej - **R**ight (PROPLIST:Right), środka – **C**enter (PROPLIST:Center) lub kropki dziesiętnej - **D**ecimal (PROPLIST:Decimal). Jest wymagane podanie jednej z tych liter.

indent Opcjonalna liczba całkowita, zamknięta w nawiasach, określająca wcięcie przy wybranym wyrównywaniu. Może to być wartość ujemna. Przy wyrównywaniu do lewej (**L**) (PROPLIST:LeftOffset), *indent* definiuje lewy margines; przy wyrównywaniu do prawej (**R**) (PROPLIST:RightOffset) lub do kropki dziesiętnej (**D**) (PROPLIST:DecimalOffset) – definiuje prawy margines; przy centrowaniu (**C**) (PROPLIST:CenterOffset) – definiuje przesunięcie od środka pola (wartość ujemna – w lewo).

modifiers: Opcjonalne znaki specjalne, wymienione poniżej, modyfikujące format wyświetlanego pola lub grupy. Dla jednego pola lub grupy możemy użyć wielu modyfikatorów *modifiers*.

Modyfikatory

***** Gwiazdka (PROPLIST:Color) oznacza, że w czterech polach LONG jest umieszczona informacja o kolorach pola. Znajduje się ona bezpośrednio po polu danych w kolejce QUEUE (lub w łańcuchu atrybutu FROM). Cztery kolory dotyczą normalnego koloru pisania, normalnego koloru tła, wyróżnionego koloru pisania i wyróżnionego koloru tła (w takiej właśnie kolejności). Nie dotyczy raportów REPORT.

Y Modyfikator Y (PROPLIST:CellStyle) oznacza predefiniowany styl dla pola (kolumny) znajdujący się w polu LONG występującym bezpośrednio po polu danych w kolejce QUEUE (lub w łańcuchu atrybutu FROM). Pole LONG zawiera liczbę identyfikującą element w tablicy stylów związanej z kontrolką LIST dostępną za pośrednictwem właściwości PROPSTYLE: (wymieniono je w dalszej części). Nie dotyczy to raportów REPORT.

Styl dla całej kolumny może zostać określony za pomocą PROPLIST:ColStyle. Jeśli stosujemy PROPLIST:ColStyle, pole LONG nie jest konieczne w kolejce QUEUE, ale bez niego nie możemy nadawać indywidualnych stylów wybranym komórkom kolumny.

I Modyfikator I (PROPLIST:Icon) oznacza, że przy lewej krawędzi kolumny będzie wyświetlana ikona. Numer ikony jest zawarty w polu LONG występującym bezpośrednio po polu danych kolejki QUEUE (lub w łańcuchu atrybutu FROM). Pole LONG zawiera liczbę wskazującą element w liście ikon związanych z kontrolką LIST dostępną za pośrednictwem właściwości PROP:IconList. Jeśli dodatkowo jest dodany modyfikator * określający kolory, pole LONG dla modyfikatora I musi wystąpić po nich. Nie dotyczy raportów REPORT.

J Modyfikator J (PROPLIST:IconTrn) oznacza wyświetlanie przeźroczystej ikony. Pozostałe jego cechy są takie same, jak w przypadku modyfikatora I. Nie dotyczy raportów REPORT.

T *[(suppress)]*

Modyfikator T (PROPLIST:Tree) powoduje, że w kontrolce LIST jest wyświetlana lista elementów w postaci struktury drzewiastej. Liczba poziomów drzewa jest zapisana w polu LONG występującym bezpośrednio po polu danych kolejki QUEUE (lub w łańcuchu atrybutu FROM). Jeśli dodatkowo zostały określone modyfikatory ***** oraz **I**, to pole LONG musi występować po związanych z nimi polach LONG. Stan zwinięcia/rozwinięcia poziomów drzewa jest określony poprzez znak wartości zapisanej w polu LONG (dodatnia – drzewo rozwinięte, ujemna – drzewo zwinięte). Nie dotyczy raportów REPORT.

Opcjonalny parametr *suppress* może zawierać **1** (PROPLIST:TreeOffset) w celu nadania głównemu poziomowi numeru jeden (1) zamiast zero (0). W ten sposób wartość -1 oznacza zwinięcie głównego poziomu. Może on również zawierać **R** (PROPLIST:TreeRoot) powodujące brak wyświetlania linii łączących dla poziomu głównego, **L** (PROPLIST:TreeLines) powodujące brak wyświetlania linii łączących pomiędzy wszystkimi poziomami, **B** (PROPLIST:TreeBoxes) powodujące brak wyświetlania przycisków rozwijania poziomów lub **I** (PROPLIST:TreeIndent) powodujące brak wcinania poziomów (pociąga to za sobą brak wyświetlania linii łączących i przycisków rozwijania).

~header~ [justification [(indent)]]

Łańcuch nagłówka zamknięty w znakach tyldy (PROPLIST:Header), po którym następuje opcjonalny parametr wyrównywania (**L** = PROPLIST:HeaderLeft, **R** = PROPLIST:HeaderRight, **C** = PROPLIST:HeaderCenter lub **D** = PROPLIST:HeaderDecimal,) i/lub wartość wcięcia *indent* zamknięta w nawiasach (PROPLIST:HeaderLeftOffset, PROPLIST:HeaderRightOffset, PROPLIST:HeaderCenterOffset lub PROPLIST:HeaderDecimalOffset), wyświetla nagłówek na początku listy. Nagłówek stosuje to samo wyrównywanie i wcięcie, co pole, chyba, że określono dla niego odmienne wartości.

@picture@

Wzorzec *picture* (PROPLIST:Picture) formatuje pole do wyświetlania. Końcowy znak @ jest wymagany do oznaczenia końca wzorca *picture*, tak więc wzorzec wyświetlania typu @N12~PLN~ może być stosowany bez obaw, że powstaną niepożądane dwuznaczności.

? Znak zapytania (PROPLIST:Locator) definiuje pole lokatora dla listy COMBO posiadającej pole selektora. Dla rozwijalnych list wielokolumnowych jest to wartość wyświetlana w aktualnie podświetlonym elemencie. Nie dotyczy raportów REPORT.

#number#

Liczba *number* zamknięta w znakach (#) (PROPLIST:FieldNo) wskazuje pole kolejki QUEUE przeznaczone do wyświetlenia. Kolejne pola łańcucha formatowania bez *#number#* są pobierane w porządku występowania, po pobraniu pola *#number#*. Na przykład, #2# w pierwszym polu łańcucha formatowania oznacza rozpoczęcie od drugiego pola w kolejce QUEUE, z pominięciem pierwszego. Jeśli liczba pól określonych w łańcuchu formatowania jest większa lub równa liczbie pól kolejki QUEUE, format „zatacza koło", wracając do pierwszego pola kolejki QUEUE.

_ Znak podkreślenia (PROPLIST:Underline) powoduje podkreślenie pola.

/ Ukośnik (PROPLIST:LastOnLine) wymusza pojawienie się następnego pola w nowej linii (stosuje się tylko dla pól w grupie).

| Pionowa kreska (PROPLIST:RightBorder) umieszcza pionową linię po prawej stronie pola.

M Litera M (PROPLIST:Resize) umożliwia zmianę rozmiaru kolumny pola (grupy pól) dynamicznie przez użytkownika. Użytkownik może chwycić myszką prawą krawędź kolumny i przesuwać ją zmieniając tym samym rozmiar kolumny. Nie dotyczy raportów REPORT.

F Litera F (PROPLIST:Fixed) powoduje utworzenie stałej kolumny, która pozostaje cały czas w liście, nawet wtedy gdy przewijamy w bok jej zawartość (potrzebny atrybut HSCROLL). Stałe pola lub grupy muszą się znajdować na początku listy. Modyfikator ten jest ignorowany jeśli pole znajduje się wewnątrz grupy. Nie dotyczy raportów REPORT.

S(*integer*)

Modyfikator S poprzedzający wartość *integer* (PROPLIST:Scroll) ujętą w nawiasy dodaje do grupy pasek przewijania. Wartość *integer* określa całkowitą liczbę jednostek dialogowych, o które może być przewinięta kolumna. Dzięki temu szerokie pola mogą być wyświetlane w kolumnach o niewielkiej szerokości. Modyfikator ten jest ignorowany, jeśli został umieszczony przy polu znajdującym się wewnątrz grupy. Nie dotyczy

raportów REPORT.

Format grup pól

[*multiple field-specifiers* **]** [**(***size***)**] [*modifiers*]

multiple field-specifiers

Lista specyfikatorów pól ujęta w nawiasy kwadratowe (**[ ]**) powodujące traktowanie jej jako pojedynczej jednostki przeznaczonej do wyświetlania.

size Opcjonalna wartość całkowita, zamknięta w nawiasach, określająca szerokość grupy (PROPLIST:Width). Jeśli zostanie pominięta, rozmiar jest wyliczany w oparciu o listę pól.

modifiers Modyfikatory grup pól oddziałujące na całą grupę lub poszczególne pola. Są to te same modyfikatory, co wymienione powyżej (dla pól), za wyjątkiem *, I, T oraz #number#. Jeśli chcemy, by dana właściwość odnosiła się do grupy, dodajemy do niej PROPLIST:Group.

Format wyświetlania pól kolejki QUEUE

Porządek pól pojawiających się w kolejce QUEUE przeznaczonej do wyświetlenia w kontrolce LIST jest ważny. Ponieważ istnieje kilka modyfikatorów wymagających istnienia w kolejce pól przeznaczonych na przechowywanie skojarzonych z nimi wartości, poniżej przedstawiono zasady porządkowania pól, których należy przestrzegać:

1. Pole zawierające dane przeznaczone do wyświetlenia (zawsze).
2. Pole stylu **Y** (jeśli modyfikator Y występuje lub jest ustawiona właściwość PROPLIST:CellStyle).
3. Pole modyfikatora **T** dla struktury drzewiastej (jeśli modyfikator T występuje lub jest ustawiona właściwość PROPLIST:Tree).
4. Pole dla modyfikatora **I** lub **J** (jeśli modyfikator I lub J występuje lub jest ustawiona właściwość PROPLIST:Icon bądź PROPLIST:IconTrn).
5. Pole zawierające normalny kolor pisania dla modyfikatora ***** (jeśli modyfikator * występuje lub jest ustawiona właściwość PROPLIST:Color).
6. Pole zawierające normalny kolor tła dla modyfikatora ***** (jeśli modyfikator * występuje lub jest ustawiona właściwość PROPLIST:Color).
7. Pole zawierające wyróżniony kolor pisania dla modyfikatora ***** (jeśli modyfikator * występuje lub jest ustawiona właściwość PROPLIST:Color).
8. Pole zawierające wyróżniony kolor tła dla modyfikatora ***** (jeśli modyfikator * występuje lub jest ustawiona właściwość PROPLIST:Color).

Właściwości dynamiczne

Właściwości pojedynczych pól i grup w wielokolumnowej kontrolce LIST lub COMBO również mogą być zmieniane za pomocą ekwiwalentów przypisywanych do ich właściwości (PROPLIST:Item). Właściwości te eliminują konieczność tworzenia kompletnych atrybutów FORMAT do zmiany pojedynczej właściwości pojedynczego pola w liście LIST.

Są to właściwości tablicowe wymagające podania numeru elementu tablicy występującego po ekwiwalencie (oddzielamy je przecinkiem) w celu określenia, której kolumny kontrolki LIST lub COMBO dotyczą. Wszystkie one zawierają łańcuch pusty (‘’) jeśli nie występują lub jeden (1) w przeciwnym wypadku.

Przykład:

```
  PROGRAM
  MAP
DisplayList          PROCEDURE
PrintList            PROCEDURE
RandomAlphaData      PROCEDURE(*STRING)
  END

TreeDemo    QUEUE                        ! kolejka pola FROM listy elementów
FName         STRING(20)
ColorNFG      LONG(COLOR:White)          ! normalny kolor pisania dla FName
ColorNBG      LONG(COLOR:Maroon)         ! normalny kolor tła dla FName
ColorSFG      LONG(COLOR:Yellow)         ! wyróżniony kolor pisania dla FName
ColorSBG      LONG(COLOR:Blue)           ! wyróżniony kolor tła dla FName
IconField     LONG                       ! numer ikony dla FName
TreeLevel     LONG                       ! poziom drzewa
LName         STRING(20)
Init          STRING(4)
            END
  CODE
  DisplayList
  PrintList

DisplayList  PROCEDURE
Win          WINDOW('List Boxes'),AT(0,0,366,181),SYSTEM,DOUBLE
               LIST,AT(0,34,366,146),FROM(TreeDemo),USE(?Show),HVSCROLL, |
                 FORMAT('80L*IT~First Name~*80L~Last Name~16C~Initials~')
             END
  CODE
  LOOP X# = 1 TO 20
    RandomAlphaData(TreeDemo.FName)
    TreeDemo.IconField = ((X#-1) % 4) + 1     ! przypisanie numeru ikony
    TreeDemo.TreeLevel = ((X#-1) % 4) + 1     ! przypisanie poziomu drzewa
    RandomAlphaData(TreeDemo.LName)
    RandomAlphaData(TreeDemo.Init)
    ADD(TreeDemo)
  END
  OPEN(Win)
  ?Show{PROP:iconlist,1} = ICON:VCRback !Icon 1 = <
  ?Show{PROP:iconlist,2} = ICON:VCRrewind !Icon 2 = <<
  ?Show{PROP:iconlist,3} = ICON:VCRplay !Icon 3 = >
  ?Show{PROP:iconlist,4} = ICON:VCRfastforward !Icon 4 = >>
  ACCEPT
  END

RandomAlphaData   PROCEDURE(Field) !MAP Prototype is: RandomAlphaData(*STRING)
  CODE
  CLEAR(Field)
  RandomSize# = RANDOM(1,SIZE(Field))     ! rozmiar określony losowo
  Field[1] = CHR(RANDOM(65,90))           ! rozpoczęcie losową dużą literą
  LOOP Z# = 2 to RandomSize#              ! wypełnienie każdego znaku
    Field[Z#] = CHR(RANDOM(97,122))       ! losową małą literą
  END

PrintList   PROCEDURE
DemoQ       QUEUE
FName         STRING(20)
ColorNFG1     LONG
ColorNBG1     LONG
ColorSFG1     LONG(COLOR:Black)           ! kolor pisania dla FName
```

```
ColorSBG1     LONG(COLOR:White)          ! kolor tła dla FName
LName         STRING(20)
ColorNFG2     LONG
ColorNBG2     LONG
ColorSFG2     LONG(COLOR:Black)          ! kolor pisania dla LName
ColorSBG2     LONG(COLOR:White)          ! kolor tła dla LName
Init          STRING(4)
ColorNFG3     LONG
ColorNBG3     LONG
ColorSFG3     LONG(COLOR:Black)          ! kolor pisania dla Init
ColorSBG3     LONG(COLOR:White)          ! kolor tła dla Init
Wage          REAL
ColorNFG4     LONG
ColorNBG4     LONG
ColorSFG4     LONG(COLOR:Black)          ! kolor pisania dla Wage
ColorSBG4     LONG(COLOR:White)          ! kolor tła dla Wage
            END

MojRaport   REPORT,AT(1000,1000,6500,9000),THOUS
CustDetail    DETAIL,AT(0,0,6500,200)
                LIST,AT(0,0,6000,200),FORMAT(''),FROM(DemoQ),USE(?Show)
              ..
  CODE
  LOOP X# = 1 TO 20
    CLEAR(DemoQ)
    RandomAlphaData(DemoQ.FName)
    RandomAlphaData(DemoQ.LName)
    RandomAlphaData(DemoQ.Init)
    DemoQ.Wage = RANDOM(100000,1000000)/100
    ADD(DemoQ)
  END
  OPEN(CustRpt)
  SETTARGET(CustRpt)
  IF RANDOM(0,1)
    ?Show{PROP:format} = '2000L*~First Name~2000L*~Last Name~500L*~Intls~1000L*~Wage~|'
  ELSE
    ?Show{PROP:format} = '2000L*~First Name~2000L*~Last ' & |
      'Name~500L*~Intls~1000D(400)*~Wage~|'
    ?Show{PROPLIST:Header,1} = 'First Field'                          ! zmienia tekst nagłówka pierwszej kolumny
    ?Show{PROPLIST:Header + PROPLIST:Group,1} = 'First Group'
  END                                                                ! zmienia tekst nagłówka pierwszej grupy
  LOOP X# = 1 TO RECORDS(DemoQ)
    GET(DemoQ,X#)
    PRINT(CustDetail)
  END
  CLOSE(CustRpt)
  FREE(DemoQ)
```

Style

Poniższe właściwości wykorzystuje się do definiowania tablic stylów dostępnych dla kolumn posiadających modyfikator **Y** lub do stosowania w połączeniu z właściwością PROPLIST:ColStyle. Definiuje się w ten sposób tablicę dostępnych stylów. Przypisując numer elementu tablicy do pola LONG związanego z modyfikatorem Y lub do właściwości PROPLIST:ColStyle powodujemy wyświetlenie danej komórki lub kolumny w określonym stylu.

PROPSTYLE:FontName

Właściwość tablicowa ustawiająca lub określająca nazwę czcionki dla numeru stylu wskazanego jako element tablicy.

PROPSTYLE:FontSize

Właściwość tablicowa ustawiająca lub określająca rozmiar czcionki dla numeru stylu wskazanego jako element tablicy.

PROPSTYLE:FontColor

Właściwość tablicowa ustawiająca lub określająca kolor czcionki dla numeru stylu wskazanego jako element tablicy.

PROPSTYLE:FontStyle

Właściwość tablicowa ustawiająca lub określająca styl czcionki (waga, itp.) dla numeru stylu wskazanego jako element tablicy.

PROPSTYLE:TextColor

Właściwość tablicowa ustawiająca lub określająca kolor tekstu dla numeru stylu wskazanego jako element tablicy (to samo, co kolor czcionki).

PROPSTYLE:BackColor

Właściwość tablicowa ustawiająca lub określająca kolor tła dla numeru stylu wskazanego jako element tablicy.

PROPSTYLE:TextSelected

Właściwość tablicowa ustawiająca lub określająca kolor tekstu wyróżnionego dla numeru stylu wskazanego jako element tablicy.

PROPSTYLE:BackSelected

Właściwość tablicowa ustawiająca lub określająca kolor tła wyróżnionego dla numeru stylu wskazanego jako element tablicy.

PROPSTYLE:Picture

Właściwość tablicowa ustawiająca lub określająca wzorzec wyświetlania związany z numerem stylu wskazanego jako element tablicy.

Przykład:

```
?list{PROPSTYLE:FontName, 1} = 'Arial'                ! konfiguruje styl wartości dodatniej
?list{PROPSTYLE:FontSize, 1} = 11
?list{PROPSTYLE:FontStyle, 1} = FONT:Regular
?list{PROPSTYLE:TextColor, 1} = COLOR:Yellow
?list{PROPSTYLE:BackColor, 1} = COLOR:Black
?list{PROPSTYLE:TextSelected, 1} = COLOR:Yellow
?list{PROPSTYLE:BackSelected, 1} = COLOR:Blue
?list{PROPSTYLE:Picture, 1} = '@n11.2'
?list{PROPSTYLE:FontName, 2} = 'Arial'                ! konfiguruje styl wartości ujemnej
?list{PROPSTYLE:FontSize, 2} = 11
?list{PROPSTYLE:FontStyle, 2} = FONT:Bold
?list{PROPSTYLE:TextColor, 2} = COLOR:Red
?list{PROPSTYLE:BackColor, 2} = COLOR:White
?list{PROPSTYLE:TextSelected, 2} = COLOR:Red
?list{PROPSTYLE:BackSelected, 2} = COLOR:Yellow
?list{PROPSTYLE:Picture, 2} = '@n(13.2)'
?list{PROPLIST:ColStyle,1} = 1                        ! kolumna 1 stosuje styl dodatni
?list{PROPLIST:ColStyle,2} = 2                        ! kolumna 2 stosuje styl ujemny
```

Inne właściwości okienka typu LIST

Przedstawione poniżej właściwości nie są częścią atrybutu FORMAT, mogą jednak dynamicznie wpływać na wygląd kontrolki LIST lub COMBO.

PROPLIST:BackColor

Właściwość tablicowa ustawiająca lub określająca domyślny kolor tła dla tekstu w kolumnie o numerze określonym jako element tablicy. To ustawienie może zostać przykryte przez standardowy mechanizm kolorowania komórek.

PROPLIST:BackSelected

Właściwość tablicowa ustawiająca lub określająca domyślny kolor tła wyróżnionego dla tekstu w kolumnie o numerze określonym jako element tablicy. To ustawienie może zostać przykryte przez standardowy mechanizm kolorowania komórek.

PROPLIST:TextColor

Właściwość tablicowa ustawiająca lub określająca domyślny kolor tekstu dla tekstu w kolumnie o numerze określonym jako element tablicy. To ustawienie może zostać przykryte przez standardowy mechanizm kolorwania komórek.

PROPLIST:TextSelected

Właściwość tablicowa ustawiająca lub określająca domyślny kolor tekstu wyróżnionego dla tekstu w kolumnie o numerze określonym jako element tablicy. To ustawienie może zostać przykryte przez standardowy mechanizm kolorwania komórek.

PROPLIST:Exists

Właściwość tablicowa dająca w rezultacie wartość 1 jeśli kolumna o numerze wskazanym jako element tablicy istnieje. Na przykład, ?List{PROPLIST:Exists,1} testuje, czy kolumna 1 istnieje w liście. Jest to przydatne przy przetwarzaniu uogólnionych list.

Przykład:

```
WinView   WINDOW('View'),AT(,,340,200),SYSTEM,CENTER
            LIST,AT(0,0,300,200),USE(?List),FROM(Que),FORMAT('80L~F1~80L~F2~80L~F3~')
          END
  CODE
  OPEN(WinView)
  LOOP X# = 1 TO 255
    IF ?List{PROPLIST:Exists,X#} = 1                    ! czy jest kolumna o tym numerze
      ?List{PROPLIST:TextColor,X#} = COLOR:Red
      ?List{PROPLIST:BackColor,X#} = COLOR:White
      ?List{PROPLIST:TextSelected,X#} = COLOR:Yellow
      ?List{PROPLIST:TextSelected,X#} = COLOR:Blue
    ELSE
      BREAK
    END
  END
```

Właściwości myszki w okienku typu LIST

Przedstawione poniżej właściwości określają położenie myszki w kontrolce LIST lub COMBO w momencie, gdy jest wciskany lub zwalniany jej przycisk. Właściwości te możemy także zapisywać, co jednak nie daje żadnego efektu poza tym, że przy następnym ich odczycie (w ramach tej samej pętli ACCEPT) wartość właściwości będzie właśnie taka, jaką nadaliśmy. Działanie takie może ułatwić osiągnięcie zamierzonego efektu w niektórych sytuacjach.

PROPLIST:MouseDownField	Zwraca numer pola, gdy przycisk myszki został wciśnięty.
PROPLIST:MouseDownRow	Zwraca numer wiersza, gdy przycisk myszki został wciśnięty.
PROPLIST:MouseDownZone	Zwraca numer strefy, gdy przycisk myszki został wciśnięty.
PROPLIST:MouseMoveField	Zwraca numer pola, gdy myszka jest przesuwana.
PROPLIST:MouseMoveRow	Zwraca numer wiersza, gdy myszka jest przesuwana.
PROPLIST:MouseMoveZone	Zwraca numer strefy, gdy myszka jest przesuwana.
PROPLIST:MouseUpField	Zwraca numer pola, gdy przycisk myszki został zwolniony.
PROPLIST:MouseUpRow	Zwraca numer wiersza, gdy przycisk myszki został zwolniony.
PROPLIST:MouseUpZone	Zwraca numer strefy, gdy przycisk myszki został zwolniony.

Wszystkie trzy właściwości związane z wierszem dają w rezultacie 0 dla tekstu nagłówka oraz –1 w przypadku wyjścia poza ostatni wyświetlany element.

Ekwiwalenty dla poszczególnych stref są zdefiniowane w pliku EQUATES.CLW:

LISTZONE:Field	Pole w liście LIST
LISTZONE:Right	Prawa granica pola (linia umożliwiająca zmianę rozmiarów kolumny)
LISTZONE:Header	Nagłówek pola lub grupy
LISTZONE:ExpandBox	Przycisk rozwijania poziomu listy drzewiastej
LISTZONE:Tree	Linie łączące w liście drzewiastej
LISTZONE:Icon	Ikona
LISTZONE:Nowhere	Dowolne inne miejsce

Przykład:

```
Que    QUEUE
F1       STRING(50)
F2       STRING(50)
F3       STRING(50)
       END

WinView   WINDOW('View'),AT(,,340,200),SYSTEM,CENTER
```

```
            LIST,AT(20,0,300,200),USE(?List),FROM(Que),HVSCROLL, |
              FORMAT('80L~F1~80L~F2~80L~F3~'),ALRT(MouseLeft)
          END

SaveFormat    STRING(20)
SaveColumn    BYTE
Columns       BYTE,DIM(3)

  CODE
  OPEN(WinView)
  Columns[1] = 1
  Columns[2] = 2
  Columns[3] = 3
  DO BuildListQue
  ACCEPT
    CASE EVENT()
    OF EVENT:PreAlertKey
      CYCLE                                          ! umożliwia standardową obsługę kliknięć listy
    OF EVENT:AlertKey
      IF ?List{PROPLIST:MouseDownRow} = 0                    ! sprawdzenie, czy nie był klikniety nagłówek
                                                             ! kolumny
        EXECUTE Columns[?List{PROPLIST:MouseDownField}]  ! sprawdzenie, który to nagłówek
        SORT(Que,Que.F1)
        SORT(Que,Que.F2)
        SORT(Que,Que.F3)
      END
      SaveFormat = ?List{PROPLIST:Format,?List{PROPLIST:MouseDownField}}
      ?List{PROPLIST:Format,?List{PROPLIST:MouseDownField}} = ?List{PROPLIST:Format,1}
      ?List{PROPLIST:Format,1} = SaveFormat
      SaveColumn = Columns[?List{PROPLIST:MouseDownField}]
      Columns[?List{PROPLIST:MouseDownField}] = Columns[1]
      Columns[1] = SaveColumn
      DISPLAY
  ...
  FREE(Que)

BuildListQue    ROUTINE
  LOOP Y# = 1 TO 9
    Que.F1 = 'Que.F1 - ' & Y#
    Que.F2 = 'Que.F2 - ' & RANDOM(10,99)
    Que.F3 = 'Que.F3 - ' & RANDOM(100,999)
    ADD(Que)
    ASSERT(NOT ERRORCODE())
  END
```

FROM (określenie źródła danych dla okienka typu LIST)

FROM(*source*)

FROM Określa źródło danych, które mają być wyświetlane lub drukowane w kontrolce LIST.

source Etykieta kolejki QUEUE lub pola w kolejce QUEUE, bądź stała łańcuchowa lub zmienna (zazwyczaj typu GROUP) zawierająca elementy danych, które mają być wyświetlane bądź drukowane w liście LIST. Jeśli kolejka QUEUE została dynamicznie utworzona za pomocą instrukcji NEW, po zamknięciu okna musi być zwolniona za pomocą instrukcji DISPOSE.

Atrybut **FROM** (PROP:FROM, tylko-do-zapisu) wskazuje źródło danych, które mają być wyświetlane w kontrolce LIST, COMBO lub SPIN bądź drukowane w kontrolce LIST. Jeśli w roli *source* występuje stała łańcuchowa, indywidualne elementy danych muszą być od siebie odseparowane znakiem pionowej kreski (|). Jeśli chcemy, by znak ten występował jako jeden ze znaków elementu, należy go wstawić dwukrotnie (||) – wyświetlony zostania tylko jeden. Jeśli chcemy wstawić do listy element będący łańcuchem pustym, musimy umieścić przynajmniej jedną spację pomiędzy pionowymi kreskami (| |).

Zastosowanie w oknie

W przypadku kontrolek SPIN, źródłem *source* powinno być na ogół pole kolejki QUEUE lub łańcuch znakowy. Jeśli źródłem *source* jest kolejka QUEUE posiadająca wiele pól, tylko pierwsze z nich jest wyświetlane w kontrolce SPIN.

W kontrolkach LIST oraz COMBO elementy danych są formatowane zgodnie z informacjami określonymi w atrybucie FORMAT. Jeśli w roli źródła *source* występuje etykieta kolejki QUEUE, wszystkie pola tej kolejki są wyświetlane zgodnie z definicją określoną w atrybucie FORMAT. Jeżeli w roli źródła *source* występuje etykieta pojedynczego pola kolejki QUEUE, tylko ono jest wyświetlane.

Zastosowanie w raporcie

Jeśli w roli źródła *source* występuje etykieta kolejki QUEUE, wszystkie pola tej kolejki są drukowane. Jeżeli w roli źródła *source* występuje etykieta pojedynczego pola kolejki QUEUE, tylko ono jest drukowane. Należy pamiętać, że w liście LIST jest drukowany tylko jeden element kolejki, ten który aktualnie znajduje się w jej buforze.

Jeżeli w roli źródła *source* występuje stała lub zmienna łańcuchowa, drukowany jest w liście reprezentowany przez nią łańcuch (wszystkie elementy oddzielone od siebie znakiem pionowej kreski). Elementy danych są formatowane zgodnie z informacjami określonymi w atrybucie FORMAT.

Przykład:

```
TD        QUEUE,AUTO
FName       STRING(20)
LName       STRING(20)
Init        STRING(4)
Wage        REAL
          END

MojRaport  REPORT,AT(1000,1000,6500,9000),THOUS
CustDetail   DETAIL,AT(0,0,6500,1000)
               LIST,AT(0,34,366,146),FORMAT('80L80L16L60L'),FROM(TD),USE(?Show1)
               LIST,AT(0,200,100,146),FORMAT('80L'),FROM(Fname),USE(?Show2)
             END
           END

Que1       QUEUE,PRE(Q1)
F1           LONG
F2           STRING(8)
           END

Win1       WINDOW,AT(0,0,160,400)
             LIST,AT(120,0,20,20),USE(?L1),FROM(Que1),FORMAT('5C~List~15L~Box~'),COLUMN
             COMBO(@S8),AT(120,120,20,20),USE(?C1),FROM(Q1:F2)
             SPIN(@N8.2),AT(280,0,20,20),USE(SpinVar1),FROM(Q1:F1)
             SPIN(@S4),AT(280,0,20,20),USE(SpinVar2),FROM('Mr.|Mrs.|Ms.|Dr.')
           END
```

FULL (ustawienie wyświetlania pełnoekranowego)

FULL

Atrybut **FULL** (PROP:FULL) powoduje takie rozszerzenie kontrolki by zajmowała cały dostępny obszar okna WINDOW dla dowolnego brakującego parametru atrybutu AT określającego szerokość lub wysokość.

Atrybut FULL nie może być określany dla kontrolek paska narzędzi TOOLBAR.

GRAY (włączenie efektów 3D)

GRAY

Atrybut **GRAY** (PROP:GRAY) powoduje, że okno WINDOW otrzymuje szare tło, odpowiednie przy stosowaniu trójwymiarowych kontrolek. Wszystkie kontrolki okna WINDOW posiadającego atrybut GRAY automatycznie otrzymują trójwymiarowy wygląd. Kontrolki w pasku narzędzi TOOLBAR zawsze otrzymują trójwymiarowy wygląd, bez potrzeby stosowania atrybutu GRAY.

Atrybut ten nie jest prawidłowy dla struktury APPLICATION.

Trójwymiarowy wygląd kontrolek może być wyłączany za pomocą SET3DLOOK.

Przykład:

```
! okno z kontrolkami 3-D
Win1   WINDOW,GRAY
       END
```

Porównaj:SET3DLOOK

GRID (Ustawia kolor wyświetlania linii siatki)

GRID(*rgb*)

GRID Określa kolor, którym będą wyświetlane linie siatki w liście.

rgb Stała całkowita LONG lub ULONG bądź ekwiwalent EQUATE standardowego koloru Windows, tworząca kolor w oparciu o składniki czerwieni, zieleni i błękitu. Wartość koloru jest przechowywana w trzech mniej znaczących bajtach (0, 1 i 2)

Atrybut **GRID** (PROPLIST:GRID) Określa kolor, którym będą wyświetlane linie siatki w liście COMBO lub LIST. Ekwiwalenty EQUATE dla standardowych kolorów Windows znajdują się w pliku EQUATES.CLW. Windows automatycznie odnajduje kolor sprzętowy najbardziej pasujący do tego, który został określony poprzez wartość *rgb*.

Przykład:

```
OknoPierwsze  WINDOW,AT(0,0,400,400)
                LIST,AT(0,34,366,146),FROM(TreeDemo),USE(?Show),HVSCROLL,GRID(COLOR:Red) |
                  FORMAT('80L*IT~First Name~*80L~Last Name~16C~Initials~')
              END
```

HIDE (ustawienie kontrolki jako ukrytej)

HIDE

Atrybut **HIDE** (PROP:HIDE) powoduje, że kontrolka nie pojawia się, gdy jej okno WINDOW lub APPLICATION zostanie otwarte. Wyświetlenie kontrolki musi zostać wymuszone za pomocą instrukcji UNHIDE.

W raporcie REPORT kontrolka nie będzie drukowana dopóty, dopóki nie zostanie użyta instrukcja UNHIDE. Właściwość PROP:HIDE może być używana w połączeniu z wbudowana zmienną TARGET w celu ukrycia bądź wyświetlenia bieżącego okna.

HLP (wskazanie identyfikatora pomocy kontekstowej)

HLP(*helpID*)

HLP Określa identyfikator systemu pomocy *helpID* dla aplikacji APPLICATION, okna WINDOW lub kontrolki.

helpID Stała łańcuchowa definiująca klucz stosowany przy dostępie do systemu pomocy. Może to być albo słowo kluczowe systemu pomocy, albo „łańcuch kontekstowy".

Atrybut **HLP** (PROP:HLP) określa identyfikator systemu pomocy *helpID* dla aplikacji APPLICATION, okna WINDOW lub kontrolki. Jeśli system pomocy dla aplikacji został utworzony, jest on automatycznie wyświetlany po wciśnięciu przez użytkownika klawisza F1.

Jeżeli użytkownik wciska klawisz F1 w celu wywołania systemu pomocy aplikacji APPLICATION i nie jest przy tym aktywne żadne jej menu, do zlokalizowania odpowiedniego tematu pomocy jest wykorzystywany identyfikator *helpID* aplikacji. W przeciwnych wypadkach biblioteka Clariona automatycznie używa identyfikatora *helpID* aktywnego menu, kontrolki, czy okna, przeszukując hierarchię tematów pomocy. Identyfikator *helpID* aplikacji APPLICATION znajduje się na szczycie hierarchii.

Identyfikator *helpID* może zawierać słowo kluczowe systemu pomocy, albo „łańcuch kontekstowy".

- Słowo kluczowe systemu pomocy jest frazą wyświetlaną w oknie wyszukiwania w systemie pomocy. Gdy użytkownik naciśnie klawisz F1 i istnieje tylko jeden temat systemu pomocy wskazywany przez dany identyfikator, jest on otwierany. W przypadku, gdy jeden identyfikator wskazuje na wiele tematów pomocy, jest otwierane okno wyszukiwania.

Łańcuch kontekstowy jest identyfikowany przez wiodący znak tyldy (~) w identyfikatorze *helpID*, po którym następuje unikalny identyfikator (nie są dopuszczalne spacje) związany z jednym i tylko jednym tematem systemu pomocy. Gdy użytkownik naciśnie klawisz F1 w pliku pomocy jest otwierany temat związany z łańcuchem kontekstowym. Jeśli znak tyldy nie występuje, identyfikator *helpID* jest traktowany jak słowo kluczowe systemu pomocy.

Przykład:

```
! okno z pomocą kontekstową:
MainWin  APPLICATION('My Application'),SYSTEM,MAX,ICON('MyIcon.ICO'),STATUS,HLP('~App')
           MENUBAR
             MENU('&File'),USE(?FileMenu)
               ITEM('&Open...'),USE(?OpenFile),HLP('~OpenFileHelp')
             END
           END
         END

! okno ze słowem kluczowym pomocy:
Win2     WINDOW,HLP('Window One Help')
           ENTRY(@s30),USE(SomeVariable),HLP('~Entry1Help')        ! łańcuch pomocy kontekstowej
           ENTRY(@s30),USE(SomeVariable),HLP('Control Two Help')  ! słowo kluczowe systemu pomocy
         END
```

HSCROLL, VSCROLL, HVSCROLL (ustawienie pasków przewijania)

HSCROLL

VSCROLL

HVSCROLL

Atrybuty **HSCROLL**, **VSCROLL** oraz **HVSCROLL** powodują umieszczenie pasków przewijania w oknach APPLICATION lub WINDOW, bądź w kontrolkach COMBO, LIST, IMAGE, TEXT. HSCROLL (PROP:HSCROLL) dodaje pasek poziomy w dolnej części, a VSCROLL (PROP:VSCROLL) pasek pionowy po prawej stronie. Z kolei HVSCROLL (PROP:HVSCROLL) dodaje oba paski – u dołu i po prawej stronie.

Atrybut HSCROLL jest dostępny również dla kontrolki SHEET. Powoduje to, że zakładki TAB są wyświetlane w jednym wierszu, zamiast w kilku, niezależnie od tego, ile ich jest. Na obu końcach wiersza zakładek pojawiają się przyciski umożliwiające ich przewijanie. Właściwość PROP:BrokenTabs może zostać ustawiona na FALSE w celu wyłączenia wizualnego efektu „przełamanych" zakładek.

Pionowy pasek przewijania pozwala na przewijanie zawartości w górę i w dół. Pasek poziomy umożliwia przewijanie w lewo i w prawo. Paski przewijania pojawiają się wtedy, gdy nie cała zawartość kontrolki mieści się w obszarze wyznaczonym przez jej rozmiary.

Jeśli atrybut VSCROLL nadamy kontrolce LIST posiadającej atrybut IMM, pionowy pasek przewijania będzie wyświetlany przez cały czas. Gdy użytkownik kliknie myszką w pasek przewijania, są generowane zdarzenia, jednak zawartość listy się nie przesuwa (tę akcję trzeba oprogramować samodzielnie). Możemy sprawdzać właściwość PROP:VscrollPos w celu określenia pozycji suwaka na pasku przewijania: od 0 (góra) do 255 (dół).

Atrybuty HSCROLL, VSCROLL oraz HVSCROLL mogą być także stosowane w kontrolce SPIN i określać opcjonalny, inny od domyślnego (jeden pod drugim, wskazujące w górę i w dół), układ przycisków kontrolki. Atrybut HSCROLL powoduje umieszczenie przycisków obok siebie (wskazują w lewo i w prawo). Atrybut VSCROLL umieszcza przyciski jeden nad drugim (wskazują w lewo i w prawo). Atrybut HVSCROLL umieszcza przyciski obok siebie (wskazują w górę i w dół).

Przykład:

```
! okno z suwakiem poziomym:
Win1  WINDOW,HSCROLL,RESIZE
      END

! okno z suwakiem pionowym:
Win2  WINDOW,VSCROLL,RESIZE
      END

! okno z suwakiem poziomym i pionowym:
Win2  WINDOW,HVSCROLL,RESIZE
      END
```

ICON (wskazanie ikony)

ICON([*file*])

ICON Określa ikonę wyświetlaną w oknie APPLICATION, WINDOW lub kontrolce.

file Stała łańcuchowa zawierająca nazwę pliku graficznego w formacie .ICO, .GIF, .JPG lub .PCX bądź ekwiwalent EQUATE dla standardowych ikon systemu Windows. Plik z grafiką jest automatycznie linkowany do pliku .EXE aplikacji jako jej zasób.

Atrybut **ICON** (PROP:ICON) określa ikonę wyświetlaną w oknie APPLICATION, WINDOW lub kontrolce. W przypadku okna APPLICATION lub WINDOW, atrybut ICON umożliwia także występowanie przycisku zwijającego okno do postaci ikony i musi wskazywać plik z grafiką poprzez parametr *file*. Przycisk zwijający okno do postaci ikony pojawia się w prawej części paska tytułowego okna (trójkącik skierowany w dół w Windows 3.x i podkreślenie w Windows 95/98/NT). Gdy użytkownik kliknie myszką przycisk zwijający, okno jest zmniejszane do ikony – bez zatrzymywania jego działania. Jeśli minimalizowane jest okno aplikacji APPLICATION lub okno WINDOW nie posiadające atrybutu MDI, ikona z pliku *file* jest wyświetlana na pulpicie systemu operacyjnego; gdy jest minimalizowane okno WINDOW posiadające atrybut MDI, ikona z pliku *file* jest wyświetlana w obszarze roboczym aplikacji.

W przypadku kontrolek BUTTON, RADIO oraz CHECK, atrybut ICON wskazuje ikonę wyświetlaną w kontrolce. Grafika z pliku *file* jest wyświetlana na przycisku reprezentującym kontrolkę. W przypadku kontrolek RADIO oraz CHECK atrybut ICON powoduje utworzenie przycisków „wciskanych”, tzn. takich które pozostają w jednym z dwóch stanów: wciśniętym lub wyciśniętym.

Ekwiwalenty EQUATE dla standardowych ikon Windows są zawarte w pliku EQUATES.CLW. Poniżej podano przykłady kilku z nich:

ICON:None	brak ikony
ICON:Question	?
ICON:Exclamation	!
ICON:Asterisk	*
ICON:VCRtop	>>|
ICON:VCRrewind	<<
ICON:VCRback	<
ICON:VCRplay	>
ICON:VCRfastforward	>>
ICON:VCRbottom	|<<
ICON:VCRlocate	?

Jeśli nazwa pliku ikon przypisana do właściwości PROP:Icon posiada dołączony w nawiasach kwadratowych numer (IconFile.DLL[1]) oznacza to, że dany plik zawiera wiele ikon, a podany numer oznacza numer ikony w pliku (licząc od zera). Jeśli nazwę pliku ikon poprzedza znak tyldy (~IconFile.ICO), oznacza to, że plik został dołączony do projektu jako zasób i nie znajduje się w oddzielnym pliku na dysku.

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL,ICON('MyIcon.ICO')
            OPTION('Option'),USE(OptVar)
              RADIO('Radio 1'),AT(120,0,20,20),USE(?R1),ICON('Radio1.ICO')
              RADIO('Radio 2'),AT(140,0,20,20),USE(?R2),ICON('Radio2.GIF')
            END
            CHECK('&A'),AT(0,120,20,20),USE(?C7),ICON(ICON:Asterisk)
            BUTTON('&1'),AT(120,0,20,20),USE(?B7),ICON(ICON:Question)
          END
```

Porównaj: ICONIZE, MAX, MAXIMIXE, IMM

ICONIZE (wymuszenie otwarcia okna jako zwiniętego do ikony)

ICONIZE

Atrybut **ICONIZE** (PROP:ICONIZE) powoduje, że okno APPLICATION lub WINDOW jest otwierane w postaci zminimalizowanej do ikony. Gdy aplikacja APPLICATION lub okno WINDOW pozbawione atrybutu MDI jest zminimalizowane, ikona zawarta w pliku *file* jest wyświetlana na pulpicie systemu operacyjnego Windows (Windows 3.x) lub w systemowym pasku zadań pojawia się jedynie wizytówka okna (Windows 95/98/NT 4.0). Gdy jest zminimalizowane okno WINDOW z atrybutem MDI, ikona z pliku *file* (Windows 3.x) lub wizytówka okna (Windows 95/98/NT 4.0) jest wyświetlana w obszarze roboczym okna aplikacji APPLICATION.

Właściwość PROP:ICONIZE wbudowanej zmiennej systemowej SYSTEM daje w rezultacie wartość 1 jeśli ustawienia pliku PIF (Program Information File) aplikacji nakazują otwarcie jej w postaci zminimalizowanej.

Przykład:

```
! okno z przyciskiem Miimalizuj, otwarte w postaci ikony:
Win2   WINDOW,ICON('MyIcon.ICO'),ICONIZE
       END
```

Porównaj: ICON, IMM

IMM (wymuszenie natychmiastowego zgłaszania zdarzenia)

IMM

Atrybut **IMM** (PROP:IMM) powoduje bezzwłoczne generowanie zdarzenia.

Zastosowanie w oknie

W oknie WINDOW lub APPLICATION atrybut IMM powoduje natychmiastowe generowanie zdarzenia, gdy tylko użytkownik przesunie lub zmieni rozmiar okna. Powoduje to, przed wykonaniem właściwej akcji, wygenerowanie jednego z następujących zdarzeń:

EVENT:Move	EVENT:Size	EVENT:Restore
EVENT:Maximize	EVENT:Iconize	

Jeśli kod obsługujący te zdarzenia opiera się na wykona instrukcję CYCLE, przypisana im standardowa akcja nie zostanie wykonana. W ten sposób możemy uniemożliwić użytkownikowi przesuwanie lub zmianę rozmiarów okna.

Gdy tylko akcja zostanie wykonana, jest generowane jedno lub kilka z poniższych zdarzeń:

EVENT:Moved	EVENT:Sized	EVENT:Restored
EVENT:Maximized	EVENT:Iconized	

Wiele zdarzeń poakcyjnych może być generowanych z tego powodu, że niektóre akcje dają kilka efektów. Na przykład, gdy użytkownik kliknie przycisk maksymalizujący, jest generowane zdarzenie EVENT:Maximize. Jeśli nie występuje instrukcja CYCLE w obsłudze tego zdarzenia, jest podejmowana odpowiednia akcja, a następnie są generowane zdarzenia EVENT:Maximized, EVENT:Moved oraz EVENT:Sized. Jest tak dlatego, bo maksymalizacja okna pociąga za sobą zmianę jego położenia i rozmiaru.

Zastosowanie w kontrolce

W przypadku kontrolki REGION atrybut IMM powoduje generowanie zdarzenia za każdym razem, gdy wskaźnik myszki znajdzie się nad obszarem (EVENT:MouseIn), porusza się w obszarze (EVENT:MouseMove) lub opuszcza obszar (EVENT:MouseOut) wyznaczony przez jej atrybut AT. Dokładna pozycja wskaźnika myszki może zostać odczytana za pomocą funkcji MOUSEX oraz MOUSEY.

W przypadku kontrolki BUTTON atrybut IMM powoduje generowanie zdarzenia EVENT:Accepted w momencie, gdy lewy przycisk myszki został wciśnięty na kontrolce. Zdarzenie to jest generowane tak długo, jak długo przycisk ten pozostaje wciśnięty. Atrybuty DELAY oraz REPEAT kontrolki BUTTON pozwalają na określenie częstotliwości, z jaką zdarzenie jest generowane.

Atrybut IMM powoduje natychmiastowe generowania zdarzenia za każdym razem, gdy użytkownik używa klawiszy w aktywnej kontrolce LIST lub COMBO, co zazwyczaj pociąga za sobą konieczność odświeżenia kolejki QUEUE z nią związanej. Oznacza to, że wszystkie klawisze są niejawnie zgłaszane jako wymagające obsługi dla danej kolejki (ALRT). Gdy użytkownik wciśnie klawisz znakowy, jest generowane zdarzenie EVENT:NewSelection.

W kontrolkach ENTRY oraz SPIN zdarzenie EVENT:NewSelection jest generowane za każdym razem, gdy zmieni się ich zawartość lub pozycja kursora. Jeśli chcemy wykonać jakieś działanie tylko wtedy, gdy zmienia się zawartość, musimy zachować

poprzednią zawartość i porównywać ją z aktualną (wykorzystując na przykład właściwość PROP:ScreenText).

W przypadku arkuszy zakładek SHEET zdarzenie EVENT:NewSelection jest generowane wtedy, gdy użytkownik kliknie zakładkę TAB (również wtedy, gdy jest to aktualna zakładka). Może to być przydatne wtedy, gdy w jednym oknie mamy kilka kontrolek SHEET.

Przykład:

```
Win2   WINDOW('Some Window'),AT(58,11,174,166),MDI,DOUBLE,MAX,IMM
         LIST,AT(109,48,50,50),USE(?List),FROM('Que'),IMM
         BUTTON('&Ok'),AT(111,108,,),USE(?Ok)
         BUTTON('&Cancel'),AT(111,130,,),USE(?Cancel)
       END

  CODE
  OPEN(Win2)
  ACCEPT
    CASE EVENT()
    OF EVENT:Move                ! uniemożliwienie przesuwania okna
      CYCLE
    OF EVENT:Maximized           ! gdy zmaksymalizowane
      ?List{PROP:Height} = 100   ! zmień rozmiar listy
    OF EVENT:Restored            ! gdy przyrócono poprzendi rozmiar
      ?List{PROP:Height} = 50    ! zmień rozmiar listy
    END
  END
```

Porównaj: RESIZE, MAX, ICON, DELAY, REPEAT

INS, OVR (ustawienie trybu wprowadzania)

INS
OVR

Atrybuty **INS** i **OVR** (PROP:INS oraz PROP:OVR) określają tryb wprowadzania dla kontrolek ENTRY i TEXT okna posiadającego atrybut MASK. Atrybut INS powoduje włączenie trybu wstawiania, atrybut OVR - zastępowania. Tryby te są aktywne tylko w oknach posiadających atrybut MASK.

JOIN (połączone przyciski przewijania zakładek TAB)

JOIN

Atrybut **JOIN** (PROP:JOIN) nadany kontrolce SHEET powoduje, że jej zakładki TAB są wyświetlane w jednej linii zamiast w wielu, niezależnie od tego ile ich jest. Przyciski przewijania zakładek (lewy i prawy bądź górny i dolny) są umieszczane obok siebie, po prawej stronie arkusza.

KEY (ustawienie klawisza skrótu)

KEY(*keycode*)

KEY Określa klawisz skrótu dla kontrolki

keycode Kod klawisza lub ekwiwalent etykiety kodu klawisza.

Atrybut **KEY** (PROP:KEY) wskazuje klawisz skrótu powodujący niezwłoczne przekazanie aktywności wprowadzania do kontrolki lub wykonanie związanej z nią akcji. Wymienione poniżej kontrolki otrzymują aktywność wprowadzania:

COMBO	CUSTOM	ENTRY	GROUP	LIST
OPTION	PROMPT	SPIN	TEXT	

Kontrolki, w przypadku których zostanie wykonana związana z nimi akcja, są następujące:

BUTTON	CHECK	CUSTOM	RADIO	MENU
ITEM.				

Przykład:

```
MainWin  APPLICATION('My Application'),SYSTEM,MAX,ICON('MyIcon.ICO'),STATUS
           MENUBAR
             MENU('&Edit'),USE(?EditMenu)
               ITEM('Cu&t'),USE(?CutText),KEY(CtrlX),STD(STD:Cut),DISABLE
               ITEM('&Copy'),USE(?CopyText),KEY(CtrlC),STD(STD:Copy),DISABLE
               ITEM('&Paste'),USE(?PasteText),KEY(CtrlV),STD(STD:Paste),DISABLE
             END
           END
           TOOLBAR
             COMBO(@S8),AT(120,120,20,20),USE(?C1),FROM(Q1:F2),KEY(F1Key)
             LIST,AT(120,0,20,20),USE(?L1),FROM(Que1),KEY(F2Key)
             SPIN(@N8.2),AT(280,0,20,20),USE(SpinVar1),FROM(Q),KEY(F3Key)
             TEXT,AT(20,0,40,40),USE(E2),KEY(F4Key)
             PROMPT('Enter &Data in E2:'),AT(10,200,20,20),USE(?P2),KEY(F5Key)
             ENTRY(@S8),AT(100,200,20,20),USE(E2),KEY(F6Key)
             BUTTON('&1'),AT(120,0,20,20),USE(?B7),KEY(F7Key)
             CHECK('&A'),AT(0,120,20,20),USE(?C7),KEY(F8Key)
             OPTION('Option'),USE(OptVar),KEY(F9Key)
               RADIO('Radio 1'),AT(120,0,20,20),USE(?R1),KEY(F10Key)
               RADIO('Radio 2'),AT(140,0,20,20),USE(?R2),KEY(F11Key)
             END
           END
         END
```

LANDSCAPE (ustawienie poziomej orientacji strony raportu)

LANDSCAPE

Atrybut **LANDSCAPE** (PROP:LANDSCAPE) nadany raportowi REPORT powoduje, że jest on drukowany domyślnie w orientacji poziomej (landscape). Jeśli atrybut LANDSCAPE został pominięty, drukowanie domyślnie odbywa się w orientacji pionowej (portrait).

Przykład:

```
Report    REPORT,PRE('Rpt'),LANDSCAPE        ! domyślne ustawienie landscape
            ! deklaracje struktury raportu
          END
```

LEFT, RIGHT, ABOVE, BELOW (określenie pozycji zakładki TAB)

LEFT([*width*])
RIGHT([*width*])
ABOVE([*width*])
BELOW([*width*])

width Stała całkowita określająca szerokość zakładki TAB w jednostkach dialogowych. Dla atrybutu LEFT jest to właściwość PROP:LeftOffset (równoważna z {PROP:LEFT,2}). Dla atrybutu RIGHT jest to właściwość PROP:RightOffset (równoważna z {PROP:RIGHT,2}). Dla atrybutu ABOVE jest to właściwość PROP:AboveSize (równoważna z {PROP:ABOVE,2}). Dla atrybutu BELOW jest to właściwość PROP:BelowSize (równoważna z {PROP:BELOW,2}).

Atrybuty **LEFT**, **RIGHT**, **ABOVE** oraz **BELOW** dla kontrolki SHEET określają pozycję jej kontrolek TAB. Atrybut LEFT (PROP:LEFT) powoduje umieszczenie zakładek TAB po lewej stronie arkusza zakładek, RIGHT (PROP:RIGHT) – po prawej stronie, ABOVE (PROP:ABOVE) – powyżej, a BELOW (PROP:BELOW) – poniżej.

Parametr *width* umożliwia określenie rozmiaru kontrolki TAB. Tekst pojawia się w zakładce TAB w orientacji poziomej, chyba, że określimy atrybut UP lub DOWN.

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            SHEET,AT(0,0,320,175),USE(SelectedTab),BELOW                    ! zakładki pod arkuszem
              TAB('Tab One'),USE(?TabOne)
                OPTION('Option 1'),USE(OptVar1),KEY(F10Key),HLP('Option1Help')
                  RADIO('Radio 1'),AT(20,0,20,20),USE(?R1)
                  RADIO('Radio 2'),AT(40,0,20,20),USE(?R2)
                END
                PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P1)
                ENTRY(@S8),AT(100,140,32,20),USE(E1)
                PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P2)
                ENTRY(@S8),AT(100,240,32,20),USE(E2)
              END
              TAB('Tab Two'),USE(?TabTwo)
                PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P3)
                ENTRY(@S8),AT(100,140,32,20),USE(E3)
                PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P4)
                ENTRY(@S8),AT(100,240,32,20),USE(E4)
              END
            END
            BUTTON('Ok'),AT(100,180,20,20),USE(?Ok)
            BUTTON('Anuluj'),AT(200,180,20,20),USE(?Cancel)
          END
```

LEFT, RIGHT, CENTER, DECIMAL (ustawienie wyrównywania)

LEFT([*offset*])
RIGHT([*offset*])
CENTER([*offset*])
DECIMAL([*offset*])

offset Stała całkowita określająca wielkość przesunięcia od punktu wyrównywania. Określona jest w jednostkach dialogowych (o ile nie określono atrybutu THOUS, MM lub POINTS dla raportu REPORT). Dla atrybutu LEFT jest to właściwość PROP:LeftOffset (równoważne z {PROP:LEFT,2}). Dla atrybutu RIGHT jest to właściwość PROP:RightOffset (równoważne z {PROP:RIGHT,2}). Dla atrybutu CENTER jest to właściwość PROP:CenterOffset (równoważne z {PROP:CENTER,2}). Dla atrybutu DECIMAL jest to właściwość PROP:DecimalOffset (równoważne z {PROP:DECIMAL,2}).

Atrybuty **LEFT**, **RIGHT**, **CENTER** oraz **DECIMAL** określają sposób wyrównywania drukowanych danych. LEFT (PROP:LEFT) powoduje wyrównywanie do lewej, RIGHT (PROP:RIGHT) – do prawej, CENTER (PROP:CENTER) – środkowanie, DECIMAL (PROP:DECIMAL) – wyrównywanie do punktu kropki dziesiętnej. W przypadku atrybutu LEFT parametr *offset* określa wcięcie w stosunku do lewej krawędzi. W przypadku atrybutu RIGHT parametr *offset* określa wielkość przesunięcia w stosunku do prawej krawędzi. Parametr *offset* atrybutu CENTER określa przesunięcie w stosunku do środka (liczba ujemna oznacza przesunięcie w lewo, dodatnia – w prawo). Dla atrybutu DECIMAL parametr *offset* określa przesunięcie w prawo w stosunku do punktu kropki dziesiętnej.

Zastosowanie w oknie

Następujące kontrolki pozwalają na stosowanie jedynie atrybutów LEFT lub RIGHT (bez parametru *offset*): BUTTON, CHECK, RADIO.

Następujące kontrolki pozwalają na stosowanie atrybutów LEFT(*offset*), RIGHT(*offset*), CENTER(*offset*), DECIMAL(*offset*): COMBO, ENTRY, LIST, SPIN, STRING.

Kontrolka TEXT umożliwia stosowanie atrybutów LEFT(*offset*), RIGHT(*offset*), CENTER(*offset*).

Zastosowanie w raporcie

Następujące kontrolki pozwalają na stosowanie jedynie atrybutów LEFT lub RIGHT (bez parametru *offset*): CHECK, GROUP, OPTION, RADIO.

Następujące kontrolki pozwalają na stosowanie atrybutów LEFT(*offset*), RIGHT(*offset*), CENTER(*offset*) lub DECIMAL(*offset*): LIST, STRING.

Kontrolka TEXT umożliwia stosowanie atrybutów LEFT, RIGHT oraz CENTER (bez parametru *offset*).

Przykład:

```
Rpt     REPORT,AT(1000,1000,6500,9000),THOUS
Detail    DETAIL,AT(0,0,6500,1000)
            LIST,AT(0,20,100,146),FORMAT('800L'),FROM(Fname),USE(?Show2),LEFT(100)
          END
        END

OknoPierwsze  WINDOW,AT(0,0,160,400)
                COMBO(@S8),AT(120,120,20,20),USE(?C1),FROM(Q1:F2),RIGHT(4)
                LIST,AT(120,0,20,20),USE(?L1),FROM(Que1),CENTER
                SPIN(@N8.2),AT(280,0,20,20),USE(SpinVar1),FROM(Q),DECIMAL(8)
                TEXT,AT(20,0,40,40),USE(E2),LEFT(8)
                ENTRY(@S8),AT(100,200,20,20),USE(E2),LEFT(4)
                CHECK('&A'),AT(0,120,20,20),USE(?C7),LEFT
                OPTION('Option'),USE(OptVar)
                  RADIO('Radio 1'),AT(120,0,20,20),USE(?R1),LEFT
                  RADIO('Radio 2'),AT(140,0,20,20),USE(?R2),RIGHT
                END
              END
```

LINEWIDTH (ustawia grubość linii)

LINEWIDTH(*width*)

LINEWIDTH Określa grubość linii LINE lub grubość ramki kontrolek BOX i ELLIPSE.

width Dodatnia stała całkowita określająca grubość w pikselach.

Atrybut **LINEWIDTH** (PROP:LINEWIDTH) określa grubość linii kontrolki LINE lub grubość ramki kontrolek BOX i ELLIPSE.

Przykład:

```
MojRaport   REPORT,AT(1000,1000,6500,9000),THOUS
Detail        DETAIL,AT(0,0,6500,1000),USE(?DetailOne)
                LINE,AT(105,78,-49,0),USE(?Line1),LINEWIDTH(3)        ! linia grubości 3 pikseli
                BOX,AT(182,27,50,50),USE(?Box1),LINEWIDTH(3)          ! prostokąt z ramką o grubości 3 pikseli
                STRING(@N$11.2),AT(6000,1500,500,500),USE(Pre:F1)
              END
            END

window  WINDOW('Caption'),AT(,,260,100),GRAY
          LINE,AT(105,78,-49,0),USE(?Line1),LINEWIDTH(3)              ! linia grubości 3 pikseli
          BOX,AT(182,27,50,50),USE(?Box1),LINEWIDTH(3)                ! prostokąt z ramką o grubości 3 pikseli
        END
```

LINK (tworzy powiązanie obiektu kontrolki OLE z plikiem)

LINK(*filename*)

LINK Powoduje utworzenie powiązania obiektu kontrolki OLE z plikiem danych specyficznym dla aplikacji serwera OLE.

filename Stała łańcuchowa zawierająca nazwę pliku.

Atrybut **LINK** (PROP:LINK, tylko-do-zapisu) powoduje utworzenie powiązania obiektu kontrolki OLE z plikiem danych specyficznym dla aplikacji serwera OLE. Składnia parametru *filename* musi być w pełni kwalifikowana, tzn. zawierać oznaczenie dysku, ścieżkę dostępu i nazwę pliku, za wyjątkiem sytuacji, gdzie plik występuje w tym samym katalogu, co aplikacja sterująca OLE.

Przykład:

```
OknoPierwsze  WINDOW,AT(0,0,200,200)
                OLE,AT(10,10,160,100),USE(?OLEObject),LINK('Book1.XLS')  ! arkusz Excel
                  MENUBAR
                    MENU('&Clarion App')
                      ITEM('&Deactivate Object'),USE(?DeactOLE)
                    END
                  END
                END
              END
```

MARK (tryb zaznaczania wielu elementów)

MARK(*flag*)

MARK Włącza tryb umożliwiający zaznaczenie wielu elementów.

flag Etykieta pola kolejki QUEUE.

Atrybut **MARK** (PROP:MARK, tylko-do-zapisu) włącza tryb umożliwiający jednoczesne zaznaczenie wielu elementów kontrolki LIST lub COMBO. Gdy element listy LIST jest zaznaczony, odpowiednie pole *flag* jest ustawiane na wartość *prawda* (1). Każdy zaznaczony element jest automatycznie podświetlany w liście LIST lub COMBO. Zmiana wartości pola *flag* pociąga za sobą odświeżenie wyglądu listy LIST lub COMBO.

W listach LIST lub COMBO, w których zastosowano atrybut MARK, nie powinno się stosować atrybutu IMM.

Przykład:

```
Que1      QUEUE
MarkFlag    BYTE
F1          LONG
F2          STRING(8)
          END

OknoPierwsze  WINDOW,AT(0,0,160,400),SYSTEM
                LIST,AT(120,0,20,20),USE(?L1),FROM(Que1.F1),MARK(Que1.MarkFlag)
                COMBO(@S8),AT(120,120,,),USE(?C1),FROM(Que1.F2),MARK(Que1.MarkFlag)
              END
  CODE
  DO LoadQue                          ! załaduj kolejkę Que1 danymi
  OPEN(OknoPierwsze)
  ACCEPT
    CASE EVENT()
    OF EVENT:CloseWindow
      BREAK
    END
  END
  LOOP X# = 1 to RECORDS(Que1)        ! pętla dla elementów kolejki
    GET(Que1,X#)
    IF Que1.MarkFlag                  ! jeśli użytkownik oznaczył element
      DO ProcessMarked                ! przetwarzaj go
    END
  END
```

MASK (włącza tryb kontroli wzorca danych)

MASK

Atrybut **MASK** (PROP:MASK) powoduje włączenie trybu kontroli wzorca przy wprowadzaniu dla określonej kontrolki COMBO, ENTRY lub SPIN bądź wszystkich kontrolek w oknie (gdy umieszczono go w deklaracji okna WINDOW). Przełączanie wartości PROP:MASK dla okna ma wpływ tylko na kontrolki utworzone po tym – nie dotyczy kontrolek już istniejących.

Tryb kontroli wzorca przy wprowadzaniu oznacza, że każdy znak jest automatycznie sprawdzany, czy jest zgodny ze wzorcem określonym dla kontrolki (np. tylko liczby we wzorcach numerycznych itp.). Wymusza to na użytkowniku wprowadzania danych w formacie określonym przez wzorzec wyświetlania kontrolki. Pominięcie atrybutu MASK powoduje dopuszczenie przez Windows do wprowadzania dowolnych znaków. Są one formatowane dopiero wtedy, gdy kontrolka zostanie zatwierdzona przez użytkownika. Dzięki temu użytkownik ma możliwość wprowadzania danych w wygodnym dla siebie formacie. Gdy użytkownik wprowadzi dane w formacie innym niż określa to wzorzec kontrolki, biblioteka uruchomieniowa stara się rozpoznać zastosowany przez niego format i przekonwertować go do formatu zgodnego ze wzorcem wyświetlania kontrolki. Na przykład, jeśli użytkownik wpisze “January 1, 1995” w kontrolce o wzorcu @D1, biblioteka uruchomieniowa sformatuje wprowadzone dane do postaci “1/1/95”. To działanie zachodzi po wprowadzeniu danych i przejściu do innej kontrolki. Jeśli biblioteka uruchomieniowa nie jest w stanie rozpoznać użytego formatu, nie aktualizuje ona wartości zmiennej USE, z którą jest związana kontrolka. Rozlega się wówczas sygnał dźwiękowy, a kursor pozostaje w kontrolce oczekując na wpisanie poprawnych danych.

Przykład:

```
! okno z włączoną kontrolą wzorca wprowadzania
Win2   WINDOW,MASK
       END

! kontrolki z włączoną kontrolą wzorca wprowadzania
Win2   WINDOW
         COMBO(@P(###) ###-####P),AT(120,120,20,20),USE(Phone),FROM(Q1:F2),MASK
         SPIN(@N8.2),AT(280,0,20,20),USE(SpinVar1),FROM(Q),MASK
         ENTRY(@D2),AT(100,200,20,20),USE(DateField),MASK
       END
```

MAX (kontrolka maksymalizacji lub wyliczenie maksimum w raporcie)

MAX([*variable*])

MAX Powoduje wyświetlanie przycisku maksymalizującego w oknie APPLICATION lub WINDOW bądź wylicza wartość maksymalną dla kontrolek STRING w oparciu o wartości zmiennej występującej w polu USE tej kontrolki.

variable Etykieta zmiennej numerycznej, w której będą przechowywane wartości pośrednie wyliczane przez AVE. Umożliwia to tworzenie podliczeń i podliczeń częściowych. Wartość w zmiennej *variable* jest wewnętrznie aktualizowana przez mechanizm drukowania, z tego względu można stosować ten sposób tylko w strukturach raportu REPORT.

Atrybut **MAX** (PROP:MAX) powoduje umieszczenie przycisku Maksymalizuj w oknie APPLICATION lub WINDOW bądź wylicza wartość maksymalną dla zmiennych USE związanej z kontrolką STRING w raporcie REPORT.

Zastosowanie w oknie

Przycisk Maksymalizuj pojawia się w prawym, górnym rogu okna. Gdy użytkownik go wciśnie okno APPLICATION lub okno WINDOW nie posiadające atrybutu MDI jest powiększane na cały pulpit Windows. Okno WINDOW z atrybutem MDI jest powiększane na cały dostępny obszar roboczy okna APPLICATION. Po powiększeniu okna, przycisk Maksymalizuj jest zastępowany automatycznie przyciskiem Przywróć, który przywraca poprzednie rozmiary okna.

Zastosowanie w raporcie

Atrybut MAX powoduje drukowanie maksymalnej wartości zmiennej występującej w polu USE kontrolki typu STRING raportu. O ile nie występuje atrybut TALLY, rezultat jest wyliczany zgodnie z poniższymi zasadami:

- Pole MAX znajdujące się w strukturze DETAIL jest wyliczane za każdym razem, gdy struktura DETAIL zawierająca kontrolkę jest drukowana.
- Pole MAX znajdujące się w stopce FOOTER grupy jest wyliczane za każdym razem, gdy jest drukowana dowolna struktura DETAIL zawarta w strukturze BREAK zawierającej kontrolkę. Pozwala na drukowanie wartości maksymalnej w grupie.
- Pole MAX znajdujące się w stopce FOOTER strony jest wyliczane za każdym razem, gdy jest drukowana dowolna struktura DETAIL w dowolnej strukturze BREAK. Pozwala na określenie wartości maksymalnej na stronie.
- Pole MAX znajdujące się w nagłówku HEADER nie ma sensu, gdyż żaden detal nie jest drukowany w czasie drukowania nagłówka.

Wartość maksymalna MAX jest resetowana wtedy, gdy są określone dla niej atrybuty RESET lub PAGE.

Przykład:

```
! okno z przyciskiem Maksymalizuj:
Win2  WINDOW,MAX
      END

MojRaport  REPORT,AT(1000,1000,6500,9000),THOUS
Grupa1       BREAK(LocalVar),USE(?BreakOne)
Grupa2       BREAK(Pre:Key1),USE(?BreakTwo)
Detail       DETAIL,AT(0,0,6500,1000),USE(?DetailOne)
               STRING(@N$11.2),AT(6000,1500,500,500),USE(Pre:F1)
             END
             FOOTER,AT(0,0,6500,1000),USE(?BreakOneGroupFoot)
               STRING('Group Maximum:'),AT(5500,500)
               STRING(@N$11.2),AT(6000,500),USE(Pre:F1),MAX(LocalVar),RESET(Grupa2)
             END
           END
           FOOTER,AT(0,0,6500,1000),USE(?BreakOneGroupFoot)
             STRING('Grand Maximum:'),AT(5500,500)
             STRING(@N$11.2),AT(6000,500),USE(LocalVar),MAX,TALLY(?BreakTwo)
           END
         END
       END
```

Porównaj: ICONIZE, ICON, MAXIMIXE, IMM, TALLY, RESET, PAGE

MAXIMIZE (otwarcie okna w postaci zmaksymalizowanej)

MAXIMIZE

Atrybut **MAXIMIZE** (PROP:MAXIMIZE) powoduje otwarcia okna APPLICATION lub WINDOW w postaci zmaksymalizowanej. Zmaksymalizowane okno APPLICATION lub okno WINDOW pozbawione atrybutu MDI zajmuje cały pulpit Windows. Okno WINDOW posiadające atrybut MDI zajmuje cały obszar roboczy aplikacji APPLICATION.

Za pomocą właściwości PROP:MAXIMIZE wbudowanej zmiennej SYSTEM możemy określić ustawienia określone w pliku Windows PIF dla danej aplikacji. Jeśli właściwość ta jest równa 1, to okno aplikacji jest otwierane w postaci zmaksymalizowanej.

Przykład:

```
! okno z przyciskiem Maksymalizuj, otwarte w postaci zmaksymalizowanej:
Win2  WINDOW,MAX,MAXIMIZE
      END
```

Porównaj: MAX, IMM

MDI (okno wewnętrzne MDI)

MDI

Atrybut **MDI** (PROP:MDI, tylko-do-odczytu) określa, że dane okno WINDOW jest oknem wewnętrznym aplikacji APPLICATION. Okna MDIS są otwierane w obszarze roboczym okna aplikacji. Są one automatycznie przemieszczane wraz z przemieszczanym oknem aplikacji APPLICATION, znikają, gdy okno aplikacji jest minimalizowane. Okno WINDOW z atrybutem MDIS nie może zostać otwarte dopóty, dopóki nie zostanie uaktywniona aplikacja APPLICATION.

Okna niezależne

Wewnętrzne okna MDIS są od siebie niezależne; użytkownik może się przełączyć na inne okno wewnętrzne tej samej lub innej aplikacji. Okno MDI nie może działać w tym samym wątku co aplikacja APPLICATION. Z tego powodu dowolne okno wywoływane bezpośrednio z poziomu aplikacji APPLICATION musi być uruchamiane w postaci oddzielnego wątku za pomocą procedury START.

Okna modalne aplikacji

Okna WINDOW bez atrybutu MDI operuję niezależnie od dowolnych otwartych wcześniej aplikacji APPLICATION. Mogą one jednakże wstrzymać działanie aplikacji APPLICATION jeśli ona lub dowolne z jej okien wewnętrznych MDI znajduje się w tym samym wątku, co okno bez atrybutu MDI. Okno bez atrybutu MDI otwarte w wątku MDI staje się oknem modalnym aplikacji wstrzymującym działanie aplikacji do momentu, gdy nie zostanie zamknięte (za wyjątkiem sytuacji, gdy jest otwierane we własnym, oddzielnym wątku). Nie blokuje się w ten sposób możliwości działania na innych aplikacjach działających w danym momencie w Windows, użytkownik może się bez przeszkód do nich przełączać. Okno MDI nie może być otwarte w tym samym wątku, co już otwarte okno pozbawione atrybutu MDI.

Przykład:

```
Win2  WINDOW,MDI          ! okno wewnętrzne MDI
      END
```

Porównaj: MODAL, THREAD

META (drukowanie .VBX w postaci metapliku .WMF)

META

Atrybut **META** (PROP:META) powoduje drukowanie kontrolki .VBX jako metapliku Windows (.WMF), zamiast w postaci mapy bitowej. Powoduje to na ogół uzyskanie lepszej jakości wydrukowanej grafiki, mniejszą zajętość pamięci i szybsze drukowanie. Z drugiej jednak strony nie wszystkie kontrolki VBX obsługują metapliki. Jeśli nie jesteśmy pewni, czy dana kontrolka VBX obsługuje wyjście META, powinniśmy przeprowadzić test i na jego podstawie pozostawić lub usunąć atrybut META.

MIN (wyliczenie wartości minimalnej w raporcie)

MIN([*variable*])

MIN Wylicza wartość minimalną dla kontrolek STRING w oparciu o wartości zmiennej występującej w polu USE tej kontrolki.

variable Etykieta zmiennej numerycznej, w której będą przechowywane wartości pośrednie wyliczane przez AVE. Umożliwia to tworzenie podliczeń i podliczeń częściowych. Wartość w zmiennej *variable* jest wewnętrznie aktualizowana przez mechanizm drukowania, z tego względu można stosować ten sposób tylko w strukturach raportu REPORT.

Atrybut **MIN** (PROP:MIN) powoduje drukowanie minimalnej wartości zmiennej występującej w polu USE kontrolki typu STRING raportu. O ile nie występuje atrybut TALLY, rezultat jest wyliczany zgodnie z poniższymi zasadami:

- Pole MIN znajdujące się w strukturze DETAIL jest wyliczane za każdym razem, gdy struktura DETAIL zawierająca kontrolkę jest drukowana.
- Pole MIN znajdujące się w stopce FOOTER grupy jest wyliczane za każdym razem, gdy jest drukowana dowolna struktura DETAIL zawarta w strukturze BREAK zawierającej kontrolkę. Pozwala na drukowanie wartości minimalnej w grupie.
- Pole MIN znajdujące się w stopce FOOTER strony jest wyliczane za każdym razem, gdy jest drukowana dowolna struktura DETAIL w dowolnej strukturze BREAK. Pozwala na określenie wartości minimalnej na stronie.
- Pole MIN znajdujące się w nagłówku HEADER nie ma sensu, gdyż żaden detal nie jest drukowany w czasie drukowania nagłówka.

Wartość maksymalna MIN jest resetowana wtedy, gdy są określone dla niej atrybuty RESET lub PAGE.

Przykład:

```
MojRaport  REPORT,AT(1000,1000,6500,9000),THOUS
Grupa1       BREAK(LocalVar),USE(?BreakOne)
Grupa2         BREAK(Pre:Key1),USE(?BreakTwo)
Detail           DETAIL,AT(0,0,6500,1000),USE(?DetailOne)
                   STRING(@N$11.2),AT(6000,1500,500,500),USE(Pre:F1)
                 END
                 FOOTER,AT(0,0,6500,1000),USE(?BreakOneGroupFoot)
                   STRING('Group Minimum:'),AT(5500,500)
                   STRING(@N$11.2),AT(6000,500),USE(Pre:F1),MIN(LocalVar),RESET(Grupa2)
                 END
               END
               FOOTER,AT(0,0,6500,1000),USE(?BreakOneGroupFoot)
                 STRING('Grand Minimum:'),AT(5500,500)
                 STRING(@N$11.2),AT(6000,500),USE(LocalVar),MIN,TALLY(?BreakTwo)
               END
             END
           END
```

MODAL (systemowe okno modalne)

MODAL

Atrybut **MODAL** (PROP:MODAL, tylko-do-odczytu) określa, że okno WINDOW jest systemowym oknem modalnym dla programów 16-bitowych. Oznacza to, że żadne inne okno (w tym samym lub innym programie) nie może stać się oknem aktywnym – okno MODAL przejmuje wyłączną kontrolę nad komputerem. Okna MODAL są zazwyczaj stosowane do wyświetlania komunikatów o błędach lub komunikatów wymagających określonej reakcji użytkownika, np. „Włóż dyskietkę do napędu A:".

Zastosowanie MODAL w aplikacjach 32-bitowych nie przynosi efektu. Biblioteka Microsoft Win32 API nie obsługuje systemowych okien modalnych.

Okna modalne aplikacji

Okno WINDOW bez atrybutu MODAL może być oknem modalnym aplikacji lub oknem niezależnym. Okno modalne aplikacji to okno bez atrybutu MDI otwarte jako aktywne okno wątku MDI. Okno modalne aplikacji nie pozwala użytkownikowi na przejście do innego wątku w tej samej aplikacji, nie blokuje jednak możliwości przełączenia się na inny program Windows.

Okna niezależne

Oknem niezależnym jest okno wewnętrzne MDI nie posiadające atrybutu MODAL. Pozwala ono na przełączanie się na inne okna za pomocą myszki, klawiatury, poleceń systemu menu. Gdy to nastąpi, nowe okno otrzymuje sterowanie i pojawia się na pierwszym planie. Dowolne okno, które nie znajduje się na czele listy okien swojego wątku nie może otrzymać sterowania, nawet od okna niezależnego.

Przykład:

```
Win2  WINDOW,MODAL          ! okno modalne dla systemu
      END
```

Porównaj: MDI, THREAD

MSG (komunikat paska stanu)

MSG(*text*)

MSG Określa tekst *text* przeznaczony do wyświetlania w pasku stanu aplikacji.

text Stała łańcuchowa zawierająca komunikat wyświetlany w pasku stanu aplikacji.

Atrybut **MSG** (PROP:MSG) określa *text* przeznaczony do wyświetlania w pierwszej sekcji paska stanu aplikacji. W deklaracji kontrolki, MSG definiuje *text*, który ma być wyświetlany w momencie, gdy kontrolka posiada aktywność wprowadzania. Dla kontrolek, które nie mogą posiadać stałej aktywności (z atrybutem SKIP lub umieszczonych w pasku narzędzi TOOLBAR lub w oknie z atrybutem TOOLBOX), *text* jest wyświetlany wtedy, gdy zatrzymamy wskaźnik myszki na kontrolce.

W strukturach APPLICATION lub WINDOW, MSG określa *text*, który będzie wyświetlany w pierwszej sekcji pasku stanu w przypadku, gdy aktywne będą kontrolki nie posiadające własnego atrybutu MSG.

Przykład:

```
MainWin  APPLICATION('My Application'),SYSTEM,MAX,ICON('MyIcon.ICO'),STATUS
           MENUBAR
             MENU('&File'),USE(?FileMenu)
               ITEM('&Open...'),USE(?OpenFile),MSG('Open a file')
               ITEM('&Close'),USE(?CloseFile),DISABLE,MSG('Close the open file')
               ITEM(),SEPARATOR
               ITEM('E&xit'),USE(?MainExit),MSG('Exit the program')
             END
           END
         END

OknoPierwsze  WINDOW,AT(0,0,160,400),MSG('Enter Data')                    ! domyślny MSG do użycia
                COMBO(@S8),AT(120,120,20,20),USE(?C1),FROM(Q1:F2),MSG('Enter or Select')
                TEXT,AT(20,0,40,40),USE(E2)                               ! użyty domyślny MSG
                ENTRY(@S8),AT(100,200,20,20),USE(E2)                      ! użyty domyślny MSG
                CHECK('&A'),AT(0,120,20,20),USE(?C7),MSG('On or Off')
                OPTION('Option 1'),USE(OptVar),MSG('Pick One or Two')
                  RADIO('Radio 1'),AT(120,0,20,20),USE(?R1)
                  RADIO('Radio 2'),AT(140,0,20,20),USE(?R2)
                END
              END
```

Porównaj: STATUS

NOBAR (brak podświetlenia)

NOBAR

Atrybut **NOBAR** (PROP:NOBAR) powoduje, że aktualnie wybrany element w liście LIST jest podświetlany tylko wtedy, gdy kontrolka LIST jest aktywna.

NOCASE (grupowanie BREAK niezależne od wielkości liter)

NOCASE

Atrybut **NOCASE** (PROP:NOCASE) powoduje, że struktura grupująca BREAK raportu REPORT jest niezależna od wielkości liter. Porównanie sprawdzające, czy nie zmieniła się wartość decydująca o grupowaniu, traktuje wówczas jednakowo duże i małe litery ASCII. Wszystkie znaki w polu grupującym i zapisanej wartości porównywanej są zamieniane na wielkie litery przed dokonaniem porównania. Ta konwersja nie ma oczywiście wpływu na wielkość liter danych zapisanych w pliku. Atrybut NOCASE nie daje żadnego efektu dla znaków nie będących znakami alfabetu.

Przykład:

```
Report   REPORT
           BREAK(BreakVariable),NOCASE          ! grupowanie niezależne od wielkości liter
             HEADER
               STRING(@n4),USE(BreakVariable)
             END
Detail       DETAIL
               STRING(@n4),USE(SomeField)
             END
           END
         END
```

Porównaj: BREAK

NOMERGE (brak scalania menu)

NOMERGE

Atrybut **NOMERGE** (PROP:NOMERGE) oznacza, że menu MENUBAR lub pasek narzędzi TOOLBAR okna WINDOW nie powinien być łączony globalnym menu lub globalnym paskiem narzędzi.

Atrybut NOMERGE umieszczony w menu MENUBAR aplikacji APPLICATION powoduje, że staje się ono menu lokalnym i jest wyświetlane tylko wtedy, gdy są otwierane okna bez atrybutu MDI oraz, że nie występuje menu globalne. Atrybut NOMERGE w pasku narzędzi TOOLBAR aplikacji APPLICATION oznacza, że pasek narzędzi jest lokalny i jest wyświetlany tylko wtedy, gdy są otwierane okna bez atrybutu MDI oraz, że nie występuje globalny pasek narzędzi.

Jeśli okna MDI nie posiadają atrybutu NOMERGE, ich paski menu i paski narzędzi są automatycznie łączone z menu globalnym i globalnym paskiem narzędzi i są następnie wyświetlane w oknie aplikacji APPLICATION. Gdy atrybut NOMERGE jest nadany, pasek menu i pasek narzędzi okna WINDOW zastępują globalne menu I globalny pasek narzędzi. Wówczas w oknie aplikacji są wyświetlane system menu i pasek narzędzi okna posiadającego aktywność.

Pasek menu MENUBAR lub pasek narzędzi TOOLBAR zdefiniowane dla okna WINDOW bez atrybutu MDI nigdy nie są scalane z globalnym menu I globalnym paskiem narzędzi – pojawiają się one w oknie WINDOW.

Przykład:

```
! okno sterujące aplikacji MDI z lokalnym menu i paskiem narzędzi:
MainWin  APPLICATION('My Application'),SYSTEM,MAX,ICON('MyIcon.ICO'),STATUS
           MENUBAR,NOMERGE
             ITEM('E&xit'),USE(?MainExit)
           END
           TOOLBAR,NOMERGE
             BUTTON('Exit'),USE(?MainExitButton)
           END
         END

! okno MDI z własnym menu i paskiem narzędzi zastępującym menu I pasek narzędzi aplikacji:
MDIChild  WINDOW('Dialog Window'),MDI,SYSTEM,MAX,STATUS
            MENUBAR,NOMERGE
              MENU('Edit'),USE(?EditMenu)
                ITEM('Cu&t'),USE(?CutText),KEY(CtrlX),STD(STD:Cut)
                ITEM('Copy'),USE(?CopyText),KEY(CtrlC),STD(STD:Copy)
                ITEM('Paste'),USE(?PasteText),KEY(CtrlV),STD(STD:Paste)
              END
            END
            TOOLBAR,NOMERGE
              BUTTON('Cut'),USE(?CutButton),STD(STD:Cut)
              BUTTON('Copy'),USE(?CopyButton),STD(STD:Copy)
              BUTTON('Paste'),USE(?PasteButton),STD(STD:Paste)
            END
            TEXT,HVSCROLL,USE(Pre:Field),MSG('Enter some text here')
            BUTTON('&OK'),USE(?Exit),DEFAULT
          END
```

Porównaj: MENUBAR, TOOLBAR

NOSHEET (ukrycie arkusza zakładek TAB)

NOSHEET

Atrybut **NOSHEET** (PROP:NOSHEET) nadany kontrolce SHEET powoduje, że zakładki TAB są wyświetlane bez widocznego arkusza otaczającego inne kontrolki.

OPEN (otwarcie obiektu kontrolki OLE z pliku)

OPEN(*object*)

OPEN Powoduje otwarcie zapisanego obiektu dla kontrolki OLE z pliku OLE Compound Storage.

object Stała łańcuchowa zawierająca nazwę pliku OLE Compound Storage oraz jego obiektu, który chcemy otworzyć

Atrybut **OPEN** (PROP:OPEN, tylko-do-zapisu) powoduje otwarcie zapisanego w pliku OLE Compound Storage obiektu *object* dla kontrolki OLE. Gdy obiekt zostaje otwarty , zapisana wersja właściwości kontenera jest powtórnie ładowana, tak więc nie ma potrzeby określania właściwości dla otwartego obiektu. Składnia parametrów obiektu *object* musi być określania w formie: *NazwaPliku\!NazwaObiektu*.

Przykład:

```
OknoPierwsze  WINDOW,AT(0,0,200,200)
                OLE,AT(10,10,160,100),USE(?OLEObject),OPEN('SavFile.OLE\!MyObject')
                  MENUBAR
                    MENU('&Clarion App')
                      ITEM('&DeactivateObject'),USE(?DeactOLE)
                    END
                  END
                END
              END
```

PAGE (resetowanie wyliczeń po stronie raportu)

PAGE

Atrybut **PAGE** (PROP:PAGE) powoduje, że pola wyliczeniowe CNT, SUM, AVE, MIN oraz MAX są zerowane po każdej stronie raportu.

PAGEAFTER (wymuszenie nowej strony po wydrukowaniu struktury)

PAGEAFTER([*newpage*])

PAGEAFTER Powoduje drukowanie nowej strony, po wydrukowaniu struktury.

newpage Stała całkowita lub wyrażenie stałe określające numer strony drukowany na następnej stronie (PROP:PageAfterNum, równoważne z {PROP:PageAfter,2}). Jeśli ma wartość zero (0) lub został pominięty, wymuszenie drukowania nowej strony nie zachodzi. Wartość minus jeden (-1) powoduje, że nowy numer strony jest kontynuacją dotychczasowej numeracji.

Atrybut **PAGEAFTER** (PROP:PAGEAFTER) powoduje, że wydrukowanie sekcji DETAIL lub nagłówka grupy HEADER bądź stopki grupy FOOTER pociąga za sobą wymuszenie drukowania nowej strony. Oznacza to, że struktura, dla której określono atrybut PAGEAFTER jest drukowana, po niej są drukowane stopka strony FOOTER, podkład strony FORM oraz nagłówek nowej strony HEADER.

Parametr *newpage*, o ile występuje, resetuje automatyczną numerację stron i wznawia ją od określonego numeru.

Przykład:

```
MojRaport   REPORT
              HEADER
                ! elementy struktury
              END
Grupa1        BREAK(SomeVariable)
                HEADER
                  ! elementy struktury
                END
CustDetail      DETAIL
                  ! elementy struktury
                END
                FOOTER,PAGEAFTER(-1)          ! stopka strony, inicjuje przepełnienie
                  ! elementy struktury
                END
              END
              FOOTER
                ! elementy struktury
              END
            END
```

PAGEBEFORE (wymuszenie nowej strony przed drukowaniu struktury)

PAGEBEFORE([*newpage*])

PAGEBEFORE Powoduje drukowanie struktury na nowej stronie.

newpage Stała całkowita lub wyrażenie stałe określające numer strony drukowany na następnej stronie (PROP:PageBeforeNum, równoważne z {PROP:PageBefore,2}). Jeśli ma wartość zero (0) lub został pominięty, wymuszenie drukowania nowej strony nie zachodzi. Wartość minus jeden (-1) powoduje, że nowy numer strony jest kontynuacją dotychczasowej numeracji.

Atrybut **PAGEBEFORE** (PROP:PAGEBEFORE) powoduje, że wydrukowanie sekcji DETAIL lub nagłówka grupy HEADER bądź stopki grupy FOOTER ma miejsce zawsze na nowej stronie. Oznacza to, że najpierw jest drukowana stopka strony FOOTER, następnie jej podkład FORM oraz nagłówek nowej strony HEADER. Po tym dopiero jest drukowana struktura, dla której określono atrybut PAGEBEFORE

Parametr *newpage*, o ile występuje, resetuje automatyczną numerację stron i wznawia ją od określonego numeru.

Przykład:

```
MojRaport  REPORT
             HEADER
               ! elementy struktury
             END
Grupa1       BREAK(SomeVariable)
               HEADER,PAGEBEFORE(-1)      ! nagłówek grupy, inicjuje przepełnienie
                 ! elementy struktury
               END
CustDetail     DETAIL
                 ! elementy struktury
               END
               FOOTER
                 ! elementy struktury
               END
             END
             FOOTER
               ! elementy struktury
             END
           END
```

PAGENO (drukowanie numeru strony raportu)

PAGENO

Atrybut **PAGENO** (PROP:PAGENO) powoduje, że w kontrolce STRING jest drukowany aktualny numer strony raportu.

PALETTE (określa liczbę kolorów sprzętowych)

PALETTE(*colors*)

PALETTE Określa liczbę kolorów sprzętowych wykorzystywanych w oknie.

colors Stała całkowita określająca liczbę kolorów sprzętowych wyświetlanych w oknie.

Atrybut **PALETTE** (PROP:PALETTE) w oknie APPLICATION lub WINDOW określa ile kolorów palety sprzętowej jest wykorzystywanych w danym oknie. Stosuje się to tylko w trybach sprzętowych, w których jest wykorzystywana paleta oraz są dostępne zapasowe kolory (nie zarezerwowane przez system) – w praktyce oznacza to tryb 256 kolorów. Wymusza to stosowanie szczególnego zestawu kolorów dla grafiki. Grafika 24-bitowa (16.7M kolorów) nie korzysta z palety sprzętowej. Wartości PALETTE powyżej 256 nie są zalecane.

Wartość zwracana przez właściwość PROP:PALETTE jest liczbą aktualnie przypisanych kolorów. Ponieważ system standardowo rezerwuje 20 kolorów w trybie 256 kolorów, ustawienie PROP:PALETTE = 256 i natychmiastowe odczytanie wartości da nam w rezultacie 236.

Przykład:

```
OknoPierwsze  WINDOW,AT(0,0,160,400),PALETTE(256)          ! wyświetla 256 kolorów
                IMAGE,AT(120,120,20,20),USE(ImageField)
              END
```

Porównaj: IMAGE

PAPER (określa rozmiar papieru dla raportu)

PAPER([*type*] [, *width*] [, *height*])

PAPER Definiuje rozmiar papieru dla raportu.

type Stała całkowita lub ekwiwalent EQUATE identyfikujący standardowy dla Windows rozmiar papieru. Odpowiednie ekwiwalenty EQUATES są umieszczone w pliku PRNPROP.CLW.

width Stała całkowita lub wyrażenie stałe określające szerokość papieru (PROPPRINT:paperwidth, równoważne z {PROPPRINT:PAPER,2}).

height Stała całkowita lub wyrażenie stałe określające wysokość papieru (PROPPRINT:paperheight, równoważne z {PROPPRINT:PAPER,3}).

Atrybut **PAPER** (PROPPRINT:PAPER) w strukturze REPORT definiuje rozmiar papieru stosowany przez raport. Parametry *width* i *height* są wymagane tylko wtedy, gdy jako typ *type* został wskazany ekwiwalent PAPER:User. Należy pamiętać, że nie wszystkie drukarki obsługują wszystkie rozmiary papierów.

Wartości zawarte w parametrach *width* oraz *height* są domyślnie określane w jednostkach dialogowych, o ile nie został określony atrybut THOUS, MM lub POINTS. Jednostki dialogowe są zdefiniowane jako jedna czwarta średniej szerokości znaku na jedną ósmą średniej wysokości znaku. Rozmiar jednostki dialogowej zależy od domyślnej czcionki raportu. Taki układ miar bazuje na czcionce wskazanej przez atrybut FONT raportu lub na domyślnej czcionce drukarki.

Przykład:

```
Raport1   REPORT,AT(1000,1000,6500,9000),THOUS,PAPER(PAPER:Custom,8500,7000)
            ! drukuje na papierze 8.5" x 7"
            ! deklaracje raportu
          END

Raport2   REPORT,AT(72,72,468,648),POINTS,PAPER(PAPER:A4)
            ! drukuje na papierze A4
            ! deklaracje raportu
          END
```

PASSWORD (maskowanie wprowadzanych znaków)

PASSWORD

Atrybut **PASSWORD** (PROP:PASSWORD) powoduje, że znaki wprowadzane do kontrolki ENTRY nie są wyświetlane (są automatycznie zastępowane znakami *). Standardowe funkcje Wytnij (Cut) i Kopiuj (Copy) systemu Windows są wyłączone dla kontrolek posiadających atrybut PASSWORD.

PREVIEW (wysłanie raportu do metaplików)

PREVIEW(*queue*)

PREVIEW Powoduje, że raport jest kierowany do metaplików Windows; każda strona w oddzielnym pliku.

queue Etykieta kolejki QUEUE lub pola w kolejce QUEUE, w której są zapisane nazwy metaplików.

Atrybut **PREVIEW** (PROP:PREVIEW, tylko-do-zapisu) w raporcie REPORT powoduje wysłanie raportu do metaplików Windows; dla każdej strony raportu oddzielny plik. Atrybut PREVIEW wskazuje kolejkę *queue*, w której są umieszczane nazwy tworzonych metaplików. Pliki te są tymczasowymi plikami tworzonymi wewnętrznie przez bibliotekę Clariona (ich nazwy zawierają kompletne ścieżki dostępu – do 64 znaków). Są one usuwane z dysku w momencie zamknięcia raportu za pomocą instrukcji CLOSE; o ile nie określiliśmy właściwości PROP:TempNameFunc pozwalającej na podanie własnych nazw dla plików tymczasowych.

Jest możliwe utworzenie okna i umieszczenie w nim kontrolki IMAGE wyświetlającej strony raportu. Właściwości {PROP:text} kontrolki IMAGE przypisujemy wówczas kolejkę *queue* zawierającą nazwy utworzonych plików tymczasowych. W ten sposób użytkownik aplikacji może obejrzeć raport przed jego wydrukowaniem. Włączona właściwość {PROP:flushpreview} powoduje wysłanie metaplików na drukarkę.

Przykład:

```
SomeReport  PROCEDURE
WMFQue      QUEUE                   ! kolejka zawierająca nazwy plików .WMF
PageImage     STRING(64)
            END
NextEntry   BYTE(1)                 ! licznik elementów kolejki

Report      REPORT,PREVIEW(WMFQue.PageImage)   ! raport z atrybutem PREVIEW
DetailOne     DETAIL
                ! kontrolki raportu
              END
            END

ViewReport  WINDOW('View Report'),AT(0,0,320,200),MDI,MAX,HVSCROLL
              IMAGE(''),AT(0,0,320,180),USE(?ImageField)
              BUTTON('View Next Page'),AT(0,180,60,20),USE(?NextPage),DEFAULT
              BUTTON('Print Report'),AT(80,180,60,20),USE(?PrintReport)
              BUTTON('Exit Without Printing'),AT(160,180,60,20),USE(?ExitReport)
            END
  CODE
  OPEN(Report)
  SET(SomeFile)                                  ! kod generujący raport
  LOOP
    NEXT(SomeFile)
    IF ERRORCODE() THEN BREAK.
    PRINT(DetailOne)
  END
  ENDPAGE(Report)
  OPEN(ViewReport)                               ! otwarcie okna podglądu raportu
  GET(WMFQue,NextEntry)                          ! pobranie pierwszego elementu kolejki
  ?ImageField{PROP:text} = WMFQue.PageImage      ! załadowanie pierwszej strony raportu
  ACCEPT
    CASE ACCEPTED()
    OF ?NextPage
      NextEntry += 1                             ! inkrementacja licznika elementów
```

```
    IF NextEntry > RECORDS(WMFQue) THEN CYCLE.  ! kontrola końca raportu
    GET(WMFQue,NextEntry)                       ! pobranie następnego elementu kolejki
    ?ImageField{PROP:text} = WMFQue.PageImage   ! załadowanie następnej strony raportu
    DISPLAY                                     ! i wyświetlenie jej
  OF ?PrintReport
    Report{PROP:flushpreview} = TRUE            ! wymiecenie plików do drukarki
    BREAK                                       ! i wyjście z procedury
  OF ?ExitReport
    BREAK                                       ! wyjście z procedury
  END
END
CLOSE(ViewReport)                               ! zamknięcie okna
FREE(WMFQue)                                    ! zwolnienie kolejki
CLOSE(Report)                                   ! zamkniecie raportu (usunięcie wszystkich .WMF)
RETURN                                          ! zwrot sterowania
```

RANGE (określenie ograniczeń zakresu)

RANGE(*lower*,*upper*)

RANGE Określa prawidłowy zakres wartości danych, które mogą być wprowadzane przez użytkownika w kontrolce SPIN lub zakres danych wyświetlanych w kontrolce PROGRESS.

lower Stała numeryczna określająca dolną granicę (włącznie) prawidłowych wartości danych (PROP:RangeLow, równoważne z {PROP:Range,1}).

upper Stała numeryczna określająca górną granicę (włącznie) prawidłowych wartości danych (PROP:RangeHigh, równoważne z {PROP:Range,2}).

Atrybut **RANGE** (PROP:RANGE) określa prawidłowe ograniczenia wartości danych dla określonego zakresu w kontrolce SPIN. RANGE definiuje również zakres wartości wyświetlanych w kontrolce PROGRESS. Atrybut ten działa w połączeniu z atrybutem STEP kontrolki SPIN. W kontrolce SPIN atrybut STEP umożliwia użytkownikowi operowanie na wartościach z prawidłowego zakresu. Właściwość PROP:RangeHigh daje w rezultacie „+Nieskończoność" jeśli nie ustawiono atrybutu RANGE. Właściwość PROP:RangeLow daje w rezultacie „-Nieskończoność" jeśli nie ustawiono atrybutu RANGE

Przykład:

```
OknoPierwsze  WINDOW,AT(0,0,160,400)
                SPIN(@N4.2),AT(280,0,20,20),USE(SpinVar1),RANGE(.05,9.95),STEP(.05)
                SPIN(@n3),AT(280,0,20,20),USE(SpinVar2),RANGE(5,995),STEP(5)
              END
```

READONLY (ustawienie tylko do wyświetlania)

READONLY

Atrybut **READONLY** (PROP:READONLY) powoduje, że kontrolka COMBO, ENTRY, SPIN lub TEXT służy tylko do wyświetlania danych. Kontrolka taka może otrzymywać aktywność wprowadzania (można w niej umieszczać kursor), nie jest jednak możliwe wprowadzanie w niej danych. Jeśli użytkownik spróbuje wprowadzać dane, jest generowany sygnał dźwiękowy.

REPEAT (ustawienie częstotliwości powtarzania przycisku)

REPEAT(*time*)

REPEAT Określa częstotliwość generowania zdarzenia.

time Stała całkowita określająca częstotliwość powtarzania przycisku; w setnych sekundy.

Atrybut **REPEAT** (PROP:REPEAT) określa częstotliwość generowania zdarzenia dla automatycznie powtarzalnych przycisków. Dla kontrolki BUTTON z atrybutem IMM, jest to częstotliwość powtarzania zdarzenia EVENT:Accepted. Dla kontrolki SPIN, jest to częstotliwość powtarzania zdarzenia EVENT:NewSelection generowanego przez przyciski kontrolki zmieniające jej wartość.

Przypisanie wartości 0 do właściwości PROP:REPEAT resetuje domyślne ustawienia, dowolna inna wartość ustawia odpowiednią częstotliwość powtarzania.

Przykład:

```
MDIChild    WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
              BUTTON('Press Me'),AT(10,10,40,20),USE(?PressMe),IMM,REPEAT(100)  ! 1/sekundę
              SPIN(@n3),AT(60,10,40,10),USE(SpinVar),RANGE(0,999),REPEAT(100)   ! 1/ sekundę
            END
  CODE
  OPEN(MDIChild)
  ?PressMe{PROP:Delay} = 50          ! ustawienie opóźnienia na 1/2 sekundy
  ?SpinVar{PROP:Delay} = 50          ! ustawienie opóźnienia na 1/2 sekundy
  ?PressMe{PROP:Repeat} = 5          ! zresetowanie powtórzenia na 5 setnych sekundy
  ?SpinVar{PROP:Repeat} = 5          ! zresetowanie powtórzenia na 5 setnych sekundy
```

Porównaj: IMM, DELAY

REQ (pole musi być wypełnione)

REQ

Atrybut **REQ** (PROP:REQ) powoduje, że kontrolka ENTRY lub TEXT nie może pozostać pusta bądź zerowa. Atrybut REQ dla kontrolki ENTRY lub TEXT nie jest sprawdzany dopóty, dopóki nie zostanie wciśnięty przycisk BUTTON posiadający atrybut REQ lub nie zostanie wywołana procedura INCOMPLETE(). Po wciśnięciu przycisku BUTTON z atrybutem REQ lub po wywołaniu procedury INCOMPLETE(), wszystkie kontrolki ENTRY oraz TEXT posiadające atrybut REQ są sprawdzane, czy posiadają niezerowe (niepuste) wartości. Pierwsza kontrolka, dla której wykryto nieprawidłowości (nie zawierająca danych), otrzymuje aktywność wprowadzania.

RESET (resetuje podliczenia)

RESET(*breaklevel*)

RESET Resetuje pola CNT, SUM, AVE, MIN oraz MAX nadając im wartość 0.

breaklevel Etykieta struktury BREAK.

Atrybut **RESET** (PROP:RESET) wskazuje grupę raportu, dla której następuje wyzerowanie pól CNT, SUM, AVE, MIN i MAX. Właściwość PROP:RESET daje w rezultacie wartość 0 jeśli atrybut nie występuje, w przeciwnym wypadku rezultatem jest poziom zagnieżdżenia danej grupy.

Przykład:

```
MojRaport  REPORT,AT(1000,1000,6500,9000),THOUS
Grupa1       BREAK(Pre:Key1)
               HEADER,AT(0,0,6500,1000)
                 STRING('Group Head'),AT(3000,500,1500,500),FONT('Arial',18)
               END
Detail         DETAIL,AT(0,0,6500,1000)
                 STRING(@N$11.2),AT(6000,1500,500,500),USE(Pre:F1)
               END
               FOOTER,AT(0,0,6500,1000)
                 STRING('Group Total:'),AT(5500,500,1500,500)
                 STRING(@N$11.2),AT(6000,500,500,500),USE(Pre:F1),SUM,RESET(Grupa1)
               END
             END
           END
```

RESIZE (ustawia zmienną wysokość kontrolki TEXT)

RESIZE

Atrybut **RESIZE** (PROP:RESIZE) powoduje, że wysokość kontrolki TEXT zmienia się w zależności od ilości danych w niej drukowanych, aż do maksymalnej wysokości określonej przez atrybut AT kontrolki. Parametr *height* atrybutu AT detalu DETAIL, nagłówka HEADER bądź stopki FOOTER zawierającej kontrolkę TEXT musi być nieokreślony (default), by atrybut RESIZE dawał zamierzony efekt.

Przykład:

```
MojRaport   REPORT,AT(1000,1000,6500,9000),THOUS
Detail        DETAIL,AT(0,0,6500,)                                    ! domyślna wysokość detalu
                STRING(@N$11.2),AT(500,500,500,),USE(Pre:F1)
                TEXT,AT(500,1000,500,5000),USE(Pre:Memo1),RESIZE  ! wysokość drukowania do 5”
              END
            END
```

RIGHT (określa pozycję MENU)

RIGHT

Atrybut **RIGHT** (PROP:RIGHT) powoduje, że MENU jest umieszczane po prawej stronie paska menu.

ROUND (zaokrągla narożniki BOX)

ROUND

Atrybut **ROUND** (PROP:ROUND) powoduje, że prostokąt BOX otrzymuje zaokrąglone narożniki.

SCROLL (umożliwia przewijanie kontrolki)

SCROLL

Atrybut **SCROLL** (PROP:SCROLL) powoduje, że kontrolka przesuwa się wraz z zawartością okna WINDOW w sytuacji, gdy jest ono przewijane. Dzięki temu możemy tworzyć okna „wirtualne" o rozmiarach większych, niż fizyczny rozmiar ekranu.

Występowanie atrybutu SCROLL oznacza, że kontrolka pozostaje w stałej pozycji względem lewego, górnego narożnika okna, niezależnie od tego, czy jest w danej chwili widoczna, czy też nie. Oznacza to, że kontrolka porusza się w momencie, gdy zawartość okna jest przewijana.

Pominięcie atrybutu SCROLL powoduje, że kontrolka pozostaje w stałej pozycji względem lewego, górnego narożnika aktualnie widocznej części okna. Oznacza to, że pozostaje ona w stałej pozycji, podczas gdy zawartość okna jest przewijana. Jest to przydatne w odniesieniu do kontrolek, które powinny być cały czas widoczne dla użytkownika (np. przyciski OK i Anuluj).

Umieszczanie w oknie kontrolek z atrybutem SCROLL i kontrolek go pozbawionych może doprowadzić do sytuacji, w której kontrolki będą zajmowały tę samą pozycję, czyli będą się wzajemnie przykrywały. Będzie się to zdarzać, gdyż kontrolki z atrybutem SCROLL poruszają się, a kontrolki bez tego atrybutu pozostają w stałej pozycji. Takie sytuacje są tymczasowe i można je skorygować odpowiednio przewijając zawartość okna. Możemy zapobiec ich występowaniu rozmieszczając odpowiednio kontrolki w oknie. Na przykład, możemy umieścić wszystkie kontrolki pozbawione atrybutu SCROLL w dolnej części okna, a kontrolki z atrybutem SCROLL – nad nimi, od lewej do prawej. W ten sposób przystosujemy okno do przewijania w poziomie (okno WINDOW powinno posiadać atrybut HSCROLL, a nie posiadać atrybut VSCROLL i HVSCROLL).

SEPARATOR (wstawia linie oddzielającą elementy menu)

SEPARATOR

Atrybut **SEPARATOR** określony dla elementu ITEM w menu MENU powoduje, że dany element jest poziomą linią oddzielającą od siebie inne elementy w danym menu. Dla takiego elementu ITEM nie określamy żadnych innych atrybutów.

SINGLE (ustawienie TEXT do wprowadzania jednowierszowego)

SINGLE

Atrybut **SINGLE** (PROP:SINGLE) powoduje, że kontrolka umożliwia wprowadzanie jednowierszowe. Jest to specyficzne wykorzystanie kontrolki TEXT w miejsce kontrolki ENTRY dla języków, w których teksty wprowadza się od prawej do lewej (hebrajski, arabski).

SKIP (pomijanie przez klawisz Tab lub warunkowe drukowanie)

SKIP

Atrybut **SKIP** (PROP:SKIP) nadany kontrolce okna powoduje, że użytkownik może uzyskać do niej dostęp klikając ją bezpośrednio myszką lub wciskając klawisz skrótu, kursor nie znajdzie się w kontrolce przy przechodzeniu pomiędzy nimi za pomocą klawisza TAB. Kontrolki wprowadzania danych otrzymują aktywność wprowadzania tylko na czas wprowadzania danych, kontrolka nie zatrzymuje aktywności. Kontrolki nie służące do wprowadzania danych nie otrzymują ani nie zachowują aktywności wprowadzania (tak samo zachowują się kontrolki w pasku narzędzi lub w okienku narzędziowym TOOLBOX). Gdy wskaźnik myszki znajduje się nad kontrolką posiadającą atrybut SKIP, tekst określony przez atrybut MSG kontrolki jest wyświetlany w pasku stanu.

Atrybut SKIP kontrolki raportu STRING lub TEXT określa, że ma ona być drukowana tylko wtedy, gdy zmienna wskazana w jej polu USE zawiera dane. Jeśli zmienna ta nie zawiera danych, kontrolka STRING lub TEXT nie jest drukowana, zaś pozostałe kontrolki są przesuwane, by zapełnić wolne miejsce. Rozwiązanie takie jest przydatne zwłaszcza przy drukowaniu nalepek adresowych, gdyż zapobiega powstawaniu zbędnych pustych linii w adresach.

Przykład:

```
MojRaport  REPORT,AT(1000,1000,6000,9000),THOUS
Detail       DETAIL,AT(0,0,2000,1000)                                  ! detal stałej wysokości
               STRING(@s35),AT(250,250,500,),USE(Pre:Name)
               STRING(@s35),AT(250,250,500,),USE(Pre:Address1)
               STRING(@s35),AT(250,250,500,),USE(Pre:Address2),SKIP    ! nie drukuj, jeśli puste
               STRING(@s35),AT(250,250,500,),USE(CityStateZip)         ! i podnieś do góry
             END
           END
```

SPREAD (równe rozmieszczenie zakładek)

SPREAD

Atrybut **SPREAD** (PROP:SPREAD) powoduje równomierne rozmieszczenie zakładek TAB w arkuszu SHEET.

STATUS (ustawienie paska stanu)

STATUS([*widths*])

STATUS Powoduje wyświetlanie paska stanu.

widths Lista stałych całkowitych (oddzielonych od siebie przecinkami) określających rozmiar każdej sekcji paska stanu. Jeśli parametr ten zostanie pominięty pasek stanu nie występuje.

Atrybut **STATUS** (PROP:STATUS) powoduje wyświetlanie paska stanu w dolnej części okna APPLICATION lub WINDOW. Pasek stanu okna z atrybutem MDI jest zawsze wyświetlany w dolnej części okna APPLICATION. Okno WINDOW nie posiadające atrybutu MDI wyświetla pasek stanu w swej dolnej części. Jeśli atrybut STATUS nie występuje w oknie APPLICATION lub WINDOW, pasek stanu nie jest wyświetlany.

Pasek stanu może być podzielony na wiele sekcji zdefiniowanych poprzez parametr *widths*. Rozmiar każdej z sekcji jest określony w jednostkach dialogowych. Wartość ujemna oznacza, że dana sekcja może zmieniać swój rozmiar, ale nie może być mniejsza niż minimalna wartość (odpowiadająca wartości bezwzględnej). Jeśli parametr *widths* nie występuje, jest tworzona pojedyncza sekcja bez określonego rozmiaru minimalnego, co odpowiada atrybutowi w postaci STATUS(-1).

Właściwość PROP:STATUS zawiera szerokości *widths* dla każdej sekcji paska stanu przechowywane jako oddzielne elementy tablicy. Wartość zero (0) umieszczona w elemencie tablicy oznacza jej zakończenie.

Pierwsza sekcja paska stanu jest zawsze przeznaczona do wyświetlania tekstów określonych przez atrybut MSG. Łańcuch tekstowy atrybutu MSG jest wyświetlany w pasku stanu tak długo, jak kontrolka, z którą jest związany posiada aktywność wprowadzania. Kontrolka lub element systemu menu pozbawiony atrybutu MSG powoduje przywrócenie sekcji paska stanu jej poprzedniej zawartości (łańcuch pusty lub tekst związany z wcześniejszą kontrolką).

Tekst może zostać umieszczony lub odczytany z dowolnej sekcji paska stanu za pomocą właściwości PROP:StatusText. Właściwość PROP:StatusText jest tablicą zawierająca teksty dla każdej sekcji paska stanu. Ostatnim elementem tej tablic musi być wartość zero (0). Tekst jest wyświetlany dopóty, dopóki nie zostanie zastąpiony innym tekstem.

Przykład:

```
! okno aplikacji z jedno-strefowym paskiem stanu:
MainWin  APPLICATION,STATUS
         END

! okno aplikacji z dwu-strefowym paskiem stanu:
Win1     WINDOW,STATUS(160,160)
         END
  CODE
  OPEN(Win1)
  Win1{PROP:STATUS,3} = 160                  ! dodaj sekcję paska stanu
  Win1{PROP:STATUS,4} = 0                    ! i zakończ tablicę
  Win1{PROP:StatusText,3} = 'Hello Zone 3'   ! umieść tekst w nowej sekcji
```

Porównaj: MSG

STD (określenie standardowej akcji)

STD(*behavior*)

STD Określa standardową akcję *behavior* systemu Windows.

behavior Stała całkowita lub ekwiwalent EQUATE określający identyfikator standardowej akcji Windows

Atrybut **STD** (PROP:STD) powoduje, że kontrolka uaktywnia jedną ze standardowych akcji Windows. Akcja ta jest automatycznie wykonywana przez bibliotekę uruchomieniową i nie pociąga za sobą generowania zdarzeń.

Ekwiwalenty EQUATE dla standardowych akcji Windows są zawarte w pliku EQUATES.CLW. Poniżej przedstawiono przykłady niektórych z nich:

STD:WindowList	Lista otwartych okien MDI
STD:TileWindow	Rozmieszcza równomiernie otwarte okna
STD:CascadeWindow	Kaskaduje otwarte okna
STD:ArrangeIcons	Rozmieszcza ikony okien
STD:HelpIndex	Wyświetla spis treści systemu pomocy aplikacji
STD:HelpSearch	Wyświetla indeks systemu pomocy aplikacji

Przykład:

```
MDIChild  WINDOW('Child One'),MDI,SYSTEM,MAX
            MENUBAR
              MENU('Edit'),USE(?EditMenu)
                ITEM('Undo'),USE(?UndoText),KEY(CtrlZ),STD(STD:Undo)
                ITEM('Cu&t'),USE(?CutText),KEY(CtrlX),STD(STD:Cut)
                ITEM('Copy'),USE(?CopyText),KEY(CtrlC),STD(STD:Copy)
                ITEM('Paste'),USE(?PasteText),KEY(CtrlV),STD(STD:Paste)
              END
            END
            TOOLBAR
              BUTTON('Cut'),USE(?CutButton),ICON(ICON:Cut),STD(STD:Cut)
              BUTTON('Copy'),USE(?CopyButton),ICON(ICON:Copy),STD(STD:Copy)
              BUTTON('Paste'),USE(?PasteButton),ICON(ICON:Paste),STD(STD:Paste)
            END
          END
```

STEP (ustawia skok wartości kontrolki SPIN)

STEP(*count*)

STEP Określa wartość (krok), o którą jest zwiększana/zmniejszana wartość kontrolki a SPIN.

count Stała numeryczna określająca wielkość wartości kroku.

Atrybut **STEP** (PROP:STEP) określa wartość (krok), o którą jest zwiększana/zmniejszana wartość kontrolki SPIN w ramach prawidłowego zakresu RANGE. Domyślnym krokiem STEP jest wartość 1.0.

Przykład:

```
OknoPierwsze  WINDOW,AT(0,0,160,400)
                SPIN(@N4.2),AT(280,0,20,20),USE(SpinVar1),RANGE(.05,9.95),STEP(.05)
                SPIN(@N3),AT(280,0,20,20),USE(SpinVar2),RANGE(5,995),STEP(5)
                SPIN(@T3),AT(280,0,20,20),USE(SpinVar3),RANGE(1,8640000),STEP(6000)
              END
```

STRETCH (dopasowanie rozmiaru obiektu OLE)

STRETCH

Atrybut **STRETCH** (PROP:STRETCH, tylko-do-zapisu) powoduje zmianę rozmiaru obiektu OLE tak, by kompletnie wypełniał obszar kontrolki kontenera OLE określony przez jej atrybut AT. Omawiany atrybut nie zachowuje proporcji obiektu.

SUM (wyliczanie sumy w raporcie)

SUM([*variable*])

SUM Wylicza sumę dla kontrolek STRING w oparciu o wartości zmiennej występującej w polu USE tej kontrolki.

variable Etykieta zmiennej numerycznej, w której będą przechowywane wartości pośrednie wyliczane przez SUM. Umożliwia to tworzenie podliczeń i podliczeń częściowych. Wartość w zmiennej *variable* jest wewnętrznie aktualizowana przez mechanizm drukowania, z tego względu można stosować ten sposób tylko w strukturach raportu REPORT.

Atrybut **SUM** (PROP:SUM) powoduje drukowanie sumy wartości zmiennej występującej w polu USE kontrolki typu STRING raportu. O ile nie występuje atrybut TALLY, rezultat jest wyliczany zgodnie z poniższymi zasadami:

- Pole SUM znajdujące się w strukturze DETAIL jest wyliczane za każdym razem, gdy struktura DETAIL zawierająca kontrolkę jest drukowana.
- Pole SUM znajdujące się w stopce FOOTER grupy jest wyliczane za każdym razem, gdy jest drukowana dowolna struktura DETAIL zawarta w strukturze BREAK zawierającej kontrolkę. Pozwala to na sumowanie wartości wchodzących w skład grupy.
- Pole SUM znajdujące się w stopce FOOTER strony jest wyliczane za każdym razem, gdy jest drukowana dowolna struktura DETAIL w dowolnej strukturze BREAK. Pozwala to na sumowanie wartości drukowanych na stronie.
- Pole SUM znajdujące się w nagłówku HEADER nie ma sensu, gdyż żaden detal nie jest drukowany w czasie drukowania nagłówka.

Suma jest resetowana wtedy, gdy są określone dla niej atrybuty RESET lub PAGE.

Przykład:

```
MojRaport  REPORT,AT(1000,1000,6500,9000),THOUS
Grupa1       BREAK(LocalVar),USE(?BreakOne)
Grupa2         BREAK(Pre:Key1),USE(?BreakTwo)
Detail           DETAIL,AT(0,0,6500,1000),USE(?DetailOne)
                   STRING(@N$11.2),AT(6000,1500,500,500),USE(Pre:F1)
                 END
                 FOOTER,AT(0,0,6500,1000),USE(?BreakOneGroupFoot)
                   STRING('Group Total:'),AT(5500,500)
                   STRING(@N$11.2),AT(6000,500),USE(Pre:F1),SUM(LocalVar),RESET(Grupa2)
                 END
               END
               FOOTER,AT(0,0,6500,1000),USE(?BreakOneGroupFoot)
                 STRING('Grand Total:'),AT(5500,500)
                 STRING(@N$11.2),AT(6000,500),USE(LocalVar),SUM,TALLY(?BreakTwo)
               END
             END
           END
```

SYSTEM (włącza menu systemowe)

SYSTEM

Atrybut **SYSTEM** (PROP:SYSTEM) powoduje, że w oknie APPLICATION lub WINDOW występuje menu systemowe (nazywane też menu sterującym). Menu systemowe zawiera kilka standardowych poleceń umożliwiających: zamknięcie okna (Close – Zamknij), zwinięcie okna do ikony (Minimize – Minimalizuj), rozwinięcie okna na cały pulpit (Maximize – Maksymalizuj) i inne. Opcje dostępne w menu systemowym zależą od atrybutów nadanych danemu oknu.

Przykład:

```
! okno aplikacji z menu systemowym:
MainWin  APPLICATION,SYSTEM
         END

! okno z menu systemowym:
Win1     WINDOW,SYSTEM
         END
```

TALLY (określa miejsca kalkulacji pól wyliczeniowych)

TALLY(*points*)

TALLY Określa, kiedy mają być wyliczane wartości AVE, CNT, MAX, MIN lub SUM.

points Lista etykiet, oddzielonych od siebie przecinkami, struktur DETAIL i/lub BREAK, dla których ma nastąpić kalkulacja pól wyliczeniowych.

Atrybut **TALLY** (PROP:TALLY) określa, kiedy mają być wyliczane wartości AVE, CNT, MAX, MIN lub SUM. Odpowiednie pole wyliczeniowe jest kalkulowana za każdym razem, gdy jest drukowana jedna ze struktur DETAIL wymienionych w liście *points* lub też, w przypadku struktury BREAK, gdy nastąpi zmiana grupy.

Przykład:

```
MojRaport  REPORT,AT(1000,1000,6500,9000),THOUS
Grupa1       BREAK(LocalVar),USE(?BreakOne)
Grupa2         BREAK(Pre:Key1),USE(?BreakTwo)
                 HEADER,AT(0,0,6500,1000),USE(?GroupHead)
                   STRING('Group Head'),AT(3000,500,1500,500),FONT('Arial',18)
                 END
Detail           DETAIL,AT(0,0,6500,1000),USE(?DetailOne)
                   STRING(@N$11.2),AT(6000,1500,500,500),USE(Pre:F1)
                 END
               END
               FOOTER,AT(0,0,6500,1000),USE(?BreakOneGroupFoot)
                 STRING('Group Total:'),AT(5500,500,1500,500)
                 STRING(@N$11.2),AT(6000,500,500,500),USE(Pre:F1),CNT,TALLY(Grupa2)
               END
             END
           END
  CODE
  OPENCustRpt)
  CustRpt$?Pre:F1{PROP:Tally} = ?BreakOne          ! zmienia miejsce obliczeń na Grupa1
```

THOUS, MM, POINTS (określa układ miar raportu)

THOUS
MM
POINTS

Atrybuty **THOUS**, **MM** oraz **POINTS** umożliwiają określenie układu miar dla raportu REPORT, wg którego są pozycjonowane jego kontrolki. Atrybut **THOUS** (PROP:THOUS) powoduje wybranie, jako jednostkę podstawową, tysięczną cala, **MM** (PROP:MM) - milimetr, a **POINTS** (PROP:POINTS) – punkt (na cal przypadają siedemdziesiąt dwa punkty, zarówno w pionie, jak i w poziomie). Jeśli nie występuje żadne z wymienionych atrybutów, domyślną jednostką jest jednostka dialogowa. Jest ona zdefiniowana jako jedna czwarta średniej szerokości znaku na jedną ósmą średniej wysokości znaku. Rozmiar jednostki dialogowej zależy od domyślnej czcionki raportu określonej atrybutem FONT raportu REPORT lub od domyślnej czcionki systemowej Windows.

TILED (sąsiadujące kopie grafiki)

TILED

Atrybut **TILED** (PROP:TILED) powoduje, że grafika wyświetlana w kontrolce IMAGE lub w tle okna bądź paska narzędzi jest powielana tyle razy, ile potrzeba do zapełnienia kontrolki, czy tła. Każda kopia jest wyświetlana w oryginalnym rozmiarze.

Przykład:

```
OknoPierwsze  WINDOW,AT(,,380,200),MDI
                IMAGE('MyWall.GIF'),AT(0,0,380,200),TILED
              END

MDIChild      WINDOW('Child One'),MDI,SYSTEM,MAX
                MENUBAR
                  MENU('Edit'),USE(?EditMenu)
                    ITEM('Undo'),USE(?UndoText),KEY(CtrlZ),STD(STD:Undo)
                    ITEM('Cu&t'),USE(?CutText),KEY(CtrlX),STD(STD:Cut)
                    ITEM('Copy'),USE(?CopyText),KEY(CtrlC),STD(STD:Copy)
                    ITEM('Paste'),USE(?PasteText),KEY(CtrlV),STD(STD:Paste)
                  END
                END
                TOOLBAR,USE(?Toolbar),WALLPAPER('MyWall.GIF'),TILED
                  BUTTON('Cut'),USE(?CutButton),ICON(ICON:Cut),STD(STD:Cut),FLAT
                  BUTTON('Copy'),USE(?CopyButton),ICON(ICON:Copy),STD(STD:Copy),FLAT
                  BUTTON('Paste'),USE(?PasteButton),ICON(ICON:Paste),STD(STD:Paste),FLAT
                END
              END

OknoPierwsze  WINDOW,AT(,,380,200),MDI,WALLPAPER('MyWall.GIF'),TILED
              END
```

Porównaj: CENTERED, WALLPAPER

TIMER (ustawia zdarzenie okresowe)

TIMER(*period*)

TIMER Określa zdarzenie okresowe.

period Stała całkowita lub wyrażenie stałe określające odstęp czasowy, w setnych sekundy. Maksymalna wartość okresu *period* wynosi 6553 (ograniczenie systemu Windows). Jeśli parametr ma wartość zerową (0), nie są generowane zdarzenia.

Atrybut **TIMER** (PROP:TIMER) powoduje generowanie okresowego zdarzenia niezależnego od pola EVENT:Timer, w odstępach czasowych określonych przez *period*. Procedura FOCUS() daje w rezultacie numer kontrolki posiadającej aktywność w momencie wystąpienia zdarzenia.

Jeśli okno posiadające atrybut TIMER nie posiada aktywności wprowadzania w momencie wystąpienia zdarzenia EVENT:Timer, aktywne okno najpierw odbiera zdarzenie EVENT:Suspend, następnie okno posiadające atrybut TIMER rejestruje zdarzenie EVENT:Timer. Po wystąpieniu zdarzenia EVENT:Suspend w oknie posiadającym aktywność wprowadzania, jest generowane zdarzenie EVENT:Resume przed innymi zdarzeniami generowanymi dla okna. Jednocześnie zdarzenie EVENT:Resume nie zostanie wygenerowane dopóty, dopóki istnieją inne zdarzenia, które musi obsłużyć dane okno (okno jest wstrzymywane a zdarzenia okresowe kontynuują przetwarzanie dopóki jest to niezbędne w oknie posiadającym aktywność).

Przykład:

```
RunClock    PROCEDURE
ShowTime     LONG

! okno ze zdarzeniem zegarowym zachodzącym co 1 sekundę:
Win1     WINDOW,TIMER(100)
           STRING(@T4),USE(ShowTime)
         END
  CODE
  OPEN(Win1)
  ShowTime = CLOCK()
  ACCEPT
    CASE EVENT()
    OF EVENT:Timer
      ShowTime = CLOCK()
      DISPLAY
    END
  END
  CLOSE(Win1)
```

TIP (tekst podpowiedzi)

TIP(*string*)

TIP Określa tekst wyświetlany w momencie, gdy użytkownik ustawi nieruchomo na kontrolce wskaźnik myszki.

string Stała łańcuchowa definiująca tekst do wyświetlenia.

Atrybut **TIP** (PROP:TIP) nadany kontrolce definiuje tekst wyświetlany w momencie, gdy użytkownik ustawi na niej nieruchomo wskaźnik myszki. Mimo, że nie ma ograniczeń co do długości tego tekstu, powinien on być taki, by mógł zostać wyświetlony na ekranie.

Chociaż omawiany atrybut jest prawidłowy dla dowolnej kontrolki, która może posiadać aktywność wprowadzania, najczęściej stosuje się go dla przycisków BUTTON posiadających dodatkowo atrybut ICON, które są umieszczone w pasku narzędzi TOOLBAR. Pozwala to użytkownikowi na szybkie ustalenie, jakiego rodzaju akcję uruchamia dany przycisk, bez potrzeby odwoływania się do systemu pomocy.

Automatyczne wyświetlanie podpowiedzi może zostać wyłączone dla wybranej kontrolki lub całego okna poprzez zastosowanie właściwości PROP:NoTips I nadanie jej wartości 1. Możemy też wyłączyć podpowiedzi w całej aplikacji nadając wartość 1 właściwości PROP:NoTips określonej dla wbudowanej zmiennej SYSTEM.

Opóźnienie, z jakim jest wyświetlony tekst podpowiedzi określamy dla całej aplikacji nadając odpowiednią wartość (w setnych sekundy) właściwości PROP:TipDelay wbudowanej zmiennej SYSTEM. Dotyczy to aplikacji 16-bitowych; opóźnienie dla aplikacji 32-bitowych jest jednym z parametrów systemowych Windows.

Przykład:

```
OknoPierwsze  WINDOW,AT(0,0,160,400)
                TOOLBAR
                  BUTTON('E&xit'),USE(?MainExitButton),ICON(ICON:hand),TIP('Exit Window')
                  BUTTON('&Open'),USE(?OpenButton),ICON(ICON:Open),TIP('Open a File')
                END
                COMBO(@S8),AT(120,120,20,20),USE(?C1),FROM(Q1:F2)
                ENTRY(@S8),AT(100,200,20,20),USE(E2)
              END
```

TOOLBOX (ustawienie okienka narzędziowego)

TOOLBOX

Atrybut **TOOLBOX** (PROP:TOOLBOX) określa, że dane okno WINDOW znajduje się „zawsze na wierzchu" i, jeśli dodatkowo nadamy mu atrybut DOCK, może być dokowane do którejś z krawędzi okna głównego aplikacji. Ani okno WINDOW ani jego kontrolki nie zachowują aktywności wprowadzania. Kontrolki zachowują się tak, jakby posiadały atrybut SKIP.

Standardowo okno WINDOW z atrybutem TOOLBOX działa jako samodzielny wątek udostępniając narzędzia oknom posiadającym aktywność wprowadzania. Atrybut MSG kontrolek okna powoduje wyświetlanie tekstów w pasku stanu w momencie, gdy wskaźnik myszki znajdzie się nad daną kontrolką.

Przykład:

```
    PROGRAM
MainWin  APPLICATION('My Application')
           MENUBAR
             MENU('File'),USE(?FileMenu)
               ITEM('E&xit'),USE(?MainExit),LAST
             END
             MENU('Edit'),USE(?EditMenu)
               ITEM('Use Tools'),USE(?UseTools)
             END
           END
         END
Pre:Field          STRING(400)
UseToolsThread BYTE
ToolsThread    BYTE

  CODE
  OPEN(MainWin)
  ACCEPT
    CASE ACCEPTED()
    OF ?MainExit
      BREAK
    OF ?UseTools
      UseToolsThread = START(UseTools)
    END
  END

UseTools  PROCEDURE                          ! procedura stosujacą okno narzędzi
MDIChild    WINDOW('Use Tools Window'),MDI
              TEXT,HVSCROLL,USE(Pre:Field)
              BUTTON('&OK'),USE(?Exit),DEFAULT
            END
  CODE
  OPEN(MDIChild)                             ! otwarcie okna
  DISPLAY                                    ! i wyświetlenie go
  ToolsThread = START(Tools)                 ! rozwinięcie okna narzędzi
  ACCEPT
    CASE EVENT()                             ! kontrola zdarzeń zdefiniowanych przez użytkownika
    OF 401h                                  ! wysłanych do kontrolek okna narzędzi
      Pre:Field += ' ' & FORMAT(TODAY(),@D1)  ! dodanie daty na koniec pola
    OF 402h
      Pre:Field += ' ' & FORMAT(CLOCK(),@T1)  ! dodanie czasu na koniec pola
    END
    CASE ACCEPTED()
    OF ?Exit
      POST(400h,,ToolsThread)                ! sygnał do zamknięcia okna narzędzi
      BREAK
    END
```

```
  END
  CLOSE(MDIChild)

Tools  PROCEDURE                          ! procedura okna narzędzi
Win1     WINDOW('Tools'),TOOLBOX
           BUTTON('Date'),USE(?Button1)
           BUTTON('Time'),USE(?Button2)
         END
  CODE
  OPEN(Win1)
  ACCEPT
    IF EVENT() = 400h THEN BREAK.         ! kontrola sygnału zamknięcia okna
    CASE ACCEPTED()
    OF ?Button1
      POST(401h,,UseToolsThread)          ! wysłanie sygnału o dacie
    OF ?Button2
      POST(402h,,UseToolsThread)          ! wysłanie sygnału o czasie
    END
  END
  CLOSE(Win1)
```

Porównaj: DOCK

TRN (ustawienie przeźroczystości kontrolki)

TRN

Atrybut **TRN** (PROP:TRN) nadany kontrolce powoduje, że jej znaki są wypisywane lub drukowane w sposób przeźroczysty (bez tła). W efekcie pojawiają się tylko punkty lub piksele tworzące drukowane lub wyświetlane znaki. Dzięki temu możemy na przykład efektownie umieszczać napisy na grafikach.

Przykład:

```
MojRaport   REPORT,AT(1000,1000,6500,9000),THOUS
              FORM,AT(0,0,6500,9000)
                IMAGE('PIC.BMP'),USE(?I1)AT(0,0,6500,9000)          ! grafika pełnostronicowa
                STRING('jakiś napis'),AT(10,0,20,20),USE(?S1),TRN
                ! przeźroczysty napis na grafice
              END
            END

OknoPierwsze  WINDOW,AT(0,0,160,400)
                IMAGE('PIC.BMP'),USE(?I1),FULL                      ! grafika zajmująca całe okno
                STRING('jakiś napis'),AT(10,0,20,20),USE(?S1),TRN
                ! przeźroczysty napis na grafice
              END
```

UP, DOWN (ustawienie orientacji tekstu zakładki TAB)

UP
DOWN

Atrybuty **UP** oraz **DOWN** kontrolki SHEET określają orientację tekstów tytułowych zakładek TAB. Atrybut UP (PROP:UP) powoduje, że tekst jest wyświetlany w pionie, z dołu do góry. Atrybut DOWN (PROP:DOWN) powoduje, że tekst jest wyświetlany w pionie, z góry do dołu. Jeśli nadamy jednocześnie oba atrybuty: UP i DOWN, tekst jest wyświetlany poziomo, „do góry nogami" (PROP:UpsideDown).

Przykład:

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
           SHEET,AT(0,0,320,175),USE(SelectedTab),RIGHT,DOWN        ! zakładka po prawej, napis w dół
             TAB('Tab One'),USE(?TabOne)
               PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P1)
               ENTRY(@S8),AT(100,140,32,20),USE(E1)
               PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P2)
               ENTRY(@S8),AT(100,240,32,20),USE(E2)
             END
             PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P3)
             ENTRY(@S8),AT(100,140,32,20),USE(E3)
             PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P4)
             ENTRY(@S8),AT(100,240,32,20),USE(E4)
           END
         END
         BUTTON('Ok'),AT(100,180,20,20),USE(?Ok)
         BUTTON('Anuluj'),AT(200,180,20,20),USE(?Cancel)
       END
```

USE (ustawienie ekwiwalentu etykiety pola lub zmiennej kontrolki)

USE(| *label* | [, *number*] [, *equate*])
| *variable* |

USE Wskazuje etykietę ekwiwalentu pola lub zmienną.

label Etykieta ekwiwalentu pola, którego dotyczy kontrolka lub struktura kodu wykonywalnego. Musi się zaczynać znakiem zapytania (?) I spełniać wszystkie kryteria dotyczące etykiet w języku Clarion.

variable Etykieta zmiennej, do której jest przypisywana wartość wprowadzona przez użytkownika do kontrolki. Etykieta zmiennej, wraz z wiodącym znakiem zapytania (*?EtykietaZmiennej*) staje się etykietą ekwiwalentu pola dla kontrolki, chyba, że jest stosowany parametr *equate*.

number Stała całkowita określająca liczbę stanowiącą ekwiwalent nadawany przez kompilator etykiecie pola kontrolki (PROP:Feq, równoważne z {PROP:USE,2}).

equate Etykieta ekwiwalentu pola umożliwiająca odwoływanie się do kontrolki w kodzie wykonywalnym, gdy nazwa *variable* jest już wykorzystywana w tej samej strukturze. Dostarcza to mechanizm zapewniający unikalność ekwiwalentów pól.

Atrybut **USE** (PROP:USE) określa etykietę ekwiwalentu pola kontrolki lub struktury, bądź zmienną aktualizowaną za pośrednictwem danej kontrolki. Atrybut USE z parametrem *label* umożliwia po prostu odwoływania się do kontrolki lub struktury w kodzie wykonywalnym. Atrybut USE z parametrem *variable* umożliwia aktualizację wartości zmiennej wartościami wprowadzanymi przez operatora do kontrolki, bądź też drukowanie wartości kontrolki w raporcie. Parametr *number* atrybutu USE umożliwia określenie numeru pola przypisywanego przez kompilator kontrolce. Numer ten jest również używany w roli numeru początkowego dla numerowania wszystkich następnych kontrolek nie posiadających parametru *number* w swoich atrybutach USE; ich numery pól są zwiększane lub zmniejszane o 1.

Występowanie dwóch (lub więcej) kontrolek o tych samych zmiennych *variable* atrybutu USE w jednym oknie WINDOW lub APPLICATION będzie pociągało za sobą próbę utworzenia dla nich wszystkich takich samych etykiet ekwiwalentu pola. Jednakże, gdy kompilator wykryje taką sytuację, wszystkie etykiety ekwiwalentu pola dla danej zmiennej USE są odrzucane. Tym samym staje się niemożliwe odwoływanie do tych kontrolek w kodzie wykonywalnym, dzięki czemu nie powstanie zamieszanie związane z określeniem, której z kontrolek dane wywołanie dotyczy. Umyślne spowodowanie powyższej sytuacji umożliwia na przykład wyświetlanie wartości zmiennej w różnych formatach. Powyższą sytuację możemy eliminować stosując dla kontrolek parametr *equate*.

Zapis do właściwości PROP:USE powoduje zmianę zmiennej wykorzystywanej przez atrybut USE. Odczyt tej właściwości pozwala na określenie aktualnej wartości zmiennej USE. Właściwość PROP:Feq ustawia i daje w rezultacie numer pola dla kontrolki.

Zastosowanie w oknie

Niektóre kontrolki i struktury umożliwiają tylko stosowanie etykiety ekwiwalentu pola *label* jako parametru atrybutu USE. Są to: PROMPT, IMAGE, LINE, BOX, ELLIPSE, GROUP, RADIO, REGION, MENU, BUTTON oraz TOOLBAR.

Atrybut USE z parametrem *variable* zapewnia aktualizację, przez użytkownika, zmiennej związanej z kontrolką. Dotyczy to kontrolek: ITEM z atrybutem CHECK, ENTRY, OPTION, SPIN, TEXT, LIST, COMBO, CHECK oraz CUSTOM.

Właściwość PROP:ListFeq jest równoważna z {PROP:USE,3} i ustawia etykietę ekwiwalentu pola dla listy będącej częścią kontrolki COMBO lub dla kontrolki LIST posiadającej atrybut DROP.

Właściwość PROP:ButtonFeq jest równoważna z {PROP:USE,4} i ustawia etykietę ekwiwalentu pola dla przycisku rozwijającego listę COMBO lub dla kontrolki LIST posiadającej atrybut DROP.

Zastosowanie w raporcie

Niektóre kontrolki i struktury umożliwiają tylko stosowanie etykiety ekwiwalentu pola *label* jako parametru atrybutu USE. Są to: IMAGE, LINE, BOX, ELLIPSE, GROUP, RADIO, FORM, BREAK, DETAIL, HEADER oraz FOOTER.

Atrybut USE z parametrem *variable* zapewnia aktualizację, przez użytkownika, zmiennej związanej z kontrolką. Dotyczy to kontrolek: OPTION, TEXT, LIST, CHECK lub CUSTOM. Kontrolki STRING mogą stosować zarówno parametr *label*, jak i *variable*.

Wszystkim kontrolkom i strukturom raportu REPORT numery są przydzielane automatycznie przez kompilator. Domyślnie numeracja zaczyna się od (1), kolejne numery są zwiększane o (1) dla każdej kontrolki raportu REPORT. Parametr *number* atrybutu USE pozwala na określenie aktualnego numeru pola nadawanego kontrolce lub strukturze przez kompilator. Numer ten jest stosowany także jako nowy numer startowy przy numerowaniu kolejnych kontrolek nie posiadających parametru *number* w swoich atrybutach USE; ich numery pól są zwiększane o 1.

Przykład:

```
MainWin  APPLICATION('My Application'),SYSTEM,MAX,ICON('MyIcon.ICO'),STATUS
           MENUBAR
             MENU('&File'),USE(?FileMenu)
               ITEM('&Open...'),USE(?OpenFile)
               ITEM('&Close'),USE(?CloseFile),DISABLE
               ITEM('E&xit'),USE(?MainExit)
             END
           END
           TOOLBAR,USE(?Toolbar)
             BUTTON('Exit'),USE(?MainExitButton)
             ENTRY(@S8),AT(100,160,20,20),USE(E2)
             ENTRY(@S8),AT(100,200,20,20),USE(E3,100)              ! pole numer 100
             ENTRY(@S8),AT(100,240,20,20),USE(E2,,?Number2:E2)
           END
         END

MojRaport REPORT,AT(1000,1000,6500,9000),THOUS
Detail     DETAIL,AT(0,0,6500,1000),USE(?Detail)
             STRING('Group Total:'),AT(5500,500,1500,500),USE(?Constant)
             ! etykieta ekwiwalentu pola
             STRING(@N$11.2),AT(6000,1500,500,500),USE(Pre:F1)
             ! zmienna USE
           END
         END
```

```
  CODE
  OPEN(MainWin)
  DISABLE(?E2)                    ! wyłącza pierwszą kontrolkę wprowadzania
  DISABLE(100)                    ! wyłącza drugą kontrolkę wprowadzania
  DISABLE(?Number2:E2)            ! wyłącza trzecią kontrolkę wprowadzania
  PrintRpt(CustRpt,?Detail)       ! przekazuje raport i ekwiwalent detalu do prcoedury drukowania
  ACCEPT
  END

PrintRpt  PROCEDURE(RptToPrint,DetailNumber)
  CODE
  OPEN(RptToPrint)                ! otwiera przekazany raport
  PRINT(RptToPrint,DetailNumber)  ! drukuje jego detal
  CLOSE(RptToPrint)               ! zamyka przekazany raport
```

Porównaj: Etykiety ekwiwalentów pól

VALUE (przypisuje wartość zmiennej związanej z RADIO lub CHECK)

```
VALUE(  | string                | )
        | truevalue , falsevalue |
```

VALUE Określa wartość przypisywaną zmiennej wskazanej w atrybucie USE struktury OPTION, jaką przyjmuje ona w momencie wybrania przez użytkownika kontrolki RADIO. W drugim przypadku jest to wartość przypisywana zmiennej wskazywanej przez atrybut USE kontrolki CHECK w momencie, gdy jest ona zaznaczana (lub zaznaczenie jest usuwane) przez użytkownika.

string Stała łańcuchowa zawierająca wartość przypisywaną do zmiennej wskazywanej przez atrybut USE struktury OPTION.

truevalue Stała łańcuchowa zawierająca wartość przypisywaną do zmiennej wskazywanej przez atrybut USE kontrolki CHECK, gdy jest ona zaznaczana (PROP:TrueValue, równoważne z {PROP:Value,1}).

falsevalue Stała łańcuchowa zawierająca wartość przypisywaną do zmiennej wskazywanej przez atrybut USE kontrolki CHECK, gdy jej zaznaczanie jest usuwane (PROP:FalseValue, równoważne z {PROP:Value,2}).

Atrybut **VALUE** (PROP:VALUE) w kontrolce RADIO określa wartość, która jest automatycznie nadawana zmiennej wskazanej w atrybucie USE struktury OPTION, do której należy ta kontrolka, w chwili, gdy została ona zaznaczona przez użytkownika. Atrybut ten zastępuje parametr *text* kontrolki RADIO.

Atrybut VALUE w kontrolce CHECK określa wartość automatycznie nadawaną zmiennej wskazywanej przez atrybut USE tej kontrolki, w momencie gdy została ona zaznaczona przez użytkownika lub to zaznaczenie zostało usunięte. To ustawienie przykrywa domyślne przypisane wartości 1 i 0.

Wszystkie zasady automatycznej konwersji danych dotyczą również wartości przypisywanych do zmiennych wskazywanych przez atrybuty USE kontrolek. Z tego względu, jeśli *string, truevalue* lub *falsevalue* zawierają tylko dane numeryczne i zmienna USE jest typu numerycznego, rejestruje ona wartość numeryczną.

W kontrolkach ENTRY, SPIN oraz COMBO może być stosowana właściwość PROP:VALUE w celu sprawdzenia wartości, która powinna być umieszczona w zmiennej za pomocą UPDATE (lub wtedy, gdy kontrolka traci aktywność wprowadzania) bez aktualizowania zmiennej. Może to spowodować generowanie zdarzenia EVENT:Rejected, jeśli istnieje taka konieczność.

Przykład:

```
Win  WINDOW,AT(0,0,180,400)
       OPTION('Option 1'),USE(OptVar1),MSG('Pick One or Two')
         RADIO('Radio 1'),AT(120,0,20,20),USE(?R1),VALUE('10')      ! OptVar1 otrzymuje 10
         RADIO('Radio 2'),AT(140,0,20,20),USE(?R2),VALUE('20')      ! OptVar1 otrzymuje 20
       END
       OPTION('Option 2'),USE(OptVar2),MSG('Pick One or Two')
         RADIO('Radio 1'),AT(120,0,20,20),USE(?R1),VALUE('10')      ! OptVar2 otrzymuje '10'
         RADIO('Radio 2'),AT(140,0,20,20),USE(?R2),VALUE('20')      ! OptVar2 otrzymuje '20'
       END
       CHECK('Check 1'),AT(160,0),USE(Check1),VALUE('T','F')
     END
```

VCR (ustawienie kontrolki VCR)

VCR([*field*])

VCR Umieszcza przyciski w stylu Video Cassette Recorder (VCR) w kontrolce LIST lub COMBO.

field Etykieta ekwiwalentu pola wskazująca kontrolkę ENTRY pełniącą rolę lokatora dla listy LIST lub COMBO (PROP:VcrFeq, równoważne z {PROP:VCR,2}). Ta kontrolka ENTRY musi się pojawić w strukturze okna WINDOW przed kontrolką LIST (lub COMBO).

Atrybut **VCR** (PROP:VCR) umieszcza przyciski w stylu **V**ideo **C**assette **R**ecorder (VCR) w kontrolce LIST lub COMBO. Przyciski w stylu VCR umożliwiają przemieszczanie się w liście. Jest siedem przycisków tworzących VCR:

|< Początek listy (EVENT:ScrollTop)

<< Strona w górę (EVENT:PageUp)

< Element w górę (EVENT:ScrollUp)

? Wyszukaj (EVENT:Locate)

> Element w dół (EVENT:ScrollDown)

>> Strona w dół (EVENT:PageDown)

>| Koniec lsity (EVENT:ScrollBottom)

Parametr *field* nazywa kontrolkę, która otrzymuje aktywność wprowadzania po wciśnięciu przez użytkownika przycisku **?**. Gdy użytkownik wprowadzi dane do lokatora *field* i naciśnie klawisz TAB, w liście LIST lub COMBO zostanie podświetlony element najbardziej pasujący do wprowadzonego wzoru. jeśli nie zostanie określony żaden parametr *field*, wciśnięcie przycisku **?** nie daje żadnego efektu. Jeśli chcemy uniknąć wyświetlania przycisku **?**, ustawiamy właściwość PROP:VCR na TRUE zamiast nadawać atrybut VCR w deklaracji kontrolki LIST lub COMBO.

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            LIST,AT(140,0,20,20),USE(?L1),FROM(Que),HVSCROLL
            ENTRY(@S8),AT(100,200,20,20),USE(E2)                    ! kontrolka lokatora dla L2
            LIST,AT(140,100,20,20),USE(?L2),FROM(Que),HVSCROLL,VCR(?E2)
            ! VCR z aktywnym lokatorem
          END
  CODE
  OPEN(MDIChild)
  ?L1{PROP:VCR} = TRUE            ! przyciski VCR bez przycisku ?
  ACCEPT
  END
```

WALLPAPER (ustawienie grafiki tła)

WALLPAPER(*image*)

WALLPAPER Określa grafikę stanowiącą tło paska narzędzi lub okna.

image Stała łańcuchowa określająca nazwę pliku zawierającego grafikę do wyświetlenia.

Atrybut **WALLPAPER** (PROP:WALLPAPER) określa grafikę *image* stanowiącą tło paska narzędzi lub okna. Rozmiar grafiki jest dobierany tak, by wypełniła cały obszar paska narzędzi lub obszar roboczy okna, chyba, że został określony atrybut TILED lub CENTERED.

Przykład:

```
MDIChild  WINDOW('Child One'),MDI,SYSTEM,MAX
            MENUBAR
              MENU('Edit'),USE(?EditMenu)
                ITEM('Undo'),USE(?UndoText),KEY(CtrlZ),STD(STD:Undo)
                ITEM('Cu&t'),USE(?CutText),KEY(CtrlX),STD(STD:Cut)
                ITEM('Copy'),USE(?CopyText),KEY(CtrlC),STD(STD:Copy)
                ITEM('Paste'),USE(?PasteText),KEY(CtrlV),STD(STD:Paste)
              END
            END
            TOOLBAR,USE(?Toolbar),WALLPAPER('MyWall.GIF')
              BUTTON('Cut'),USE(?CutButton),ICON(ICON:Cut),STD(STD:Cut),FLAT
              BUTTON('Copy'),USE(?CopyButton),ICON(ICON:Copy),STD(STD:Copy),FLAT
              BUTTON('Paste'),USE(?PasteButton),ICON(ICON:Paste),STD(STD:Paste),FLAT
            END
          END

OknoPierwsze  WINDOW,AT(,,380,200),MDI,WALLPAPER('MyWin.GIF')
              END
```

Porównaj: CENTERED, TILED

WITHNEXT (eliminuje zjawisko owdowienia)

WITHNEXT([*siblings*])

WITHNEXT Powoduje, że dana struktura jest drukowana zawsze na tej samej stronie, co struktura bezpośrednio po niej następująca.

siblings Stała całkowita lub wyrażenie stałe określające liczbę struktur drukowanych na tej samej stronie. Jeśli pominiemy, domyślną wartością jest jeden.

Atrybut **WITHNEXT** (PROP:WITHNEXT) powoduje, że struktura DETAIL, nagłówek grupy HEADER bądź stopka grupy FOOTER są zawsze drukowane na tej samej stronie, co określona liczba struktur bezpośrednio po niej następujących. Dzięki temu struktura nigdy nie będzie drukowana na stronie „osamotniona"; eliminujemy występowanie „owdowiałych" struktur. Struktura „owdowiała" to nagłówek grupy lub pierwszy jej element, wydrukowany na poprzedniej stronie i odseparowany od pozostałych elementów.

Parametr *siblings*, jeśli występuje, określa liczbę struktur, które muszą być wydrukowane na tej samej stronie. By były policzone, struktury te muszą należeć do tej samej, bądź zagnieżdżonej, grupy BREAK. Muszą być elementami ze sobą powiązanymi. Dowolna struktura nie należąca do tej samej lub zagnieżdżonej grupy BREAK nie zostanie policzona jako jedna z wymaganych do osiągnięcia liczby *siblings*.

Przykład:

```
MojRaport  REPORT
Grupa1       BREAK(SomeVariable)
               HEADER,WITHNEXT(2)        ! zawsze drukowany z dwiema strukturami
                 ! elementy struktury
               END
CustDetail     DETAIL,WITHNEXT()         ! zawsze drukowany z jedną strukturą
                 ! elementy struktury
               END
               FOOTER
                 ! elementy struktury
               END
             END
           END
```

WITHPRIOR (eliminuje zjawisko osierocenia)

WITHPRIOR([*siblings*])

WITHPRIOR Powoduje, że dana struktura jest drukowana zawsze na tej samej stronie, co struktura bezpośrednio ją poprzedzająca

siblings Stała całkowita lub wyrażenie stałe określające liczbę struktur drukowanych na tej samej stronie. Jeśli pominiemy, domyślną wartością jest jeden.

Atrybut **WITHPRIOR** (PROP:WITHPRIOR) powoduje, że struktura DETAIL, nagłówek grupy HEADER bądź stopka grupy FOOTER są zawsze drukowane na tej samej stronie, co określona liczba struktur bezpośrednio je poprzedzających. Dzięki temu struktura nigdy nie będzie drukowana na stronie „osamotniona"; eliminujemy występowanie „osieroconych" struktur. Struktura „osierocona" to nagłówek grupy lub ostatni jej element, wydrukowany na następnej stronie i odseparowany od pozostałych elementów.

Parametr *siblings*, jeśli występuje, określa liczbę struktur, które muszą być wydrukowane na tej samej stronie. By były policzone, struktury te muszą należeć do tej samej, bądź zagnieżdżonej, grupy BREAK. Muszą być elementami ze sobą powiązanymi. Dowolna struktura nie należąca do tej samej lub zagnieżdżonej grupy BREAK nie zostanie policzona jako jedna z wymaganych do osiągnięcia liczby *siblings*.

Przykład:

```
MojRaport   REPORT
Grupa1        BREAK(SomeVariable)
                HEADER
                  ! elementy struktury
                END
CustDetail      DETAIL,WITHPRIOR()          ! zawsze drukowany z jedną strukturą
                  ! elementy struktury
                END
                FOOTER,WITHPRIOR(2)         ! zawsze drukowany z dwiema strukturami
                  ! elementy struktury
                END
              END
            END
```

WIZARD (ukrywa zakładki arkusza SHEET)

WIZARD

Atrybut **WIZARD** (PROP:WIZARD) powoduje, że zakładki TAB kontrolki SHEET są ukrywane. Pozwala to m.in. na tworzenie kreatorów, w których przechodzenie pomiędzy poszczególnymi zakładkami umożliwiają przyciski, na przykład „Dalej” i „Wstecz”.

ZOOM (Ustawia powiększanie obiektu OLE)

ZOOM

Atrybut **ZOOM** (PROP:ZOOM, tylko-do-zapisu) powoduje, że rozmiar obiektu OLE jest dopasowywany, z zachowaniem proporcji, do rozmiaru kontrolki kontenera OLE określonego przez jej atrybut AT.

10 - WYRAŻENIA

Przegląd

Wyrażenie jest formułą matematyczną, łańcuchową lub logiczną dającą w rezultacie jakąś wartość. Wyrażenie może być wykorzystywane w instrukcjach podstawiania, jako parametr procedury, indeks tablicy, czy warunek instrukcji warunkowych IF, CASE, LOOP lub EXECUTE.

Wyrażenia mogą zawierać wartości stałe, zmienne, funkcje, wszystkie połączone operatorami logicznymi, arytmetycznymi, czy łańcuchowymi.

Obliczanie wyrażeń

Wyrażenia są obliczane w oparciu o standardową kolejność operacji algebraicznych. Pierwszeństwo operacji jest kontrolowane w oparciu o typ operatora i położenie nawiasów. Każda operacja daje wynik pośredni (wewnętrzny) wykorzystywany później przy ich składaniu. Nawiasy mogą być stosowane do grupowania operacji wewnątrz wyrażenia. Wyrażenia są obliczane począwszy od najbardziej zagnieżdżonych nawiasów, a kończąc na nawiasach zewnętrznych.

Kolejność poziomów w obliczaniu wyrażeń, od najwyższego do najniższego i od lewej do prawej w ramach poziomu, jest następująca:

Poziom 1	( )	Nawiasy grupujące
Poziom 2	-	Minus (znak ujemności)
Poziom 3	Wywołanie funkcji	Pobranie zwracanej wartości
Poziom 4	^	Potęgowanie
Poziom 5	* / %	Mnożenie, dzielenie, reszta z dzielenia
Poziom 6	+ -	Dodawanie, odejmowanie
Poziom 7	&	Konkatenacja (łączenie łańcuchów)
Poziom 8	= < >	Porównanie logiczne
Poziom 9	NOT AND OR XOR	Wyrażenia boolowskie

Wyrażenia mogą dawać w rezultacie wartości liczbowe, łańcuchowe, logiczne (określenie prawda/fałsz). Wyrażenie może w ogóle nie zawierać operatorów, może być pojedynczą zmienną, wartością stała, czy rezultatem funkcji.

Operatory

Operatory arytmetyczne

Operator arytmetyczny wykonuje działanie arytmetyczne na dwóch operandach w celu uzyskania wynikającej z niego wartości. Operatorami arytmetycznymi są:

+	Dodawanie	A+B daje w rezultacie sumę A i B
-	Odejmowanie	A-B daje w rezultacie różnicę A i B
*	Mnożenie	A*B daje w rezultacie iloczyn A i B
/	Dzielenie	A/B daje w rezultacie iloraz A i B
^	Potęgowanie	A^B daje w rezultacie A podniesione do potęgi B
%	Reszta	A%B daje w rezultacie resztę z dzielenia A przez B

Ponieważ Clarion jest językiem przewidzianym do zastosować biznesowych, dzielenie przez zero daje w rezultacie wartość 0. Istnieje jednak ustawienie projektu (zero_divide, patrz *Programmer's Guide*) przywracające typowe zachowanie, przy którym wykonanie operacji dzielenia przez 0 daje w rezultacie błąd działania.

Operator konkatenacji łańcuchów

Znak ampersand (&) jest operatorem konkatenacji i służy do łączenia łańcuchów. Długość łańcucha wynikowego jest sumą długości łączonych łańcuchów. Dane numeryczne mogą być łączone z łańcuchami, zmiennymi numerycznymi, wartościami stałymi. W wielu przypadkach stosowne jest stosowanie funkcji CLIP usuwającej z łączonego łańcucha końcowe spacje.

Przykład:

```
CLIP(Imie) & ' ' Inicjal & '. ' & Nazwisko  ! Konkatenacja pól zawierających imię, inicjał i nazwisko
'TopSpeed Corporation' & ', Inc.'           ! Konkatenacja dwóch stałych
```

Porównaj: CLIP, Wyrażenia numeryczne, Reguły konwersji danych, FORMAT

Operatory logiczne

Operator logiczny na ogół porównuje dwa operandy bądź wyrażenia i daje w rezultacie wartość „prawda" bądź „fałsz". Mamy do dyspozycji dwa typy operatorów logicznych: warunkowe i boolowskie.

Operatory warunkowe porównują dwie wartości lub wyrażenia. Operatory boolowskie łączą łańcuchy, wartości numeryczne lub wyrażenia logiczne w celu określenia logiki prawda-fałsz. Operatory mogą być łączone, wtedy otrzymujemy operatory złożone.

Operatory warunkowe

=	Znak równości
<	Mniejszy niż
>	Większy niż

Operatory boolowskie

NOT	Negacja logiczna
~	Negacja logiczna
AND	Iloczyn logiczny
OR	Suma logiczna
XOR	Logiczna różnica symetryczna

Operatory łączone

<>	Różne od
~=	Różne od
NOT =	Różne od
<=	Mniejsze lub równe
=<	Mniejsze lub równe
~>	Nie większe od
NOT>	Nie większe od
>=	Większe lub równe
=>	Większe lub równe
~<	Nie mniejsze od
NOT<	Nie mniejsze od

Podczas obliczania wyrażenia logicznego wszystkie niezerowe wartości numeryczne i niepuste łańcuchy są przyjmowane jako prawda, a wartości zerowe i łańcuchy puste – jako fałsz.

Przykład:

	Rezultaty wyrażeń logicznych
A = B	Prawda, gdy A jest równe B
A < B	Prawda, gdy A jest mniejsze od B
A > B	Prawda, gdy A jest większe od B
A <> B, A ~= B, A NOT = B	Prawda, gdy A jest różne od B
A ~< B, A >= B, A NOT < B	Prawda, gdy A nie jest mniejsze od B
A ~> B, A <= B, A NOT > B	Prawda, gdy A nie jest większe od B
~ A, NOT A	Prawda, gdy A jest nieokreślone (null) lub równe zero
A AND B	Prawda, gdy A jest prawdą i B jest prawdą
A OR B	Prawda, gdy A jest prawdą lub B jest prawdą lub A i B jest prawdą
A XOR B	Prawda, gdy A jest prawdą lub B jest prawdą, ale A i B jednocześnie nie są prawdą

Stałe

Stałe numeryczne

Stałe numeryczne są niezmiennymi wartościami numerycznymi. Mogą one występować w deklaracjach danych, jako parametry procedur, czy też jako właściwości. Stała numeryczna występuje domyślnie w postaci dziesiętnej (podstawa 10), może też być reprezentowana w postaci binarnej (podstawa 2), ósemkowej (podstawa 8), szesnastkowej (podstawa 16), czy też w notacji naukowej. Stosowanie znaków formatujących, takich jak znak dolara ($), czy przecinek nie jest dopuszczalne w stałych numerycznych; można jedynie używać znaku plus lub minus oraz kropki dziesiętnej.

Stałe numeryczne w postaci dziesiętnej mogą składać się z wiodącego znaku ujemnego (minus), części całkowitej i opcjonalnej części ułamkowej.

Stałe binarne mogą zawierać opcjonalny, wiodący znak ujemny (minus), cyfry z zakresu od 0 do 1 i kończący znak „B” lub „b”.

Stałe ósemkowe mogą zawierać opcjonalny, wiodący znak ujemny (minus), cyfry z zakresu od 0 do 7 i kończący znak „O” lub „o”.

Stałe szesnastkowe mogą zawierać opcjonalny, wiodący znak ujemny (minus), cyfry z zakresu od 0 do 9 i znaki alfabetu od „A” do „F” odpowiadające liczbom z zakresu od 10 do 15 oraz kończący znak „H” lub „h”. Jeśli pierwszym znakiem stałej szesnastkowej jest litera alfabetu, musi ja poprzedzać cyfra 0.

Przykład:

```
-924            ! stałe dziesiętne
76.346
+76.346
1011b           ! stałe binarne
-1000110B
3403o           ! stałe w formacie ósemkowym
-7041312O
-1FFBh          ! stałe heksadecymalne
0CD1F74FH
```

Stałe łańcuchowe

Stała łańcuchowa jest zbiorem znaków zamkniętych w pojedynczych apostrofach. Maksymalną długością stałej łańcuchowej jest 255 znaków. Znaki, których nie można wprowadzić z klawiatury można umieszczać w stałych łańcuchowych – należy je umieścić w nawiasach tójkątnych, np. sekwencja <10><13> odpowiada klawiszowi Enter. Kody znaków ASCII mogą być reprezentowane przez stałe w formacie dziesiętnym, szesnastkowym, binarnym, ósemkowym. Umieszczenie w stałej łańcuchowej lewego nawiasu trójkątnego (<) pociąga za sobą wyszukanie pozycji, w której będzie występował znak prawego nawiasu trójkątnego (>). Dlatego też, jeśli chcemy umieścić w łańcuchu znaków po prostu znak (<) musimy wpisać go dwa razy. Jest to sygnał dla kompilatora, że nie rozpoczynamy wpisywania kodu znaku, a chcemy po prostu wstawić znak (<). Podobna sytuacja występuje w przypadku apostrofów, które pełnią rolę ograniczeń łańcucha znaków. Jeśli chcemy w nim umieścić znak apostrofu – wpisujemy dwa apostrofy, które nie są od siebie oddzielone żadnym innym znakiem (również spacją).

Kolejne wystąpienia wielu jednakowych znaków w łańcuchu mogą być reprezentowane przez *licznik powtórzeniowy*. Liczba powtórzeń danego znaku powinna być umieszczona w nawiasach klamrowych ({ }) bezpośrednio po znaku, który ma zostać powtórzony. Jeśli chcemy umieścić w łańcuchu lewy nawias klamrowy ({) musimy go wpisać, podobnie jak we wcześniej wymienionych przypadkach, dwa razy ({{).

Znak ampersand (&) jest zawsze poprawny w stałych łańcuchowych. W niektórych przypadkach, w zależności od rodzaju przypisania, może on być interpretowany jako indykator klawisza skrótu, np. w kontrolkach PROMPT; znak, przed którym występuje, zostanie po prostu podkreślony. Jeśli chcemy tego uniknąć, wpisujemy znak ampersand dwa razy (&&).

Przykład:

```
'jakiś napis'      ! stała łańcuchowa
'It''s a girl!'    ! stała łańcuchowa z osadzonym apostrofem
'<27,15>'          ! dziesiętne kody ASCII
'A << B'           ! stała łańcuchowa z osadzonym znakiem mniejszości dla wyrażenia A < B
'*{20}'            ! dwadzieścia znaków gwiazdki, specjalna notacja  powtarzająca
''                 ! łańcuch nieokreślony (pusty)
```

Typy wyrażeń

Wyrażenia numeryczne

Wyrażenia numeryczne mogą być używane jako parametry procedur, warunki instrukcji IF, CASE, LOOP, czy EXECUTE, lub też jako część instrukcji przypisania, której wynik jest umieszczany w zmiennej numerycznej. Wyrażenie numeryczne może zawierać operatory arytmetyczne i operator konkatenacji, nie może zawierać natomiast operatorów logicznych.

Stałe i zmienne łańcuchowe użyte w wyrażeniach numerycznych są konwertowane do pośrednich wartości liczbowych. Jeśli został zastosowany operator konkatenacji, wartość pośrednia jest konwertowana do wartości numerycznej po operacji konkatenacji.

Przykład:

```
Licznik + 1       ! dodaje 1 do Licznika
(1 - N * N) / R   ! N pomnożone przez N, odjęte od 1 i podzielone przez R
58 & 7854555      ! Konkatenacja numeru kierunkowego i numeru telefonu
```

Porównaj: Reguły konwersji danych

Wyrażenia łańcuchowe

Wyrażenia łańcuchowe mogą być stosowane jako parametry procedur i atrybutów lub też jako część źródłowa instrukcji przypisania, której wynik jest umieszczany w zmiennej łańcuchowej.

Wyrażenia łańcuchowe mogą zawierać pojedyncze łańcuchy lub zmienne numeryczne, jak również złożone kombinacje podwyrażeń, funkcji i operacji.

Przykład:

```
Lancuch    STRING(30)
Nazwa      STRING(10)
Waga       STRING(3)
Telefon    LONG

  CODE
    ! Konkatenacja stałej i zmiennej
    Lancuch = 'Adres:' & Kli:Adres
    !Konkatenacja wartości stałej i zmiennej sformatowanej za pomocą procedury FORMAT
    Lancuch = 'Telefon:' & ' 58-' & FORMAT(Telefon,@P###-####P)
    ! Konkatenacja stałej i zmiennej
    Lancuch = Waga & 'kg'
```

Porównaj: CLIP, Operator konkatenacji, Reguły konwersji danych, FORMAT

Wyrażenia logiczne

Wyrażenie logiczne wyliczające wartość prawda-fałsz dla warunkowych struktur sterujących IF, LOOP UNTIL oraz LOOP WHILE. Przepływa sterowania zależy od rezultatu (prawda lub fałsz) wyrażenia. Wyrażenia logiczne są wyliczane od lewej do prawej. Prawy operand AND, OR lub XOR wyrażenia logicznego powinien być wyliczany tylko wtedy, gdy może wpłynąć na rezultat. Nawiasy powinny być używany w celu wykluczenia nieoczekiwanej kolejności wyliczenia oraz w celu wymuszenia takiej kolejności. Poziomy określające kolejność operatorów logicznych przedstawiają się następująco:

Poziom 1	Operatory warunkowe
Poziom 2	~, NOT
Poziom 3	AND
Poziom 4	OR, XOR

Przykład:

```
LOOP UNTIL KEYBOARD()       ! Prawda, gdy użytkownik naciśnie dowolny klawisz
END

IF A = B THEN RETURN.       ! RETURN jeśli A jest równe B

LOOP WHILE ~ Done#          ! Loop dopóki fałsz (Done# = 0)
    !instrukcje
END

IF A >= B OR (C > B AND E = D) THEN RETURN.
  ! Prawda jeśli a >= b, również prawda gdy c > b oraz e = d.
  ! Druga część wyrażenia (po OR) jest wyliczana tylko wtedy , gdy pierwsza część nie jest
  ! prawdą.
```

Porównaj: IF, LOOP

Właściwości

[target] [$] [control] { property [, element] }

target Etykieta struktury APPLICATION, WINDOW, REPORT, VIEW lub FILE, etykieta BLOB, bądź jedna z wbudowanych zmiennych: TARGET, PRINTER lub SYSTEM. Jeśli parametr ten zostanie pominięty, przyjmowany jest kontekst reprezentowany aktualnie przez TARGET.

$ Wymagany rozgranicznik jeśli są określone zarówno *target* jak i *control*. Jeśli *target* lub *control* jest pominięte, również ten parametr pomijamy.

control Numer pola lub ekwiwalent etykiety pola dla kontrolki będącej częścią struktury *target* (APPLICATION, WINDOW lub REPORT), której właściwość nas interesuje. Jeśli pomijamy ten parametr, musi być określony *target*. Parametr *control* musi zostać pominięty, jeśli *target* jest strukturą FILE, BLOB bądź zmienną wbudowaną PRINTER lub SYSTEM.

property Stała całkowita, EQUATE lub zmienna określająca właściwość (atrybut), którą chcemy zmienić. Może to być również łańcuch, gdy dotyczy właściwości kontrolki .VBX.

element Stała lub zmienna całkowita określająca zmieniany element (dla właściwości *properties* występujących w postaci tablicy).

Składnia wyrażeń właściwości umożliwia dostęp do wszystkich atrybutów (właściwości) struktur APPLICATION, WINDOW, REPORT lub dowolnych kontrolek należących do tych struktur. W celu określenia atrybutu struktury APPLICATION, WINDOW, REPORT, VIEW lub FILE omijamy część *control* wyrażenia właściwości. Jeśli odnosimy się do kontrolki bieżącego okna omijamy część *target* wyrażenia.

Struktury danych raportu REPORT nigdy nie są domyślnym *kontekstem*. Jeżeli chcemy, w trakcie działania aplikacji, zmienić bieżący kontekst na raport REPORT musimy zastosować do tego instrukcję SETTARGET. Możemy to również uzyskać wyraźnie określając, że etykieta raportu REPORT jest kontekstem *target* zanim zmienimy dowolną właściwość struktury lub kontrolki wchodzącej w jej skład.

Wyrażenia właściwości mogą być używane w instrukcjach języka Clarion wszędzie tam, gdzie dopuszczalne jest stosowanie wyrażeń łańcuchowych. Mogą tez stanowić część źródłową lub docelową instrukcji prostego przypisania. Nie można ich stosować w instrukcjach przypisań operatorowych, takich jak +=, *=, itp. Przypisanie nowej wartości do właściwości jest instrukcją prostego przypisania, gdzie właściwość stanowi część docelową, a wartość – część źródłową. Określenie aktualnej wartości właściwości jest instrukcją prostego przypisania, gdzie część źródłową stanowi właściwość, a docelową – zmienna, w której ma być umieszczona odczytana wartość.

Wyrażenie właściwości może być także stosowane jako instrukcja wykonywalna (pozbawiona instrukcji przypisania), gdy stanowi metodę wywoływaną dla kontrolki VBX, OLE lub OCX.

Wszystkie właściwości w czasie działania programu są traktowane jako dane łańcuchowe; kompilator automatycznie dokonuje wszelkich niezbędnych konwersji typów danych. Dowolna właściwość nie posiadająca parametrów jest właściwością binarną (przełącznikiem). Właściwości binarne „występują” lub „nie występują”. Oznacza to, w tym pierwszym przypadku wartość ‘1’ lub ‘’ (null) – w drugim

przypadku. Zmiana wartości właściwości binarnej na ‘’ (null), ‘0’ (zero) lub dowolny łańcuch nienumeryczny oznacza, że właściwość taka „nie występuje”. Zmiana tej wartości na dowolna inną wartość, niż wymienione wcześniej, oznacza, że dana właściwość „występuje”.

Większość z właściwości może być zarówno sprawdzana (odczyt), jak i ustawiana (zapis). Jednak niektóre z nich są przeznaczone tylko do odczytu i nie mogą być zmieniane. Przypisanie wartości do właściwości typu „tylko do odczytu” nie daje żadnego efektu. Istnieją również właściwości „tylko do zapisu”, których wartości nie możemy odczytać.

Niektóre właściwości są tablicami zawierającymi wiele wartości. Składnia umożliwiająca odwołanie się do konkretnej właściwości *element* w tablicy polega na zastosowaniu przecinka, jako rozdzielnika właściwości *property* i jej elementu *element*.

Zmienne wbudowane (built-in)

W standardowej bibliotece uruchomieniowej Clarion zostały zdefiniowane trzy zmienne wbudowane: TARGET, PRINTER oraz SYSTEM. Używa się ich tylko podczas przypisywania wartości właściwościom kontrolek w celu zidentyfikowania właściwego kontekstu (*target*).

TARGET standardowo odnosi się do okna, które aktualnie posiada aktywność sterowania (focus). Może być jednakże ustawiany także jako referencja do okna innego wątku, czy też aktualnie drukowanego raportu. Dostajemy dzięki temu możliwość wpływania na właściwości kontrolek i okien znajdujących się w innych wątkach, a także dynamicznej zmiany właściwości kontrolek raportu podczas jego drukowania. Instrukcja SETTARGET zmienia referencję zmiennej TARGET.

PRINTER odnosi się tylko i wyłącznie do właściwości drukarki, które będą zastosowane przy następnym otwartym raporcie REPORT; i oczywiście wszystkich kolejnych, dopóki nie zostaną zmienione.

SYSTEM określa globalne właściwości wykorzystywane przez całą aplikację. Istnieje pewna liczba właściwości (czasu działania aplikacji), które wykorzystują zmienną SYSTEM do zmiany swych wartości.

Przykład:

```
MainWin APPLICATION('Moja aplikacja),SYSTEM,MAX,ICON('MyIcon.ICO'),STATUS,RESIZE
            MENUBAR
              MENU('File'),USE(?FileMenu)
                ITEM('Open...'),USE(?OpenFile)
                ITEM('Close'),USE(?CloseFile),DISABLE
                ITEM('E&xit'),USE(?MainExit)
              END
              MENU('Help'),USE(?HelpMenu)
                ITEM('Contents'),USE(?HelpContents),STD(STD:HelpIndex)
                ITEM('Search for Help On...'),USE(?HelpSearch),STD(STD:HelpSearch)
                ITEM('How to Use Help'),USE(?HelpOnHelp),STD(STD:HelpOnHelp)
                ITEM('About MyApp...'),USE(?HelpAbout)
              ..
              TOOLBAR
                BUTTON('Open'),USE(?OpenButton),ICON(ICON:Open)
            ..

  CODE
    OPEN(MainWin)
    MainWin{PROP:Text} = 'A New Title'            ! Zmiana tytułu okna
    ?OpenButton{PROP:ICON} = ICON:Asterisk        ! Zmiana ikony przycisku
    ?OpenButton{PROP:AT,1} = 5                    ! Zmiana pozycji x przycisku
    ?OpenButton{PROP:AT,2} = 5                    ! Zmiana pozycji y przycisku
```

```
IF MainWin$?HelpContents{PROP:STD} <> STD:HelpIndex
  MainWin$?HelpContents{PROP:STD} = STD:HelpIndex
END
MainWin{PROP:MAXIMIZE} = 1                ! Rozwinięcie na cały ekran
ACCEPT
  CASE ACCEPTED()                         ! Która kontrolka została wybrana?
  OF ?OpenFile                            ! Wybrano polecenie Open... z menu
  OROF ?OpenButton                        ! Wciśnięto przycisk Open w pasku narzędzi
    START(OpenFileProc)                   ! Start nowego wątku
  OF ?MainExit                            ! Wybrano polecenie Exit z menu
  OROF ?MainExitButton                    ! Wciśnięto przycisk Exit w pasku narzędzi
    BREAK                                 ! Przerwanie pętli ACCEPT
  OF ?HelpAbout                           ! Wybór polecenia About...
    HelpAboutProc                         ! Wywołanie procedury z wizytówką aplikacji
  END
END
CLOSE(MainWin)                            ! Zamknięcie APPLICATION
RETURN
```

Porównaj: SETTARGET, Właściwości runtime

Wyliczanie wyrażeń dynamicznych (runtime)

Clarion daje nam możliwość wyliczania wyrażeń dynamicznie tworzonych podczas działania aplikacji, a nie tylko tych zdefiniowanych „na sztywno" w trakcie pisania jej kodu. Dzięki temu aplikacja napisana w Clarionie posiada zdolność do konstruowania wyrażeń „w locie". Pozwala to tym samym jej końcowemu użytkownikowi na samodzielne wprowadzanie wyrażeń, które mają zostać wyliczone.

Wyrażenie jest matematyczną lub logiczną formułą, która daje w efekcie wartość; nie jest ono kompletną instrukcją języka Clarion. Wyrażenia mogą zawierać wartości stałe, zmienne lub procedury, które zwracają rezultat. Wszystkie te składniki muszą być połączone operatorami logicznymi i/lub arytmetycznymi.

Wyrażenie może występować jako część źródłowa instrukcji przypisania, parametr procedury, indeks tablicy, warunek instrukcji IF, CASE, LOOP lub EXECUTE.

Dowolna zmienna oraz większość wewnętrznych procedur Clariona może występować jako część łańcucha tworzącego wyrażenie dynamiczne (runtime). Procedury definiowane przez użytkownika spełniające określone warunki (opisane w dokumentacji instrukcji BIND) również mogą być stosowane w łańcuchach tworzących wyrażenia dynamiczne.

Wszystkie standardowe składniki wyrażeń Clariona można stosować w łańcuchach wyrażeń dynamicznych. Dotyczy to znaków grupujących, wszystkich operatorów arytmetycznych, logicznych i łańcuchowych. Wyrażenia dynamiczne są wyliczane tak jak i inne wyrażenia Clariona z zachowaniem wszystkich standardowych reguł kolejności operatorów opisanych w punkcie Obliczanie wyrażeń.

Istnieją trzy podstawowe reguły, których musimy przestrzegać stosując wyrażenia dynamiczne:

- Zmienne użyte w wyrażeniu dynamicznym muszą być zadeklarowane za pomocą instrukcji BIND.
- Wyrażenie musi zostać zbudowane. Pociąga to za sobą odpowiednie połączenie w jeden łańcuch parametrów określonych (wybranych) przez użytkownika lub umożliwienie mu bezpośredniego wpisania wyrażenia, co wymaga oczywiście znajomości składni języka Clarion.
- Wyrażenie jest przekazywane do funkcji EVALUATE, która oblicza jego rezultat. Jeśli wyrażenie nie jest prawidłowym wyrażeniem w sensie składni języka Clarion, zwracany jest kod błędu (odczytujemy za pomocą funkcji ERRORCODE).

Gdy tylko wyrażenie zostanie wyliczone, jego rezultat może być wykorzystywany tak samo, jak rezultat dowolnego innego wyrażenia statycznego. Na przykład, łańcuch wyrażenia dynamicznego pozwala na zdefiniowanie filtra eliminującego z widoku rekordy nie spełniające określonych warunków (wyrażenie FILTER struktury VIEW pośrednio zawsze jest łańcuchem wyrażenia dynamicznego).

BIND (deklaruje zmienną dla łańcucha wyrażenia dynamicznego)

```
BIND( | name,variable    | )
      | name,procedure   |
      | group             |
```

BIND Identyfikuje zmienne, które mogą być stosowane w wyrażeniach dynamicznych.

name Stała łańcuchowa zawierająca identyfikator, który będzie stosowany w wyrażeniach dynamicznych. Może on być taki sam, jak etykieta *variable* lub *procedure*.

variable Etykieta dowolnej zmiennej (włączając w to pola FILE, GROUP, czy QUEUE) lub przekazany parametr. Jeśli jest to tablica, musi ona mieć tylko jeden wymiar.

procedure Etykieta procedury języka Clarion, która daje w rezultacie wartość typu STRING, REAL lub LONG. Jeśli do tej procedury są przekazywane parametry, muszą one być typu STRING (przekazywanie przez wartość, nie przez adres) i nie mogą to być parametry pomijalne.

group Etykieta struktury GROUP, RECORD lub QUEUE zadeklarowanej z atrybutem BINDABLE.

Instrukcja **BIND** deklaruje nazwę logiczną stosowaną do identyfikowania zmiennej lub procedury zdefiniowanej przez użytkownika, które mają być wykorzystywane w wyrażeniach dynamicznych (definiowanych w trakcie działania programu). Ta zmienna lub własna procedura musi zostać zidentyfikowana, za pomocą instrukcji BIND, zanim zostanie użyta w wyrażeniu łańuchowym wykorzystywanym przez funkcję EVALUATE, czy atrybut FILTER struktury VIEW.

BIND(*name,variable*)

Nazwa *name* jest wykorzystywana w wyrażeniach w miejsce zmiennej o etykiecie *variable*.

BIND(*name,procedure*)

Nazwa *name* jest wykorzystywana w wyrażeniach w miejsce etykiety procedury *procedure*.

BIND(*group*)

Deklaruje wszystkie zmienne struktury GROUP, RECORD lub QUEUE (posiadającej atrybut BINDABLE) jako dostępne dla wyrażeń dynamicznych. Zawartość atrybutu NAME każdej zmiennej staje się logiczną nazwą wykorzystywaną w wyrażeniach dynamicznych. Jeśli nie występuje atrybut NAME, stosowana jest etykieta zmiennej (włączając w to prefiks). Struktury GROUP, RECORD i QUEUE zadeklarowane z atrybutem BINDABLE otrzymują miejsce w pliku .EXE przydzielone do przechowywania nazw wszystkich elementów danych wchodzących w skład takiej struktury. Pociąga to za sobą zwiększenie objętości programu (tym samym – wzrost zapotrzebowania na zasoby systemowe). Dodatkowo trzeba pamiętać, że im więcej

zmiennych bindujemy w danym czasie, tym wolniej działa funkcja EVALUATE. Z tego między innymi względu BIND(*group*) powinno być stosowane tylko wtedy, gdy zdecydowaną większość pól struktury będziemy wykorzystywali w wyrażeniu dynamicznym.

Przykład:

```
    PROGRAM
      MAP
        AllCapsFunc(STRING),STRING                          ! Procedura Clarion
      END

Header      FILE,DRIVER('Clarion'),PRE(Hea),BINDABLE    ! Deklaracja układu nagłówka pliku
OrderKey      KEY(Hea:OrderNumber)
Record        RECORD
OrderNumber     LONG
ShipToName      STRING(20)
                . .
StringVar   STRING(20)

  CODE
    BIND('ShipName',Hea:ShipToName)
    BIND('SomeFunc',AllCapsFunc)
    BIND('StringVar',StringVar)
    StringVar = 'SMITH'
    CASE EVALUATE('StringVar = SomeFunc(ShipName)')
    OF ''
      IF ERRORCODE()
        MESSAGE('Error ' & ERRORCODE() & ' -- ' & ERROR())
      ELSE
        MESSAGE('Unkown error evaluating expression')
      END
    OF '0'
      DO NonSmithProcess
    OF '1'
      DO SmithProcess
    END

AllCapsFunc  PROCEDURE(PassedString)
  CODE
    RETURN(UPPER(PassedString))
```

Porównaj: UNBIND, EVALUATE, PUSHBIND, POPBIND, FILTER

EVALUATE (oblicza rezultat łańcucha wyrażenia dynamicznego)

EVALUATE(*expression*)

EVALUATE Wylicza wyrażenie dynamiczne reprezentowane przez łańcuch..

expression Stała lub zmienna łańcuchowa zawierająca wyrażenie do wyliczenia.

Funkcja **EVALUATE** wylicza wyrażenie *expression* i zwraca rezultat w postaci wartości STRING. Jeśli wyrażenie *expression* nie spełnia reguł poprawnego wyrażenia Clarion, rezultatem jest łańcuch pusty (‘’) i ustawiany jest numer błędu, który można odczytać za pomocą funkcji ERRORCODE. Logiczne wyrażenie daje w rezultacie łańcuch zawierający napis (‘0’) lub (‘1’), podczas gdy wyrażenie arytmetyczne daje w rezultacie faktyczną wartość (w postaci łańcucha).

Jeśli chcemy w wyrażeniu użyć znaku mniejszości (<) musimy wpisać go dwa razy (<<), by uniknąć błędów kompilacji. Im więcej zmiennych zbindujemy w jednym czasie, tym wolniej będzie działać funkcja EVALUATE. Z tego powodu instrukcja BIND(*group*) powinna być używana tylko wtedy, gdy większość pól struktury *group* będzie wykorzystywana w wyrażeniu dynamicznym. Instrukcja UNBIND służy do zwolnienia wszystkich zmiennych i procedur użytkownika, które nie są w danej chwili wykorzystywane w wyrażeniu dynamicznym.

Typ rezultatu: STRING

Komunikaty błędów: 800 Illegal Expression

801 Variable Not Found

Przykład:

```
    MAP
      AllCapsFunc PROCEDURE(STRING),STRING                    ! Procedura Clarion
    END

Header      FILE,DRIVER('Clarion'),PRE(Hea),BINDABLE          ! Deklaracja układu nagłówka pliku
OrderKey      KEY(Hea:OrderNumber)
Record        RECORD
OrderNumber     LONG
ShipToName      STRING(20)
              . .
StringVar     STRING(20)

  CODE
    BIND('ShipName',Hea:ShipToName)
    BIND('SomeFunc',AllCapsFunc)
    BIND('StringVar',StringVar)
    StringVar = 'SMITH'
    CASE EVALUATE('StringVar = SomeFunc(ShipName)')
    OF ''
      IF ERRORCODE() THEN MESSAGE('Error ' & ERRORCODE() & ' -- ' & ERROR()).
    OF '0'
      DO NonSmithProcess
    OF '1'
      DO SmithProcess
    END

AllCapsFunc  PROCEDURE(PassedString)
  CODE
    RETURN(UPPER(PassedString))
```

Porównaj: BIND, UNBIND, PUSHBIND, POPBIND, FILTER

POPBIND (odtwarza obszar nazw łańcucha wyrażenia dynamicznego)

POPBIND

Instrukcja **POPBIND** odtwarza przestrzeń nazw zdefiniowaną przez poprzednią instrukcję BIND. Przywraca to poprzedni obszar zmiennych stosowanych przez poprzednią instrukcję BIND.

Przykład:

```
SomeProc       PROCEDURE
OrderNumber      LONG
Item             LONG
Quantity         SHORT
  CODE
    BIND('OrderNumber',OrderNumber)
    BIND('Item',Item)
    BIND('Quantity',Quantity)
    AnotherProc                              ! Wywołanie innej procedury
    UNBIND('OrderNumber',OrderNumber)
    UNBIND('Item',Item)
    UNBIND('Quantity',Quantity)

AnotherProc  PROCEDURE
OrderNumber  LONG
Item         LONG
Quantity     SHORT
  CODE
    PUSHBIND                             ! Utworzenie nowego zasięgu dla BIND
    BIND('OrderNumber',OrderNumber)      ! Bindowanie zmiennych z tymi samymi nazwami w nowym zasięgu
    BIND('Item',Item)
    BIND('Quantity',Quantity)
    !Do some Processing
    UNBIND('OrderNumber')
    UNBIND('Item')
    UNBIND('Quantity')
    POPBIND                              ! Przywrócenie poprzedniego zasięgu dla BIND
```

Porównaj: PUSHBIND, EVALUATE

PUSHBIND (zachowuje obszar nazw łańcucha wyrażenia dynamicznego)

PUSHBIND([*clearflag*])

PUSHBIND Tworzy nowy obszar dla występujących po sobie instrukcji BIND.

clearflag Stała lub zmienna całkowita zawierająca wartość zero (0) lub jeden (1). Gdy jest to zero, obszar nazw instrukcji BIND jest czyszczony ze wszystkich zmiennych i procedur zbindowanych wcześniej. Gdy jest to jeden wszystkie zmienne I procedury zbindowane wcześniej są zachowane. Pominięcie *clearflag* jest jednoznaczne z ustawieniem tego parametru na wartość 0.

Instrukcja **PUSHBIND** tworzy nowy obszar dla następujących po sobie instrukcji BIND. Obszar ten kończy się w momencie wystąpienia instrukcji POPBIND. Powoduje to utworzenie nowego obszaru dla następujących po sobie instrukcji BIND, umożliwiając tworzenie nowych nazw dla zmiennych bez powodowania konfliktów z nazwami z poprzedniego obszaru.

Przykład:

```
SomeProc      PROCEDURE
OrderNumber   LONG
Item          LONG
Quantity      SHORT
  CODE
    BIND('OrderNumber',OrderNumber)
    BIND('Item',Item)
    BIND('Quantity',Quantity)
    AnotherProc                                  ! Wywołanie innej procedury
    UNBIND('OrderNumber',OrderNumber)
    UNBIND('Item',Item)
    UNBIND('Quantity',Quantity)

AnotherProc  PROCEDURE
OrderNumber  LONG
Item         LONG
Quantity     SHORT
  CODE
    PUSHBIND                          ! Utworzenie nowego zasięgu dla BIND
    BIND('OrderNumber',OrderNumber)   ! Bindowanie zmiennych z tymi samymi nazwami w nowym zasięgu
    BIND('Item',Item)
    BIND('Quantity',Quantity)
    !Do some Processing
    UNBIND('OrderNumber')
    UNBIND('Item')
    UNBIND('Quantity')
    POPBIND                           ! Przywrócenie poprzedniego zasięgu dla BIND
```

Porównaj: POPBIND, EVALUATE

UNBIND (zwalnia zmienną dla łańcucha wyrażenia dynamicznego)

UNBIND([*name*])

UNBIND Zwalnia zmienne stosowane w łańcuchu wyrażenia dynamicznego.

name Stała łańcuchowa określająca identyfikator ewaluatora wyrażenia dynamicznego. Jeśli pominiemy ten parametr, wszystkie zbindowane zmienne są zwalniane.

Instrukcja **UNBIND** zwalnia logiczne nazwy zbindowane wcześniej za pomocą instrukcji BIND. Im więcej zmiennych zbindujemy w danym czasie, tym wolniej działa funkcja EVALUATE. Instrukcja UNBIND powinna być wykorzystywana do zwalniania wszystkich zmiennych i procedur użytkownika, które nie są już wykorzystywane w łańcuchu wyrażenia dynamicznego.

Przykład:

```
PROGRAM
  MAP
    AllCapsFunc(STRING),STRING                    ! Procedura Clarion
  END

Header    FILE,DRIVER('Clarion'),PRE(Hea)         ! Deklaracja układu nagłówka pliku
AcctKey       KEY(Hea:AcctNumber)
OrderKey      KEY(Hea:OrderNumber)
Record        RECORD
AcctNumber      LONG
OrderNumber     LONG
ShipToName      STRING(20)
ShipToAddr      STRING(20)
ShipToCity      STRING(20)
ShipToState     STRING(20)
ShipToZip       STRING(20)
              . .
Detail        FILE,DRIVER('Clarion'),PRE(Dtl),BINDABLE    ! Bindowalna struktura RECORD
OrderKey      KEY(Dtl:OrderNumber)
Record          RECORD
OrderNumber     LONG
Item            LONG
Quantity        SHORT
                . .

  CODE
    BIND('ShipName',Hea:ShipToName)
    BIND(Dtl:Record)
    BIND('SomeFunc',AllCapsFunc)
    UNBIND('ShipName')                            ! UNBIND zmiennej
    UNBIND('SomeFunc')                            ! UNBIND procedury języka Clarion
    UNBIND                                        ! UNBIND wszystkich zbindowanych zmeinnych

AllCapsFunc PROCEDURE(PassedString)
  CODE
    RETURN(UPPER(PassedString))
```

Porównaj: BIND, EVALUATE, PUSHBIND, POPBIND

11 - PRZYPISANIA

Instrukcje przypisania

Proste przypisania

```
destination = source
```

destination Etykieta zmiennej lub właściwość kontrolki.

source Stała numeryczna lub łańcuchowa, zmienna, procedura, wyrażenie bądź właściwość struktury danych.

Znak = powoduje przypisanie wartości *source* do *destination*; innymi słowy – skopiowanie wartości reprezentowanej przez wyrażenie *source* do zmiennej *destination*. Jeśli *destination* i *source* mają odmienne typy danych, wykonywana jest automatycznie konwersja zgodnie z zasadami konwersji danych obowiązującymi w języku Clarion.

Przykład:

```
StringVar    STRING(10)
LongVar      LONG
RealVar      REAL

  CODE
    StringVar = 'JONES'                    ! Zmienna = stała łańcuchowa
    RealVar = 3.14159                      ! Zmienna = stała numeryczna
    RealVar = SQRT(1 - Sine * Sine)        ! Zmienna = rezultat funkcji
    LongVar = B + C + 3                    ! Zmienna = wyrażenie numeryczne
    StringVar = CLIP(FirstName) & ' ' Initial & '. ' & LastName
    ! Zmienna = string expression
    StringVar = '10'                       ! Przypisanie danej numerycznej do łańcucha, następnie
    Longvar = StringVar                    ! automatyczna konwersja rezultatu i Longvar powinna zawierać: 10
```

Porównaj: Reguły konwersji danych, Wyrażenia właściwości

Przypisania operatorowe

```
Destination += source
Destination -= source
Destination *= source
Destination /= source
Destination ^= source
Destination %= source
```

destination Etykieta zmiennej; nie może to być właściwość kontrolki.

source Stała, zmienna lub wyrażenie.

Instrukcje przypisań operatorowych przeprowadzają operacje na *destination* i *source*, następnie umieszczają wynik w *destination*. Instrukcje przypisania operatorowego są znacznie bardziej efektywne, niż odpowiadające im zwykłe operacje

Przykład:

Funkcjonalne ekwiwalenty przypisań operatorowych

```
A += 1      A = A + 1
A -= B      A = A - B
A *= -5     A = A * -5
A /= 100    A = A / 100
A ^= I + 1  A = A ^ (I + 1)
A %= 7      A = A % 7
```

Porównaj: Reguły konwersji danych, Wyrażenia właściwości

Głębokie przypisanie

Destination :=: source

destination Etykieta struktury typu GROUP, RECORD lub QUEUE, bądź tablica wielowymiarowa.

source Etykieta struktury typu GROUP, RECORD lub QUEUE, bądź stała numeryczna lub łańcuchowa, zmienna, procedura lub wyrażenie.

Znak **:=:** wykonuje instrukcję głębokiego przypisania, które przeprowadza wiele przypisań elementów struktury *destination* do odpowiadających im elementów struktury *source*. Przypisania są wykonywane tylko pomiędzy wewnętrznymi zmiennymi poszczególnych struktur, które posiadają dokładnie takie same etykiety (wyłączając prefiksy). Kompilator wyszukuje w zagnieżdżonych strukturach GROUP pasujących do siebie etykiet. Zmienna w strukturze *destination*, dla której nie stwierdzono odpowiednika w strukturze *source*, nie jest zmieniana.

Wykonanie głębokiego przypisania ma taki sam skutek, jak przypisania wykonane dla poszczególnych zmiennych struktury złożonej. Oznacza to, że znajdują tu zastosowanie reguły konwersji, tak samo jak przy zwykłym przypisywaniu zmiennych. Na przykład, etykieta z zagnieżdżonej struktury GROUP *source* może pasować do etykiety z zagnieżdżonej struktury GROUP *destination*, a może również pasować do zwykłej zmiennej. W tym przypadku, zagnieżdżona struktura *source* jest przypisywana do struktury *destination* jako STRING, w taki sposób, w jaki jest obsługiwane zwykłe przypisanie grupy GROUP.

Nazwa tablicy *source* może pasować do tablicy *destination*. W takim przypadku, każdy element tablicy *source* jest przypisywany do odpowiadającego mu elementu tablicy *destination*. Jeśli tablica *source* posiada więcej lub mniej elementów, niż tablica *destination*, tylko pasujące elementy są przypisywane do elementów tablicy *destination*.

Jeśli *destination* jest zmienna tablicową nie będącą częścią struktury GROUP, RECORD lub QUEUE, a *source* jest stałą, zmienną lub wyrażeniem, to każdy element tablicy *destination* jest inicjowany wartością *source*. Jest to bardziej efektywna metoda inicjalizowania elementów tablicy określona wartością, niż wykonanie np. odpowiedniej pętli LOOP.

Destination lub *source* mogą być nazwami struktury CLASS, która, w takim wypadku, jest traktowana jak grupa GROUP. W takim wypadku postąpimy niezgodnie z koncepcją enkapsulacji, gdyż głębokie przypisanie jest operacją ignorująca strukturę, stąd nie jest tu zalecane.

Przykład:

```
Group1      GROUP
S             SHORT
L             LONG
            END


Group2      GROUP
L             SHORT
S             REAL
T             LONG
            END
ArrayField  SHORT,DIM(1000)

  CODE
    Group2 :=: Group1                 ! Jest równoważne do:
                                      ! Group2.S = Group1.S
                                      ! Group2.L = Group1.L
                                      ! i przeprowadza wszelkie niezbędne konwersje danych
    ArrayField :=: 7                  ! Jest równoważne do:
                                      ! LOOP I# = 1 to 1000
                                        ! ArrayField[I#] = 7
                                      ! END
```

Porównaj: GROUP, RECORD, QUEUE, DIM

Przypisania referencyjne

```
destination &= source
```

destination Etykieta zmiennej referencyjnej.

source Może to być:

- Etykieta zmiennej lub struktury danych tego samego typu, co typ, do którego referencję stanowi *destination.*
- Etykieta innej zmiennej referencyjnej, tego samego typu, co *destination.*
- Procedura, która zwraca rezultat, którego typ może być zarejestrowany przez *destination.*
- Wyrażenie (dające wartość LONG, tak jak rezultat funkcji ADDRESS), które definiuje adres zmiennej w pamięci, która jest tego samego typu, co typ wskazywany przez *destination* (musi to być referencja na prosty typ danych, za wyjątkiem typów STRING, CSTRING oraz PSTRING).
- Wbudowana zmienna nieokreślona NULL.

Znak **&=** powoduje wykonanie instrukcji przypisania referencyjnego. Instrukcja przypisania referencyjnego przypisuje referencję do zmiennej *source* do zmiennej referencyjnej *destination*. Gdy jest używana w wyrażeniach warunkowych (takich, jak np. instrukcja IF), instrukcja przypisania referencyjnego ustala zgodność referencji (czy dwie zmienne referencyjne wskazują na tę samą rzecz?).

W zależności od typu danych na który wskazuje referencja, zmienna referencyjna *destination* może otrzymać albo adres pamięci *source*, albo bardziej złożoną wewnętrzną strukturę danych (opisującą lokalizację i typ danej *source*).

Gdy *source* jest wbudowana zmienną NULL, przypisanie referencyjne może albo wyczyścić zmienną referencyjną *destination*, albo sprawdzić, czy referencja nie jest czasem nieokreślona (w wyrażeniach warunkowych).

Deklaracje zmiennej referencyjnej *destination* i jej źródła *source* muszą się pokrywać, za wyjątkiem sytuacji, gdy *destination* jest zadeklarowane jako zmienna typu ANY); przypisania referencyjne nie pociągają za sobą automatycznej konwersji typów. Dla przykładu, instrukcja przypisania referencyjnego do zmiennej *destination* zadeklarowanej jako &QUEUE musi zawierać *source*, którym jest albo inna zmienna referencyjna &QUEUE, albo etykieta struktury QUEUE, albo adres procedury w postaci ADDRESS(JakaśKolejka). Z drugiej strony, jeśli *destination* jest referencją do łańcucha (&STRING), *source* może być strukturą danych traktowaną jak dana łańcuchowa, gdy jest adresowana jako pojedyncza jednostka (GROUP, RECORD, QUEUE, MEMO).

Przykład:

```
Queue1      QUEUE
ShortVar      SHORT
LongVar1      LONG
LongVar2      LONG
            END
QueueRef   &QUEUE                    ! Referencja do QUEUE, tylko
Queue1Ref &Queue1                    ! Referencja do QUEUE zdefiniowana dokładnie jako Queue1, tylko
LongRef    &LONG                     ! Referencja do LONG, tylko
LongRef2   &LONG                     ! Referencja do LONG, tylko

  CODE
    QueueRef &= Queue1                    ! Przypisanie referencji QUEUE
    Queue1Ref &= Queue1                   ! Przypisanie referencji QUEUE
    IF Queue1Ref &= QueueRef              ! Czy wskazują tę samą kolejkę QUEUE?
      MESSAGE('Both Pointing at same QUEUE')
    END
    IF SomeCondition                      ! Ewaluacja warunku
      LongRef &= Queue1.LongVar1          ! i referencja na odpowiednią zmienną
    ELSE
      LongRef &= Queue1.LongVar2
    END
    LongRef += 1                          ! Inkrementacja albo LongVar1 albo LongVar2
                                          ! w zależności, która jest referencjonowana
    IF LongRef2 &= NULL                   ! Wykrycie zmiennej referencyjnej bez referencji i
      LongRef2 &= LongRef                 ! utworzenie drugiej referencji do tej samej danej
    END
    LongRef &= ADDRESS(Queue1.LongVar1) ! Przypisanie referencyjne adresu danej prostego typu danych
```

Porównaj: Zmienne referencyjne, ANY, NEW

CLEAR (wyczyszczenie zmiennej)

CLEAR(*label* [, *n*])

CLEAR Usuwa wartość ze zmiennej.

label Etykieta zmiennej (za wyjątkiem zmiennych typu BLOB), grupy GROUP, rekordu RECORD, kolejki QUEUE, klasy CLASS lub pliku FILE. Jeśli zmienna posiada atrybut DIM, czyszczona jest zawartość całej tablicy (wszystkich jej elementów). Pojedynczy element tablicy nie może być czyszczony za pomocą CLEAR.

n Stała numeryczna; wartość 1 lub -1. Jeśli jest pominięta, wartości numeryczne są zerowane, wartość zmiennych łańcuchowych jest zamieniana na spacje, zmienne typu PSTRING i CSTRING są ustawiane jako zmienne o zerowej długości.

Instrukcja **CLEAR** usuwa wartość ze zmiennej *label*. Występowanie parametru *n* oznacza wartość, która jest usuwana w czyszczonej zmiennej różną niż 0 lub spację. Jeśli *n* jest równe 1, zmienna *label* jest ustawiana na najwyższą dopuszczalną dla jej typu danych wartość. W przypadku zmiennych typu STRING, PSTRING oraz CSTRING jest to zawsze kod ASCII 255. Jeśli *n* jest równe -1, zmienna *label* jest ustawiana na najniższą możliwą dla jej typu wartość. W przypadku zmiennych typu STRING jest to zerowy kod ASCII (0). Dla typów PSTRING oraz CSTRING długość łańcucha staje się zerowa.

Jeśli *label* reprezentuje strukturę typu GROUP, RECORD lub QUEUE, wszystkie zmienne w tej strukturze są czyszczone, a wszystkie zmienne referencyjne stają się nieokreślone - NULL.

Jeśli *label* odnosi się do struktury FILE, a parametr *n* jest pominięty, wszystkie zmienne struktury FILE (włączając w to również pola MEMO i/lub BLOB) są czyszczone.

Jeśli *label* identyfikuje strukturę CLASS lub obiekt dziedziczący ze struktury CLASS, wszystkie jego zmienne są czyszczone i wszystkie zmienne referencyjne stają się nieokreślone - NULL.

Przykład:

```
MyQue    QUEUE
F1         LONG
F2         STRING(20)
F3         &CSTRING            ! Referencja do CSTRING
F4         ANY                 ! ANY może być zmienną referencyjną do dowolnego prostego typu danych
         END
  CODE
    CLEAR(MyQue)                 ! Równoważne z:
                                 ! MyQue.F1 = 0
                                 ! MyQue.F2 = ''
                                 ! MyQue.F3 &= NULL
                                 ! MyQue.F4 &= NULL
    CLEAR(Count)                 ! Wyczyszczenie zmiennej
    CLEAR(Cus:Record)            ! Wyczyszczenie struktury rekordu
    CLEAR(Customer)              ! Wyczyszczenie struktury rekordu, memo i blob
    CLEAR(Amount,1)              ! Wyczyszczenie zmiennej do najwyższej możliwej wartości
    CLEAR(Amount,-1)             ! Wyczyszczenie zmiennej do najniższej możliwej wartości
```

Porównaj: Instrukcje przypisania referencyjnego, GROUP, RECORD, QUEUE, DIM, CLASS, ANY

Reguły konwersji typów danych

Język Clarion zapewnia automatyczną konwersję różnych typów danych. Istnieją sytuacje, że pewne przypisania będą generowały nierówne zbiory źródłowe i wynikowe. Może to prowadzić do nieoczekiwanych błędów.

Typy podstawowe

W celu zapewnienia automatycznej konwersji typów danych, Clarion stosuje wewnętrznie cztery typy bazowe. Do tych typów są konwertowane wszystkie dane, na których są wykonywane operacje. Typami tymi są : STRING, LONG, DECIMAL oraz REAL. Wszystkie z nich są standardowymi typami danych Clariona.

Typ bazowy STRING jest stosowany jako typ pośredni dla wszystkich operacji łańcuchowych. Typy LONG, DECIMAL i REAL są wykorzystywane w operacjach arytmetycznych. O tym, który z nich zostanie zastosowany, decyduje oryginalny typ danych, rodzaj operacji i operandy.

Poniżej pogrupowano typy danych wg ich typów bazowych.

Typ bazowy LONG:
BYTE
SHORT
USHORT
LONG
DATE
TIME
Stałe całkowite
Łańcuchy zadeklarowane ze wzorcem @P

Typ bazowy DECIMAL:
ULONG
DECIMAL
PDECIMAL
STRING(@Nx.y)
Stałe DECIMAL

Typ bazowy REAL:
SREAL
REAL
BFLOAT4
BFLOAT8
STRING(@Ex.y)
Stałe w notacji naukowej
Parametry o nieokreślonych typach (? i *?)

Typ bazowy STRING:
STRING
CSTRING
PSTRING
Stałe łańcuchowe

Typy DATE i TIME są najpierw konwertowane do standardowego formatu daty lub czasu Clariona i tym samym otrzymują typ bazowy LONG dla wszystkich operacji.

W większości przypadków, wewnętrzne wykorzystanie typów bazowych przez Clarion jest niewidoczne dla programisty I nie wymaga uwzględnienia podczas kodowania aplikacji. Jednakże, opracowywanie aplikacji biznesowych, a w szczególności przetwarzanie danych numerycznych posiadających część ułamkową

(waluta), stosowanie typów, dla których typ DECIMAL jest typem bazowym, różni się nieco od przetwarzania danych o typie bazowym REAL.

- DECIMAL pozwala na stosowanie 31 cyfr znaczących, podczas gdy REAL – tylko 15.
- DECIMAL automatycznie zaokrągla liczby z precyzją określona w deklaracji danych, podczas gdy REAL będzie nam stwarzał problemy z zaokrągleniami wynikające z translacji z formatu dziesiętnego (podstawa 10) do binarnego (podstawa 2) w celu dokonania obliczeń przez jednostkę zmiennoprzecinkową.
- Na komputerach nie posiadających jednostki zmiennoprzecinkowej (koprocesora arytmetycznego), przetwarzanie liczb typu DECIMAL jest wyraźnie szybsze, niż przetwarzanie liczb typu REAL.
- Operacje na liczbach DECIMAL są ściślej związane z konwencjonalną arytmetyką dziesiętną.

Operacje i procedury BCD

Clarion posiada bibliotekę Binary Coded Decimal (BCD) dla operacji i procedur wykonywanych w sposób podobny jak tradycyjna arytmetyka i jej obliczenia przeprowadzane na papierze. Operacje te wykorzystują wewnętrzne, pośrednie wartości o dokładności 31 cyfr po obu stronach kropki dziesiętnej.

Wielką zaletą operacji BCD jest to, że bardzo łatwo je "zobaczyć", ponieważ wykonują się tak samo, jak podczas obliczeń dokonywanych ołówkiem na papierze. Wystarczy wyobrazić sobie prowadzenie obliczeń poczynając od prawej strony i skreślając cyfry, które zostały już dodane, odjęte, czy pomnożone.

Dysponowanie 31 stałymi pozycjami po obu stronach kropki dziesiętnej nie pozwala na przedstawienie tych wszystkich liczb w systemie BCD, które mogą być przedstawione za pomocą typu REAL. Dlatego zrozumienie zasad konwersji jest bardzo przydatne.

Generalnie, jedyne przypadki, gdy wystąpi niedopełnienie mają miejsce przy dzieleniu przez wielokrotność liczby 3. Na przykład:

```
100000/3 = 33333.33333333333333333333333333333
(100000/3)-INT(100000/3)*100000 = 33333.33333333333333333333333300000
```

Czas obliczeń BCD ściśle zależy od typu danych; proporcje są analogiczne, jak w przypadku obliczeń dokonywanych ręcznie. Im dłuższa liczba, tym dłuższy czas obliczeń. Istnieją jednak pewne triki (np. manipulowanie potęgą liczby dziesięć) pozwalające na znaczne przyspieszenie działania biblioteki BCD.

Następujące operacje mogą być wykonywane jako operacje BCD:

Dodawanie (+), **Odejmowanie** (-), **Mnożenie** (*)

Są wykonywane jako operacja BCD, gdy żaden z operandów nie jest typu bazowego REAL (są LONG lub DECIMAL) i jeden posiada typ bazowy DECIMAL. Dowolne cyfry pojawiające się na prawo od $1e^{31}$ znikają, dowolne cyfry na lewo od $1e^{-30}$ są zaokrąglane.

Dzielenie (/) Jest wykonywane jako operacja BCD, gdy żaden z operandów nie jest typu bazowego REAL (oba są LONG lub DECIMAL). Dowolne cyfry pojawiające się na prawo od $1e^{31}$ znikają, dowolne cyfry na lewo od $1e^{-30}$ są zaokrąglane.

Eksponent (^) Jest wykonywany jako operacja BCD, gdy pierwszy operand ma typ bazowy DECIMAL lub LONG, a drugi - LONG. Dowolne cyfry pojawiające się na prawo od $1e^{31}$ znikają, dowolne cyfry na lewo od $1e^{-30}$ są zaokrąglane.

ABS() Usuwa znak ze zmiennej lub wartości pośredniej DECIMAL i daje w rezultacie wartość DECIMAL.

INT() Zaokrągla odrzucając część dziesiętną wartości pośredniej DECIMAL i daje w rezultacie wartość DECIMAL.

ROUND() Jeśli drugi parametr posiada typ bazowy LONG lub DECIMAL, zaokrąglenie jest wykonywane jako operacja BCD, która daje w rezultacie wartość DECIMAL. ROUND jest bardzo efektywne jako operacja BCD I powinno być wykorzystywane przy porównywaniu danych REAL i DECIMAL z określona precyzją.

Typy konwersji i rezultaty pośrednie

Wewnętrznie rezultaty pośrednie BCD mogą być przechowywane z dokładnością do 31 cyfr po obu stronach kropki dziesiętnej. Tak więc dwie liczby DECIMAL mogą być dodane z pełną dokładnością. Jednakże zapisywanie wartości pośrednich do zmiennych docelowych może prowadzić do utraty precyzji. Obowiązują tu następujące reguły:

Decimal(x,y) = BCD

Najpierw wartość BCD jest zaokrąglana do **y** pozycji dziesiętnych. Jeśli rezultat przekracza liczbę cyfr określona przez **x**, początkowe cyfry są usuwane (odpowiada to wyrównywaniu wokół licznika dziesiętnego).

Integer = BCD

Wszystkie cyfry na prawo od kropki dziesiętnej są ignorowane. Liczba dziesiętna jest następnie konwertowana na całkowitą z pełną dokładnością, a następnie pobrana z modulo 2^{32}.

String(@Nx.y) = BCD

Wartość BCD jest zaokrąglana do **y** pozycji dziesiętnych, rezultat jest umieszczany w łańcuchu opartym o wzorzec. Jeśli wystąpi przepełnienie, w rezultacie otrzymamy ciąg znaków (####).

Real = BCD

Pobierane jest 15 najbardziej znaczących cyfr, kropka dziesiętna jest odpowiednio dostosowywana.

Dla tych operacji i procedur, które nie posiadają obsługi typu DECIMAL, dane DECIMAL są konwertowane najpierw do typu REAL. W przypadku, gdy w wartości DECIMAL występuje więcej niż 15 cyfr, prowadzi to do utraty precyzji.

Uwaga: Parametry o nieokreślonym typie danych posiadają niejawnie typ bazowy REAL. Z tego powodu dane o typie bazowym DECIMAL przekazywane w postaci parametrów o nieokreślonym typie, zachowują tylko 15 cyfr. Dane o typie bazowym DECIMAL powinny być przekazywane jako parametry typu *DECIMAL, co pozwoli zapobiec utracie precyzji.

W momencie wyliczania wyrażenia za pomocą EVALUATE (lub przetwarzania filtra widoku) jest stosowany typ bazowy REAL.

Konwersja przy prostym przypisaniu

Poniżej zostały przedstawione zasady konwersji obowiązujące w instrukcjach prostego przypisania:

BYTE = **(SHORT, USHORT, LONG lub ULONG)**

W zmiennej docelowej jest zapisywanych 8 mniej znaczących bitów wartości źródłowej.

(DECIMAL, PDECIMAL, REAL, SREAL, BFLOAT8 lub BFLOAT4)

Wartość źródłowa jest najpierw konwertowana do typu LONG, co powoduje obcięcie części dziesiętnej, następnie zmienna docelowa otrzymuje wartość odpowiadającą 8 mniej znaczącym bitom wartości LONG.

(STRING, CSTRING lub PSTRING)

Wartość źródłowa musi być wartością numeryczną bez znaków formatujących. Wartość źródłowa jest konwertowana do typu LONG, co powoduje obcięcie części dziesiętnej, następnie zmienna docelowa otrzymuje wartość odpowiadającą 8 mniej znaczącym bitom wartości LONG.

SHORT = **BYTE**

Zmienna docelowa otrzymuje wartość wartości źródłowej.

(USHORT, LONG lub ULONG)

Zmienna docelowa otrzymuje wartość odpowiadającą 16 mniej znaczącym bitom wartości źródłowej.

(DECIMAL, PDECIMAL, REAL, SREAL, BFLOAT8 lub BFLOAT4)

Wartość źródłowa jest konwertowana do typu LONG, co powoduje obcięcie części dziesiętnej, następnie zmienna docelowa otrzymuje wartość odpowiadającą 16 mniej znaczącym bitom wartości LONG.

(STRING, CSTRING lub PSTRING)

Wartość źródłowa musi być wartością numeryczną bez znaków formatujących. Wartość źródłowa jest konwertowana do typu LONG, co powoduje obcięcie części dziesiętnej, następnie zmienna docelowa otrzymuje wartość odpowiadającą 16 mniej znaczącym bitom wartości LONG.

USHORT = **BYTE**

Zmienna docelowa otrzymuje wartość wartości źródłowej.

(SHORT, LONG lub ULONG)

Zmienna docelowa otrzymuje wartość odpowiadającą 16 mniej znaczącym bitom wartości źródłowej.

(DECIMAL, PDECIMAL, REAL, SREAL, BFLOAT8 lub BFLOAT4)

Wartość źródłowa jest konwertowana do typu LONG, co powoduje obcięcie części dziesiętnej, następnie zmienna docelowa otrzymuje wartość odpowiadającą 16 mniej znaczącym bitom wartości LONG.

(STRING, CSTRING lub PSTRING)

Wartość źródłowa musi być wartością numeryczną, bez osadzonych znaków formatujących. Wartość źródłowa jest konwertowana do typu LONG, co powoduje obcięcie części dziesiętnej, następnie zmienna docelowa otrzymuje wartość odpowiadającą 16 mniej znaczącym bitom wartości LONG.

LONG = **(BYTE, SHORT, USHORT lub ULONG)**

Zmienna docelowa otrzymuje wartość i znak wartości źródłowej.

(DECIMAL, PDECIMAL, REAL, SREAL, BFLOAT8 lub BFLOAT4)

Zmienna docelowa otrzymuje wartość wartości źródłowej, włączając w to znak, do liczby 2^{31}. Jeśli liczba jest większa od 2^{31}, zmienna docelowa otrzymuje wartość modulo 2^{31}. Część dziesiętna jest obcinana.

(STRING, CSTRING lub PSTRING)

Wartość źródłowa musi być wartością numeryczną, bez osadzonych znaków formatujących. Wartość źródłowa jest konwertowana do typu REAL, następnie do typu LONG.

DATE = **(BYTE, SHORT, USHORT lub ULONG)**

Zmienna docelowa otrzymuje datę formatu Btrieve z wartości źródłowej zawierającej datę zapisaną w standardowym formacie daty Clariona.

(DECIMAL, PDECIMAL, REAL, SREAL, BFLOAT8 lub BFLOAT4)

Wartość źródłowa jest konwertowana do typu LONG, jako standardowa data Clariona, co powoduje obcięcie części dziesiętnej, następnie zmienna docelowa otrzymuje datę w formacie Btrieve.

(STRING, CSTRING lub PSTRING)

Wartość źródłowa musi być wartością numeryczną, bez osadzonych znaków formatujących. Wartość źródłowa jest konwertowana do typu LONG, jako standardowa data Clariona, co powoduje obcięcie części dziesiętnej, następnie zmienna docelowa otrzymuje datę w formacie Btrieve.

TIME = **(BYTE, SHORT, USHORT lub ULONG)**

Zmienna docelowa otrzymuje czas formatu Btrieve z wartości źródłowej zawierającej czas zapisany w standardowym formacie czasu Clariona.

(DECIMAL, PDECIMAL, REAL, SREAL, BFLOAT8 lub BFLOAT4)

Wartość źródłowa jest konwertowana do typu LONG, jako standardowy czas Clariona, co powoduje obcięcie części dziesiętnej, następnie zmienna docelowa otrzymuje czas w formacie Btrieve.

(STRING, CSTRING, PSTRING)

Wartość źródłowa musi być wartością numeryczną, bez osadzonych znaków formatujących. Wartość źródłowa jest konwertowana do typu LONG, jako standardowy czas Clariona, co powoduje obcięcie części dziesiętnej, następnie zmienna docelowa otrzymuje czas w formacie Btrieve.

ULONG = **(BYTE, SHORT lub USHORT)**

Wartość źródłowa jest konwertowana do typu LONG, następnie zmienna docelowa otrzymuje wszystkie 32 bity wartości LONG.

LONG Zmienna docelowa otrzymuje wszystkie 32 bity wartości źródłowej.
(DECIMAL, PDECIMAL, REAL, SREAL, BFLOAT8 lub BFLOAT4)

Wartość źródłowa jest konwertowana do typu LONG, co powoduje obcięcie części dziesiętnej, następnie zmienna docelowa otrzymuje wszystkie 32 bity wartości LONG.

(STRING, CSTRING lub PSTRING)

Wartość źródłowa musi być wartością numeryczną, bez osadzonych znaków formatujących. Wartość źródłowa jest konwertowana do typu LONG, co powoduje obcięcie części dziesiętnej, następnie zmienna docelowa otrzymuje wszystkie 32 bity wartości LONG.

REAL = **(BYTE, SHORT, USHORT, LONG lub ULONG)**

Zmienna docelowa otrzymuje pełną część całkowitą i znak wartości źródłowej.

(DECIMAL, PDECIMAL, SREAL, BFLOAT8 lub BFLOAT4)

Zmienna docelowa otrzymuje znak, część całkowitą i część dziesiętną wartości źródłowej.

(STRING, CSTRING, PSTRING)

Wartość źródłowa musi być wartością numeryczną, bez osadzonych znaków formatujących. Zmienna docelowa otrzymuje znak, część całkowitą i część dziesiętną wartości źródłowej. Końcowe spacje są ignorowane.

SREAL = **(BYTE, SHORT, USHORT, LONG lub ULONG)**

Zmienna docelowa otrzymuje wartość i znak wartości źródłowej.

(DECIMAL, PDECIMAL lub REAL)

Zmienna docelowa otrzymuje znak, część całkowitą i część ułamkową wartości źródłowej.

(STRING, CSTRING lub PSTRING)

Wartość źródłowa musi być wartością numeryczną, bez osadzonych znaków formatujących. Zmienna docelowa otrzymuje znak, część całkowitą i część dziesiętną liczby. Końcowe spacje są ignorowane.

BFLOAT8 = **(BYTE, SHORT, USHORT, LONG lub ULONG)**

Zmienna docelowa otrzymuje wartość i znak wartości źródłowej.

(DECIMAL, PDECIMAL lub REAL)

Zmienna docelowa otrzymuje znak, część całkowitą i część ułamkową wartości źródłowej.

(STRING, CSTRING lub PSTRING)

Wartość źródłowa musi być wartością numeryczną, bez osadzonych znaków formatujących. Zmienna docelowa otrzymuje znak, część całkowitą i część dziesiętną liczby. Końcowe spacje są ignorowane.

BFLOAT4 = **(BYTE, SHORT, USHORT, LONG lub ULONG)**

Zmienna docelowa otrzymuje wartość i znak wartości źródłowej.

(DECIMAL, PDECIMAL lub REAL)

Zmienna docelowa otrzymuje znak, część całkowitą i część ułamkową wartości źródłowej.

(STRING, CSTRING lub PSTRING)

Wartość źródłowa musi być wartością numeryczną, bez osadzonych znaków formatujących. Zmienna docelowa otrzymuje znak, część całkowitą i część dziesiętną liczby. Końcowe spacje są ignorowane.

DECIMAL = **(BYTE, SHORT, USHORT, LONG, ULONG lub PDECIMAL)**

Zmienna docelowa otrzymuje znak i wartość wartości źródłowej, z odpowiednim zaokrągleniem i wyrównaniem.

(REAL lub SREAL)

Zmienna docelowa otrzymuje znak, część całkowitą i bardziej znaczącą część ułamka wartości źródłowej. Bardziej znaczącą część ułamka jest zaokrąglana w zmiennej docelowej.

(STRING, CSTRING, PSTRING)

Wartość źródłowa musi być wartością numeryczną, bez osadzonych znaków formatujących. Zmienna docelowa otrzymuje znak, część całkowitą i część dziesiętną liczby. Końcowe spacje są ignorowane.

PDECIMAL = **(BYTE, SHORT, USHORT, LONG, ULONG lub DECIMAL)**

Zmienna docelowa otrzymuje wartość i znak wartości źródłowej, z odpowiednim zaokrągleniem i wyrównaniem.

(REAL, SREAL, BFLOAT8 lub BFLOAT4)

Zmienna docelowa otrzymuje znak, część całkowitą i bardziej znaczącą część ułamka wartości źródłowej. Bardziej znaczącą część ułamka jest zaokrąglana w zmiennej docelowej.

(STRING, CSTRING lub PSTRING)

Wartość źródłowa musi być wartością numeryczną, bez osadzonych znaków formatujących. Zmienna docelowa otrzymuje znak, część całkowitą i część dziesiętną liczby. Końcowe spacje są ignorowane.

STRING = **(BYTE, SHORT, USHORT, LONG lub ULONG)**

Zmienna docelowa otrzymuje znak i niesformatowaną liczbę wartości źródłowej. Wartość jest wyrównywana do lewej.

(DECIMAL, PDECIMAL, REAL, SREAL, BFLOAT8 lub BFLOAT4)

Zmienna docelowa otrzymuje znak, część całkowitą i część ułamkową wartości źródłowej (zaokrągloną zgodnie ze wzorcem). Wartość jest wyrównywana do lewej.

CSTRING = **(BYTE, SHORT, USHORT, LONG lub ULONG)**

Zmienna docelowa otrzymuje znak i niesformatowaną liczbę wartości źródłowej. Wartość jest wyrównywana do lewej.

(DECIMAL, PDECIMAL, REAL, SREAL, BFLOAT8 lub BFLOAT4)

Zmienna docelowa otrzymuje znak, część całkowitą i część ułamkową wartości źródłowej (zaokrągloną zgodnie ze wzorcem). Wartość jest wyrównywana do lewej.

PSTRING = **(BYTE, SHORT, USHORT, LONG lub ULONG)**

Zmienna docelowa otrzymuje znak i niesformatowaną liczbę wartości źródłowej. Wartość jest wyrównywana do lewej.

(DECIMAL, PDECIMAL, REAL, SREAL, BFLOAT8 lub BFLOAT4)

Zmienna docelowa otrzymuje znak, część całkowitą i część ułamkową wartości źródłowej (zaokrągloną zgodnie ze wzorcem). Wartość jest wyrównywana do lewej.

12 – STEROWANIE KODEM

Struktury sterujące

ACCEPT (procesor zdarzeń)

```
ACCEPT
   statements
END
```

ACCEPT Pętla obsługi zdarzeń.

statements Instrukcje kodu wykonywalnego.

Pętla **ACCEPT** jest pętlą obsługi zdarzeń przetwarzającą zdarzenia generowane przez system Windows dla okien aplikacji APPLICATION i zwykłych okien WINDOW. Pętla ACCEPT i okno są ze sobą ściśle powiązane. Po otwarciu okna jest uruchamiana pętla ACCEPT przechwytująca i obsługująca wszystkie zdarzenia z nim związane.

ACCEPT operuje w sposób podobny, co pętla LOOP w powiązaniu z instrukcjami BREAK i CYCLE, które mogą być wywoływane wewnątrz niej. Pętla ACCEPT obsługuje cykl dla każdego zdarzenia, które wymaga określonej reakcji programu. ACCEPT oczekuje na przesłanie przez bibliotekę runtime Clariona zdarzenia, które powinno być przetworzone przez program, następnie obsługuje je za pomocą swoich instrukcji *statements*. W czasie, gdy ACCEPT znajduje się w stanie oczekiwania, sterowanie posiada biblioteka runtime Clariona, której zadanie polega na automatycznej obsłudze typowych zdarzeń systemowych nie wymagających reakcji programu, takich jak np. odświeżenie ekranu. Aktualna zawartość wszystkich kontrolek typu STRING reprezentujących na ekranie zawartość zmiennych (dla okna stanowiącego wątek bieżący) jest automatycznie odświeżana przy każdym cyklu pętli ACCEPT. Eliminuje to konieczność bezpośredniego używania instrukcji DISPLAY. Zawartość zmiennych reprezentowanych przez pozostałe kontrolki jest automatycznie odświeżana przy każdym zdarzeniu związanym z taką kontrolką (za wyjątkiem kontrolek posiadających włączoną właściwość PROP:Auto powodującą odświeżanie za każdym razem, gdy następuje cykl pętli ACCEPT). Wewnątrz pętli ACCEPT program identyfikuje zdarzenia za pomocą następujących funkcji:

EVENT() Daje w rezultacie wartość identyfikującą zdarzenie, które zaszło. Stałe reprezentujące poszczególne zdarzenia zostały zdefiniowane w pliku EQUATES.CLW.

FIELD() Daje w rezultacie numer pola, którego kontrolki dotyczy zdarzenie; o ile jest to zdarzenie związane z kontrolką.

ACCEPTED() Daje w rezultacie numer pola, którego kontrolka została zaakceptowana (zdarzenie EVENT:Accepted).

SELECTED() Daje w rezultacie numer pola, którego kontrolka została wyselekcjonowana (zdarzenie EVENT:Selected).

FOCUS() Daje w rezultacie numer pola, którego kontrolka jest aktywna - posiada sterowanie, niezależnie od tego, jakie zdarzenie zaszło.

MOUSEX() Daje w rezultacie współrzędną x kursora myszki.

MOUSEY() Daje w rezultacie współrzędną y kursora myszki

Dwa zdarzenia powodują natychmiastowe, bezwarunkowe przerwanie pętli ACCEPT. Są to zdarzenia sygnalizujące zamknięcie okna (EVENT:CloseWindow) lub zamknięcie aplikacji (EVENT:CloseDown). W kodzie programu nie musimy umieszczać obsługi tych zdarzeń, gdyż ich obsługa jest wykonywana automatycznie. Możemy to jednak zrobić i w ten sposób uzyskać możliwość wykonania dodatkowych operacji, na przykład potwierdzenia ze strony użytkownika, że na pewno chce zamknąć dane okno, czy aplikację. W takim przypadku stosujemy dodatkowo instrukcję CYCLE, która powoduje rozpoczęcie wykonania pętli ACCEPT od początku. W podobny sposób możemy zapewnić sobie obsługę innych zdarzeń: EVENT:Move, EVENT:Size, EVENT:Restore, EVENT:Maximize oraz EVENT:Iconize. Instrukcja CYCLE pozwala na przerwanie wykonywania domyślnej ich obsługi i rozpoczęcie pętli ACCEPT od nowa.

Przykład:

```
CODE
  OPEN(Window)
  ACCEPT                          ! Pętla obsługi zdarzeń
    CASE FIELD()
    OF 0                          ! Obsługa zdarzeń niezlaeżnych od pól
      CASE EVENT()
      OF EVENT:Move
        CYCLE                     ! Nie zezwalaj użytkownikowi na przemieszczanie okna
      OF EVENT:Suspend
        CASE FOCUS()
        OF ?Field1
          ! Zapisanie
        END
      OF EVENT:Resume
        ! Odtworzenie
      END
    OF ?Field1 !Handle events for Field1
      CASE EVENT()
      OF EVENT:Selected
        ! kod pre-edycyjny dla pola field1
      OF EVENT:Accepted
        ! kod kompletujący dla pola field1
      END
    END
```

Porównaj: EVENT, APPLICATION, WINDOW, FIELD, FOCUS, ACCEPTED, SELECTED, CYCLE, BREAK

CASE (struktura wykonania warunkowego)

```
CASE condition
OF expression [ TO expression ]
   statements
[ OROF expression [ TO expression ] ]
   statements
[ ELSE ]
   statements
END
```

CASE Inicjuje strukturę wykonania warunkowego.

condition Zmienna numeryczna lub łańcuchowa bądź wyrażenie.

OF Instrukcje *statements* następujące po **OF** są wykonywane gdy wyrażenie *expression* umieszczone po OF jest równe warunkowi *condition* instrukcji CASE. W strukturze CASE może występować wiele opcji OF.

expression Stała lub zmienna łańcuchowa, numeryczna bądź wyrażenie.

TO **TO** umożliwia określenie zakresu wartości dla OF lub OROF. Instrukcje *statements* występujące po OF (lub OROF) są wykonywane wtedy, gdy wartość warunku *condition* mieści się w określonym przez *expressions* zakresie. Wyrażenie *expression* występujące po OF (lub OROF) musi zawierać ograniczenie dolne. Wyrażenie *expression* występujące po TO musi zawierać ograniczenie górne.

OROF Instrukcje *statements* występujące po **OROF** są wykonywane, gdy wyrażenie *expression* umieszczone przy opcji OROF lub OF jest równe warunkowi *condition* struktury CASE. Może występować wiele opcji OROF związanych z opcją OF. Opcja OROF opcjonalnie może zostać umieszczona w oddzielnej linii.

ELSE Instrukcje *statements* występujące po **ELSE** są wykonywane, gdy wszystkie poprzedzające je opcje OF i OROF zostały wyliczone jako nie spełniające warunku struktury CASE. Opcja ELSE nie jest wymagana, jednak jeśli występuje, musi być ostatnią opcją struktury CASE.

statements Kod źródłowy języka Clarion.

Struktura **CASE** selektywnie wykonuje pierwszy zestaw instrukcji *statements*, którego wyrażenie *expression* lub zakres wyrażeń *expressions* spełnia warunek *condition* struktury CASE.

Struktury CASE mogą być zagnieżdżane w innych strukturach wykonywalnych; zachodzi też sytuacja odwrotna. Struktura CASE musi być zakończona instrukcją END (lub znakiem kropki).

W sytuacjach, gdy logika programu dopuszcza zastosowanie struktury CASE albo złożonych struktur IF/ELSIF, ta pierwsza zapewnia uzyskanie dużo bardziej

efektywnego kodu wykonywalnego. Instrukcja EXECUTE generuje najbardziej efektywny kod dla tych szczególnych przypadków, gdy warunki są wyliczane jako liczby całkowite z zakresu od 1 do n.

Przykład:

```
CASE ACCEPTED()                          ! Podprogram ewaluacji edycji pola
OF ?Name                                 ! Jeśli jest to pole Name
  ERASE(?Address,?Zip)                   ! wykasuj od Address do Zip
  GET(NameFile,NameKey)                  ! pobierz rekord
  CASE Action                            ! Określ akcję
  OF 1                                   ! dodanie rekordu – który nie istnieje
    IF NOT ERRORCODE()                   ! powinien być błąd pliku
      ErrMsg = 'ALREADY ON FILE'         ! w przeciwnym wypadku wyświetl komunikat
      DISPLAY(?Address,?Zip)             ! wyświetlaj o dadresu do kodu pocztowego
      SELECT(?Name)                      ! wprowadź ponownie Name
    END
  OF 2 OROF 3                            ! zmiana lub usunięcie – rekord istnieje
    DISPLAY(?Address,?Zip)               ! wyświetlaj o dadresu do kodu pocztowego
  END                                    ! koniec akcji CASE
  CASE Name[1]                           ! Pobierz pierwszą literę Name
  OF 'A' TO 'M'                          ! Przetwarzaj pierwszą połówkę alfabetu
  OROF 'a' TO 'm'
    DO FirstHalf
  OF 'N' TO 'Z' OROF 'n' TO 'z'          ! Przetwarzaj drugą połówkę alfabetu
    DO SecondHalf
  END                                    ! Koniec
OF ?Address                              ! Jeśli pole jest adresem
  DO AddressVal                          ! wywołaj podprogram walidacji
END                                      ! Koniec accepted()
```

Porównaj: EXECUTE, IF

EXECUTE (struktura wykonania instrukcji)

```
EXECUTE expression
   statement 1
   statement 2
   [ BEGIN
      statements
   END ]
      statement n
   [ ELSE ]
      statement
   END
```

EXECUTE Inicjuje strukturę wykonania pojedynczej instrukcji.

expression Wyrażenie numeryczne lub zmienna zawierająca liczbę całkowitą.

statement 1 Pojedyncza instrukcja wykonywana tylko wtedy, gdy wyrażenie *expression* jest równe 1.

statement 2 Pojedyncza instrukcja wykonywana tylko wtedy, gdy wyrażenie *expression* jest równe 2.

BEGIN **BEGIN** oznacza początek struktury zawierającej pewną liczbę linii kodu. Struktura BEGIN jest traktowana przez strukturę EXECUTE jak pojedyncza instrukcja. Struktura BEGIN jest kończona znakiem kropki lub słowem kluczowym END.

statement n Pojedyncza instrukcja wykonywana tylko wtedy, gdy wyrażenie *expression* jest równe *n*.

ELSE Instrukcja *statement* występująca po **ELSE** jest wykonywana wtedy, gdy wyrażenie *expression* zostanie wyliczone jako leżące poza zakresem od 1 do *n*, gdzie *n* jest zdefiniowane jako całkowita liczba pojedynczych instrukcji umieszczonych pomiędzy EXECUTE a ELSE.

statement Pojedyncza instrukcja wykonywana tylko wtedy, gdy wyrażenie *expression* znajduje się poza dopuszczalnym zakresem wartości.

Struktura **EXECUTE** wybiera pojedynczą wykonywalną instrukcję (lub wykonywalną strukturę kodu) bazując na wartości wyrażenia *expression*. Struktura EXECUTE musi być zakończona instrukcją END lub znakiem kropki

Jeśli wyrażenie *expression* jest równe 1, wykonywana jest pierwsza instrukcja - *statement 1*. Jeżeli wyrażenie *expression* jest równe 2, wykonywana jest druga instrukcja - *statement*, itd. W przypadku, gdy wartość wyrażenia *expression* jest równa zero lub jest większa, niż całkowita liczba instrukcji (lub struktur traktowanych jako pojedyncze instrukcje), wykonywana jest instrukcja *statement* ujęta w klauzuli ELSE. Jeśli ELSE nie występuje, wykonywanie programu jest kontynuowane od pierwszej instrukcji występującej po strukturze EXECUTE.

Struktury EXECUTE mogą być zagnieżdżane w innych strukturach wykonywalnych (i odwrotnie). Należą do nich IF, CASE, LOOP, EXECUTE, czy BEGIN.

W sytuacjach, gdy logika programu dopuszcza zastosowanie jednej ze struktur: EXECUTE, CASE lub IF/ELSE, ta pierwsza generuje najbardziej efektywny kod i powinna być preferowana.

Przykład:

```
EXECUTE Transact                        ! Ewaluacja Transact
  ADD(Customer)                         ! Wykonaj jeśli Transact = 1
  PUT(Customer)                         ! Wykonaj jeśli Transact = 2
  DELETE(Customer)                      ! Wykonaj jeśli Transact = 3
END                                     ! Koniec execute
EXECUTE CHOICE()                        ! Ewaluacja procedury CHOICE()
  OrderPart                             ! Wykonaj jeśli CHOICE() = 1
  BEGIN                                 ! Wykonaj jeśli CHOICE() = 2
    SavVendor" = Vendor
    UpdVendor
    IF Vendor <> SavVendor"
      Mem:Message = 'VENDOR NAME CHANGED'
  . .
  CASE VendorType                       ! Wykonaj jeśli CHOICE() = 3
  OF 1
    UpdPartNo1
  OF 2
    UpdPartNo2
  END
  RETURN                                ! Wykonaj jeśli CHOICE() = 4
END                                     ! Koniec execute
EXECUTE SomeValue
  DO OneRoutine
  DO TwoRoutine
ELSE
  MESSAGE('SomeValue did not contain a 1 or 2')
END
```

Porównaj: BEGIN, CASE, IF

IF (struktura wykonania warunkowego)

```
IF logical expression [ THEN ]
   statements
[ ELSIF logical expression [ THEN ]
   statements ]
[ ELSE
   statements ]
END
```

IF	Inicjuje strukturę warunkowego wykonania instrukcji.
logical expression	Zmienna, procedura lub wyrażenie obliczające warunek. Sterowanie jest uzależnione od rezultatu (prawda lub fałsz) wyrażenia. Wartość zerowa (lub łańcuch pusty) jest traktowana jako fałsz, pozostałe wartości są traktowane jako prawda.
THEN	Instrukcje *statements* występujące po THEN są wykonywane wtedy, gdy poprzedzające je wyrażenie logiczne *logical expression* daje w rezultacie wartość „prawda”. Jeśli THEN jest używane, musi być umieszczane w tej samej linii co **IF** lub **ELSIF**.
statements	Instrukcja wykonywalna lub sekwencja takich instrukcji.
ELSIF	Wyrażenie logiczne *logical expression* występujące po **ELSIF** jest wyliczane tylko wtedy, gdy wszystkie poprzednie warunki **IF** lub **ELSIF** zostały wyliczone jako „fałsz”.
ELSE	Instrukcje *statements* występujące po **ELSE** są wykonywane tylko wtedy, gdy wszystkie poprzednie warunki **IF** lub **ELSIF** zostały wyliczone jako „fałsz”. Instrukcja ELSE nie jest wymagana, jeśli występuje musi być ostatnią opcją w strukturze IF.

Struktura **IF** steruje wykonywaniem programu bazując na jednym (lub więcej) wyrażeniu logicznym *logical expressions*. Struktury IF mogą posiadać dowolną liczbę grup ELSIF. Struktury IF mogą być zagnieżdżane w innych strukturach wykonywalnych i odwrotnie. Każda struktura IF musi zostać zakończona słowem kluczowym END lub znakiem kropki.

Przykład:

```
IF Cus:TransCount = 1                                    ! Jeśli nowy kient
  AcctSetup                                              ! wywołaj procedurę konfiguracji konta
ELSIF Cus:TransCount > 10 AND Cus:TransCount < 100       ! Jeśli zwykły klient
  DO RegularAcct                                         ! przetwarzaj konto
ELSIF Cus:TransCount > 100                               ! Jeśli specjalny klient
  DO SpecialAcct                                         ! przetwarzaj konto
ELSE                                                     ! w przeciwnym wypadku
  DO NewAcct                                             ! przetwarzaj konto
  IF Cus:Credit THEN CheckCredit ELSE CLEAR(Cus:CreditStat).
                                                         ! zweryfikuj stan kredytu
END
IF ERRORCODE() THEN ErrHandler(Cus:AcctNumber,Trn:InvoiceNbr).      ! Obsługuj błędy
```

Porównaj: EXECUTE, CASE

LOOP (pętla iteracyjna)

```
label   LOOP [  | count TIMES                    | ]
                | i = initial TO limit [ BY step ] |
                | UNTIL logical expression       |
                | WHILE logical expression       |
            statements
        | END                                    |
        | UNTIL logical expression               |
        | WHILE logical expression               |
```

LOOP Inicjuje iteracyjną strukturę wykonywania instrukcji.

count Stała lub zmienna całkowita, wyrażenie określające liczbę powtórzeń (TIMES) instrukcji *statements* w pętli LOOP.

TIMES Określa liczbę powtórzeń *count* instrukcji *statements*.

i Etykieta zmiennej, która automatycznie zwiększa (lub zmniejsza, gdy *step* jest ujemne) swoją wartość w każdej iteracji.

= *initial* Stała lub zmienna numeryczna, wyrażenie określające początkową wartość zmiennej iteracyjnej (*i*); przy pierwszym przejściu pętli LOOP.

TO *limit* Stała lub zmienna numeryczna, wyrażenie określające końcową wartość zmiennej iteracyjnej (*i*) dla pętli LOOP. Gdy *i* jest większe od *limit (*lub mniejsze, w przypadku, gdy *step* jest wartością ujemną) pętla LOOP kończy swoje działanie. Zmienna *i* zawiera wówczas ostatnią wartość większą (lub mniejszą) od *limit*.

BY *step* Stała lub zmienna numeryczna, wyrażenie określające wartość o jaką jest zwiększana (zmniejszana) wartość zmiennej *i* w każdej iteracji pętli LOOP. Jeśli BY *step* zostanie pominięte, przyjmuje się, że zmienna *i* zmienia swą wartość o 1.

UNTIL Gdy jest umieszczone w pętli LOOP, wylicza wyrażenie logiczne *logical expression* przed każdą iteracją. Gdy kończy strukturę LOOP, UNTIL wylicza *logical expression* po każdej iteracji. Jeśli wyrażenie logiczne *logical expression* ma wartość „prawda", pętla LOOP kończy swe działanie.

WHILE Gdy jest umieszczone w pętli LOOP, wylicza wyrażenie logiczne *logical expression* przed każdą iteracją. Gdy kończy strukturę LOOP, WHILE wylicza *logical expression* po każdej iteracji. Jeśli wyrażenie logiczne *logical expression* ma wartość „fałsz", pętla LOOP kończy swe działanie.

logical expression Zmienna numeryczna lub łańcuchowa, wyrażenie, bądź procedura. Wyrażenie logiczne *logical expression* wylicza warunek. Sterowanie zależy od rezultatu (prawda lub fałsz) wyrażenia. Wartość zerowa lub łańcuch pusty są traktowane jako „fałsz", pozostałe wartości – jako „prawda".

statements Instrukcja wykonywalna lub sekwencja takich instrukcji

Struktura **LOOP** powtarza wykonywanie instrukcji *statements* w niej umieszczonych. Struktury LOOP mogą być zagnieżdżane w innych strukturach wykonywalnych i odwrotnie. Każda struktura LOOP musi zostać zakończona słowem kluczowym END lub znakiem kropki, bądź instrukcją UNTIL lub WHILE.

Pętla LOOP bez określonego na jej początku warunku wykonuje się nieprzerwanie dopóty, dopóki nie napotka instrukcji BREAK lub RETURN. Instrukcja BREAK przerywa pętlę LOOP i przekazuje sterowanie do pierwszej instrukcji występującej w kodzie po strukturze LOOP. Instrukcja CYCLE bezzwłocznie przekazuje sterowanie na początek pętli LOOP i zaczyna się kolejna jej iteracja; instrukcje występujące po CYCLE nie są wykonywane.

Wyrażenia logiczne *logical expression* pętli LOOP UNTIL oraz LOOP WHILE są zawsze wyliczona na początku pętli LOOP, zanim zostaną wykonane jej instrukcje *statements*. Jeśli wyrażenie *logical expression* da w rezultacie fałsz w pierwszej iteracji, instrukcje *statements* pętli LOOP nie wykonają się ani razu. Jeśli chcemy uzyskać pętlę LOOP, która zawsze wykona swoje instrukcje, musimy umieszczać UNTIL oraz WHILE na końcu pętli LOOP; w roli jej terminatorów.

Przykład:

```
LOOP                              ! Nieskończona pętla
  Char = GetChar()                ! pobierz znak
  IF Char <> CarrReturn           ! jeśli to nie jest powrót karetki
    Field = CLIP(Field) & Char    ! dodaj znak
  ELSE                            ! w przeciwnym wypadku
    BREAK                         ! przerwij pętlę
. .                               ! koniec IF, koniec LOOP

IF ERRORCODE()                    ! w przypadku błędu
  LOOP 3 TIMES                    ! powtarzaj trzy razy
    BEEP                          ! alarmuj dźwiękiem
. .                               ! koniec IF, koniec LOOP

LOOP I# = 1 TO 365 BY 7           ! Powtarzaj, zwiększając I# o 7 za każdym razem
  GET(DailyTotal,I#)              ! czytaj co 7 rekord
  DO WeeklyJob
END                               ! I# zawiera 372, gdy LOOP się kończy

LOOP I# = 10 TO 1 BY -1           ! Powtarzaj, zmniejszając I# o 1 za każdym razem
  DO SomeRoutine
END                               ! I# zawiera (0), gdy LOOP się kończy

SET(MasterFile)                   ! Ustaw na pierwszy rekord
LOOP UNTIL EOF(MasterFile)        ! Przetwarzaj wszystkie rekordy
  NEXT(MasterFile)                ! odczytaj rekord
  ProcMaster                      ! wywołaj procedurę
END

LOOP WHILE KEYBOARD()             ! Wyczyść bufor klawiatury
  ASK                             ! bez przetwarzania klawiszy
UNTIL KEYCODE() = EscKey          ! ale przerywając pętlę, gdy zostanie wciśnięty Escape
```

Porównaj: BREAK, CYCLE

Instrukcje przekazywania sterowania

BREAK (przerywa wykonywanie pętli)

BREAK [*label*]

BREAK Przekazuje sterowanie do pierwszej instrukcji występującej po instrukcji kończącej pętlę LOOP lub ACCEPT.

label Etykieta instrukcji LOOP lub ACCEPT. Musi to być etykieta zagnieżdżonej pętli zawierającej instrukcję BREAK.

Instrukcja **BREAK** bezzwłocznie przerywa wykonywanie pętli LOOP lub ACCEPT i przekazuje sterowanie do pierwszej instrukcji występującej po instrukcji kończącej pętlę LOOP (END, WHILE lub UNTIL) bądź pętlę ACCEPT (END).

Instrukcja BREAK może być stosowana tylko w pętlach LOOP lub ACCEPT. Zastosowanie opcjonalnej etykiety *label* pozwala na przerwanie wielopoziomowo zagnieżdżonych pętli eliminując konieczność zastosowania GOTO.

Przykład:

```
          LOOP                          ! pętla
            ASK                         ! czekaj na klawisz
            IF KEYCODE() = EscKey       ! jeśli wciśnięto Esc
              BREAK                     ! przerwij pętlę
            ELSE                        ! w przeciwnym wypadku
              BEEP                      ! sygnalizuj dźwiękiem
            END
          END

Loop1     LOOP                          ! Loop1 jest etykietą
            DO ParentProcess
Loop2       LOOP                        ! Loop2 jest etykietą
              DO ChildProcess
              IF SomeCondition
                BREAK Loop1             ! przerwanie obu zagnieżdżonych pętli
              END
            END
          END

ACCEPT                                  ! pętla ACCEPT
  CASE ACCEPTED()
  OF ?Ok
    CallSomeProc
  OF ?Cancel
    BREAK                               ! przerwanie pętli
  END
END
```

Porównaj: LOOP, CYCLE, ACCEPT

CYCLE (skok na początek pętli)

CYCLE [*label*]

CYCLE Przekazuje sterowanie z powrotem na początek pętli LOOP lub ACCEPT.

label Etykieta instrukcji LOOP lub ACCEPT, do której ma zostać zwrócone sterowanie. Musi to być etykieta zagnieżdżonej pętli zawierającej instrukcję CYCLE.

Instrukcja **CYCLE** przekazuje bezzwłocznie sterowanie na początek pętli LOOP lub ACCEPT. Instrukcja CYCLE może być stosowana tylko w pętlach LOOP lub ACCEPT. Zastosowanie opcjonalnej etykiety *label* umożliwia przechodzenie na początek pętli znajdujących się na wyższych poziomach zagnieżdżenia (w ramach tych pętli występuje aktualna pętla); eliminując w ten sposób korzystania z nie zalecanej instrukcji GOTO.

W pętli ACCEPT, dla większości obsługiwanych zdarzeń, CYCLE przerywa domyślną akcję, zanim zostanie ona wykonana (na przykład EVENT:Move).

Przykład:

```
        SET(MasterFile)                     ! ustawienie na pierwszy rekord
        LOOP                                ! przetwarzanie wszystkich rekorów
          NEXT(MasterFile)                  ! odczytanie rekordu
          IF ERRORCODE() THEN BREAK.        ! przerwanie pętli, jeśli koniec pliku
          DO MatchMaster                    ! porównanie
          IF NoMatch                        ! jeśli nie znaleziono
            CYCLE                           ! skok na początek pętli
          END
          DO TransVal                       ! walidacja
          PUT(MasterFile)                   ! zapis rekordu
        END

Loop1   LOOP                                ! Loop1 jest etykietą
          DO ParentProcess
Loop2     LOOP                              ! Loop2 jest etykietą
            DO ChildProcess
            IF SomeCondition
              CYCLE Loop1                   ! skok na początek pętli Loop1
            END
          END
        END
```

Porównaj: LOOP, BREAK, ACCEPT

DO (wywołanie podprogramu ROUTINE)

DO *label*

DO Wykonanie podprogramu ROUTINE.

label Etykieta instrukcji ROUTINE.

Instrukcja **DO** jest stosowana do wykonania podprogramu ROUTINE lokalnego dla programu lub procedury, w której została użyta. Gdy podprogram ROUTINE kończy swe działanie, sterowanie jest przekazywane do instrukcji występującej bezpośrednio za instrukcją DO. Podprogram ROUTINE może być wywoływany tylko wewnątrz sekcji CODE zawierającej kod źródłowy podprogramu.

Przykład:

```
DO NextRecord          !Call the next record routine
DO CalcNetPay          !Call the calc net pay routine
```

Porównaj: EXIT, ROUTINE

EXIT (wyjście z podprogramu ROUTINE)

EXIT

Instrukcja **EXIT** powoduje niezwłoczne opuszczenie podprogramu ROUTINE i przekazanie sterowania do instrukcji występującej bezpośrednio po instrukcji DO, za pomocą której podprogram został wywołany. Jest to działanie różne od RETURN, gdyż ta ostatnia instrukcja powoduje wyjście z procedury lub z programu, również wtedy, gdy została wykonana w podprogramie ROUTINE.

Instrukcja EXIT nie jest wymagana. Podprogram nie posiadający instrukcji EXIT kończy się automatycznie w momencie, gdy wszystkie jego instrukcje zostaną wykonane.

Przykład:

```
CalcNetPay   ROUTINE
  IF GrossPay = 0                  ! Jeśli brak
    EXIT                           ! wyjdź z podprogramu
  END
  NetPay = GrossPay - FedTax - Fica
  QtdNetPay += NetPay
  YtdNetPay += NetPay
```

Porównaj: DO, RETURN

GOTO (skok do etykiety)

GOTO *target*

GOTO Bezwarunkowo przekazuje sterowanie do instrukcji wskazanej przez etykietę *target*.

target Etykieta instrukcji wewnątrz programu, procedury lub podprogramu ROUTINE.

Instrukcja **GOTO** bezwarunkowo przekazuje sterowanie do wskazanej instrukcji. Etykieta *target* zastosowana w instrukcji GOTO nie może być etykietą procedury, ani etykietą podprogramu ROUTINE. Zasięg instrukcji GOTO jest ograniczony do aktualnie wykonywanego podprogramu ROUTINE lub procedury – nie może to być etykieta leżąca poza podprogramem bądź procedurą.

Nie zaleca się intensywnego wykorzystywania instrukcji GOTO; lepszą ich alternatywą jest użycie pętli LOOP.

Przykład:

```
ComputeIt PROCEDURE(Level)
  CODE
  IF Level = 0 THEN GOTO PassCompute.      ! pomiń wyliczenie jeśli brak Level
    Rate = Level * MarkUp                  ! wylicz Rate
    RETURN(Rate)                           ! i zwróć jako rezultat
PassCompute   RETURN(999999)               ! zwróć zmyślony numer
```

Porównaj: LOOP

RETURN (powrót do miejsca wywołania)

RETURN([*expression*])

RETURN Przerywa wykonanie programu PROGRAM lub procedury PROCEDURE.

expression Wyrażenie *expression* określa rezultat procedury przekazywany do miejsca, z którego została ona wywołana. Może to być wartość nieokreślona NULL, jeśli procedura daje w rezultacie referencję.

Instrukcja **RETURN** przerywa wykonanie programu PROGRAM lub procedury PROCEDURE i przekazuje sterowanie do miejsca, z którego została wywołana. Gdy RETURN jest wykonywana z poziomu sekcji kodu (CODE) programu PROGRAM, kończy on swoje działanie, wszystkie pliki zostają zamknięte, a sterowanie przekazane do systemu operacyjnego.

Instrukcja RETURN jest wymagana w programach i procedurach, w których prototypach określono typ zwracanego rezultatu. Jeśli RETURN nie jest użyte w procedurze lub w programie na końcu kodu wykonywalnego jest umieszczana niejawna instrukcja RETURN. Koniec kodu wykonywalnego jest zdefiniowany jako koniec pliku kodu źródłowego bądź początek następnej procedury lub podprogramu ROUTINE.

Instrukcja RETURN w procedurze (jawna lub niejawna) automatycznie zamyka wszystkie lokalne struktury APPLICATION, WINDOW, REPORT i VIEW otwarte przez tą procedurę. Nie powoduje ona automatycznego zamknięcia globalnych lub statycznych (w odniesieniu do modułu) struktur APPLICATION, WINDOW, REPORT, czy też VIEW. Zamyka ona natomiast i zwalnia pamięć przydzieloną lokalnym kolejkom QUEUE zadeklarowanym bez atrybutu STATIC.

Przykład:

```
IF Done# THEN RETURN.                ! wyjdź, jeśli zrobione
DayOfWeek   PROCEDURE(Date)          ! procedura zwracająca dzień tygodnia
RetVal          STRING(9)
  CODE
  EXECUTE Date % 7                   ! Określ dzień tygodnia dla danej daty
    RetVal = 'Monday'
    RetVal = 'Tuesday'
    RetVal = 'Wednesday'
    RetVal = 'Thursday'
    RetVal = 'Friday'
    RetVal = 'Saturday'
  ELSE
    RetVal = 'Sunday'
  END
  RETURN(RetVal)                     ! i zwróć odpowiedni łańcuch
```

Porównaj: PROCEDURE, Typy rezultatów procedur

DODATEK A - DDE, OLE I .OCX

Dynamiczna Wymiana Danych (DDE- Dynamic Data Exchange)

Przegląd DDE

Dynamiczna wymiana danych (Dynamic Data Exchange - DDE) jest potężnym narzędziem systemu Windows umożliwiającym bezpośrednie przekazywanie danych pomiędzy działającymi aplikacjami. Pozwala to użytkownikowi na pracę z danymi w ich naturalnym (natywnym) formacie (obsługiwanym przez aplikację, z której są pobierane) oraz na zapewnieniu ich aktualności w aplikacji, do której są przekazywane.

DDE bazuje na „konwersacji" (połączeniu) ustanowionej pomiędzy dwoma działającymi aplikacjami Windows. Jedna z tych aplikacji działa jako serwer DDE udostępniając swoje dane, druga jest klientem DDE odbierającym udostępniane dane. Pojedyncza aplikacja może występować zarówno w roli serwera, jak i klienta DDE, udostępniając i pobierając dane z innych aplikacji obsługujących technikę DDE. Pomiędzy serwerem i klientem DDE może być jednocześnie nawiązanych wiele „konwersacji".

By być serwerem DDE, aplikacja napisana w Clarionie musi:

- Otworzyć przynajmniej jedno okno, gdyż wszystkie serwery DDE muszą być związane z oknem (i jego pętlą ACCEPT).
- Zarejestrować się w systemie Windows jako serwer DDE korzystając z procedury DDESERVER.
- Udostępnić dane klientowi DDE za pomocą instrukcji DDEWRITE.
- Przerwać powiązanie za pomocą instrukcji DDECLOSE, gdy łącze DDE nie jest już potrzebne. Można także umożliwić to w sytuacji, gdy użytkownik zamyka serwer DDE lub okno, dla którego zostało nawiązane to połączenie.

By być klientem DDE, aplikacja napisana w Clarionie musi:

- Otworzyć przynajmniej jedno okno, gdyż wszystkie zdarzenia DDE muszą być obsługiwane przez pętlę ACCEPT okna.
- Otworzyć powiązanie z serwerem DDE i zgłosić się jako jego klient za pomocą procedury DDECLIENT.
- Zgłosić do serwera zapotrzebowanie na dane za pomocą instrukcji DDEREAD lub wywołać obsługę ze strony serwera za pomocą instrukcji DDEEXECUTE.
- Przerwać powiązanie za pomocą instrukcji DDECLOSE, gdy łącze DDE nie jest już potrzebne. Można także umożliwić automatyczne przerwanie łącza w sytuacji, gdy użytkownik zamyka klienta DDE lub okno, dla którego zostało nawiązane to połączenie.

Prototypy procedur obsługujących DDE są umieszczone w pliku DDE.CLW, który musimy dołączać do naszej aplikacji za pomocą instrukcji INCLUDE w strukturze MAP. Proces DDE powoduje wysyłanie zdarzeń niezależnych od pól, specyficznych dla DDE, do pętli ACCEPT okna, które otworzyło połączenie pomiędzy dwoma aplikacjami jako serwer bądź klient.

Zdarzenia DDE

Proces obsługi DDE koncentruje się na przetwarzaniu kilku niezależnych od pól zdarzeń specyficznych tylko dla DDE. Zdarzenia te są przesyłane do pętli ACCEPT okna, które ustanowiło powiązanie pomiędzy dwiema aplikacjami, albo jako klient, albo jako serwer DDE.

Poniższe zdarzenia są przesyłane tylko do serwera DDE:

EVENT:DDErequest	Klient prosi o przesłanie elementu danych
EVENT:DDEadvise	Klient prosi o ciągłą aktualizację elementu danych
EVENT:DDEexecute	Klient wywołuje instrukcję DDEEXECUTE
EVENT:DDEpoke	Klient odsyła dane

Zdarzenia, które są przesyłane tylko do klienta DDE przedstawiono poniżej:

EVENT:DDEdata	Serwer dostarcza zaktualizowany element danych
EVENT:DDEclosed	Serwer kończy połączenie DDE

W odpowiedzi na jedno z tych zdarzeń mogą być wywoływane określone procedury umożliwiające operowanie na danych i udostępnianie różnych informacji. Są to:

DDECHANNEL()	Zwraca uchwyt (handle) do serwera lub klienta DDE
DDEITEM()	Zwraca element lub łańcuch polecenia przekazany do serwera za pomocą instrukcji DDEREAD lub DDEEXECUTE
DDEAPP()	Zwraca nazwę aplikacji
DDETOPIC()	Zwraca nazwę tematu

Gdy program napisany w Clarionie tworzy serwer DDE, zewnętrzne programy klienckie mogą się z nim łączyć i występować o dane. Każdemu wystąpieniu o dane towarzyszy łańcuch (w specjalnym formacie akceptowanym przez serwer) identyfikujący element danych. Jeśli serwer DDE zna wartość tego elementu danych, przekazuje ją automatycznie do klienta, bez generowania dodatkowych zdarzeń. Jeśli nie, do pętli ACCEPT okna serwera jest przesyłane zdarzenie EVENT:DDErequest lub EVENT:DDEadvise

Gdy program napisany w Clarionie tworzy klienta DDE, może łączyć się z zewnętrznymi programami udostępniającymi dane. Jeśli serwer zna wartość elementu danych, przekazuje ją do klienta bez generowania dodatkowych zdarzeń. Jeśli klient ustanowi „gorące” połączenie z serwerem, do pętli ACCEPT okna klienta jest przesyłane zdarzenie EVENT:DDEdata, za każdym razem, gdy serwer wysyła nową wartość dla danego elementu danych.

Procedury DDE

DDEACKNOWLEDGE (wysyła potwierdzenie z serwera DDE)

DDEACKNOWLEDGE(*response*)

DDEACKNOWLEDGE — Wysyła potwierdzenie bieżącej instrukcji DDEPOKE lub DDEEXECUTE przesłanej do serwera DDE.

response — Stała lub zmienna całkowita bądź wyrażenie o wartości zero (0) lub jeden (1) określające potwierdzenie negatywne lub pozytywne.

Procedura **DDEACKNOWLEDGE** umożliwia programowi serwera DDE niezwłoczne potwierdzenie dla instrukcji DDEPOKE proszącej o udostępnienie danych lub instrukcji DDEEXECUTE przesyłającej polecenia. Pozwala to aplikacji klienckiej na kontynuację swego działania. Instrukcja CYCLE po zdarzeniu EVENT:DDEpoke lub EVENT:DDEexecute również sygnalizuje pozytywne potwierdzenie do klienta, DDEACKNOWLEDGE umożliwia wysłanie potwierdzenia negatywnego.

Przykład:

```
! kod aplikacji klienta zawiera:
WinOne        WINDOW,AT(0,0,160,400)
              END
SomeServer    LONG
DDEChannel    LONG
  CODE
    OPEN(WinOne)
    DDEChannel = DDECLIENT('MyServer','System')      ! otwiera kanał do aplikacji MyServer
    DDEEXECUTE(DDEChannel,'[ShowList]')              ! mówi jej, że ma coś zrobić

! kod aplikacji serwera zawiera:
WinOne        WINDOW,AT(0,0,160,400)
              END
DDEChannel    LONG
  CODE
    OPEN(WinOne)
    DDEChannel = DDESERVER('MyServer','System')      ! otwiera kanał
    ACCEPT
      CASE EVENT()
      OF EVENT:DDEExecute
        CASE DDEVALUE()                              ! sprawdza wymaganą akcję
        OF 'ShowList'
          DDEACKNOWLEDGE(1)                          ! wysyła  pozytywne potwierdzenie
          DO ShowList                                ! i podejmuje akcję
        ELSE                                         ! jeśli wymagana akcja jest nieznana
          DDEACKNOWLEDGE(0)                          ! wysyła  negatywne potwierdzenie
        END
      END
    END
```

Porównaj: DDEPOKE, DDEEXECUTE

DDEAPP (zwraca nazwę aplikacji serwera)

DDEAPP( )

Procedura **DDEAPP** daje w rezultacie łańcuch zawierający nazwę aplikacji w kanale DDE, która właśnie wysłała zdarzenie DDE. Odpowiada to funkcjonalnie pierwszemu parametrowi przekazywanemu do procedury DDESERVER lub DDECLIENT w momencie ustanowienia kanału DDE.

Typ rezultatu: STRING

Przykład:

```
ClientApp     STRING(20)
WinOne        WINDOW,AT(0,0,160,400)
                STRING(@S20),AT(5,5,90,20),USE(ClientApp)
              END
TimeServer    LONG
DateServer    LONG
FormatTime    STRING(5)
FormatDate    STRING(8)

  CODE
  OPEN(WinOne)
  TimeServer = DDESERVER('SomeApp','Time')       ! otwiera jako serwer
  DateServer = DDESERVER('SomeApp','Date')       ! otwiera jako serwer
  ACCEPT
    CASE EVENT()
    OF EVENT:DDErequest
      CASE DDECHANNEL()
      OF TimeServer
        ClientApp = DDEAPP()                     ! pobiera nazwę klienta
        DISPLAY                                  ! i wyświetla na ekranie
        FormatTime = FORMAT(CLOCK(),@T1)
        DDEWRITE(TimeServer,DDE:manual,'Time',FormatTime)
      OF DateServer
        ClientApp = DDEAPP()                     ! pobiera nazwę klienta
        DISPLAY                                  ! i wyświetla na ekranie
        FormatDate = FORMAT(TODAY(),@D1)
        DDEWRITE(DateServer,DDE:manual,'Date',FormatDate)
      END
    END
  END
```

Porównaj: DDECLIENT, DDESERVER

DDECHANNEL (zwraca numer kanału DDE)

DDECHANNEL( )

Procedura **DDECHANNEL** daje w rezultacie liczbę całkowitą LONG zawierającą numer kanału DDE, przez który aplikacja klienta bądź serwera wysłała właśnie zdarzenie DDE. Odpowiada to funkcjonalnie wartości zwracanej przez procedurę DDESERVER lub DDECLIENT w momencie ustanowienia kanału DDE.

Typ rezultatu: LONG

Przykład:

```
WinOne      WINDOW,AT(0,0,160,400)
            END
TimeServer  LONG
DateServer  LONG
FormatTime  STRING(5)
FormatDate  STRING(8)

  CODE
  OPEN(WinOne)
  TimeServer = DDESERVER('SomeApp','Time')          ! otwiera jako serwer
  DateServer = DDESERVER('SomeApp','Date')          ! otwiera jako serwer
  ACCEPT
    CASE EVENT()
    OF EVENT:DDErequest
      CASE DDECHANNEL() !Check which channel
      OF TimeServer
        FormatTime = FORMAT(CLOCK(),@T1)
        DDEWRITE(TimeServer,DDE:manual,'Time',FormatTime)
      OF DateServer
        FormatDate = FORMAT(TODAY(),@D1)
        DDEWRITE(DateServer,DDE:manual,'Date',FormatDate)
      END
    END
  END
```

Porównaj: DDECLIENT, DDESERVER

DDECLIENT (zwraca numer kanału DDE klienta)

DDECLIENT([*application*] [, *topic*])

DDECLIENT Daje w rezultacie nowy numer kanału DDE klienta.

application Stała lub zmienna łańcuchowa zawierająca nazwę aplikacji serwera, z którą się łączymy. Zazwyczaj jest to nazwa aplikacji. Jeśli pominiemy, połączenie jest nawiązywane z pierwszym dostępnym serwerem DDE.

topic Stała lub zmienna łańcuchowa zawierająca nazwę tematu specyficznego dla danej aplikacji. Jeśli pominiemy, brany jest pod uwagę pierwszy temat dostępny w aplikacji *application*.

Procedura **DDECLIENT** zwraca nowy numer kanału klienta DDE dla aplikacji *application* i tematu *topic*. Jeśli aplikacja *application* nie jest aktualnie uruchomiona, DDECLIENT zwraca wartość zero (0). Zazwyczaj, gdy kanał DDE jest otwierany jako kliencki, *application* jest nazwą aplikacji serwera. Temat *topic* jest łańcuchem albo rejestrowanym w Windows przez *application* albo reprezentującym pewne wartości informujące aplikację *application*, jakie dane ma dostarczyć. Możemy zastosować procedurę DDEQUERY do określenia, czy *applications* oraz *topics* są aktualnie zarejestrowane w Windows.

Typ rezultatu: LONG

Przykład:

```
DDEReadVal   REAL
WinOne       WINDOW,AT(0,0,160,400)
               ENTRY(@s20),USE(DDEReadVal)
             END
ExcelServer  LONG

  CODE
  OPEN(WinOne)
  ExcelServer = DDECLIENT('Excel','MySheet.XLS')
    ! otwórz jako klient arkusza Excel
  IF NOT ExcelServer                        ! jeśli serwer nie działa
    MESSAGE('Please start Excel')           ! informuje użytkownika i uruchamia go
    RETURN                                  ! następna próba
  END
  DDEREAD(ExcelServer,DDE:auto,'R5C5',DDEReadVal)
  ACCEPT
    CASE EVENT()
    OF EVENT:DDEdata                        ! gdy nadchodzą zmienione dane od Excela
      PassedData(DDEReadVal)                ! przetwarzaj je
    END
  END
```

Porównaj: DDEQUERY, DDEWRITE, DDESERVER

DDECLOSE (przerywa połączenie serwera DDE)

DDECLOSE(*channel*)

DDECLOSE Zamyka otwarty kanał DDE.

channel Etykieta zmiennej całkowitej LONG zawierającej numer kanału – wartość zwracaną przez procedurę DDESERVER lub DDECLIENT.

Procedura **DDECLOSE** pozwala klientowi DDE na zamknięcie wskazanego kanału *channel*. Kanał *channel* jest automatycznie zamykany, gdy okno, które go otworzyło, zostanie zamknięte.

Raportowane błędy: 601 Nieprawidłowy kanał DDE

602 Kanał DDE nie jest otwarty

605 Przekroczenie czasu

Przykład:

```
WinOne      WINDOW,AT(0,0,160,400)
            END
SomeServer  LONG

  CODE
  OPEN(WinOne)
  SomeServer = DDECLIENT('SomeApp','MyTopic')       ! otwiera jako klienta
  ACCEPT
  END
  DDECLOSE(SomeServer)
```

Porównaj: DDECLIENT, DDESERVER

DDEEXECUTE (wysyła polecenie do serwera DDE)

DDEEXECUTE(*channel*, *command*)

DDEEXECUTE	Wysyła łańcuch poleceń do otwartego kanału klienta DDE.
channel	Etykieta zmiennej lub stałej całkowitej LONG zawierającej numer kanału – wartość zwracaną przez procedurę DDECLIENT.
command	Stała lub zmienna łańcuchowa zawierająca specyficzne dla aplikacji polecenia do wykonania przez serwer.

Procedura **DDEEXECUTE** umożliwia klientowi DDE przekazanie *command* do serwera. Polecenie *command* musi być w formacie, który aplikacja serwera potrafi rozpoznać i wykonać. Serwer nie musi być programem napisanym w Clarionie. Zgodnie z przyjętą konwencją, łańcuch *command* jest ujmowany w nawiasy kwadratowe ([ ]).

Serwer DDE napisany w Clarionie może używać procedury DDEVALUE() do określenia, jakie polecenie *command* przesłał klient DDE. Instrukcja CYCLE użyta po EVENT:DDEexecute sygnalizuje pozytywne potwierdzenie do klienta, który przesłał polecenie *command*. DDEACKNOWLEDGE może przesłać zarówno pozytywne, jak i negatywne potwierdzenie.

Raportowane błędy:

601 Nieprawidłowy kanał DDE

602 Kanał DDE nie jest otwarty

603 DDEEXECUTE nie powiodło się

605 Przekroczenie czasu

Generowane zdarzenia: EVENT:DDEexecute Klient wysłał polecenie.

Przykład:

```
! kod aplikacji klienta zawiera:
WinOne       WINDOW,AT(0,0,160,400)
             END
SomeServer   LONG
DDEChannel   LONG

  CODE
  OPEN(WinOne)
  DDEChannel = DDECLIENT('PROGMAN','PROGMAN')
    ! otwórz kanał do Windows Program Manager
  DDEEXECUTE(DDEChannel,'[CreateGroup(Clarion Applications)]')
    ! utwórz nową grupę programów
  DDEEXECUTE(DDEChannel,'[ShowGroup(1)]')                    ! wyświetl to
  DDEEXECUTE(DDEChannel,'[AddItem(MYAPP.EXE,My Program,PROGMAN.EXE,2)]')
    ! utwórz nowy element w grupie
    ! stosując drugą ikonę z progman.exe
```

Porównaj: DDEACKNOWLEDGE, DDEVALUE

DDEITEM (zwraca element serwera)

DDEITEM( )

Procedura **DDEITEM** zwraca łańcuch zawierający nazwę elementu dla bieżącego zdarzenia DDE. Jest to element wywoływany przez DDEREAD lub element danych dostarczany przez DDEPOKE.

Typ rezultatu: STRING

Przykład:

```
WinOne      WINDOW,AT(0,0,160,400)
            END
Server      LONG
FormatTime  STRING(5)
FormatDate  STRING(8)

  CODE
  OPEN(WinOne)
  Server = DDESERVER('SomeApp','Clock')          ! otwiera jako serwer dla tematu
  ACCEPT
    CASE EVENT()
    OF EVENT:DDErequest
      CASE DDEITEM()
      OF 'Time'
        FormatTime = FORMAT(CLOCK(),@T1)
        DDEWRITE(Server,DDE:manual,'Time',FormatTime)
      OF 'Date'
        FormatDate = FORMAT(TODAY(),@D1)
        DDEWRITE(Server,DDE:manual,'Date',FormatDate)
      END
    OF EVENT:DDEadvise
      CASE DDEITEM()
      OF 'Time'
        FormatTime = FORMAT(CLOCK(),@T1)
        DDEWRITE(Server,1,'Time',FormatTime)
      OF 'Date'
        FormatDate = FORMAT(TODAY(),@D1)
        DDEWRITE(Server,60,'Date',FormatDate)
      END
    END
  END
```

Porównaj: DDEREAD, DDEEXECUTE, DDEPOKE

DDEPOKE (przesyła nieproszone dane do serwera DDE)

DDEPOKE(*channel*, *item*, *value*)

DDEPOKE	Wysyła nieproszone dane poprzez otwarty kanał klienta DDE do serwera DDE.
channel	Etykieta zmiennej całkowitej LONG zawierającej numer kanału klienta – wartość zwracaną przez procedurę DDECLIENT.
item	Stała lub zmienna łańcuchowa zawierający specyficzny dla aplikacji element przeznaczony do odbierania nieoczekiwanych danych.
value	Stała lub zmienna łańcuchowa zawierająca dane do umieszczenia w elemencie *item.*

Procedura **DDEPOKE** umożliwia klientowi DDE komunikowanie nieproszonych danych do serwera. Temat *item* i wartość *value* muszą być w formacie, który aplikacja serwera potrafi rozpoznać i obsłużyć. Serwer nie musi być programem napisanym w Clarionie.

Serwer DDE napisany w Clarionie może używać procedur DDEITEM() oraz DDEVALUE() do określenia, co właściwie przesłał klient. Instrukcja CYCLE użyta po EVENT:DDEpoke sygnalizuje pozytywne potwierdzenie do klienta, który przesłał dane. DDEACKNOWLEDGE może przesłać zarówno pozytywne, jak i negatywne potwierdzenie.

Raportowane błędy:

601 Nieprawidłowy kanał DDE

602 Kanał DDE nie jest otwarty

603 DDEPOKE nie powiodło się

605 Przekroczenie czasu

Generowane zdarzenia: EVENT:DDEpoke Klient wysłał dane, które nie były żądane

Przykład:

```
WinOne      WINDOW,AT(0,0,160,400)
            END
DDEChannel  LONG

  CODE
  OPEN(WinOne)
  DDEChannel = DDECLIENT('Excel','System')                    ! otwiera kanał  to Excel
  DDEEXECUTE(DDEChannel,'[NEW(1)]')                           ! tworzy nowy arkusz
  DDEEXECUTE(DDEChannel,'[Save.As("DDE_CHART.XLS")]')         ! zapisuje go jako DDE_CHART.XLS
  DDECLOSE(DDEChannel)                                        ! zamyka konwersację
  DDEChannel = DDECLIENT('Excel','DDE_CHART.XLS')             ! otwiera kanał  do arkusza
  DDEPOKE(DDEChannel,'R1C2','Widgets')                        ! przesyła dane
  DDEPOKE(DDEChannel,'R1C3','Gadgets')
  DDEPOKE(DDEChannel,'R2C1','East')
  DDEPOKE(DDEChannel,'R3C1','West')
  DDEPOKE(DDEChannel,'R2C2','450')
  DDEPOKE(DDEChannel,'R3C2','275')
  DDEPOKE(DDEChannel,'R2C3','340')
  DDEPOKE(DDEChannel,'R3C3','390')
  DDEEXECUTE(DDEChannel,'[SELECT("R1C1:R3C2")]')              ! podświetla dane
  DDEEXECUTE(DDEChannel,'[NEW(2,2)]')                         ! i tworzy nowy wykres
    ! wyślij kilka dodatkowych poleceń do formatowania wykresu i pracy z nim
  DDECLOSE(DDEChannel)                                        ! zamyka kanał, gdy wykonane
```

Porównaj: DDEACKNOWLEDGE, DDEITEM, DDEVALUE

DDEQUERY (zwraca zarejestrowane serwery DDE)

DDEQUERY([*application*] [, *topic*])

DDEQUERY Zwraca aktualnie działające serwery DDE.

application Stała lub zmienna łańcuchowa zawierająca nazwę aplikacji, z którą się łączymy. Zazwyczaj jest to nazwa aplikacji. Jeśli pominiemy, zwracane są wszystkie zarejestrowane z określonym tematem *topic* aplikacje.

topic Stała lub zmienna łańcuchowa zawierająca nazwę tematu specyficznego dla danej aplikacji. Jeśli pominiemy, zwracane są wszystkie tematy dostępne w aplikacji *application*.

Procedura **DDEQUERY** daje w rezultacie łańcuch zawierający nazwę aktualnie dostępnej aplikacji *applications* serwera DDE i jej tematów *topics*. Jeśli parametr *topic* jest pominięty, są zwracane wszystkie tematy danej aplikacji. Jeśli parametr *application* jest pominięty, zwracane są wszystkie zarejestrowane aplikacje *applications* posiadające określony temat *topic*. jeśli zostaną pominięte wszystkie parametry, DDEQUERY zwraca w rezultacie wszystkie aktualnie dostępne serwery DDE.

Format danych w zwracanym łańcuchu jest następujący: *application:topic*, może on zawierać wiele par *application* oraz *topic* oddzielonych od siebie przecinkami (np. ‘Excel:MySheet.XLS,ClarionApp:DataFile.DAT’).

Typ rezultatu: STRING

Przykład:

```
! poniższy kod przykładowy nie obsługuje DDEADVISE
WinOne   WINDOW,AT(0,0,160,400)
         END
SomeServer  LONG
ServerString  STRING(200)

  CODE
  OPEN(WinOne)
  LOOP
    ServerString = DDEQUERY()                          ! zwraca wszystkie zarejestrowane serwery
    IF NOT INSTRING('SomeApp:MyTopic',ServerString,1,1)
      MESSAGE('Open SomeApp, Please')
    ELSE
      BREAK
    END
  END
  SomeServer = DDECLIENT('SomeApp','MyTopic')          ! otwiera jako klienta
  ACCEPT
  END
  DDECLOSE(SomeServer)
```

DDEREAD (pobiera daną z serwera DDE)

DDEREAD(*channel*, *mode*, *item* [, *variable*])

DDEREAD	Pobiera dane z poprzednio otwartego kanału klienta DDE.
channel	Etykieta zmiennej całkowitej LONG zawierającej numer kanału klienta – wartość zwracaną przez procedurę DDECLIENT.
mode	Ekwiwalent EQUATE definiujący typ łącza danych: DDE:auto, DDE:manual lub DDE:remove (zdefiniowane w EQUATES.CLW).
item	Stała lub zmienna łańcuchowa zawierający specyficzną dla aplikacji nazwę elementu danych do pobrania.
variable	Nazwa zmiennej, w której jest umieszczana wczytana dana. Jeśli pominiemy, a trybem *mode* jest DDE:remove, wszystkie połączenia do *item* są kasowane.

Procedura **DDEREAD** pozwala klientowi DDE na odczyt danej z kanału *channel* i zapisanie jej w zmiennej *variable*. Typ aktualizacji jest określony przez parametr *mode*. Parametr *item* dostarcza do serwera pewne wartości łańcuchowe informujące go, jakie specyficzne elementy danych są żądane. Format i struktura łańcucha *item* zależy od aplikacji serwera.

Jeśli tryb *mode* jest określony jako DDE:auto, zmienna *variable* jest w sposób ciągły aktualizowana przez serwer („gorące" łącze). Zdarzenie EVENT:DDEdata jest generowana za każdym razem, gdy zmienna *variable* jest aktualizowana przez serwer.

Jeśli tryb *mode* jest określony jako DDE:manual, zmienna *variable* jest aktualizowana raz i nie są generowane żadne zdarzenia. Sprawdzenie, czy wartość się nie zmieniła, wymaga wysłania kolejnych żądań DDEREAD do serwera („zimne" łącze).

Jeśli tryb *mode* jest określony jako DDE:remove, poprzednie "gorące" łącze do zmiennej *variable* jest przerywane. W przypadku, gdy przy trybie *mode* równym DDE:remove zostanie pominięta zmienna *variable*, przerywane są wszystkie poprzednie „gorące" łącza do elementu *item*, bez względu na to, jakie zmienne *variables* były podłączone. Oznacza to, że klient musi wysłać do serwera następne wywołanie DDEREAD w celu sprawdzenia, czy nie nastąpiła zmiana wartości.

Raportowane błędy:

601 Nieprawidłowy kanał DDE

602 Kanał DDE nie jest otwarty

605 Przekroczenie czasu

Generowane zdarzenia: Do aplikacji kleinta są przesyłane następoujące zdarzenia:

EVENT:DDEdata	Serwer dostarczył i zaktualizował element danych dla gorącego łącza.
EVENT:DDEclosed	Serwer przerwał połączenie DDE.

Przykład:

```
WinOne    WINDOW,AT(0,0,160,400)
          END
ExcelServer   LONG(0)
DDEReadVal REAL

  CODE
  OPEN(WinOne)
  ExcelServer = DDECLIENT('Excel','MySheet.XLS')
    ! otwórz jako klienta arkusza Excel
  IF NOT ExcelServer                    ! jeśli serwer nie działa
    MESSAGE('Please start Excel')       ! poinformuj uzytkownika i uruchom go
    CLOSE(WinOne)
    RETURN
  END
  DDEREAD(ExcelServer,DDE:auto,'R5C5',DDEReadVal)
    ! wymaga ciągłej aktualizacji z serwera
  ACCEPT
    CASE EVENT()
    OF EVENT:DDEdata                    ! gdy nadchodzą zmienione dane z Excela
      PassedData(DDEReadVal)            ! wywołaj procedurę do ich przetworzenia
    END
  END
```

Porównaj: DDEQUERY, DDEWRITE, DDESERVER

DDESERVER (zwraca kanał serwera DDE)

DDESERVER([*application*] [, *topic*])

DDESERVER Zwraca nowy numer kanału DDE serwera.

application Stała lub zmienna łańcuchowa zawierająca nazwę aplikacji, z którą się łączymy. Zazwyczaj jest to nazwa aplikacji. Jeśli pominiemy, stosowana jest nazwa pliku aplikacji (bez rozszerzenia).

topic Stała lub zmienna łańcuchowa zawierająca nazwę tematu specyficznego dla danej aplikacji. Jeśli pominiemy, aplikacja odpowie na każde żądanie udostępnienia danych.

Procedura **DDESERVER** zwraca numer nowego kanału serwera DDE dla aplikacji *application* i tematu *topic*. Numer kanału określa temat *topic*, w ramach którego serwer *application* ma dostarczać dane. Pozwala to pojedynczej aplikacji *application* napisanej w Clarionie na rejestrowanie się jako serwer DDE dla wielu tematów *topics*.

Typ rezultatu: LONG

Przykład:

```
DDERetVal STRING(20)
WinOne    WINDOW,AT(0,0,160,400)
            ENTRY(@s20),USE(DDERetVal)
          END
MyServer  LONG

  CODE
  OPEN(WinOne)
  MyServer = DDESERVER('MyApp','DataEntered')          ! otwiera jako serwer
  ACCEPT
    CASE EVENT()
    OF EVENT:DDErequest                                ! jeśli wymagane są dane jednorazowo
      DDEWRITE(MyServer,DDE:manual,'DataEntered',DDERetVal)      ! udostepniej je jednorazowo
    OF EVENT:DDEadvise                                 ! gdy wymagana jest stała aktualizacja
      DDEWRITE(MyServer,15,'DataEntered',DDERetVal)
        ! sprawdzaj, czy sa zmiany co 15 sekund
        ! dostarczaj dane, gdy się zmieniły
    END
  END
```

Porównaj: DDECLIENT, DDEWRITE

DDETOPIC (zwraca temat serwera)

DDETOPIC( )

Procedura **DDETOPIC** zwraca łańcuch zawierający nazwę tematu dla kanału DDE , przez który zostało właśnie przesłane zdarzenie DDE.

Typ rezultatu: STRING

Przykład:

```
WinOne    WINDOW,AT(0,0,160,400)
          END
TimeServer   LONG
DateServer   LONG
FormatTime   STRING(5)
FormatDate   STRING(8)

  CODE
  OPEN(WinOne)
  TimeServer = DDESERVER('SomeApp')          ! otwiera jako serwer
  ACCEPT
    CASE EVENT()
    OF EVENT:DDErequest
      CASE DDETOPIC()                        ! pobierz wymagany element
      OF 'Time'
        FormatTime = FORMAT(CLOCK(),@T1)
        DDEWRITE(TimeServer,DDE:manual,'Time',FormatTime)
      OF 'Date'
        FormatDate = FORMAT(TODAY(),@D1)
        DDEWRITE(DateServer,DDE:manual,'Date',FormatDate)
      END
    END
  END
```

Porównaj: DDEREAD, DDECLIENT, DDESERVER

DDEVALUE (zwraca wartość danej przesłanej do serwera)

DDEVALUE( )

Procedura **DDEVALUE** zwraca w rezultacie łańcuch zawierający dane przesłane do serwera DDE Clarion przez instrukcję DDEPOKE lub przez polecenie przeznaczone do wykonania przez instrukcję DDEEXECUTE.

Typ rezultatu: STRING

Przykład:

```
WinOne    WINDOW,AT(0,0,160,400)
          END
TimeServer  LONG
TimeStamp   FILE,DRIVER(ASCII),PRE(Tim)
Record        RECORD
FormatTime      STRING(5)
FormatDate      STRING(8)
Message         STRING(50)
              . .

  CODE
  OPEN(WinOne)
  TimeServer = DDESERVER('TimeStamp')          ! otwiera jako serwer
  ACCEPT
    CASE EVENT()
    OF EVENT:DDEpoke
      OPEN(TimeStamp)
      Tim:FormatTime = FORMAT(CLOCK(),@T1)
      Tim:FormatDate = FORMAT(TODAY(),@D1)
      Tim:Message = DDEVALUE()                 ! pobierz daną
      ADD(TimeStamp)
      CLOSE(TimeStamp)
      CYCLE                                    ! zapewnij potwierdzenie
    END
  END
```

Porównaj: DDEPOKE, DDEEXECUTE

DDEWRITE (dostarcza dane do klienta DDE)

DDEWRITE(*channel*, *mode*, *item* [, *variable*])

DDEWRITE	Dostarcza dane do otwartego kanału serwera DDE.
channel	Etykieta zmiennej całkowitej LONG zawierającej numer kanału serwera – wartość zwracaną przez procedurę DDESERVER.
mode	Stała lub zmienna całkowita zawierająca interwał czasowy (w sekundach), co który są dostarczane zmiany zmiennej *variable*, bądź ekwiwalent EQUATE definiujący typ łącza danych: DDE:auto, DDE:manual lub DDE:remove (zdefiniowane w EQUATES.CLW).
item	Stała lub zmienna łańcuchowa zawierający specyficzną dla aplikacji nazwę elementu danych przeznaczonego do dostarczenia.
variable	Nazwa zmiennej dostarczającej dane. Jeśli pominiemy, a tryb *mode* jest określony jako DDE:remove, wszystkie łącza do elementu *item* są kasowane.

Procedura **DDEWRITE** pozwala serwerowi DDE na udostępnianie klientowi danych zmiennej *variable*. Parametr *item* dostarcza wartości łańcuchowe informujące, jaki specyficzny element danych jest żądany. Format i struktura łańcucha *item* zależy od aplikacji serwera. *server*.

Typ przeprowadzanej aktualizacji zależy od parametru *mode*. Jeśli tryb *mode* jest określony jako DDE:auto, klient rejestruje bieżącą wartość zmiennej *variable*, zaś biblioteki wewnętrzne utrzymują ją, tak, by mogła być udostępniona na następne żadanie tego lub innego klienta. Jeśli klient tworzy „gorące" łącze, każda zmiana zmiennej *variable* powinna być wychwytywana przez program Clarion, tak by mógł on wywołać nową instrukcję DDEWRITE aktualizując wartość u klienta.

Jeśli tryb *mode* jest określony jako DDE:manual, zmienna *variable* jest aktualizowana raz. Jeśli klient tworzy „gorące" łącze, każda zmiana zmiennej *variable* powinna być wychwytywana przez program Clarion, tak by mógł on wywołać nową instrukcję DDEWRITE aktualizując wartość u klienta. Właściwość PROP:DDETimeOut może być używana do ustawiania lub pobierania wartości przekroczenia czasu dla połączenia DDE (domyślnie jest to 5 sekund). Jeśli parametr *mode* jest dodatnią liczbą całkowitą, biblioteki wewnętrzne sprawdzają wartość zmiennej *variable* co określoną liczbę sekund. Jeśli wartość się zmieniła, klient jest automatycznie aktualizowany przez biblioteki wewnętrzne (nie jest to wymagane w kodzie). Naraża nas to na oczywiste przeciążenia, w zależności od rodzaju danych, tak więc należy korzystać z tej możliwości z rozwagą i tylko wtedy, gdy jest to naprawdę konieczne.

Jeśli tryb *mode* jest określony jako DDE:remove, poprzednie "gorące" łącze do zmiennej *variable* jest przerywane. W przypadku, gdy przy trybie *mode* równym DDE:remove zostanie pominięta zmienna *variable*, przerywane są wszystkie poprzednie „gorące" łącza do elementu *item*, bez względu na to, jakie zmienne *variables* były podłączone. Oznacza to, że klient musi wysłać do serwera następne wywołanie DDEREAD w celu sprawdzenia, czy nie nastąpiła zmiana wartości.

Raportowane błędy: 601 Nieprawidłowy kanał DDE

602 Kanał DDE nie jest otwarty

605 Przekroczenie czasu

Generowane zdarzenia: EVENT:DDErequest Klient zgłasza zapotrzebowanie na element danych (zimne łącze).

EVENT:DDEadvise Klient zgłasza zapotrzebowanie na ciągła aktualizację elementu danych (gorące łącze).

Przykład:

```
DDERetVal STRING(20)
WinOne    WINDOW,AT(0,0,160,400)
            ENTRY(@s20),USE(DDERetVal)
          END
MyServer  LONG

  CODE
  OPEN(WinOne)
  MyServer = DDESERVER('MyApp','DataEntered')          ! otwiera jako serwer
  ACCEPT
    CASE EVENT()
    OF EVENT:DDErequest                                ! gdy dane są wymagane jednorazowo
      DDEWRITE(MyServer,DDE:manual,'DataEntered',DDERetVal)
                                                       ! dostarcz je jednorazowo
    OF EVENT:DDEadvise                                 ! gdy serwer ma dostarczać dane zaktualizowane
      DDEWRITE(MyServer,15,'DataEntered',DDERetVal)
      ! sprawdzaj co 15 sekund, czy nie nastąpiły zmieny
      ! i dostarczaj zmienione dane
    END
  END
```

Porównaj: DDEQUERY, DDEREAD, DDESERVER

Łączenie i osadzanie obiektów

Przegląd OLE

Łączenie i osadzanie obiektów - Object Linking and Embedding (OLE) umożliwia dołączanie lub osadzanie "obiektów" jednej aplikacji do „dokumentów" (struktur danych) innej aplikacji. Aplikacja tworząca obiekt i zarządzająca nim jest serwerem OLE, podczas gdy aplikacja zawierająca obiekt jest kontrolerem OLE (nazywana też jest czasami klientem OLE).

„Obiekty" OLE są strukturami danych właściwymi dla aplikacji serwera OLE (np. wykresami z arkusza kalkulacyjnego, rysunkami). Obiekt jest umieszczany w „oknie kontenera" w aplikacji kontrolera OLE. W Clarionie „okno kontenera" to kontrolka OLE. Implementacja OLE w Clarionie pozwala aplikacjom Clariona na występowanie w roli kontrolera OLE – łączenie I osadzanie obiektów z dowolnych serwerów OLE. Clarion obsługuje również OLE Automation, technikę dającą kontrolerowi OLE dynamiczną kontrolę nad serwerem OLE za pośrednictwem jego języka makropoleceń.

Łączenie obiektów

Łączenie obiektów oznacza, najogólniej rzecz biorąc, że kontroler OLE przechowuje „wskaźniki" na obiekty będące strukturami danych (np. arkusz kalkulacyjny) lub ich elementami (np. zakres komórek arkusza kalkulacyjnego). Gdy obiekt jest łączony do kontrolera OLE, ten ostatni zawiera tylko niezbędne informacje umożliwiające np. jego zlokalizowanie. Są one przechowywane albo w postaci pola BLOB, albo pliku OLE Compound Storage.

Osadzanie obiektów

Osadzenie obiektu oznacza, że aplikacja kontrolera OLE przechowuje cały obiekt, niezależnie od aplikacji serwera OLE. Obiekt osadzony przez kontroler OLE nie jest przechowywany w postaci oddzielnego pliku danych, do którego dostęp mógłby mieć serwer OLE. Aplikacja kontrolera OLE całkowicie przechowuje aktywny obiekt, który jest zapisany w polu BLOB lub w pliku OLE Compound Storage.

Zarządzanie obiektem OLE

Dowolny obiekt w aplikacji kontrolera OLE, dołączony lub osadzony, jest zarządzany przez aplikację serwera OLE, która go utworzyła, a nie przez kontroler OLE. Oznacza to, że gdy użytkownik zechce dokonać zmian w obiekcie, kontroler OLE uaktywnia w tym celu aplikację serwera OLE. Istnieją dwa sposoby na uaktywnienie serwera OLE: aktywacja „in-place" oraz tryb „open-mode".

Aktywacja in-place

Aktywacja in-place oznacza, że z punktu widzenia użytkownika, pozostaje on w aplikacji kontrolera OLE, do której są dołączane menu i paski narzędzi serwera OLE, który z kolei staje się aktywną aplikacją „wewnętrzną". Edytowany obiekt jest obramowywany (gruba ramka z kreskami ukośnymi) w celu zaznaczenia, że znajduje się on w trybie edycji. Jeśli aplikacja serwera OLE posiada paski narzędzi, pojawiają się one albo jako paski rozwijalne, albo paski dołączone do jednej z krawędzi okna głównego, albo kombinacja obu tych stylów.

Aktywacja trybu open-mode

Aktywacja trybu open-mode oznacza, że użytkownik jest przełączany do aplikacji serwera OLE działającej w oddzielnym oknie. Edytowany obiekt pojawia się w tym oknie gotowy do edycji, podczas gdy obiekt w oknie kontrolera OLE jest otaczany pogrubioną ramką oznaczającą, że znajduje się on w trybie edycji.

Właściwości kontenera OLE

Istnieje pewna liczba właściwości powiązanych z kontrolką kontenera OLE, które działają tylko z obiektami OLE (nie działają z kontrolkami .OCX).

Właściwości atrybutów

PROP:Create	Atrybut CREATE (pusty, gdy nie występuje). Tylko-do-zapisu.
PROP:Open	Atrybut OPEN (pusty, gdy nie występuje). Tylko-do-zapisu.
PROP:Document	Atrybut DOCUMENT (pusty, gdy nie występuje). Tylko-do-zapisu.
PROP:Link	Atrybut LINK (pusty, gdy nie występuje). Tylko-do-zapisu.
PROP:Clip	Atrybut CLIP. Atrybut przełącznikowy. Przypisanie łańcucha pustego ('') lub zera - wyłącza, przypisanie '1' lub 1 - włącza. Tylko-do-zapisu.
PROP:Stretch	Atrybut STRETCH. Atrybut przełącznikowy. Przypisanie łańcucha pustego ('') lub zera - wyłącza, przypisanie '1' lub 1 - włącza. Tylko-do-zapisu.
PROP:Autosize	Atrybut AUTOSIZE. Atrybut przełącznikowy. Przypisanie łańcucha pustego ('') lub zera - wyłącza, przypisanie '1' lub 1 - włącza. Tylko-do-zapisu.
PROP:Zoom	Atrybut ZOOM. Atrybut przełącznikowy. Przypisanie łańcucha pustego ('') lub zera - wyłącza, przypisanie '1' lub 1 - włącza. Tylko-do-zapisu.
PROP:Compatibility	Atrybut COMPATIBILITY (pusty, gdy nie występuje). Tylko-do-zapisu.

Właściwości nie zadeklarowane

PROP:Blob — Konwertuje obiekt do i z pola BLOB. Odczyt-zapis.

PROP:SaveAs — Zapisuje obiekt do pliku OLE Compound Storage. Tylko-do-zapisu. Składnia umieszczenia obiektu w pliku wygląda następująco: '*filename*\!*component*'. Na przykład:

```
?controlx{PROP:SaveAs} = 'myfile\!objectx'
```

PROP:DoVerb — Wykonuje polecenie OLE doverb z przedstawionego poniżej zbioru. Tylko-do-zapisu.

DOVERB:Primary (0)

Wywołuje podstawową akcję obiektu. Akcję tę determinuje sam obiekt, nie kontener. Jeśli obiekt obsługuje aktywację in-place, podstawowa akcja zazwyczaj uaktywnia edycję obiektu w trybie in-place.

DOVERB:Show (-1)

Informuje obiekt, że ma się pokazać w trybie do edycji lub podglądu. Wywołuje się w celu wyświetlenia nowo wstawionego obiektu w trybie początkowej edycji lub do pokazania źródeł łączenia. Jest to zazwyczaj alias różnych innych akcji zdefiniowanych dla obiektu.

DOVERB:Open (-2)

Informuje obiekt, że ma się otworzyć w trybie do edycji w oddzielnym oknie z jego kontenera (obejmuje to obiekty obsługujące aktywację in-place). Jeśli obiekt nie obsługuje aktywacji in-place, będzie to ta sama akcja, co w przypadku DOVERB:Show.

DOVERB:Hide (-3)

Informuje obiekt, że ma usunąć swój interfejs użytkownika. Dotyczy to tylko obiektów aktywowanych w trybie in-place.

DOVERB:UIActivate (-4)

Uaktywnia obiekt w trybie in-place, włącznie z jego pełnym interfejsem użytkownika, włączając w to menu, paski narzędzi, nazwę w pasku tytułowym okna kontenera.

DOVERB:InPlaceActivate (-5)

Uaktywnia obiekt w trybie in-place bez wyświetlania jego narzędzi (menu i pasków narzędzi), których użytkownik potrzebuje do zmiany działania lub wyglądu obiektu.

DOVERB:DiscardUndoState (-6)

Informuje obiekt, że ma odrzucić dowolny stan undo, który może być zarządzany bez deaktywacji obiektu.

DOVERB:Properties (-7)

Wywołuje przeglądarkę właściwości obiektu umożliwiając użytkownikowi ich zmianę.

PROP:Deactivate Deaktywuje obiekt OLE znajdujący się w trybie in-place. Odczyt-Zapis-Wykonanie.

PROP:Update Informuje obiekt OLE, że ma się zaktualizować. Odczyt-Zapis-Wykonanie.

PROP:CanPaste Czy można wkleić obiekt ze Schowka? Tylko-do-odczytu.

PROP:Paste Wkleja obiekt ze Schowka do kontrolki kontenera OLE. Odczyt-Zapis-Wykonanie.

PROP:CanPasteLink Czy obiekt może być wklejony ze schowka w postaci łącza? Tylko-do-odczytu.

PROP:PasteLink Wkleja w postaci łącza obiekt ze Schowka do kontrolki kontenera OLE. Odczyt-Zapis-Wykonanie.

PROP:Copy Kopiuje obiekt z kontrolki kontenera OLE do Schowka. Odczyt-Zapis-Wykonanie.

PROP:ReportException Raportuje wyjątki OLE (dla celów debugowania). Tylko-do-zapisu.

PROP:OLE Czy w kontenerze znajduje się obiekt OCX lub OLE? Tylko-do-odczytu.

PROP:Language Numer języka stosowanego w OLE Automation lub metodzie OCX. Numer języka angielskiego (US English) to 0409H, numery innych języków mogą zostać określone w oparciu o dane zawarte w pliku WINNT.H Windows SDK. Odczyt-Zapis.

Przykład:

```
    PROGRAM
    MAP
      INCLUDE('OCX.CLW')
  SelectOleServer PROCEDURE(OleQ PickQ),STRING
    END
    INCLUDE 'XL.CLW'                          ! stałe wykorzystywane przez Excel
    INCLUDE 'ERRORS.CLW'                      ! stałe kodów błędów

  SaveLinks  FILE,DRIVER('TopSpeed'),PRE(SAV),CREATE
  Object       BLOB
  Record       RECORD
  LinkType       STRING(1) !F = File, B = BLOB
  LinkFile       STRING(64)                 ! plik OLE Compound Storage
               END
             END

  i            LONG                         ! liczniki powtórzeń
  j            LONG

  ResultQ    QUEUE                          ! kolejka do zapisania rezultatu OLEDIRECTORY
  Name         CSTRING(64)
  CLSID        CSTRING(64)
  ProgID       CSTRING(64)
             END

  MainWin    WINDOW('OLE Demo'),AT(,,350,200),STATUS(-1,-1),SYSTEM,GRAY,RESIZE,MAX,TIMER(1)
               MENUBAR
                 MENU('&File')
                   ITEM('e&xit'),USE(?exit)
                 END
                 MENU('&Objects')
                   ITEM('Create Object'),USE(?CreateObject)
                   ITEM('Paste Object'),USE(?PasteObject)
                   ITEM('PasteLink Object'),USE(?PasteLinkObject)
                   ITEM('Save Object to BLOB'),USE(?SaveObjectBlob),DISABLE
                   ITEM('Save Object to OLE File'),USE(?SaveObjectFile),DISABLE
                   ITEM('Retrieve Saved Object'),USE(?GetObject),DISABLE
                 END
                 MENU('&Activate')
                   ITEM('&Spreadsheet'),USE(?ActiveExcel)
                   ITEM('&Any OLE Object'),USE(?ActiveOLE),DISABLE
                 END
               END
               OLE,AT(5,10,160,100),COLOR(0808000H),USE(?ExcelObject)
                 MENUBAR
                   MENU('&Clarion App')
                     ITEM('&Deactivate Excel'),USE(?DeactExcel)
                   END
                 END
               END
               OLE,AT(170,10,160,100),USE(?AnyOLEObject),AUTOSIZE
                 MENUBAR
                   MENU('&Clarion App')
                     ITEM('&Deactivate Object'),USE(?DeactOLE)
                   END
                 END
               END
             END
  CODE
    OPEN(SaveLinks)
    IF ERRORCODE()                            ! kontrola błędów przy otwarciu
      IF ERRORCODE() = NoFileErr              ! jeśli plik nie istnieje
        CREATE(SaveLinks)                     ! utwórz go
        IF ERRORCODE() THEN HALT(,ERROR()).
        OPEN(SaveLinks)                       ! następnie otwórz do użycia
```

```
    IF ERRORCODE() THEN HALT(,ERROR()).
  ELSE
    HALT(,ERROR())
  END
END
OPEN(MainWin)
?ExcelObject{PROP:Create} = 'Excel.Sheet.5'     ! utwórz obiekt arkusza Excel
DO BuildSheetData                               ! wypełnij go losowymi danymi
IF RECORDS(SaveLinks)                           ! sprawdź, czy istnieje zapisany rekord
  SET(SaveLinks)                                ! i pobierz go
  NEXT(SaveLinks)
  POST(EVENT:Accepted,?GetObject)               ! i wyświetl go
  DO MenuEnable
ELSE
  ADD(SaveLinks)                                ! lub dodaj pusty rekord
END
IF ERRORCODE() THEN HALT(,ERROR()).
ACCEPT
  CASE EVENT()
  OF EVENT:CloseWindow
    ?ExcelObject{PROP:Deactivate}               ! de-aktywuj aplikację serwera OLE
    ?AnyOLEObject{PROP:Deactivate}
  OF EVENT:Timer
    IF CLIPBOARD()
      IF ?AnyOLEObject{PROP:CanPaste}           ! czy można wkleić obiekt ze Schowka?
        IF ?PasteObject{PROP:Disable} THEN ENABLE(?PasteObject).
      ELSIF NOT ?PasteObject{PROP:Disable}
        DISABLE(?PasteObject)
      END
      IF ?AnyOLEObject{PROP:CanPasteLink}       ! czy można wkleić łącze do obiektu ze Schowka?
        IF ?PasteLinkObject{PROP:Disable} THEN ENABLE(?PasteLinkObject).
      ELSIF NOT ?PasteLinkObject{PROP:Disable}
        DISABLE(?PasteLinkObject)
      END
    END
    OF EVENT:Accepted
      CASE FIELD()
      OF ?Exit
        POST(EVENT:CloseWindow)
      OF ?CreateObject
        OLEDIRECTORY(ResultQ,0)                 ! pobierz listę zainstalowanych serwerów OLE
        ?AnyOLEObject{PROP:Create} = SelectOleServer(ResultQ)
                                                ! pozwól użytkownikowi wybrać jeden z nich
        ?AnyOLEObject{PROP:DoVerb} = 0          ! uaktywnij serwer OLE w jego trybie domyślnym
        DO MenuEnable
      OF ?PasteObject
        ?AnyOLEObject{PROP:Paste}               ! wklej obiekt
        SETCLIPBOARD('Paste Completed')         ! przypisz nie-obiektowy tekst do Schowka
        DO MenuEnable
      OF ?PasteLinkObject
        ?AnyOLEObject{PROP:PasteLink}           ! wklej łącze do obiektu
        SETCLIPBOARD('PasteLink Completed')     ! przypisz nie-obiektowy tekst do Schowka
        DO MenuEnable
      OF ?SaveObjectBlob                        ! zapisz obiekt w BLOB
        SAV:Object{PROP:Handle} = ?AnyOLEObject{PROP:Blob}
        SAV:LinkType = 'B'
        PUT(SaveLinks)
        IF ERRORCODE() THEN STOP(ERROR()).
      OF ?SaveObjectFile                        ! zapisz w pliku OLE Compound Storage
        ?AnyOLEObject{PROP:SaveAs} = 'TEST1.OLE\!Object'
        SAV:LinkFile = 'TEST1.OLE\!Object'
        SAV:LinkType = 'F'
        PUT(SaveLinks)
        IF ERRORCODE() THEN STOP(ERROR()).
      OF ?GetObject
        IF SAV:LinkType = 'F'                   ! zapisany w pliku OLE Compound Storage?
```

```
            ?AnyOLEObject{PROP:Open} = SAV:LinkFile
          ELSIF SAV:LinkType = 'B'                  ! zapisany w BLOB?
            ?AnyOLEObject{PROP:Blob} = SAV:Object{PROP:Handle}
          END
          DISPLAY
        OF ?ActiveExcel
          ?ExcelObject{PROP:DoVerb} = 0             ! aktywacja in-place Excela
        OF ?ActiveOLE
          ?AnyOLEObject{PROP:DoVerb} = 0            ! aktywacja serwera OLE w jego trybie domyślnym
        OF ?DeactExcel
          ?ExcelObject{PROP:Deactivate}             ! powrót do aplikacji Clarion
        OF ?DeactOLE
          ?AnyOLEObject{PROP:Deactivate}            ! powrót do aplikacji Clarion
        END
      END
    END

  BuildSheetData      ROUTINE                       ! wykorzystuje OLE Automation do zbudowania arkusza

    ?ExcelObject{PROP:ReportException} = TRUE       ! Excel powinien raportować błędy
    ?ExcelObject{'Application.Calculation'} = xlManual  ! wyłącz automatyczną rekalkulację
    LOOP i = 1 TO 3                                 ! wypełnij arkusz wartościami
      LOOP j = 1 TO 3
        ?ExcelObject{'Cells(' & i & ',' & j & ').Value'} = Random(100,900)
      END
      ?ExcelObject{'Cells(4,' & i & ').Value'} = 'Sum'
      ?ExcelObject{'Cells(5,' & i & ').FormulaR1C1'} = '=SUM(R[-4]C:R[-2]C)'
      ?ExcelObject{'Cells(6,' & i & ').Value'} = 'Average'
      ?ExcelObject{'Cells(7,' & i & ').FormulaR1C1'} = '=AVERAGE(R[-6]C:R[-4]C)'
    END
    ?ExcelObject{'Application.Calculation'} = xlAutomatic  ! włącz automatyczną rekalkulację
    DISPLAY
    MenuEnable ROUTINE                              ! włącz elementy menu
    ENABLE(?ActiveOLE)
    ENABLE(?SaveObjectBlob,?GetObject)
    SelectOleServer PROCEDURE(OleQ PickQ)

window  WINDOW('Choose OLE Server'),AT(,,122,159),CENTER,SYSTEM,GRAY
          LIST,AT(11,8,100,120),USE(?List),HVSCROLL, |
            FORMAT('146L~Name~@s64@135L~CLSID~@s64@20L~ProgID~@s64@'),FROM(PickQ)
          BUTTON('Select'),AT(42,134),USE(?Select)
        END
  CODE
    OPEN(window)
    SELECT(?List,1)
    ACCEPT
      CASE ACCEPTED()
      OF ?Select
        GET(PickQ,CHOICE(?List))
        IF ERRORCODE() THEN STOP(ERROR()).
        POST(EVENT:CloseWindow)
      END
    END
    RETURN(PickQ.ProgID)
```

Właściwości interfejsu

PROP:Object — Pobiera interfejs przesłany dla obiektu. Tylko-do-odczytu.

W VisualBasic pasek narzędzi i kontrolka struktury drzewiastej wykorzystują kontrolkę image-list do wyświetlenia ikon. Do powiązania kontrolki image z paskiem narzędzi używamy następującego kodu:

```
?toolbar{'ImageList'} = ?imagelist{prop:object}
```

PROP:SelectInterface — Wybiera interfejs do zastosowania w obiekcie. Tylko-do-zapisu.

```
?x{PROP:SelectInterface} = 'x.y'
?x{'z(1)'} = 1
?x{'z(2)'} = 2
    ma to samo znaczenie, co:
?x{'x.y.z(1)'} = 1
?x{'x.y.z(2)'} = 2
```

PROP:AddRef — Zwiększa licznik referencji dla interfejsu. Tylko-do-zapisu.

PROP:Release — Zmniejsza licznik referencji dla interfejsu. Tylko-do-zapisu.

Hierarchia biblioteki i obiektów Clarion OLE/OCX

Podczas projektowania i implementowania biblioteki OLE Clariona brak dostępu do drugorzędnych obiektów utworzonych przez obiekt podstawowy (przykład z Excela: ExcelUse{'Application.Charts.Add'}) nie był rozpatrywany w kategoriach wielkiego problemu, gdyż istniały inne metody dostępu do takiego obiektu (ExcelUse{'Application.Charts(Chart1).ChartWizard('&?ex{'Range(A5:C5)'}&','&xl3DPie&',7,1,0,0,2,,,,)'})

W tym czasie był znany tylko jeden znany przypadek, którego ten problem nie dotyczył. Tak jest prawdopodobnie do dzisiaj, jako że standard OLE zakłada, że obiekt implementując kolekcję musi także implementować metodę dostępu do niego poprzez indeksowanie. W związku ze specjalnym przypadkiem opisanym powyżej, gdy obiekt był tworzony przez jedną kontrolkę i przekazywany do innego obiektu jako parametr, zaimplementowano metodę, która może być bardziej lub mniej przeźroczysta dla użytkownika. Wywołanie metody zwracającej IDispatchInterface zostało przekonwertowane do specjalnej reprezentacji (znak '`', po którym następują cyfry). Ta specjalne reprezentacja jest rozpoznawana w kilku miejscach biblioteki OLE. Miejsce, które można uznać za najbardziej przydatne, to takie, gdzie interfejs może się pojawić w składni właściwości z intencją zastąpienia dowolnego poprzedniego interfejsu w dostępie do właściwości lub metod obiektu. Na przykład:

```
x=y{'charts.add()')
y{x&'p(7)'}
```

gdzie **y** jest obiektem OLE, a **x** jest typu CSTRING. Jest to przykład metody zwracającej interfejs, który jest później wykorzystywany przy dostępie do metody **p** z parametrem **7**. W tym kontekście przyszłe komplikacje wynikają ze zliczania referencji stosowanego w OLE. Oznacza to, że jeśli obiekt jest wykorzystywany więcej niż raz, musi, przed użyciem, zwiększać swój licznik referencji o jeden.

```
x=y{'charts.add()')
y{PROP:AddRef}=x
y{x&'p(7)'}
y{x&'p(7)'} ;last use of x
```

OLEDIRECTORY (pobranie listy zainstalowanych OLE/OCX)

OLEDIRECTORY(*list* , *flag* [, *bits*])

OLEDIRECTORY Pobiera listę zainstalowanych serwerów OLE lub kontrolek OCX.

list Etykieta kolejki QUEUE, w której zostanie umieszczona pobrana lista.

flag Stała lub zmienna całkowita determinująca, czy jest pobierana lista serwerów OLE (*flag* = 0), czy też kontrolek OCX (*flag* = 1).

bits Stała lub zmienna całkowita determinująca, czy jest pobierana lista 16-to, czy 32-bitowych kontrolek OCX. Jeśli parametr jest równy jeden (1), zwracane są kontrolki OCX 16-bitowe. Jeśli dwa (2) - 32-bitowe. Jeśli trzy (3) – zarówno 16-to, jak i 32-bit owe. Jeśli zero lub parametr został pominięty – program 16-bitowy zwraca kontrolki OCX 16-bitowe, a program 32-bitowy zwraca kontrolki OCX 32-bitowe.

OLEDIRECTORY pobiera listę zainstalowanych serwerów OLE lub kontrolek OCX i umieszcza ją w kolejce *list*. Kolejka *list* musi być zadeklarowana z tą samą strukturą, co kolejka OleQ QUEUE w EQUATES.CLW:

```
OleQ    QUEUE,TYPE
Name      CSTRING(64)      ! nazwa aplikacji serwera OLE
CLSID     CSTRING(64)      ! unikalny identyfikator dla systemu operacyjnego
ProgID    CSTRING(64)      ! nazwa rejestru, np: Excel.Sheet.5
          END
```

Przykład:

```
ResultQ   QUEUE(OleQ).              ! deklaruj ResultQ tak, jak OleQ w EQUATES.CLW
  CODE
  OLEDIRECTORY(ResultQ,0)           ! pobierz listę zainstalowanych serwerów OLE i umieść w kolejce ResultQ
  ?OleControl{PROP:Create} = SelectOleServer(ResultQ)    ! pozwól wybrac jeden użytkownikowi

SelectOleServer  PROCEDURE(OleQ PickQ)                  ! procedura wyboru serwera OLE
window           WINDOW('Choose OLE Server'),AT(,,122,159),CENTER,SYSTEM,GRAY
                   LIST,AT(11,8,100,120),USE(?List),HVSCROLL, |
                   FORMAT('146L~Name~@s64@135L~CLSID~@s64@20L~ProgID~@s64@'),FROM(PickQ)
                   BUTTON('Select'),AT(42,134),USE(?Select)
                 END
  CODE
  OPEN(window)
  SELECT(?List,1)
  ACCEPT
    CASE ACCEPTED()
    OF ?Select
      GET(PickQ,CHOICE(?List))
      IF ERRORCODE() THEN STOP(ERROR()).
      POST(EVENT:CloseWindow)
    ..
    RETURN(PickQ.ProgID)
```

Kontrolki OLE (.OCX)

Przegląd

Pliki kontrolek OLE (OLE custom controls) mają na ogół rozszerzenie .OCX. Są one zwyczajowo nazywane kontrolkami .OCX. Kontrolki .OCX są podobne do kontrolek .VBX – są odrębne i są przeznaczone do wykonania określonych, specyficznych zadań na rzecz programu. Z drugiej jednak strony, kontrolki .OCX nie mają ograniczeń kontrolek .VBX, a to z tego względu, że kontrolki .OCX są budowane zgodnie ze specyfikacją Microsoft OLE 2, opracowaną pod kątem zgodności pomiędzy różnymi językami (a nie tylko dla Visual Basic).

Właściwości kontrolki .OCX

PROP:Create	Atrybut CREATE (pusty, jeśli nie występuje). Tylko-do-zapisu.
PROP:DesignMode	Określenie, czy kontrolka .OCX w kontenerze znajduje się w trybie edycyjnym (czy jest ramka wokół niej)? Tylko-do-zapisu.
PROP:Ctrl	Czy to jest kontrolka .OCX? Tylko-do-odczytu.
PROP:GrabHandles	Powoduje wyświetlenie uchwytów przeniesienia kontrolki .OCX. Tylko-do-zapisu.
PROP:OLE	Czy jest kontrolka OCX lub obiekt OLE w kontenerze? Tylko-do-odczytu.
PROP:IsRadio	Czy kontrolka OCX jest przyciskiem radio? Tylko-do-odczytu.
PROP:LastEventName	Pobiera nazwę ostatniego zdarzenia przesłanego do kontrolki .OCX. Tylko-do-odczytu.
PROP:SaveAs	Zapisuje obiekt do pliku OLE Compound Storage. Tylko-do-zapisu.. Składnia umieszczenia obiektu w pliku jest następująca '*nazwa_pliku*\!k*omponent*'. Na przykład: `?controlx{PROP:SaveAs} = 'myfile\!objectx'`
PROP:ReportException	Raportuje wyjątki OLE (do celów debugowania). Tylko-do-zapisu.
PROP:DoVerb	Wykonuje polecenie OLE doverb w oparciu o poniższy zestaw poleceń. Tylko-do-zapisu. DOVERB:Primary (0) Wywołuje podstawową akcję obiektu. Obiekt, nie kontener, determinuje tę akcję. Jeśli obiekt obsługuje aktywację in-place, zazwyczaj aktywuje obiekt w trybie in-place. DOVERB:Show (-1) Informuje obiekt, że ma się wyświetlić w trybie do edycji lub podglądu. Wywoływany do wyświetlenia nowo wstawianych obiektów i wyświetlenia źródeł łączeń. Jest to

zazwyczaj alias dla pewnych innych akcji zdefiniowanych dla obiektu.

DOVERB:Open (-2)

Informuje obiekt, że ma się otworzyć w trybie do edycji w oddzielnym oknie z jego kontenera (obejmuje to obiekty obsługujące aktywację in-place). Jeśli obiekt nie obsługuje aktywacji in-place, będzie to ta sama akcja, co w przypadku DOVERB:Show.

DOVERB:Hide (-3)

Informuje obiekt, że ma usunąć swój interfejs użytkownika. Dotyczy to tylko obiektów aktywowanych w trybie in-place.

DOVERB:UIActivate (-4)

Uaktywnia obiekt w trybie in-place, włącznie z jego pełnym imterfejsem użytkownika, włączając w to menu, paski narzędzi, nazwę w pasku tytułowym okna kontenera.

DOVERB:InPlaceActivate (-5)

Uaktywnia obiekt w trybie in-place bez wyświetlania jego narzędzi (menu i pasków narzędzi), których użytkownik potrzebuje do zmiany działania lub wyglądu obiektu.

DOVERB:DiscardUndoState (-6)

Informuje obiekt, że ma odrzucić dowolny stan undo, który może być zarządzany bez deaktywacji obiektu.

DOVERB:Properties (-7)

Wywołuje przeglądarkę właściwości obiektu umożliwiając użytkownikowi ich zmianę.

PROP:Language — Numer języka stosowanego w OLE Automation lub metodzie OCX. Numer języka angielskiego (US English) to 0409H, numery innych języków mogą zostać określone w oparciu o dane zawarte w pliku WINNT.H Windows SDK. Odczyt-Zapis.

Funkcje Callback

Funkcje callback są standardową częścią programowania Windows w wielu językach programowania. Funkcja callback to procedura PROCEDURE, którą programista pisze do obsługi specyficznych sytuacji, które system operacyjny traktuje jako te, które są „sprawą" programisty. Funkcja callback jest wywoływana przez system operacyjny za każdym razem, gdy stwierdzi on zaistnienie takiej sytuacji. Funkcja callback nie występuje jako część logiki programu, jest od niej odseparowana i nie ma logicznego związku z innymi procedurami programu. Język Clarion for Windows nie wymusza na nas pisania własnych funkcji callback dla większości ogólnych, typowych zadań, czego wymagają inne języki programowania. Wynika to z tego, że biblioteka uruchomieniowa Clariona oraz pętla ACCEPT robi to za nas. Jednakże, ze względu na to, że kontrolki .OCX są pisane w innych językach programowania, wymagających funkcji callback, powinniśmy je utworzyć do obsługi zdarzeń i innych zagadnień programowych. Jako że metody klasy CLASS posiadają bezwarunkowy pierwszy parametr, nie mogą być używane w roli funkcji callback.

Istnieją trzy rodzaje funkcji callback, które możemy stworzyć dla kontrolki .OCX: procesor zdarzeń, kontroler edycji właściwości, manipulator zmiany właściwości. Możemy je nazywać tak, jak chcemy, ale mają one pewne specyficzne wymagania, co do parametrów:

Procesor zdarzeń kontrolki OCX

Prototyp procesora zdarzeń musi mieć postać:

```
OcxEventFuncName PROCEDURE(*SHORT,SIGNED,LONG),LONG
```

Parametry, które przekazuje do niego system operacyjny są następujące:

*SHORT	Parametr referencyjny służący do przekazania do następujących innych procedur biblioteki OCX: OCXGETPARAM, OCXGETPARAMCOUNT oraz OCXSETPARAM jako ich pierwszy parametr.
SIGNED	Numer pola dla kontrolki. Jest to ten sam numer, który jest reprezentowany przez etykietę ekwiwalentu pola kontrolki.
LONG	Numer zdarzenia .OCX. Ekwiwalenty dla pewnych predefiniowanych numerów zdarzeń są zawarte w pliku OCXEVENT.CLW.

Zwracana wartość LONG informuje system operacyjny, czy jest potrzebne dodatkowe przetwarzanie. Wartość zero (0) oznacza, że dodatkowe przetwarzanie jest konieczne (tak jak aktualizacja zmiennej USE, czy usunięcie zaznaczenia przycisku radio), podczas gdy dowolna inna liczba oznacza, że przetwarzanie się już zakończyło.

Przetwarzanie zdarzeń generowanych przez kontrolkę .OCX musi zachodzić bardzo szybko, gdyż dla niektórych z nich czas jest kryterium krytycznym. Z tego względu nie powinna mieć miejsca interakcja z użytkownikiem w ramach tej procedury (jak to jest w przypadku okien WINDOW, instrukcji ASK, czy procedur MESSAGE). Kod powinien przetwarzać tylko to, co jest wymagane, tak szybko jak jest to tylko możliwe (zazwyczaj oznacza to eliminację zdarzeń związanych z myszką).

Kontroler edycji właściwości kontrolki OCX

Prototyp kontrolera edycji właściwości musi mieć postać:

```
OcxPropEditFuncName PROCEDURE(SIGNED,STRING),LONG
```

Parametrami przekazywanymi do niego przez system operacyjny są:

SIGNED Numer pola dla kontrolki. Jest to ten sam numer, który jest reprezentowany przez etykietę ekwiwalentu pola kontrolki.

STRING Nazwa właściwości do edycji.

Zwracana wartość LONG określa, czy zezwolenie na edycję właściwości zostało przyznane funkcji callback. Jeśli procedura da w rezultacie zero (0), zezwolenie nie zostało udzielone i użytkownik nie ma możliwości edycji właściwości. Dowolna inna wartość, różna od zera (0), oznacza udzielenie zezwolenia I użytkownik ma możliwość edycji właściwości.

Manipulator zmiany właściwości kontrolki OCX

Prototyp manipulatora zmiany właściwości musi mieć postać:

```
OcxPropChangeProcName PROCEDURE(SIGNED,STRING)
```

Parametrami przekazywanymi do niego przez system operacyjny są:

SIGNED Numer pola dla kontrolki. Jest to ten sam numer, który jest reprezentowany przez etykietę ekwiwalentu pola kontrolki.

STRING Nazwa zmienianej właściwości.

Ta procedura jest wywoływana, gdy właściwość zostanie zmieniona.

Przykład:

```
! ten program wykorzystuje Calendar OCX dostarczany przez Microsoft wraz z Access95
! (jeden ze składników MS Office Professional for Windows 95).
  PROGRAM
  MAP
    INCLUDE('OCX.CLW')

EventFunc     PROCEDURE(*SHORT Reference,SIGNED OleControl,LONG CurrentEvent),LONG
PropChange    PROCEDURE(SIGNED OleControl,STRING CurrentProp)
PropEdit      PROCEDURE(SIGNED OleControl,STRING CurrentProp),LONG
  END
  INCLUDE('OCXEVENT.CLW')                    ! stałe wykorzystywane przez zdarzenia OCX
  INCLUDE('ERRORS.CLW')                      ! dołączenie kodów błędów

GlobalQue     QUEUE                          ! zdarzenie i zmiana wyświetlanej kolejki
F1              STRING(255)
              END

SaveDate      FILE,DRIVER('TopSpeed'),PRE(SAV),CREATE
Record          RECORD
DateField         STRING(10)
                END
              END

MainWin       WINDOW('OCX Demo'),AT(,,350,200),STATUS(-1,-1),SYSTEM,GRAY,MAX,RESIZE
                MENUBAR
                  MENU('&File')
                    ITEM('Save Date to File'),USE(?SaveObjectValue)
                    ITEM('Retrieve Saved Date'),USE(?GetObject)
                    ITEM('E&xit'),USE(?exit)
                  END
                  MENU('&Object')
                    ITEM('About Box'),USE(?AboutObject)
```

```
                    ITEM('Set Date to TODAY'),USE(?SetObjectValueToday)
                    ITEM('Set Date to 1st of Month'),USE(?SetObjectValueFirst)
                  END
                  ITEM('&Properties!'),USE(?ActiveObj)
                END
                LIST,AT(237,6,100,100),USE(?List1),HVSCROLL,FROM(GlobalQue)
                OLE,AT(5,10,200,150),USE(?OcxObject)
              END
CODE
  OPEN(SaveDate)
  IF ERRORCODE()                                    ! kontrola błędów przy otwarciu
    IF ERRORCODE() = NoFileErr                      ! jeśli plik nie istnieje
      CREATE(SaveDate)                              ! utwórz go
      IF ERRORCODE() THEN HALT(,ERROR()).
      OPEN(SaveDate)                                ! następnie otwórz do użycia
      IF ERRORCODE() THEN HALT(,ERROR()).
    ELSE
      HALT(,ERROR())
    END
  END
  OPEN(MainWin)
  ?OcxObject{PROP:Create} = 'MSACAL.MSACALCtrl.7'   ! kontrolka MS Access 95 Calendar OCX
  IF RECORDS(SaveDate)                              ! sprawdź istniejące zapisane rekordy
    SET(SaveDate)                                   ! i pobierz je
    NEXT(SaveDate)
    IF ERRORCODE() THEN STOP(ERROR()).
    POST(EVENT:Accepted,?GetObject)
  ELSE
    ADD(SaveDate)                                   ! lub dodaj jeden
    IF ERRORCODE() THEN STOP(ERROR()).
  END
  IF ?OcxObject{PROP:OLE}                           ! sprawdź obiekt OLE
    GlobalQue = 'An Object is in the OLE control'
    ADD(GlobalQue)
    IF ?OcxObject{PROP:Ctrl}                        ! przekonaj się, czy to jest OCX
      GlobalQue = 'It is an OCX Object'
      ADD(GlobalQue)
    END
  END
  DISPLAY
  OCXREGISTEREVENTPROC(?OcxObject,EventFunc)        ! zarejestruj funkcję callback obsługi zdarzeń
  OCXREGISTERPROPCHANGE(?OcxObject,PropChange)      ! zarejestruj funkcję callback obsługi
                                                    ! zmiany właściwości
  OCXREGISTERPROPEDIT(?OcxObject,PropEdit)          ! zarejestruj funkcję callback obsługi
                                                    ! edycji właściwości
  ?OcxObject{PROP:ReportException} = 1              ! włącz raportowanie błędów OCX
  ACCEPT
    CASE EVENT()
    OF EVENT:Accepted
      CASE FIELD()
      OF ?Exit
        POST(EVENT:CloseWindow)
      OF ?AboutObject
        ?OcxObject{'AboutBox'}                      ! wyświetl About Box kontrolki
      OF ?SetObjectValueToday
        ?OcxObject{'Value'} = FORMAT(TODAY(),@D1)   ! ustaw kontrolke na dzisiejszą datę
      OF ?SetObjectValueFirst
        ?OcxObject{'Value'} = MONTH(TODAY()) & '/1/' & SUB(YEAR(TODAY()),3,2)
      OF ?SaveObjectValue                           ! zapisz wartośc kontrolki w pliku
        SAV:DateField = ?OcxObject{'Value'}
        PUT(SaveDate)
        IF ERRORCODE() THEN STOP(ERROR()).
      OF ?GetObject                                 ! pobierz wartość dla kontrolki z pliku
        ?OcxObject{'Value'} = SAV:DateField
      OF ?ActiveObj
        ?OcxObject{PROP:DoVerb} = 0                 ! uaktywnij dialog właściwości kontrolki
```

```
      END
    END
  END

EventFunc   PROCEDURE(*SHORT Reference,SIGNED OleControl,LONG CurrentEvent)
Count         LONG                                          ! funkcja callback obsługi zdarzeń
Res           CSTRING(200)
Parm          CSTRING(30)

  CODE
    IF CurrentEvent <> OCXEVENT:MouseMove                   ! eliminuj zdarzenia myszki
      Res = 'Event: ' & OleControl{PROP:LastEventName}
      LOOP Count = 1 TO OCXGETPARAMCOUNT(Reference)         ! powtarzaj dla wszystkich parametrów
        Parm = OCXGETPARAM(Reference,Count)                 ! pobierając nazwę każdego parametru
        Res = CLIP(Res) & ' - ' & Parm                      ! i łącząc je ze sobą
      END
      GlobalQue = Res                                       ! przypisz do globalnej kolejki
      ADD(GlobalQue)                                        ! i dodaj jej element
      DISPLAY
    END
    RETURN(True)

PropChange  PROCEDURE(SIGNED OleControl,STRING CurrentProp)
                                                            ! funkcja callback zmiany właściwości
  CODE
    GlobalQue = 'PropChange: ' & CurrentProp & ' = ' & OleControl{CurrentProp}
    !Assign to a global QUEUE
    ADD(GlobalQue)                                          ! dodaj element do wyświetlania
    IF ERRORCODE() THEN STOP(ERROR()).

PropEdit    PROCEDURE(SIGNED OleControl,STRING CurrentProp)
                                                            ! funkcja callback edycji właściwości

  CODE
    IF MESSAGE('Allow?','Change',ICON:Question,BUTTON:Yes+BUTTON:No,BUTTON:Yes,1) |
      = BUTTON:Yes                                          ! sprawdź możliwość zmiany wartości
      RETURN(1)                                             ! pozwól na zmianę
    ELSE
      RETURN(0)                                             ! nie zezwalaj na zmianę
    END
```

Wywoływanie metod obiektu OLE

Zarówno w przypadku OLE Automation dla serwera OLE, jak i obiektów OCX/ActiveX, ich metody (procedury) wywołujemy w celu przeprowadzenia określonych akcji. jako że kontrolki OCX są sukcesorami OLE kontrolek VBX, większość producentów udostępnia kody przykładowe dla nich w oparciu o składnię języka Visual Basic (VB). Te, które są wykorzystywane w programach C++ zazwyczaj posiadają przykłady oparte na kodzie C++.

Zastosowanie tych przykładów w odpowiednim kodzie Clarion wymaga na ogół pewnej wiedzy na temat VB lub C++. W tym punkcie przedstawimy ogólne typy wywołania metod w przykładowym kodzie VB i sposoby ich przeniesienia do Clariona.

Przegląd składni metod

Do wywołania dowolnej metody OLE/OCX stosujemy składnię właściwości (property syntax). Określamy kontrolkę, której metoda dotyczy poprzez ekwiwalent etykiety pola, po nim umieszczamy nazwę metody zamkniętą w nawiasach klamrowych ({}). Kod przykładowy dostarczany z większością kontrolek OLE stosuje składnię VB/C++ „dot property" do określenia nazwy kontrolki i wywoływanej metody bądź właściwości. Na przykład, poniższy kod VB:

```
ControlName.AboutBox
```

w Clarionie przyjmie postać:

```
?Ole{'AboutBox'}
```

Ten kod powoduje wyświetlenie okienka dialogowego „About" dla kontrolki *ControlName*. Może się też zdarzyć, że ten kod VB będzie zapisany w postaci:

```
Form1.ControlName.AboutBox
```

Ta forma po prostu wskazuje dialog zawarty w obiekcie *ControlName*.

Odwołania do obiektu OLE/OCX odbywają się w kodzie Clariona zawsze poprzez etykietę ekwiwalentu pola kontrolki OLE, niezależnie od tego, jaką nazwę ma ta kontrolka w VB, a to z tego względu, że zarejestrowana nazwa obiektu jest określona w atrybucie CREATE lub OPEN kontrolki OLE. Dlatego wystarczy, że biblioteka uruchomieniowa Clarion otrzyma etykietę ekwiwalentu pola, a będzie dokładnie wiedziała, którego obiektu dotyczy dana referencja.

Translacja składni „With" VisualBasic-a

Wiele przykładów kodu dla OLE/OCX wykorzystuje strukturę VB *With ... End With* do powiązania wielu przypisań właściwości i (lub) wywołań metod do pojedynczego obiektu. W tym przypadku obiekt jest nazywany w instrukcji *With* i wszystkie przypisania właściwości oraz wywołania metod wewnątrz tej struktury rozpoczynają się znakiem kropki, po którym następuje nazwa ustawianej właściwości lub wywoływanej metody. Na przykład, poniższy kod VB:

```
With Form1.VtChart1
    'wyświetla wykres 3d z 8 kolumnami  i 8 wierszami
    .chartType = VtChChartType3dBar
    .columnCount = 8
    .rowCount = 8
    For column = 1 To 8
      For row = 1 To 8
        .column = column
        .row = row
```

```
        .Data = row * 10
      Next row
    Next column
    .ShowLegend = True
End With
```

jest przekształcany w Clarionie na kod:

```
! wyświetla wykres 3d z 8 kolumnami  i 8 wierszami
?Ole{'chartType'} = VtChChartType3dBar
?Ole{'columnCount'} = 8
?Ole{'rowCount'} = 8
LOOP column# = 1 TO 8
  LOOP row# = 1 TO 8
    ?Ole{'column'} = column#
    ?Ole{'row'} = row#
    ?Ole{'Data'} = row# * 10
  END
END
?Ole{'ShowLegend'} = True
```

Ponieważ w Clarionie nie występuje bezpośredni ekwiwalent dla struktury VB *With ... End With*, musimy umieszczać etykietę ekwiwalentu pola kontrolki OLE w każdym przypisaniu właściwości i wywołaniu metody. Pojedynczy cudzysłów (‘) w kodzie VB oznacza komentarz. VB umożliwia zagnieżdżanie struktur *With ... End With*, co może wymagać dokładnej analizy w celu określenia właściwej nazwy obiektu. Poniższy przykład ilustruje zagnieżdżenie struktur *With*:

```
With MyObject
    .Height = 100 ' Same as MyObject.Height = 100.
    .Caption = "Hello World" ' Same as MyObject.Caption = "Hello World".
    With .Font
      .Color = Red ' Same as MyObject.Font.Color = Red.
      .Bold = True ' Same as MyObject.Font.Bold = True.
    End With
End With
```

co w Clarionie przyjmuje postać:

```
?Ole{‘Height’} = 100              ! MyObject.Height = 100
?Ole{‘Caption’} = ‘Hello World’   ! MyObject.Caption = "Hello World"
?Ole{‘Font.Color’} = Red          ! MyObject.Font.Color = Red
?Ole{‘Font.Bold’} = True          ! MyObject.Font.Bold = True
```

Przekazywanie parametrów do metod OLE/OCX

Tak jak w Calrionie, w VB występują dwa sposoby przekazywania parametrów do procedur: przez wartość lub przez adres (referencję). Słowa kluczowe VB *ByVal* oraz *ByRef* identyfikują te dwie metody w kodzie VB. Te terminy oznaczają w VB to samo, co w Clarionie – przekazanie parametru przez wartość polega na przekazaniu kopii wartości zmiennej, podczas gdy przekazanie parametru przez referencję (w VB jest to sposób domyślny) powoduje przekazanie adresu zmiennej, tak że metoda może zmieniać jej wartość.

Stosowanie nawiasów

Składnia VB może, ale nie musi, używać nawiasów dla listy parametrów. Jeśli metoda VB nie zwraca rezultatu lub też, gdy on nas nie interesuje, parametry są przekazywane bez nawiasów, na przykład:

```
VtChart1.InsertColumns 6,3
```

Jeśli chcemy otrzymać rezultat, parametry umieszczamy w nawiasach:

```
ReturnValue = VtChart1.InsertColumns (6,3)
```

W składni Clariona parametry są zawsze ujęte w nawiasy. Tak więc oba przedstawione powyżej przykłady zostaną przekształcone odpowiednio na:

```
?Ole{'InsertColumns(6,3)'}
ReturnValue = ?Ole{'InsertColumns(6,3)'}
```

Przekazywanie parametrów przez wartość

Parametry w postaci wartości są przekazywane do obiektów OLE/OCX w postaci łańcuchów (wyjątkiem są parametry logiczne Boolean). Ponieważ obiekty OLE/OCX zakładają doprowadzenie do poprawności typów danych za pomocą mechanizmu VARIANT (podobnego do konwersji typów stosowanej w Clarionie), zapewnia to jak największą zgodność przy jak najmniejszym nakładzie pracy. Dowolny łańcuch wymagający podwójnego cudzysłowu (“) potrzebują włączenia dwóch cudzysłowów (””).

Parametry w postaci wartości mogą być przekazywane do obiektu OLE/OCX jako stałe lub zmienne. Poniżej przedstawiono przykłady przekazania parametrów w postaci stałych. Nie można umieszczać spacji w stałej, jeśli nie jest ona ujęta w podwójny cudzysłów (na przykład “Wartość ze spacjami”).

mamy do dyspozycji dwie drogi przekazania zmiennej Clarion do metody OLE/OCX poprzez wartość: poprzez konkatenację ze stałą łańcuchową która wywołuje metodę lub poprzez zastosowania BIND w odniesieniu do nazwy zmiennej I umieszczenia jej bezpośrednio w stałej łańcuchowej wywołującej metodę. Na przykład:

```
ColumnNumber = 6
NumberOfColumns = 3
?Ole{'InsertColumns(' & ColumnNumber & ',' & NumberOfColumns & ')'}
!Same as ?Ole{'InsertColumns(6,3)'}
```

Drugim sposobem przekazania zmiennych poprzez wartość jest ich bindowanie (BIND) i umieszczenie nazw w stałej łańcuchowej, na przykład:

```
BIND('ColumnNumber',ColumnNumber)
BIND('NumberOfColumns',NumberOfColumns)
?Ole{'InsertColumns(ColumnNumber,NumberOfColumns)'}
!Same as ?Ole{'InsertColumns(6,3)'}
```

Ta metoda czyni kod bardziej czytelnym, wymaga jednak zastosowania instrukcji BIND.

Przekazywanie parametrów przez adres (referencję)

Parametry w postaci referencji mogą być przekazywane do metod obiektu OLE/OCX tylko jako nazwane zmienne w stałej łańcuchowej. Dlatego musimy zastosować instrukcję BIND dla nazwy zmiennej I umieścić nazwę zmiennej bezpośrednio w stałej łańcuchowej wywołującej metodę wraz ze znakiem ampersand poprzedzającym nazwę zmiennej i sygnalizującym że jest ona przekazywana poprzez adres. Na przykład:

```
ColumnNumber = 6
NumberOfColumns = 3
BIND('ColumnNumber',ColumnNumber)
BIND('NumberOfColumns',NumberOfColumns)
?Ole{'InsertColumns(&ColumnNumber,&NumberOfColumns)'}
```

Parametry przekazywane przez adres są przekazywane do obiektów OLE/OCX jako typ danych zbindowanej zmiennej (z wyjątkiem parametrów logicznych Boolean). Zmienne są w rzeczywistości przekazywane jako tymczasowe zmienne łańcuchowe, dla których biblioteka Clariona automatycznie przeprowadza de-referencję, dzięki której dowolne wprowadzone w nich przez metodę OLE/OCX zmiany są przekazywane z powrotem do oryginalnych zmiennych.

Parametry logiczne

Parametry logiczne Boolean (1/0 lub True/False) są przekazywane albo przez wartość, albo przez adres. W tym pierwszym przypadku możemy przekazać stałą (1 lub 0, bądź słowa TRUE lub FALSE), na przykład:

```
?Ole{'ODBCConnect(&DataSource,1,&RetVal)'}
?Ole{'ODBCConnect(&DataSource,TRUE,&RetVal)'}
```

albo też przekazać nazwę zmiennej (po wcześniejszym wykonaniu dla niej BIND) wewnątrz wywołania "bool()", na przykład:

```
BoolParm = 1
BIND('BoolParm',BoolParm)
?Ole{'ODBCConnect(&DataSource,bool(BoolParm),&RetVal)'}
```

Bool() jest konstrukcją, która informuje analizator wyrażeń właściwości, że ma przekazać w danym przypadku wartość logiczną. Bool() jest prawidłowy tylko wewnątrz łańcucha wywołującego metodę OLE/OCX.

By przekazać parametr logiczny w postaci referencji, poprzedzamy po prostu nazwę zmiennej znakiem ampersand wewnątrz konstrukcji bool(), na przykład:

```
BIND('BoolParm',BoolParm)
?Ole{'ODBCConnect(&DataSource,bool(&BoolParm),&RetVal)'}
```

Parametry o określonej nazwie

W VB istnieją dwa sposoby na przekazywanie parametrów: poprzez pozycję lub w postaci „argumentów o określonej nazwie". Parametry pozycyjne wymagają, by w wywołaniu metody w określonej pozycji umieścić parametr lub pozostawić puste miejsce w celu określenia, że jest on pomijany. Ponieważ niektóre metody otrzymują dużą liczbę parametrów, może to spowodować powstanie długich łańcuchów z pustymi miejscami pomiędzy przecinkami, podczas gdy my chcemy na przykład przekazać tylko dwa parametry. VB rozwiązuje ten problem poprzez umożliwienie

programiście „nazwania” parametru. Dzięki temu jest możliwe wywoływania metody tylko z tymi parametrami, które chcemy jej przekazać.

Parametry o określonej nazwie nie są w sposób uniwersalny obsługiwane przez VB, tak więc producent OLE/OCX powinien tak napisać jego metody, by ich obsługa była możliwa. Plik pomocy OLE/OCX powinien określać, czy stosowanie “nazwanych” parametrów jest dopuszczalne. Jeśli nie, możemy zastosować VB Object Browser do określenia, czy tak jest.

Składnia VB dla parametrów o określonej nazwie stosuje := do przypisania wartości do nazwy parametru. Na przykład, dla poniższej instrukcji VB:

```
OpenIt(Name:=, [Exclusive]:=, [ReadOnly]:=, [Connect]:=)
```

możemy wywołać metodę stosując parametry pozycjonowane:

```
Db = OpenIt("MyFile",False,False,"ODBC;UID=Fred")
```

co w Clarionie przyjmie postać:

```
Db = ?Ole{‘OpenIt("MyFile",False,False,"ODBC;UID=Fred")’}
```

Tę samą metodę możemy wywołać w VB stosując parametry o określonej nazwie (znak podkreślenia oznacza kontynuację wiersza):

```
Db = OpenIt(Name:="MyFile",Exclusive:=False,ReadOnly:=False, _
Connect:="ODBC;UID=Fred")
```

co w Clarionie przyjmie postać:

```
Db = ?Ole{‘OpenIt(Name="MyFile",Exclusive=False,ReadOnly=False, ‘ & |
        ‘Connect="ODBC;UID=Fred")’}
```

możemy też przekazać parametry w VB w innym porządku:

```
Db = OpenIt(Connect:="ODBC;UID=Fred", _
Name:="MyFile", _
ReadOnly:=False, _
Exclusive:=False")
```

co w Clarionie będzie wyglądało tak:

```
Db = ?Ole{‘OpenIt(Connect="ODBC;UID=Fred",Name="MyFile",’ & |
        ‘Exclusive=False,ReadOnly=False‘)’}
```

Procedury biblioteki OCX

OCXREGISTERPROPEDIT (instaluje kontroler edycji właściwości)

OCXREGISTERPROPEDIT(*control* , *procedure*)

OCXREGISTERPROPEDIT	Instaluje funkcję callback pełniącą rolę kontrolera edycji właściwości.
control	Wyrażenie całkowite zawierające numer pola lub etykietę ekwiwalentu pola dla kontrolki OLE, której dotyczy działanie.
procedure	Etykieta funkcji callback dla kontrolki *control*.

OCXREGISTERPROPEDIT Instaluje funkcję callback *procedure* pełniącą rolę kontrolera edycji właściwości dla kontrolki *control*. Funkcja callback *procedure* kontroluje możliwość edycji właściwości kontrolki *control* umożliwiając ją lub nie.

Przykład:

```
OCXREGISTERPROPEDIT(?OleControl,CallbackFunc)
```

Porównaj: Funkcje callback

OCXREGISTERPROPCHANGE (instaluje manipulator zmian właściwości)

OCXREGISTERPROPCHANGE(*control* , *procedure*)

OCXREGISTERPROPCHANGE	Instaluje funkcję callback pełniącą rolę manipulatora zmian właściwości.
control	Wyrażenie całkowite zawierające numer pola lub etykietę ekwiwalentu pola dla kontrolki OLE, której dotyczy działanie.
procedure	Etykieta funkcji callback dla kontrolki *control*.

OCXREGISTERPROPCHANGE instaluje funkcję callback *procedure* pełniącą rolę manipulatora zmian właściwości dla kontrolki *control*. Jest ona wywoływana wtedy, gdy właściwości kontrolki *control* ulegną zmianie.

Przykład:

```
OCXREGISTERPROPCHANGE(?OleControl,CallbackProc)
```

Porównaj: Funkcje callback

OCXREGISTEREVENTPROC (instaluje procesor zdarzeń)

OCXREGISTEREVENTPROC(*control* , *procedure*)

OCXREGISTEREVENTPROC	Instaluje funkcję callback pełniącą rolę procesora zdarzeń.
control	Wyrażenie całkowite zawierające numer pola lub etykietę ekwiwalentu pola dla kontrolki OLE, której dotyczy działanie.
procedure	Etykieta funkcji callback dla kontrolki *control*.

OCXREGISTEREVENTPROC instaluje funkcję callback *procedure* pełniącą rolę procesora zdarzeń dla kontrolki *control*. Jest ona wywoływana w momencie przesłania przez system operacyjny dowolnego zdarzenia do kontrolki *control*.

Przykład:

```
OCXREGISTEREVENTPROC(?OleControl,CallbackProc)
```

Porównaj: Funkcje callback

OCXUNREGISTERPROPEDIT (de-instaluje kontroler edycji właściwości)

OCXUNREGISTERPROPEDIT(*control*)

OCXUNREGISTERPROPEDIT	De-instaluje funkcję callback pełniącą rolę kontrolera edycji właściwości.
control	Wyrażenie całkowite zawierające numer pola lub etykietę ekwiwalentu pola dla kontrolki OLE, której dotyczy działanie.

OCXUNREGISTERPROPEDIT de-instaluje funkcję callback *procedure* pełniącą rolę kontrolera edycji właściwości dla kontrolki *control*.

Przykład:

```
OCXUNREGISTERPROPEDIT(?OleControl)
```

Porównaj: Funkcje callback

OCXUNREGISTERPROPCHANGE (de-instaluje manipulator zmian właściwości)

OCXUNREGISTERPROPCHANGE(*control*)

OCXUNREGISTERPROPCHANGE	De-instaluje funkcję callback pełniącą rolę manipulatora zmian właściwości.
control	Wyrażenie całkowite zawierające numer pola lub etykietę ekwiwalentu pola dla kontrolki OLE, której dotyczy działanie.

OCXUNREGISTERPROPCHANGE de-instaluje funkcję callback *procedure* pełniącą rolę manipulatora zmian właściwości dla kontrolki *control.*

Przykład:

```
OCXUNREGISTERPROPCHANGE(?OleControl)
```

Porównaj: Funkcje callback

OCXUNREGISTEREVENTPROC (de-instaluje procesor zdarzeń)

OCXUNREGISTEREVENTPROC(*control*)

OCXUNREGISTEREVENTPROC	De-instaluje funkcję callback pełniącą rolę procesora zdarzeń.
control	Wyrażenie całkowite zawierające numer pola lub etykietę ekwiwalentu pola dla kontrolki OLE, której dotyczy działanie.

OCXUNREGISTEREVENTPROC de-instaluje funkcję callback *procedure* pełniącą rolę procesora zdarzeń dla kontrolki *control.*

Przykład:

```
OCXUNREGISTEREVENTPROC(?OleControl)
```

Porównaj: Funkcje callback

OCXGETPARAMCOUNT (zwraca liczbę parametrów dla bieżącego zdarzenia)

OCXGETPARAMCOUNT(*reference*)

OCXGETPARAMCOUNT Zwraca liczbę parametrów związanych z bieżącym zdarzeniem OCX.

reference Etykieta pierwszego parametru funkcji callback przetwarzającej zdarzenia.

OCXGETPARAMCOUNT zwraca liczbę parametrów związanych z bieżącym zdarzeniem OCX. Procedura ta jest właściwa tylko wtedy, gdy jest aktywna funkcja callback przetwarzająca zdarzenia .OCX.

Typ rezultatu: USHORT

Przykład:

```
OEvent    PROCEDURE(Reference,OleControl,CurrentEvent)      ! procedura callback przetwarzania zdarzeń
Count       LONG
Res         CSTRING(200)
Parm        CSTRING(30)
  CODE
    IF CurrentEvent <> OCXEVENT:MouseMove                   ! eliminuje zdarzenia myszki
      Res = 'Control ‘ & OleControl & ' Event ' & OleControl{PROP:LastEventName} & ‘:‘
      LOOP Count = 1 TO OCXGETPARAMCOUNT(Reference)         ! powtarza dla wszystkich parametrów
        Parm = OCXGETPARAM(Reference,Count)                 ! pobiera nazwę każdego parametru
        Res = CLIP(Res) & ' ' & Parm                        ! i łączy je ze sobą
      END
      GlobalQue = Res                                       ! przypisuje do kolejki globalnej
      ADD(GlobalQue)                                        ! dodaje w celu późniejszego wyświetlenia
    END                                                     ! wszystkich zdarzeń OCX i ich
    RETURN(True)                                            ! parametrów
```

Porównaj: Funkcje callback, OCXGETPARAM

OCXGETPARAM (zwraca łańcuch parametru bieżącego zdarzenia)

OCXGETPARAM(*reference* , *number*)

OCXGETPARAM Zwraca wartość parametru związanego z bieżącym zdarzeniem OCX.

reference Etykieta pierwszego parametru funkcji callback przetwarzającej zdarzenia.

number Liczba parametrów do wczytania.

OCXGETPARAM zwraca wartość określonej przez *number* liczby parametrów związanych z bieżącym zdarzeniem OCX. Procedura ta jest właściwa tylko wtedy, gdy jest aktywna funkcja callback przetwarzająca zdarzenia .OCX.

Typ rezultatu: STRING

Przykład:

```
OEvent    PROCEDURE(Reference,OleControl,CurrentEvent)      ! procedura callback przetwarzania zdarzeń
Count       LONG
Res         CSTRING(200)
Parm        CSTRING(30)

  CODE
    IF CurrentEvent <> OCXEVENT:MouseMove                   ! eliminuje zdarzenia myszki
      Res = 'Control ‘ & OleControl & ' Event ' & OleControl{PROP:LastEventName} & ‘:‘
      LOOP Count = 1 TO OCXGETPARAMCOUNT(Reference)         ! powtarza dla wszystkich parametrów
        Parm = OCXGETPARAM(Reference,Count)                 ! pobiera nazwę każdego parametru
        Res = CLIP(Res) & ' ' & Parm                        ! i łączy je ze sobą
      END
      GlobalQue = Res                                       ! przypisuje do kolejki globalnej
      ADD(GlobalQue)                                        ! dodaje w celu późniejszego
    END                                                     ! wszystkich zdarzeń OCX i ich
    RETURN(True)                                            ! parametrów
```

Porównaj: Funkcje callback,OCXSETPARAM, OCXGETPARAMCOUNT

OCXSETPARAM (ustawia łańcuch parametru bieżącego zdarzenia)

OCXSETPARAM(*reference* , *number* ,*value*)

OCXSETPARAM	Ustawia wartość parametru powiązanego z bieżącym zdarzeniem OCX.
reference	Etykieta pierwszego parametru funkcji callback przetwarzającej zdarzenia.
number	Liczba parametrów do ustawienia.
value	Stała lub zmienna łańcuchowa zawierająca wartość do ustawienia.

OCXSETPARAM ustawia wartość określonej przez *number* liczby parametrów związanych z bieżącym zdarzeniem. Jest to dopuszczalne tylko dla parametrów przekazanych przez adres (sprawdź dokumentację kontrolki .OCX w celu określenia prawidłowego zestawu parametrów do ustawienia). Jeśli modyfikacja nie jest dopuszczalna, jest ignorowany. Procedura ta jest właściwa tylko wtedy, gdy jest aktywna funkcja callback przetwarzająca zdarzenia .OCX.

Przykład:

```
OEvent   PROCEDURE(Reference,OleControl,CurrentEvent)      ! procedura callback przetwarzania zdarzeń
Count      LONG
Res        CSTRING(200)
Parm       CSTRING(30)

  CODE
    IF CurrentEvent <> OCXEVENT:MouseMove                  ! eliminuje zdarzenia myszki
      Res = 'Control ‘ & OleControl & ' Event ' & OleControl{PROP:LastEventName} & ‘:‘
      LOOP Count = 1 TO OCXGETPARAMCOUNT(Reference)        ! powtarza dla wszystkich parametrów
        Parm = OCXGETPARAM(Reference,Count)                ! pobiera nazwę każdego parametru
        Res = CLIP(Res) & ' ' & Parm                       ! i łączy je ze sobą
        OCXSETPARAM(Reference,1,’1’)                       ! zmienia wartość parametru
      END
      GlobalQue = Res                                      ! przypisuje do kolejki globalnej
      ADD(GlobalQue)                                       ! dodaje w celu późniejszego wyświetlenia
    END                                                    ! wszystkich zdarzeń OCX i ich
    RETURN(True)                                           ! parametrów
```

Porównaj: Funkcje callback, OCXGETPARAM

OCXLOADIMAGE (zwraca obiekt graficzny)

OCXLOADIMAGE(*name*)

OCXLOADIMAGE	Zwraca obiekt graficzny.
name	Wyrażenie łańcuchowe zawierające nazwę pliku lub zasobu do załadowania.

OCXLOADIMAGE zwraca obiekt graficzny. Obiekt graficzny może być przypisany do dowolnej kontrolki wykorzystującej tego typu obiekt (np. kontrolka imagelist VB)

Typ rezultatu: STRING

Przykład:

```
?imagelist{'ListImages.Add(,,' & OCXLOADIMAGE('CLOCK.BMP') & ')'}
! dodaje grafikę do kontrolki ImageList
```

DODATEK B - ZDARZENIA

Zdarzenia

W programach Clarion przeznaczonych dla Windows, większość komunikatów przekazywanych przez system Windows jest obsługiwana wewnętrznie przez procesor zdarzeń ACCEPT. Są to ogólne zdarzenia obsługiwane przez bibliotekę uruchomieniową, takie jak np. odrysowanie ekranu. Do kodu Clariona są przekazywane przez ACCEPT tylko te zdarzenia, które wymagają obsługi przez program. Efektem jest ułatwienie pracy programisty poprzez eliminację pisania kodu obsługującego podstawowe funkcje niskiego poziomu i możliwość skupienia się na implementacji funkcji wysokiego poziomu. Oczywiście jest również możliwe samodzielne pisanie kodu obsługującego niski poziom, ale tylko wtedy, gdy jest to naprawdę niezbędne. Najlepszym źródłem informacji na ten temat jest podręcznik *Programming Windows* Charles'a Petzold'a opublikowany przez Microsoft Press

Istnieją dwa typy zdarzeń przekazywanych do programu przez ACCEPT: **specyficzne dla pól** oraz **niezależne od pól**. Poniżej zostały przedstawione wykazy ekwiwalentów EQUATE dla poszczególnych zdarzeń, zdefiniowanych w pliku EQUATES.CLW.

Zdarzenia niezależne od pól

Zdarzenie **niezależne od pola** nie jest związane z żadną kontrolką, ale wymaga pewnej reakcji programuj (na przykład: zamknięcia okna, wyjścia z programu, zmiany wykonywanego wątku). Większość tych zdarzeń powoduje przejście systemu w stan modalny, na czas ich przetwarzania, a to z tego względu, że wymagają odpowiedzi zanim wykonywanie programu będzie kontynuowane.

EVENT:AlertKey

Użytkownik nacisnął klawisz skrótu wskazany przez atrybut ALRT (lub instrukcję ALERT) okna. Jest to zdarzenie, które wywołuje akcję oczekiwaną przez użytkownika po wciśnięciu klawisza skrótu.

EVENT:BuildDone

Instrukcja BUILD lub PACK zakończyła odbudowę indeksów. Jest to zdarzenie, które wywołuje akcję związaną z porządkowaniem po operacji odbudowy kluczy. Jeśli użytkownik anuluje BUILD, jest ustawiany kod błędu ERRORCODE 93.

EVENT:BuildFile

Instrukcja BUILD lub PACK odtwarza indeksy pliku. Jest to zdarzenie, które pozwala na informowanie użytkownika o postępie przeprowadzanej operacji.

EVENT:BuildKey

Instrukcja BUILD lub PACK odtwarza indeks. Jest to zdarzenie, które na informowanie użytkownika o postępie przeprowadzanej operacji.

EVENT:CloseDown

Aplikacja jest zamykana. Wysłanie (POST) tego zdarzenia zamyka aplikację. Jest to zdarzenie, które pozwala na wykonanie dodatkowego kodu związanego z zamykaniem aplikacji.

EVENT:CloseWindow

Okno jest zamykane. Wysłanie (POST) tego zdarzenia powoduje zamknięcie okna. Jest to zdarzenie, które pozwala na wykonanie dodatkowego kodu związanego z zamykaniem okna.

EVENT:Completed

Tryb AcceptAll (non-stop) zakończył przetwarzanie wszystkich kontrolek okna. Jest to zdarzenie, które pozwala na wykonanie kodu sprawdzającego poprawność wszystkich pól wprowadzania danych dla kontrolek okna i bezpieczny zapis na dysk.

EVENT:DDEadvise

Klient prosi o ciągłą aktualizację elementu danych z danego serwera DDE Clarion. Jest to zdarzenie, które pozwala na wykonanie DDEWRITE w celu dostarczenia danych do klienta za każdym razem, gdy ulegną one zmianie.

EVENT:DDEclosed

Serwer DDE kończy połączenie DDE do danego klienta DDE Clarion.

EVENT:DDEdata

Serwer DDE dostarcza zaktualizowany element danych do danego klienta DDE Clarion.

EVENT:DDEexecute

Klient przesłał polecenie do danego serwera DDE Clarion (jeśli klient jest inną aplikacją Clarion, wykonuje instrukcję DDEEXECUTE). Jest to zdarzenie, które pozwala na określenie akcji, jakiej wymaga klient i wykonanie jej, następnie wykonania instrukcji CYCLE w celu zasygnalizowania pozytywnego potwierdzenia do klienta, który przesłał polecenie.

EVENT:DDEpoke

Klient przesłał nie oczekiwane dane do danego serwera DDE Clarion. Jest to zdarzenie, które pozwala na określenie, jakie dane przesłał klient i gdzie je umieścić, następnie na wykonanie instrukcji CYCLE w celu zasygnalizowania pozytywnego potwierdzenia do klienta, który przesłał dane.

EVENT:DDErequest

Klient wystąpił o element danych do serwera DDE Clarion. Jest to zdarzenie, które pozwala na wykonanie DDEWRITE w celu jednorazowego dostarczenia danej do klienta.

EVENT:Docked

Dokowalne okno narzędziowe zostało zadokowane lub zmieniła się jego pozycja dokowania.

EVENT:Undocked

Dokowalne okno narzędziowe zostało oddokowane.

EVENT:GainFocus

Okno otrzymało aktywność wprowadzania od innego wątku. Jest to zdarzenie, które umożliwia odtworzenie dowolnych danych zachowanych przy EVENT:LoseFocus. W trakcie tego zdarzenia system jest modalny.

EVENT:Iconize

Użytkownik minimalizuje okno posiadające atrybut IMM. Jeśli w kodzie występuje instrukcja CYCLE w celu obsługi tego zdarzenia, EVENT:Iconized nie jest generowane, a jego akcja jest odrzucana. Jest to zdarzenie, które pozwala na uniemożliwienie użytkownikowi zminimalizowania okna. W trakcie tego zdarzenia system jest modalny.

EVENT:Iconized

Użytkownik zminimalizował okno posiadające atrybut IMM. Jest to zdarzenie, które pozwala na uporządkowanie elementów zależnych od rozmiaru ekranu.

EVENT:LoseFocus

Okno traci aktywność wprowadzania na rzecz innego wątku. Jest to zdarzenie, które umożliwia zachowanie dowolnych danych, dla których istnieje ryzyko, że zostaną zmienione przez ten inny wątek. W trakcie tego zdarzenia system jest modalny.

EVENT:Maximize

Użytkownik maksymalizuje okno posiadające atrybut IMM. Jeśli w kodzie występuje instrukcja CYCLE w celu obsługi tego zdarzenia, EVENT:Maximized nie jest generowane, a jego akcja jest odrzucana. Jest to zdarzenie, które pozwala na uniemożliwienie użytkownikowi zmaksymalizowania okna. W trakcie tego zdarzenia system jest modalny.

EVENT:Maximized

Użytkownik zmaksymalizował okno posiadające atrybut IMM. Jest to zdarzenie, które pozwala na uporządkowanie elementów zależnych od rozmiaru ekranu.

EVENT:Move

Użytkownik przesuwa okno posiadające atrybut IMM. Jeśli w kodzie występuje instrukcja CYCLE w celu obsługi tego zdarzenia, EVENT:Moved nie jest generowane, a jego akcja jest odrzucana. Jest to zdarzenie, które pozwala na uniemożliwienie użytkownikowi przesuwania okna. W trakcie tego zdarzenia system jest modalny.

EVENT:Moved

Użytkownik przesunął okno posiadające atrybut IMM. Jest to zdarzenie, które pozwala na uporządkowanie elementów zależnych od położenia na ekranie.

EVENT:OpenWindow

Okno jest otwierane. Jest to zdarzenie, które pozwala na wykonanie kodu związanego z inicjowaniem okna.

EVENT:PreAlertKey

Użytkownik nacisnął klawisz skrótu wskazany przez atrybut ALRT (lub instrukcję ALERT) okna. Jest to zdarzenie, które wywołuje akcję oczekiwaną przez użytkownika po wciśnięciu klawisza skrótu. Jeśli w kodzie występuje instrukcja CYCLE w celu obsługi tego zdarzenia, standardowa akcja biblioteki uruchomieniowej dla tego klawisza jest wykonywana przed wygenerowaniem EVENT:AlertKey. Jest to zdarzenie, które pozwala na określenie, czy standardowa akcja biblioteki dla klawisza ma być wykonywana, czy też nie, w uzupełnieniu do kodu określonego dla EVENT:AlertKey. W trakcie tego zdarzenia system jest modalny.

EVENT:Restore

Użytkownik przywraca poprzedni rozmiar okna posiadającego atrybut IMM. Jeśli w kodzie występuje instrukcja CYCLE w celu obsługi tego zdarzenia, EVENT:Restored nie jest generowane, a jego akcja jest odrzucana. Jest to zdarzenie, które pozwala na uniemożliwienie użytkownikowi przywrócenie rozmiaru okna. W trakcie tego zdarzenia system jest modalny.

EVENT:Restored

Użytkownik przywrócił poprzedni rozmiar okna posiadającego atrybut IMM. Jest to zdarzenie, które pozwala na uporządkowanie elementów zależnych od rozmiaru ekranu.

EVENT:Resume

Okno nadal posiada aktywność wprowadzania i przywrócone sterowanie od EVENT:Suspend. W trakcie tego zdarzenia system jest modalny.

EVENT:Size

Użytkownik zmienia rozmiar okna posiadającego atrybut IMM. Jeśli w kodzie występuje instrukcja CYCLE w celu obsługi tego zdarzenia, EVENT:Sized nie jest generowane, a jego akcja jest odrzucana. Jest to zdarzenie, które pozwala na uniemożliwienie użytkownikowi zmiany rozmiaru okna. W trakcie tego zdarzenia system jest modalny.

EVENT:Sized

Użytkownik zmienił rozmiar okna posiadającego atrybut IMM. Jest to zdarzenie, które pozwala na uporządkowanie elementów zależnych od rozmiaru ekranu.

EVENT:Suspend

Okno nadal posiada aktywność wprowadzania ale przekazuje sterowanie do innego wątku na czas obsługi zdarzenia zegarowego. W trakcie tego zdarzenia system jest modalny.

EVENT:Timer

Atrybut TIMER został uzbrojony. Jest to zdarzenie, które on wykonuje dowolne akcje okresowe, takie jak wyświetlenie zegara, czy przetwarzanie rekordów w tle na rzecz procesów wsadowych, czy raportów.

Zdarzenia specyficzne dla pól

Zdarzenie **specyficzne dla pola** zachodzi wtedy, gdy użytkownik wykona działanie związane z kontrolką wymagające określonej akcji programu.

EVENT:Accepted

Użytkownik wprowadził dane lub dokonał selekcji, następnie wcisnął klawisz TAB lub kliknął myszką inną kontrolkę. Jest to zdarzenie, które pozwala na wykonanie kodu kontroli poprawności wprowadzonych danych.

EVENT:AlertKey

Użytkownik nacisnął klawisz skrótu wskazany przez atrybut ALRT kontrolki. Jest to zdarzenie, które wywołuje akcję oczekiwaną przez użytkownika po wciśnięciu klawisza skrótu.

EVENT:ColumnResize

Użytkownik zmienił rozmiar kolumny kontrolki LIST posiadającej M w łańcuchu atrybutu FORMAT.

EVENT:Contracted

Użytkownik kliknął przycisk zwijania gałęzi drzewa w kontrolce LIST posiadającej T w łańcuchu atrybutu FORMAT.

EVENT:Contracting

Użytkownik kliknął przycisk zwijania gałęzi drzewa w kontrolce LIST posiadającej T w łańcuchu atrybutu FORMAT. Jeśli w kodzie występuje instrukcja CYCLE w celu obsługi tego zdarzenia, EVENT:Contracted nie jest generowane, a jego akcja jest odrzucana. W trakcie tego zdarzenia system jest modalny.

EVENT:Drag

Użytkownik zwolnił przycisk myszki nad poprawnym celem operacji drag-and-drop. Jest to zdarzenie, które jest przesyłane do kontrolki, z której użytkownik "przeciąga" informacje. Za pomocą obsługi tego zdarzenia możemy zrealizować przekazanie danych ze źródła do celu operacji drag-and-drop.

EVENT:Dragging

Użytkownik przeciąga myszką z kontrolki posiadającej atrybut DRAGID a wskaźnik myszki znajduje się nad potencjalnym celem operacji drag and drop. Jest to zdarzenie, które jest wysyłane do kontrolki, z której przeciągane są dane. Jest to zdarzenie pozwalające na zmianę kształtu kursora w celu poinformowania użytkownika, że wskaźnik myszki znajduje się nad poprawnym celem operacji drag and drop.

EVENT:Drop

Użytkownik zwolnił przycisk myszki nad poprawnym celem operacji drag-and-drop. Jest to zdarzenie, które jest przesyłane do kontrolki stanowiącej cel operacji drag and drop. Poprzez obsługę tego zdarzenia możemy zarejestrować dane przekazane ze źródła.

EVENT:DroppedDown

Została rozwinięta lista kontrolki LIST lub COMBO posiadającej atrybut DROP. Jest to zdarzenie, które pozwala na ukrycie innych kontrolek pozostających na ekranie, które niepotrzebnie mogą odciągać uwagę użytkownika.

EVENT:DroppingDown

Użytkownik wcisnął przycisk ze strzałką w dół w kontrolce LIST lub COMBO posiadającej atrybut DROP. Jest to zdarzenie, które pozwala na załadowanie rekordów do listy rozwijalnej.

EVENT:Expanded

Użytkownik kliknął przycisk rozwijania gałęzi drzewa w kontrolce LIST posiadającej T w łańcuchu atrybutu FORMAT.

EVENT:Expanding

Użytkownik kliknął przycisk zwijania gałęzi drzewa w kontrolce LIST posiadającej T w łańcuchu atrybutu FORMAT. Jeśli w kodzie występuje instrukcja CYCLE w celu obsługi tego zdarzenia, EVENT:Expanded nie jest generowane, a jego akcja jest odrzucana. W trakcie tego zdarzenia system jest modalny.

EVENT:Locate

W kontrolce LIST posiadającej atrybut VCR, użytkownik kliknął przycisk lokatora (?) VCR. Jest to zdarzenie, które pozwala na pokazanie (unhide) pola wprowadzania lokatora, o ile pozostaje ono ukryte.

EVENT:MouseDown

Synonim zdarzenia EVENT:Accepted dla regionu REGION posiadającego atrybut IMM (tylko dla celów czytelności kodu).

EVENT:MouseIn

Kursor myszki pojawił się w obszarze regionu REGION posiadającego atrybut IMM.

EVENT:MouseMove

Kursor myszki przesuwa się w obszarze regionu REGION posiadającego atrybut IMM.

EVENT:MouseOut

Kursor myszki opuścił obszar regionu REGION posiadającego atrybut IMM.

EVENT:MouseUp

Przycisk myszki został zwolniony w obszarze regionu REGION posiadającego atrybut IMM.

EVENT:NewSelection

Zdarzenie to jest generowane dla kontrolek LIST, COMBO, SHEET lub SPIN wtedy, gdy zostanie w nich wybrany inny element. W kontrolce ENTRY posiadającej atrybut IMM, zdarzenie to jest generowane za każdym razem, gdy zmieni się zawartość kontrolki lub nastąpi ruch kursora. Jest to zdarzenie, które pozwala na wykonanie kodu synchronizującego inne kontrolki z aktualnie podświetlonym rekordem w liście albo też na stwierdzenie, czy użytkownik wprowadził dopuszczalne dane w polu ENTRY.

EVENT:PageDown

Użytkownik wcisnął klawisz PGDN w kontrolce LIST lub COMBO posiadającej atrybut IMM. Jest to zdarzenie, które pozwala na pobranie następnej "strony" rekordów.

EVENT:PageUp

Użytkownik wcisnął klawisz PGUP w kontrolce LIST lub COMBO posiadającej atrybut IMM. Jest to zdarzenie, które pozwala na pobranie poprzedniej "strony" rekordów.

EVENT:PreAlertKey

Użytkownik nacisnął klawisz skrótu wskazany przez atrybut ALRT kontrolki. Jeśli w kodzie występuje instrukcja CYCLE w celu obsługi tego zdarzenia, standardowa akcja biblioteki uruchomieniowej dla tego klawisza jest wykonywana przed wygenerowaniem EVENT:AlertKey. Jest to zdarzenie, które pozwala na określenie, czy standardowa akcja biblioteki dla klawisza ma być wykonywana, czy też nie, w uzupełnieniu do kodu określonego dla EVENT:AlertKey. W trakcie tego zdarzenia system jest modalny.

EVENT:Rejected

Użytkownik wprowadził nieprawidłową wartość dla danego wzorca wprowadzania lub wartość leżącą poza zakresem kontrolki SPIN. Procedura REJECTCODE określa powód odrzucenia danych wprowadzonych przez użytkownika, można też zastosować właściwość PROP:ScreenText do pobrania ich z ekranu. Jest to zdarzenie, które umożliwia poinformowanie użytkownika o przyczynach odrzucenia wprowadzonych danych.

EVENT:ScrollBottom

Użytkownik wcisnął klawisz CTRL+PGDN w kontrolce LIST lub COMBO posiadającej atrybut IMM. Jest to zdarzenie, które pozwala na pobranie ostatniej "strony" rekordów.

EVENT:ScrollDown

Użytkownik przesuwa w dół podświetlenie w kontrolce LIST lub COMBO posiadającej atrybut IMM. Jest to zdarzenie, które pozwala na pobranie następnego rekordu lub na samo przesuniecie podświetlenia, gdy pobranie rekordu nie jest konieczne.

EVENT:ScrollDrag

Użytkownik przesuwał suwak w pasku przewijania kontrolki LIST lub COMBO posiadającej atrybut IMM i właśnie zwolnił przycisk myszki. Jest to zdarzenie, które pozwala na dynamiczne przewijanie wyświetlanych rekordów w oparciu o wartość właściwości PROP:VScrollPos.

EVENT:ScrollTop

Użytkownik wcisnął klawisz CTRL+PGUP w kontrolce LIST lub COMBO posiadającej atrybut IMM. Jest to zdarzenie, które pozwala na pobranie pierwszej “strony” rekordów.

EVENT:ScrollTrack

Użytkownik przesuwa suwak w pasku przewijania kontrolki LIST lub COMBO posiadającej atrybut IMM. Jest to zdarzenie, które pozwala na dynamiczne przewijanie wyświetlanych rekordów w oparciu o wartość właściwości PROP:VScrollPos.

EVENT:ScrollUp

Użytkownik przesuwa w górę podświetlenie w kontrolce LIST lub COMBO posiadającej atrybut IMM. Jest to zdarzenie, które pozwala na pobranie poprzedniego rekordu lub na samo przesuniecie podświetlenia, gdy pobranie rekordu nie jest konieczne.

EVENT:Selected

Kontrolka otrzymała aktywność wprowadzania. Jest to zdarzenie, które pozwala na wykonanie kodu inicjującego.

EVENT:TabChanging

Aktywność jest przekazywana innej zakładce kontrolki SHEET. Jest to zdarzenie, które pozwala na wykonanie kodu dla zakładki, którą opuszczamy.

EVENT:VBXevent

Wystąpiło zdarzenie VBX w kontrolce CUSTOM. Jest to zdarzenie, które pozwala na odczytanie z właściwości PROP:VBXEvent oraz PROP:VBXEventArg zdarzenia VBX i jego parametrów.

DODATEK C – WŁAŚCIWOŚCI RUNTIME

Właściwości runtime

PROP:AcceptAll

Właściwość okna WINDOW dająca wartość jeden (1), gdy tryb AcceptAll jest aktywny, bądź zero (0) – w przeciwnym wypadku. Może ona być także stosowana do przełączania stanu aktywności trybu AcceptAll (non-stop). SELECT bez parametrów zazwyczaj inicjuje tryb AcceptAll. Jest to tryb edycji pola, w którym każda kontrolka okna jest przetwarzana poprzez generowanie dla niej zdarzenia EVENT:Accepted. Pozwala to na wykonanie kodu sprawdzającego poprawność danych dla wszystkich kontrolek, włączając w to i te, których użytkownik nie uaktywniał. Tryb AcceptAll kończy się bezpośrednio po spełnieniu jednego z warunków:

SELECT(?)

Window{PROP:AcceptAll} = 0

Wartość kontrolki z atrybutem REQ jest pusta lub zerowa.

Instrukcja SELECT(?) selekcjonuje do edycji przez użytkownika tę sama kontrolkę. Ten kod zazwyczaj jest stosowany w sytuacji, gdy użytkownik wprowadził do kontrolki niewłaściwą wartość i musi te operację wykonać ponownie. Instrukcja Window{PROP:AcceptAll} = 0 powoduje wyłączenie trybu AcceptAll. Przypisanie odpowiedniej wartości do tej właściwości umożliwia zainicjowanie lub zakończenie trybu AcceptAll. Gdy kontrolka z atrybutem REQ ma wartość pusta lub zerową, tryb AcceptAll kończy się pozostawiając ją podświetloną i umożliwiającą użytkownikowi wprowadzenie wartości, bez przetwarzania żadnego innego pola. Gdy wszystkie kontrolki zostaną przetworzone pomyślnie, do okna jest przesyłane zdarzenie EVENT:Completed.

Przykład:

```
Screen      WINDOW,PRE(Scr)
              ENTRY(@N3),USE(Ctl:Code)
              ENTRY(@S30),USE(Ctl:Name),REQ
              BUTTON('OK'),USE(?OkButton),KEY(EnterKey)
              BUTTON('Cancel'),USE(?CanxButton),KEY(EscKey)
            END
  CODE
    OPEN(Screen)
    ACCEPT
      IF EVENT() = EVENT:Completed THEN BREAK.      ! Tryb AcceptAll zakończył się
      CASE ACCEPTED()
      OF ?Ctl:Code
        IF Ctl:Code > 150                           ! Jeśli wprowadzona dana jest niewłaściwa
          BEEP                                      ! ostrzeżenie dla użytkownika i
          SELECT(?)                                 ! umożliwienie ponownego jej wprowadzenia
        END
      OF ?OkButton
        Screen{PROP:AcceptAll} = 1                  ! Zainicjowanie trybu AcceptAll
    . .                                             ! Zakończenie ACCEPT i CASE ACCEPTED
```

PROP:Active

Właściwość okna WINDOW dająca wartość 1, gdy okno jest aktywne lub łańcuch pusty – w przeciwnym wypadku. Ustawienie na 1 powoduje, że okno staje się oknem pierwszoplanowym, a jego wątek – wątkiem aktywnym.

Przykład:

```
CODE
  OPEN(ThisWindow)
  X# = START(AnotherThread)            ! uruchomienie kolejnego wątku
  ACCEPT
    CASE EVENT()
    OF EVENT:LoseFocus                 ! gdy dane okno traci aktywność
      IF Y# <> X#                      ! sprawdzenie zmiany pierwszej aktywności
        ThisWindow{PROP:Active} = 1    ! i zwrot aktywności do tego wątku
        Y# = X#                        ! następnie oflagowanie kompletności pierwszej zmiany aktywności
  . . .
```

PROP:AlwaysDrop

Gdy jest ustawiona na zero, część opadająca kontrolki COMBO lub LIST posiadającej atrybut DROP jest rozwijana tylko wtedy, gdy użytkownik wciśnie przyciski rozwijania. Wciskanie klawisza strzałki w dół powoduje zmianę elementów wyświetlanych w polu edycyjnym, bez rozwijania listy. Dowolna inna wartość, różna od zera, powoduje, że część opadająca jest rozwijana albo po wciśnięciu przycisku rozwijania, albo po wciśnięciu klawisza strzałki w dół.

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            COMBO(@S20),AT(0,0,20,220),USE(MyCombo),FROM(Que),DROP(5)
          END
  CODE
  OPEN(MDIChild)
  ?MyCombo{PROP:AlwaysDrop} = 0                ! lista rozwijalna zachowuje się jak okno
```

PROP:AppInstance

Zwraca uchwyt egzemplarza (HInstance) pliku .EXE do zastosowania w wywołaniu niskiego poziomu funkcji API go potrzebującej. Używa się tylko dla wbudowanej zmiennej SYSTEM. Tylko-do-odczytu.

Przykład:

```
  PROGRAM
HInstance   LONG
  CODE
  OPEN(AppFrame)
  HInstance = SYSTEM{PROP:AppInstance}      ! pobranie uchwytu egzemplarza .EXE do późniejszego użycia
  ACCEPT
  END
```

PROP:AutoPaper

Ustawia i zwraca stan funkcji automatycznego przydziału papieru dla mechanizmu drukowania. Domyślnie funkcja ta jest włączona. Używa się tylko dla wbudowanej zmiennej SYSTEM.

Przykład:

```
PROGRAM
  CODE
  OPEN(AppFrame)
  SYSTEM{PROP:AutoPaper} = ''                    ! wyłączenie wyboru papieru best-fit
  ACCEPT
  END
```

PROP:BreakVar

Ustawia zmienną dla struktury grupującej BREAK raportu REPORT.

Przykład:

```
Report   REPORT,AT(1000,2000,6000,7000),PRE(RPT),FONT('Arial',10,,FONT:regular),THOUS
Break1     BREAK(ORD:CustNumber),USE(?Break1)
Break2       BREAK(DTL:OrderNumber),USE(?Break2)
               HEADER,AT(,,,2167)
             END
detail       DETAIL,AT(,,,385)
             END
             FOOTER,PAGEAFTER(-1)
             END
           END
           FOOTER,PAGEAFTER(-1),AT(,,,1385)
           END
         END
         FOOTER,AT(1000,9000,6000,1000)
         END
       END

  CODE
  OPEN(Report)
  Report$?Break1{PROP:BreakVar} = ORD:CustName       ! zmiana zmiennej grupującej dla Break1
```

PROP:Buffer

Właściwość okna, która umożliwia wybranie sposobu odświeżania tła. Może to znacznie zredukować miganie ekranu w niektórych sytuacjach (np. przy stosowaniu animowanych GIF-ów), ale jednocześnie zwiększyć zajętość pamięci.

Domyślną wartością jest zero (0), co powoduje rysowanie bezpośrednio do ekranu. Jest to metoda najszybsza i nie powoduje zwiększenia zajętości pamięci, może jednak, w niektórych przypadkach, powodować miganie ekranu.

Przypisanie wartości jeden (1) alokuje stały bufor pamięci dla okna. Jest to metoda dość szybka, ale powoduje większą zajętość pamięci.

Przypisanie wartości dwa (2) realokuje bufor pamięcią dla okna za każdym razem ,gdy jest konieczne odrysowanie. Jest to metoda wolniejsza, jednak nie pociąga za sobą zwiększenia zajętości pamięci, eliminując jednocześnie efekt migania ekranu.

Przykład:

```
WinView  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
         END
  CODE
  OPEN(WinView)
  WinView{PROP:Buffer} = 1                    ! stałe przydzielenie bufora dla okna
```

PROP:Checked

Zwraca aktualny stan wyświetlania kontrolki CHECK – zaznaczona (1) lub nie zaznaczona (‘’). Tylko-do-odczytu.

Przykład:

```
WinView  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
           CHECK('Check Me'),AT(20,0,20,20),USE(CheckVar)
         END
  CODE
  OPEN(WinView)
  IF ?CheckVar{PROP:Checked} = TRUE           ! czy to jest zaznaczone?
    !Do something
  END
  ACCEPT
  END
```

PROP:Child, PROP:ChildIndex

PROP:Child jest właściwością tablicową, która zwraca numer kontrolki podrzędnej znajdującej się w strukturze kontrolki nadrzędnej (takiej, jak: TAB, OPTION lub GROUP). Tylko-do-odczytu. Numer elementu jest pozycją porządkową kontrolki w strukturze nadrzędnej. Właściwość zwraca łańcuch pusty (‘’) jeśli numer elementu znajduje się poza zakresem.

PROP:ChildIndex jest właściwością tablicową, która zwraca pozycję porządkową wszystkich kontrolek podrzędnych struktury nadrzędnej (takiej, jak: TAB, OPTION lub GROUP). Tylko-do-odczytu. Numer elementu jest numerem kontrolki w strukturze nadrzędnej. Właściwość zwraca łańcuch pusty (‘’)) jeśli numer elementu znajduje się poza zakresem.

Przykład:

```
WinView  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
           OPTION('Option 1'),USE(OptVar1),KEY(F10Key),HLP('Option1Help')
             RADIO('Radio 1'),AT(0,0,20,20),USE(?R1)
             RADIO('Radio 2'),AT(20,0,20,20),USE(?R2)
           END
         END

  CODE
  OPEN(WinView)
  LOOP X# = 1 TO 99
    Y# = ?OptVar1{PROP:Child,X#}                  ! pobranie numerów kontrolek w OPTION
    IF NOT Y# THEN BREAK.
    Z# = ?OptVar1{PROP:ChildIndex,Y#}             ! pobranie pozycji porządkowych kontrolek w OPTION
    MESSAGE('Radio ' & Z# & ' is field number ' & Y#)
  END
  ACCEPT
  END
```

PROP:ChoiceFeq

Zwraca lub ustawia numer pola aktualnie wybranej zakładki TAB w kontrolce SHEET lub przycisku RADIO w strukturze OPTION.

Przykład:

```
WinView  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
           OPTION('Option 1'),USE(OptVar1),KEY(F10Key),HLP('Option1Help')
             RADIO('Radio 1'),AT(0,0,20,20),USE(?R1)
             RADIO('Radio 2'),AT(20,0,20,20),USE(?R2)
           END
         END

  CODE
  OPEN(WinView)
  ?OptVar1{PROP:ChoiceFeq} = ?R1                    ! wybór przycisku Radio 1
  ACCEPT
  END
```

PROP:ClientHandle

Właściwość okna WINDOW, która zwraca uchwyt obszaru wewnętrznego okna (obszaru, w którym znajdują się jego kontrolki). Uchwyt ten jest przekazywany do wywołań funkcji Windows API. Tylko-do-odczytu.

Przykład:

```
WinView  WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL
         END
MessageText    CSTRING('You cannot exit the program from this window ')
MessageCaption CSTRING('No EVENT:CloseDown Allowed ')
TextAddr       LONG
CaptionAddr    LONG
RetVal         SHORT

  CODE
  OPEN(WinView)
  ACCEPT
    CASE EVENT()
    OF EVENT:CloseDown
      TextAddr = ADDRESS(MessageText)
      CaptionAddr = ADDRESS(MessageCaption)
      RetVal = MessageBox(WinView{PROP:ClientHandle},TextAddr,CaptionAddr,MB_OK)
                                          ! wywołanie Windows API z zastosowaniem uchwytu okna
      CYCLE                               ! uniemożliwienie programowi zakończenia z tego okna
    END
  END
```

PROP:ClientWndProc

Właściwość okna WINDOW, która ustawia lub pobiera procedurę obsługująca komunikaty obszaru roboczego okna (bez paska tytułowego i paska stanu). Procedura ta jest stosowana w wywołaniach Windows API. Na ogół stosuje się w celu śledzenia wszystkich komunikatów Windows.

Przykład:

```
  PROGRAM
  MAP
    main
    SubClassFunc(USHORT,SHORT,USHORT,LONG),LONG,PASCAL
    MODULE('Windows') !TopSpeed Win31 Library
      CallWindowProc(LONG,UNSIGNED,SIGNED,UNSIGNED,LONG),LONG,PASCAL
    END
  END

SavedProc  LONG
PT         GROUP,PRE(PT)
X            SHORT
Y            SHORT
           END

  CODE
  Main

Main     PROCEDURE
WinView  WINDOW('View'),AT(0,0,320,200),HVSCROLL,MAX,TIMER(1)
           STRING('X Pos'),AT(1,1,,),USE(?String1)
           STRING(@n3),AT(24,1,,),USE(PT:X)
           STRING('Y Pos'),AT(44,1,,),USE(?String2)
           STRING(@n3),AT(68,1,,),USE(PT:Y)
           BUTTON('Close'),AT(240,180,60,20),USE(?Close)
         END

  CODE
  OPEN(WinView)
  SavedProc = WinView{PROP:ClientWndProc}                ! Zachowanie procedury
  WinView{PROP:ClientWndProc} = ADDRESS(SubClassFunc)
                                                         ! zmiana procedury
  ACCEPT
    CASE ACCEPTED()
    OF ?Close
      BREAK
    END
  END

SubClassFunc  PROCEDURE(hWnd,wMsg,wParam,lParam)     ! procedura
WM_MOUSEMOVE  EQUATE(0200H)                          ! do śledzenia ruchu myszki w
  CODE                                               ! obszarze roboczym okna
  CASE wMsg
  OF WM_MOUSEMOVE
    PT:X = MOUSEX()
    PT:Y = MOUSEY()
  END
  RETURN(CallWindowProc(SavedProc,hWnd,wMsg,wParam,lParam))
    ! przekazanie sterowania z powrotem
    ! do zachowanej procedury
```

PROP:ClipBits

Właściwość kontrolki IMAGE umożliwiająca umieszczanie grafik bitmapowych w Schowku systemowym – gdy jest ustawiona na jeden (1). W Schowku systemowym można umieszczać w postaci .BMP tylko grafikę formatu .BMP, .PCX lub .GIF.

Przykład:

```
WinView  WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL
           IMAGE(),AT(0,0,,),USE(?Image)
           BUTTON('Save Picture'),AT(80,180,60,20),USE(?SavePic)
           BUTTON('New Picture'),AT(160,180,60,20),USE(?NewPic)
         END
FileName STRING(64)

  CODE
  OPEN(WinView)
  DISABLE(?LastPic)
  IF NOT FILEDIALOG('Choose File to View',FileName,'BitMap|*.BMP|PCX|*.PCX',0)
    RETURN                                          ! powrót, jeśli nie wybrano pliku
  END
  ?Image{PROP:Text} = FileName
  ACCEPT
    CASE ACCEPTED()
    OF ?NewPic
      IF NOT FILEDIALOG('Choose File to View',FileName,'BitMap|*.BMP|PCX|*.PCX',0)
        BREAK                                       ! powrót, jeśli nie wybrano pliku
      END
      ?Image{PROP:Text} = FileName
    OF ?SavePic
      ?Image{PROP:ClipBits} = 1                     ! umieszczenie grafiki w Schowku
      ENABLE(?LastPic)                              ! uaktywnienie przycisku Last Picture
    END
  END
```

PROP:DDEMode

Właściwość wbudowanej zmiennej SYSTEM umożliwiająca ustawienie normalnego zachowania DDE (0, domyślnie), przy którym wszystkie zdarzenia DDE są przesyłane do okna, które otworzyło kanał DDE lub ustawienie na wartość jeden (1), która powoduje, że wszystkie zdarzenia DDE są przesyłane do pierwszoplanowego okna bieżącego wątku.

Przykład:

```
DDERetVal  STRING(20)
WinOne     WINDOW,AT(0,0,160,400)
             ENTRY(@s20),USE(DDERetVal)
           END
MyServer   LONG

  CODE
  OPEN(WinOne)
  SYSTEM{PROP:DDEMode} = 1                        ! wysyłanie zdarzeń do pierwszego okna bieżącego wątku
  MyServer = DDESERVER('MyApp','DataEntered')     ! otwarcie jako serwera
  ACCEPT
  END
```

PROP:DDETimeOut

Właściwość wbudowanej zmiennej SYSTEM umożliwiająca ustawienie i pobranie czasu oczekiwania DDE stosowanego we wszystkich poleceniach DDE. Wartość ta jest określana w setnych sekundy, a domyślną wartością jest 500.

Przykład:

```
DDERetVal STRING(20)
WinOne    WINDOW,AT(0,0,160,400)
            ENTRY(@s20),USE(DDERetVal)
          END
MyServer  LONG

  CODE
  OPEN(WinOne)
  SYSTEM{PROP:DDETimeOut} = 12000                          ! ustawienie oczekiwania 2 minutowego
  MyServer = DDESERVER('MyApp','DataEntered')              ! otwarcie jako serwera
  ACCEPT
    CASE EVENT()
    OF EVENT:DDErequest                                    ! jednorazowe żądanie danej
      DDEWRITE(MyServer,DDE:manual,'DataEntered',DDERetVal)
                                                           ! jednorazowe udostępnienie danej
    END
  END
```

PROP:DeferMove

Właściwość wbudowanej zmiennej SYSTEM opóźniająca zmianę rozmiaru i/lub przemieszczenie kontrolek dopóki nie zostanie osiągnięty koniec pętli ACCEPT lub właściwość SYSTEM{PROP:DeferMove} zostanie ustawiona na zero (0). Wyłącza to natychmiastowe wykonywanie przypisań pozycji i rozmiarów, pozwalając na wykonanie wszystkich przesunięć bibliotece uruchomieniowej i to za jednym razem (eliminuje to tymczasowe nakładanie się kontrolek). Wartość bezwzględna liczby przypisanej do SYSTEM{PROP:DeferMove} definiuje liczbę opóźnionych przesunięć, dla których miejsce jest przygotowywane wcześniej (może być zwiększana automatycznie, jeśli jest taka konieczność, lecz jest to mniej efektywne i może prowadzić do błędów). Przypisanie wartości dodatniej automatycznie resetuje PROP:DeferMove na zero przy następnym ACCEPT, podczas gdy liczba ujemna pozostawia ją ustawioną dopóty, dopóki jawnie nie zostanie wyzerowana. (0).

Przykład:

```
WinView  WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL
           IMAGE(),AT(0,0,,),USE(?Image)
           BUTTON('New Picture'),AT(160,180,60,20),USE(?NewPic)
           BUTTON('Close'),AT(80,180,60,20),USE(?Close)
         END
FileName      STRING(64)
ImageWidth    SHORT
ImageHeight   SHORT

  CODE
  OPEN(WinView)
  DISABLE(?LastPic)
  IF NOT FILEDIALOG('Choose File to View',FileName,'BitMap|*.BMP|PCX|*.PCX',0)
    RETURN                                  ! powrót, jeśli nie wybrano pliku
  END
  ?Image{PROP:Text} = FileName
  ACCEPT
    CASE ACCEPTED()
    OF ?NewPic
      IF NOT FILEDIALOG('Choose File to View',FileName,'BitMap|*.BMP|PCX|*.PCX',0)
      BREAK                                 ! powrót, jeśli nie wybrano pliku
    END
    ?Image{PROP:Text} = FileName
    SYSTEM{PROP:DeferMove} = 4              ! opóźnione przesunięcie I rozmiarowanie
    ImageWidth = ?Image{PROP:Width}         ! 1 przesunięcie
    ImageHeight = ?Image{PROP:Height}       ! 2 przesunięcie
    IF ImageWidth > 320
      ?Image{PROP:Width} = 320
      ?Image{PROP:XPos} = 0
    ELSE
      ?Image{PROP:XPos} = (320 - ImageWidth) / 2     ! centrowanie w poziomie
    END
    IF ImageHeight > 180
      ?Image{PROP:Height} = 180
      ?Image{PROP:YPos} = 0
    ELSE
      ?Image{PROP:YPos} = (180 - ImageHeight) / 2    ! centrowanie w pionie
    END
  OF ?Close
    BREAK
. .                                 ! przesunięcia i rozmiarowania zachodzą na końcu ACCEPT
```

PROP:Edit

Właściwość kontrolki LIST określająca etykietę ekwiwalentu pola kontrolki do przeprowadzenia operacji edit-in-place dla kolumny listy LIST. Jest to tablica, której numer elementu wskazuje na numer kolumny do edycji. Gdy jest różny od zero, kontrolka jest pokazywana i przemieszczana (może zmienić się jej rozmiar) do bieżącego wiersza kolumny, w której ma zostać przeprowadzona edycja danej. Przypisanie wartości zero powoduje ponowne ukrycie kontrolki wprowadzania danych.

Przykład:

```
Q     QUEUE
f1      STRING(15)
f2      STRING(15)
      END

Win1  WINDOW('List Edit In Place'),AT(0,1,308,172),SYSTEM
        LIST,AT(6,6,120,90),USE(?List),COLUMN,FORMAT('60L@s15@60L@s15@'), |
        FROM(Q),ALRT(EnterKey)
      END
?EditEntry  EQUATE(100)

  CODE
  OPEN(Win1)
  CREATE(?EditEntry,CREATE:Entry)
  SELECT(?List,1)
  ACCEPT
    CASE FIELD()
    OF ?List
      CASE EVENT()
    OF EVENT:AlertKey
      IF KEYCODE() = EnterKey
        SETKEYCODE(MouseLeft2)
        POST(EVENT:Accepted,?List)
      END
    OF EVENT:NewSelection
      IF ?List{PROP:edit,?List{PROP:column}}
        GET(Q,CHOICE())
      END
    OF EVENT:Accepted
      IF KEYCODE() = MouseLeft2
        GET(Q,CHOICE())
        ?EditEntry{PROP:text} = ?List{PROPLIST:picture,?List{PROP:column}}
        CASE ?List{PROP:column}
        OF 1
          ?EditEntry{PROP:use} = Q.F1
        OF 2
          ?EditEntry{PROP:use} = Q.F2
        END
        ?List{PROP:edit,?List{PROP:column}} = ?EditEntry
      ..
    OF ?EditEntry
      CASE EVENT()
      OF EVENT:Selected
        ?EditEntry{PROP:Touched} = 1
      OF EVENT:Accepted
        PUT(Q)
        ?List{PROP:edit,?List{PROP:column}} = 0
    ...
```

PROP:Enabled

Zwraca pusty łańcuch, gdy kontrolka nie jest dostępna, niezależnie od tego czy „sama w sobie" nie jest dostępna, czy nie jest dostępna jej kontrolka nadrzędna (OPTION, GROUP, MENU, SHEET lub TAB). Tylko-do-odczytu.

Przykład:

```
MDIChild    WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
              SHEET,AT(0,0,320,175),USE(SelectedTab)
                TAB('Tab One'),USE(?TabOne)
                PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P1)
                ENTRY(@S8),AT(100,140,32,20),USE(E1)
                PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P2)
                ENTRY(@S8),AT(100,240,32,20),USE(E2)
              END
              TAB('Tab Two'),USE(?TabTwo)
                PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P3)
                ENTRY(@S8),AT(100,140,32,20),USE(E3)
                PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P4)
                ENTRY(@S8),AT(100,240,32,20),USE(E4)
              END
            END
            BUTTON('Ok'),AT(100,180,20,20),USE(?Ok)
            BUTTON('Cancel'),AT(200,180,20,20),USE(?Cancel)
          END
  CODE
  OPEN(MDIChild)
  ACCEPT
    CASE EVENT()
    OF EVENT:Completed
      BREAK
    END
    CASE FIELD()
    OF ?Ok
      CASE EVENT()
      OF EVENT:Accepted
        SELECT
      END
    OF ?E3
      CASE EVENT()
      OF EVENT:Accepted
        IF ?E3{PROP:Enabled} AND MDIChild{PROP:AcceptAll}
          ! sprawdzenie widzialności podczas AcceptAll
          E3 = UPPER(E3)                          ! konwersja wprowadzonej danej na duże litery
          DISPLAY(?E3)                            ! i wyświetlenie przkonwertowanej danej
        END
      END
    OF ?Cancel
      CASE EVENT()
      OF EVENT:Accepted
        BREAK
      END
    END
  END
```

PROP:EventsWaiting

Właściwość okna WINDOW, która określa, czy są dla niego jakieś zdarzenia oczekujące na przetworzenie. Na ogół wykorzystuje się w Internet Connect do określenia, kiedy sformatować stronę HTML. Tylko-do-odczytu.

Przykład:

```
IF TARGET{PROP:EventsWaiting}                ! sprawdzenie, czy nie oczekują zdarzenia
  ! jakieś akcje
END
```

PROP:ExeVersion

Właściwość wbudowanej zmiennej SYSTEM zwracająca numer wersji EXE utworzonego przez Clarion for Windows. Jest to numer wersji Clarion for Windows, za pomocą której został skompilowany plik EXE, nawet gdy biblioteka runtime pochodzi z nowszej wersji (porównaj PROP:LibVersion). Pojawiło się to po raz pierwszy w Clarion for Windows wersja 1.501, tak więc PROP:ExeVersion zwraca łańcuch pusty dla wersji wcześniejszych, niż 1.501. Tylko-do-odczytu.

Przykład:

```
MESSAGE('Compiled in CW release ' & SYSTEM{PROP:ExeVersion})
```

PROP:FlushPreview

Wysyła metapliki wskazane atrybutem PREVIEW raportu REPORT do drukarki (0 = wyłączone, w przeciwnym wypadku włączone, zawsze 0 przy otwarciu raportu).

Przykład:

```
SomeReport  PROCEDURE
WMFQue      QUEUE                          ! kolejka zawierająca nazwy plików .WMF
              STRING(64)
            END
NextEntry   BYTE(1)                        ! zmienna zliczająca elementy w kolejce

Report      REPORT,PREVIEW(WMFQue)         ! raport z atrybutem PREVIEW
DetailOne     DETAIL
                ! kontrolki raportu
              END
            END

ViewReport  WINDOW('View Report'),AT(0,0,320,200),MDI,MAX,HVSCROLL
              IMAGE(),AT(0,0,320,180),USE(?ImageField)
              BUTTON('View Next Page'),AT(0,180,60,20),USE(?NextPage),DEFAULT
              BUTTON('Print Report'),AT(80,180,60,20),USE(?PrintReport)
              BUTTON('Exit Without Printing'),AT(160,180,60,20),USE(?ExitReport)
            END
  CODE
  OPEN(Report)
  SET(SomeFile)                                    ! kod generujący raport
  LOOP
    NEXT(SomeFile)
    IF ERRORCODE() THEN BREAK.
      PRINT(DetailOne)
    END
    ENDPAGE(Report)
    OPEN(ViewReport)                               ! otwarcie okna podglądu raportu
    GET(WMFQue,NextEntry)                          ! pobranie pierwszego elementu kolejki
    ?ImageField{PROP:text} = WMFQue                ! załadowanie pierwszej strony raportu
    ACCEPT
      CASE ACCEPTED()
      OF ?NextPage
        NextEntry += 1                             ! inkrementacja licznika elementów
        IF NextEntry > RECORDS(WMFQue) THEN CYCLE. ! kontrola końca raportu
        GET(WMFQue,NextEntry)                      ! pobranie następnego elementu kolejki
        ?ImageField{PROP:text} = WMFQue            ! załadowanie następnej strony raportu
        DISPLAY                                    ! i wyświetlenie jej
      OF ?PrintReport
        Report{PROP:FlushPreview} = 1              ! wymiecenie plików do drukarki
        BREAK                                      ! i wyjście z procedury
      OF ?ExitReport
        BREAK                                      ! wyjście z procedury
      END
    END
    RETURN                                   ! powrót do miejsca wywołania, automatyczne
                                             ! zamkniecie okna i raportu, zwolnienie kolejki
                                             ! automatyczne usunięcie plików tymczasowych .WMF
```

PROP:Follows

Zmienia kolejność w sekwencji tabulacji określając pozycję w strukturze nadrzędnej, którą ma zająć kontrolka. Kontrolka następuje po numerze kontrolki, który określamy w sekwencji tabulacji. Musi to określać istniejącą kontrolkę wewnątrz struktury nadrzędnej (okna, opcji, grupy, menu, raportu, detalu, itd.). Ustawienie PROP:Follows dla kontrolki REGION jest ignorowane, jako że REGION nie znajduje się w sekwencji tabulacji. Tylko-do-zapisu.

Przykład:

```
WinView   WINDOW('View Report'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            BUTTON('View Next Page'),AT(0,180,60,20),USE(?NextPage),DEFAULT
            BUTTON('Print Report'),AT(80,180,60,20),USE(?PrintReport)
            BUTTON('Exit Without Printing'),AT(160,180,60,20),USE(?ExitReport)
          END
  CODE
  OPEN(WinView)
  ! przycisk Print Report występuje normalnie po przycisku View
  ?PrintReport{PROP:Follows} = ?ExitReport
  ! teraz przycisk Print Report następuje po przycisku Exit w sekwencji tabulacji
  ACCEPT
  END
```

PROP:Handle

Zwraca uchwyt okna lub kontrolki, który będzie zastosowany w wywołaniu funkcji niskiego poziomu Windows API. Tylko-do-odczytu.

Przykład:

```
WinView   WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL
          END
MessageText     CSTRING('You cannot exit the program from this window ')
MessageCaption  CSTRING('No EVENT:CloseDown Allowed ')
TextAddress     LONG
CaptionAddress  LONG
RetVal          SHORT

  CODE
  OPEN(WinView)
  ACCEPT
    CASE EVENT()
    OF EVENT:CloseDown
      TextAddress = ADDRESS(MessageText)
      CaptionAddress = ADDRESS(MessageCaption)
      RetVal = MessageBox(WinView{PROP:Handle},TextAddress,CaptionAddress,MB_OK)
      ! wywołanie Windows API wykorzystujące uchwyt okna
      CYCLE                          ! uniemożliwa zamkniecie programu z tego okna
    END
  END
```

PROP:HeaderHeight

Zwraca wysokość nagłówka w kontrolce LIST lub COMBO. Wysokość jest określona w jednostkach dialogowych (chyba że jest aktywny PROP:Pixels). Tylko-do-odczytu.

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            LIST,AT(0,0,220,220),USE(?L1),FROM(Que),IMM,FORMAT('60L~Header Text~')
          END
  CODE
  OPEN(MDIChild)
  X# = ?L1{PROP:HeaderHeight}          ! pobranie wysokości nagłówka w jednostkach dialogowych
```

PROP:HscrollPos

Zwraca pozycję suwaka paska poziomego (od 0 do 255) w oknie, w kontrolce: IMAGE, TEXT, LIST lub COMBO, które posiadają atrybut HSCROLL. Ustawienie to daje możliwość przewijania zawartości okna lub kontrolki w poziomie.

Przykład:

```
Que    QUEUE
F1       STRING(50)
F2       STRING(50)
F3       STRING(50)
       END
WinView   WINDOW('View'),AT(,,340,200),SYSTEM,CENTER
            LIST,AT(20,0,300,200),USE(?List),FROM(Que),IMM,HVSCROLL |
              FORMAT('80L#1#80L#2#80L#3#')
          END
  CODE
    OPEN(WinView)
    DO BuildListQue
    ACCEPT
      CASE FIELD()
      OF ?List
        CASE EVENT()
        OF EVENT:ScrollDrag
          CASE (?List{PROP:HscrollPos} % 200) + 1
          OF 1
            ?List{PROP:Format} = '80L#1#80L#2#80L#3#'
          OF 2
            ?List{PROP:Format} = '80L#2#80L#3#80L#1#'
          OF 3
            ?List{PROP:Format} = '80L#3#80L#1#80L#2#'
          END
          DISPLAY
    ...
    FREE(Que)
    BuildListQue ROUTINE
    LOOP 15 TIMES
      Que.F1 = 'F1F1F1F1'
      Que.F2 = 'F2F2F2F2'
      Que.F3 = 'F3F3F3F3'
      ADD(Que)
    END
```

PROP:IconList

Tablica, która ustawia lub zwraca ikony wyświetlane w kontrolce LIST sformatowanej do wyświetlania ikon (zazwyczaj lista typu tree). Jeśli do nazwy pliku ikony jest dołączony numer w nawiasach kwadratowych, oznacza to, że dany plik zawiera wiele ikon, a numer określa tę, którą pobieramy (liczymy od zera). Jeśli nazwa pliku jest poprzedzona znakiem tyldy (~), (~IconFile.ICO), oznacza to, że plik jest wlinkowany do projektu jako zasób, a nie znajduje się na dysku.

Przykład:

```
    PROGRAM
    MAP
RandomAlphaData   PROCEDURE(*STRING Field)
    END

TreeDemo  QUEUE,PRE()              ! kolejka pola FROM listy elementów
FName       STRING(20)
ColorNFG    LONG                   ! normalny kolor pisania dla FName
ColorNBG    LONG                   ! normalny kolor tła dla FName
ColorSFG    LONG                   ! wyróżniony kolor pisania dla FName
ColorSBG    LONG                   ! wyróżniony kolor tła dla FName
IconField   LONG                   ! numer ikony dla FName
TreeLeve    LONG                   ! poziom drzewa
LName       STRING(20)
Init        STRING(4)
            END

Win         WINDOW('List Boxes'),AT(0,0,366,181),SYSTEM,DOUBLE
              LIST,AT(0,34,366,146),FROM(TreeDemo),USE(?Show),HVSCROLL, |
                FORMAT('80L*IT~First Name~*80L~Last Name~16C~Initials~')
              END
  CODE
  LOOP 20 TIMES
    RandomAlphaData(FName)
    ColorNFG = COLOR:White          ! przypisanie kolorów
    ColorNBG = COLOR:Maroon
    ColorSFG = COLOR:Yellow
    ColorSBG = COLOR:Blue
    IconField = ((x#-1) % 4) + 1    ! przypisanie numeru ikony
    TreeLevel = ((x#-1) % 4) + 1    ! przypisanie poziomu drzewa
    RandomAlphaData(LName)
    RandomAlphaData(Init)
    ADD(TD)
  END
  OPEN(Win)
  ?Show{PROP:iconlist,1} = ICON:VCRback       ! Ikona 1 = <
  ?Show{PROP:iconlist,2} = ICON:VCRrewind     ! Ikona 2 = <<
  ?Show{PROP:iconlist,3} = 'VCRdown.ico'      ! Ikona 3 = > nie linkowana do projektu
  ?Show{PROP:iconlist,4} = '~VCRnext.ico'     ! Ikona 4 = >>linkowana do projektu
  ACCEPT
  END

RandomAlphaData   PROCEDURE(*STRING Field)
  CODE
  y# = RANDOM(1,SIZE(Field))                  ! losowy rozmiar wypełnienia
  LOOP x# = 1 to y#                           ! wypełnienie każdego znaku
    Field[x#] = CHR(RANDOM(97,122))           ! losową małą literą
  END
```

PROP:ImageBits

Właściwość kontrolki IMAGE, która umożliwia zapisywanie i pobieranie grafik bitmapowych w niej wyświetlanych do i z pola memo. Może zostać zachowana dowolna grafika przechowywana w kontrolce. Właściwość PROP:ImageBlob wykonuje ten sam typ funkcji dla pola BLOB.

Przykład:

```
WinView     WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL
              IMAGE(),AT(0,0,,),USE(?Image)
              BUTTON('Save Picture'),AT(80,180,60,20),USE(?SavePic)
              BUTTON('New Picture'),AT(160,180,60,20),USE(?NewPic)
              BUTTON('Last Picture'),AT(240,180,60,20),USE(?LastPic)
            END

SomeFile    FILE,DRIVER('Clarion'),PRE(Fil)          ! plik z polem memo
MyMemo        MEMO(65520),BINARY
Rec           RECORD
F1              LONG
            . .
FileName    STRING(64)                               ! zmienna dla nazwy pliku
CODE
  OPEN(SomeFile)
  OPEN(WinView)
  DISABLE(?LastPic)
  IF NOT FILEDIALOG('Choose File to View',FileName,'BitMap|*.BMP|PCX|*.PCX',0)
    RETURN                                           ! powrót jeśli nie wybrano pliku
  END
  ?Image{PROP:Text} = FileName
  ACCEPT
    CASE ACCEPTED()
    OF ?NewPic
      IF NOT FILEDIALOG('Choose File to View',FileName,'BitMap|*.BMP|PCX|*.PCX',0)
        BREAK                                        ! powrót jeśli nie wybrano pliku
      END
      ?Image{PROP:Text} = FileName
    OF ?SavePic
      Fil:MyMemo = ?Image{PROP:ImageBits}            ! umieszczenie grafiki w memo
      ADD(SomeFile)                                  ! i zachowanie w pliku na dysku
      ENABLE(?LastPic)                               ! uaktywnienie przycisku Last Picture
    OF ?LastPic
      ?Image{PROP:ImageBits} = Fil:MyMemo            ! umieszczenie ostatnio zapisanego memo w grafice
    END
  END
```

PROP:ImageBlob, PROP:PrintMode

PROP:ImageBlob

Właściwość kontrolki IMAGE, która umożliwia zapisywanie i pobieranie grafik bitmapowych w niej wyświetlanych do i z pola BLOB. Może zostać zachowana dowolna grafika przechowywana w kontrolce. Właściwość PROP:ImageBits wykonuje ten sam typ funkcji dla pola memo. Większość grafik jest domyślnie zachowana w formacie mapy bitowej (za wyjątkiem PCX i GIF), jeśli nie jest ustawiona właściwość PROP:PrintMode powodująca przechowywanie w formacie natywnym.

PROP:PrintMode

Właściwość bitmapowa kontrolki IMAGE (lub wbudowanej zmiennej SYSTEM), która określa, w jaki sposób PROP:ImageBlob ma zachowywać grafikę w polu BLOB. Bit 0 określa, czy jest potrzebna informacja kodująca grafiki, a bit 1 – czy jest potrzebna informacja dekodująca. Ustawienie na 3 powoduje, że zarówno oryginalne dane i kodowane dane DIB są dostępne, umożliwiając PROP:ImageBlob zachowanie grafiki w jej natywnym formacie (np. JPG) w polu BLOB.

Przykład:

```
WinView  WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL
           IMAGE(),AT(0,0,,),USE(?Image)
           BUTTON('Save Picture'),AT(80,180,60,20),USE(?SavePic)
           BUTTON('New Picture'),AT(160,180,60,20),USE(?NewPic)
           BUTTON('Last Picture'),AT(240,180,60,20),USE(?LastPic)
         END

SomeFile FILE,DRIVER('TopSpeed'),PRE(Fil)          ! plik z polem memo
MyBlob     BLOB,BINARY
Rec        RECORD
F1           LONG
         . .
FileName STRING(64)                                ! zmienna dla nazwy pliku
CODE
  OPEN(SomeFile); OPEN(WinView)
  DISABLE(?LastPic)
  IF NOT FILEDIALOG('File to View',FileName,'Images|*.BMP;*.PCX;*.JPG;*.WMF',0)
    RETURN                                         ! powrót jeśli nie wybrano pliku
  END
  ?Image{PROP:PrintMode} = 3                       ! przechowywanie w natywnym formacie
  ?Image{PROP:Text} = FileName
  ACCEPT
    CASE ACCEPTED()
    OF ?NewPic
      IF NOT FILEDIALOG('File to View',FileName,'Images|*.BMP;*.PCX;*.JPG;*.WMF',0)
        BREAK
      END
      ?Image{PROP:Text} = FileName
    OF ?SavePic
      Fil:MyBlob{PROP:Handle} = ?Image{PROP:ImageBlob}     ! umieszczenie grafiki w BLOB
      ADD(SomeFile)                                        ! i zachowanie w pliku na dysku
      ENABLE(?LastPic)                                     ! uaktywnienie przycisku Last Picture
    OF ?LastPic
      ?Image{PROP:ImageBlob} = Fil:MyBlob{PROP:Handle}     ! umieszczenie ostatnio zapisanego BLOB
                                                           ! w grafice
    END
  END
```

PROP:InToolbar

Atrybut przełącznikowy określający, czy kontrolka znajduje się w pasku narzędzi TOOLBAR. Tylko-do-odczytu.

Przykład:

```
WinView   WINDOW('View'),AT(0,0,,),MDI,MAX,HVSCROLL,SYSTEM,RESIZE
            TOOLBAR
              BUTTON('Save Picture'),AT(80,180,60,20),USE(?SavePic)
            END
            LIST,AT(6,6,120,90),USE(?List),FORMAT('120L'),FROM(Q),IMM
          END
  CODE
  OPEN(WinView)
  IF ?SavePic{PROP:InToolbar} = TRUE
    !DO Something
  END
  ACCEPT
  END
```

PROP:Items

Zwraca lub ustawia liczbę elementów widocznych w kontrolce LIST lub COMBO.

Przykład:

```
Que    QUEUE
         STRING(30)
       END

WinView   WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL,SYSTEM
            LIST,AT(6,6,120,90),USE(?List),FORMAT('120L'),FROM(Q),IMM
          END

  CODE
  OPEN(WinView)
  SET(SomeFile)
  LOOP ?List{PROP:Items} TIMES          ! wypełnienie kolejki zgodnie z limitem widocznych elementów
    NEXT(SomeFile)
    Que = Fil:Record
    ADD(Que)
  END
  ACCEPT
  END
```

PROP:LazyDisplay

Właściwość SYSTEM , która wyłącza (ustawiona na 1) lub włącza (ustawiona na 0, domyślnie) funkcję powodującą, że wszystkie okna są całkowicie odrysowywane przed kontynuacją przetwarzania następnej instrukcji następującej po DISPLAY. Ustawienie PROP:LazyDisplay = 1 powoduje wyraźnie szybsze przetwarzanie video, gdyż odrysowania następują na końcu pętli ACCEPT, gdy już nie oczekują żadne zdarzenia. Może to zwiększyć wydajność niektórych aplikacji, może jednakże mieć także negatywny wpływ na ich wygląd.

Przykład:

```
WinView  APPLICATION('MyApp'),AT(0,0,320,200),MAX,HVSCROLL,SYSTEM
         END
  CODE
  OPEN(WinView)
  SYSTEM{PROP:LazyDisplay} = 1  ! wyłączenie dodatkowych komunikatów odświeżających ekran
                                ! dla całej aplikacji
  ACCEPT
  END
```

PROP:LFNSupport

Właściwość wbudowanej zmiennej SYSTEM w programach 16-bitowych, która zwraca jeden (1) jeśli system operacyjny obsługuje długie nazwy plików lub łańcuch pusty – w przeciwnym wypadku. Systemy operacyjne 32-bitowe obsługują długie nazwy plików. Tylko-do-odczytu.

Przykład:

```
IF SYSTEM{PROP:LFNSupport} = TRUE
  MESSAGE('Long Filenames are supported')
ELSE
  MESSAGE('Long Filenames are NOT supported')
END
```

PROP:LibHook

Właściwość tablicowa wbudowanej zmiennej SYSTEM, która ustawia procedury zastępujące wewnętrzne procedury Clarion. Dla każdej z tych procedur przypisuje się adres ADDRESS procedury zastępującej i biblioteka uruchomieniowa wywołuje w efekcie procedury zastępcze zamiast oryginalnych. Prototyp procedury zastępującej musi być dokładnie taki sam, jak procedury oryginalnej. Te właściwości zostały zaimplementowane w celu ułatwienia działania Internet Connect. Tylko-do-zapisu.

PROP:ColorDialogHook

Właściwość wbudowanej zmiennej SYSTEM, która ustawia procedurę zastępującą wewnętrzną procedurę COLORDIALOG Clariona. Równoważne z {PROP:LibHook,1}. Przypisanie adresu ADDRESS procedury zastępującej powoduje, że biblioteka uruchomieniowa wywołuje tę procedurę zamiast procedury COLORDIALOG. Przypisanie wartości zero powoduje przywrócenie wywoływania oryginalnej procedury wewnętrznej. Prototyp procedury zastępującej musi być dokładnie taki sam, jak procedury COLORDIALOG. Tylko-do-zapisu.

PROP:FileDialogHook

Właściwość wbudowanej zmiennej SYSTEM, która ustawia procedurę zastępującą wewnętrzną procedurę FILEDIALOG Clariona. Równoważne z {PROP:LibHook,2}. Przypisanie adresu ADDRESS procedury zastępującej powoduje, że biblioteka uruchomieniowa wywołuje tę procedurę zamiast procedury FILEDIALOG. Przypisanie wartości zero powoduje przywrócenie wywoływania oryginalnej procedury wewnętrznej. Prototyp procedury zastępującej musi być dokładnie taki sam, jak procedury FILEDIALOG. Tylko-do-zapisu.

PROP:FontDialogHook

Właściwość wbudowanej zmiennej SYSTEM, która ustawia procedurę zastępującą wewnętrzną procedurę FONTDIALOG Clariona. Równoważne z {PROP:LibHook,3}. Przypisanie adresu ADDRESS procedury zastępującej powoduje, że biblioteka uruchomieniowa wywołuje tę procedurę zamiast procedury FONTDIALOG. Przypisanie wartości zero powoduje przywrócenie wywoływania oryginalnej procedury wewnętrznej. Prototyp procedury zastępującej musi być dokładnie taki sam, jak procedury FONTDIALOG. Tylko-do-zapisu.

PROP:PrinterDialogHook

Właściwość wbudowanej zmiennej SYSTEM, która ustawia procedurę zastępującą wewnętrzną procedurę PRINTERDIALOG Clariona. Równoważne z {PROP:LibHook,4}. Przypisanie adresu ADDRESS procedury zastępującej powoduje, że biblioteka uruchomieniowa wywołuje tę procedurę zamiast procedury PRINTERDIALOG. Przypisanie wartości zero powoduje przywrócenie wywoływania oryginalnej procedury wewnętrznej. Prototyp procedury zastępującej musi być dokładnie taki sam, jak procedury PRINTERDIALOG. Tylko-do-zapisu.

PROP:HaltHook

Właściwość wbudowanej zmiennej SYSTEM, która ustawia procedurę zastępującą wewnętrzną procedurę HALT Clariona. Równoważne z {PROP:LibHook,5}. Przypisanie adresu ADDRESS procedury zastępującej powoduje, że biblioteka uruchomieniowa wywołuje tę procedurę zamiast procedury HALT. Przypisanie wartości zero powoduje przywrócenie wywoływania oryginalnej procedury wewnętrznej. Prototyp procedury zastępującej musi być dokładnie taki sam, jak procedury HALT. Tylko-do-zapisu.

PROP:MessageHook

Właściwość wbudowanej zmiennej SYSTEM, która ustawia procedurę zastępującą wewnętrzną procedurę MESSAGE Clariona. Równoważne z {PROP:LibHook,6}. Przypisanie adresu ADDRESS procedury zastępującej powoduje, że biblioteka uruchomieniowa wywołuje tę procedurę zamiast procedury MESSAGE. Przypisanie wartości zero powoduje przywrócenie wywoływania oryginalnej procedury wewnętrznej. Prototyp procedury zastępującej musi być dokładnie taki sam, jak procedury MESSAGE. Tylko-do-zapisu.

PROP:StopHook

Właściwość wbudowanej zmiennej SYSTEM, która ustawia procedurę zastępującą wewnętrzną procedurę STOP Clariona. Równoważne z {PROP:LibHook,7}. Przypisanie adresu ADDRESS procedury zastępującej powoduje, że biblioteka uruchomieniowa wywołuje tę procedurę zamiast procedury STOP. Przypisanie wartości zero powoduje przywrócenie wywoływania oryginalnej procedury wewnętrznej. Prototyp procedury zastępującej musi być dokładnie taki sam, jak procedury STOP. Tylko-do-zapisu.

PROP:AssertHook

Właściwość wbudowanej zmiennej SYSTEM, która ustawia procedurę zastępującą wewnętrzną procedurę ASSERT Clariona. Równoważne z {PROP:LibHook,8}. Przypisanie adresu ADDRESS procedury zastępującej powoduje, że biblioteka uruchomieniowa wywołuje tę procedurę zamiast procedury ASSERT. Przypisanie wartości zero powoduje przywrócenie wywoływania oryginalnej procedury wewnętrznej. Prototyp procedury zastępującej musi być dokładnie taki sam, jak procedury ASSERT (STRING komunikat,UNSIGNED NumerLinii). Tylko-do-zapisu.

PROP:FatalErrorHook

Właściwość wbudowanej zmiennej SYSTEM, która ustawia procedurę zastępującą wewnętrzną procedurę Clariona. Równoważne z {PROP:LibHook,9}. Przypisanie adresu ADDRESS procedury zastępującej powoduje, że biblioteka uruchomieniowa wywołuje tę procedurę zamiast procedury wewnętrznej. Przypisanie wartości zero powoduje przywrócenie wywoływania oryginalnej procedury wewnętrznej. Prototyp procedury zastępującej musi być dokładnie taki sam, jak procedury wewnętrznej (STRING komunikat,UNSIGNED NumerBłędu). Tylko-do-zapisu.

Przykład:

```
PROGRAM
MAP
  MyColorDialog PROCEDURE(<STRING>,*?,SIGNED=0),SIGNED,PROC
  MYFileDialog PROCEDURE(<STRING>,*?,<STRING>,SIGNED=0),PROC,BOOL
  MyFontDialog PROCEDURE(<STRING>,*?,<*?>,<*?>,<*?>,SIGNED = 0),BOOL,PROC
  MyPrinterDialog PROCEDURE(<STRING>,BOOL=FALSE),BOOL,PROC
  MyHalt PROCEDURE(UNSIGNED=0,<STRING>)
  MyMessage PROCEDURE(STRING,<STRING>,<STRING>,<STRING>,UNSIGNED=0,BOOL=FALSE), |
    UNSIGNED,PROC
  MyStop PROCEDURE(<STRING>)
  MyAssert PROCEDURE(<STRING>,UNSIGNED)
  MyFatalError PROCEDURE(<STRING>,UNSIGNED)
END

CODE
  !Hook all my own procedures
  SYSTEM{PROP:ColorDialogHook} = ADDRESS(MyColorDialog)
  SYSTEM{PROP:FileDialogHook} = ADDRESS(MyFileDialog)
  SYSTEM{PROP:FontDialogHook} = ADDRESS(MyFontDialog)
  SYSTEM{PROP:PrinterDialogHook} = ADDRESS(MyPrinterDialog)
  SYSTEM{PROP:HaltHook} = ADDRESS(MyHalt)
  SYSTEM{PROP:MessageHook} = ADDRESS(MyMessage)
  SYSTEM{PROP:StopHook} = ADDRESS(MyStop)
  SYSTEM{PROP:AssertHook} = ADDRESS(MyAssert)
  SYSTEM{PROP:FatalErrorHook} = ADDRESS(MyFatalError)
  !UnHook all my own procedures i return to the library procedures
  SYSTEM{PROP:ColorDialogHook} = 0
  SYSTEM{PROP:FileDialogHook} = 0
  SYSTEM{PROP:FontDialogHook} = 0
  SYSTEM{PROP:PrinterDialogHook} = 0
  SYSTEM{PROP:HaltHook} = 0
  SYSTEM{PROP:MessageHook} = 0
  SYSTEM{PROP:StopHook} = 0
  SYSTEM{PROP:AssertHook} = 0
  SYSTEM{PROP:FatalErrorHook} = 0
```

PROP:LibVersion

Właściwość wbudowanej zmiennej SYSTEM, która zwraca numer wersji biblioteki uruchomieniowej .DLL Clarion for Windows załadowanej aktualnie dla wykonywanego EXE. Jest to co innego, niż numer wersji Clarion for Windows użytej do skompilowania pliku EXE (porównaj PROP:ExeVersion). Pojawiło się to po raz pierwszy w Clarion for Windows wersja 1.501, tak więc PROP:ExeVersion zwraca łańcuch pusty dla wersji wcześniejszych, niż 1.501. Tylko-do-odczytu.

Przykład:

```
MESSAGE('Runtime DLL from release ' & SYSTEM{PROP:LibVersion})
```

PROP:Line, PROP:LineCount

Właściwość PROP:Line jest tablicą, której każdy element zawiera wiersz tekstu kontrolki TEXT. Tylko-do-odczytu.

Właściwość PROP:LineCount zwraca liczbę wierszy tekstu w kontrolce TEXT. Tylko-do-odczytu.

Przykład:

```
LineCount    SHORT
MemoLine     STRING(80)
CustRpt      REPORT,AT(1000,1000,6500,9000),THOUS
Detail1        DETAIL,AT(0,0,6500,6000)
                 TEXT,AT(0,0,6500,6000),USE(Fil:MemoField)
               END
Detail2        DETAIL,AT(0,0,6500,125)
                 STRING(@s80),AT(0,0,6500,125),USE(MemoLine)
               END
             END
  CODE
  OPEN(File)
  SET(File)
  OPEN(CustRpt)
  LOOP
    NEXT(File)
    LineCount = CustRpt$?Fil:MemoField{PROP:LineCount}
    LOOP X# = 1 TO LineCount
      MemoLine = CustRpt$?Fil:MemoField{PROP:Line,X#}
      PRINT(Detail2)
    END
  END
```

PROP:LineHeight

Ustawia lub zwraca wysokość wierszy w kontrolce LIST lub COMBO. Wysokość jest określona w jednostkach dialogowych (chyba, że aktywna jest właściwość PROP:Pixels). Dla kontrolki TEXT, zwraca wysokość komórki znaku dla czcionki kontrolki (odległość od góry jednego wiersza tekstu do góry następnego wiersza tekstu) w jednostkach, które są aktualnie używane. Tylko-do-odczytu dla kontrolki TEXT.

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            LIST,AT(0,0,220,220),USE(?L1),FROM(Que),IMM
          END
  CODE
  OPEN(MDIChild)
  ?L1{PROP:LineHeight} = 8                ! ustawienie wysokości na 8 jednostek dialogowych
```

PROP:MaxHeight, PROP:MaxWidth, PROP:MinHeight, PROP:MinWidth

PROP:MaxHeight ustawia lub zwraca maksymalną wysokość okna, które może zmieniać swój rozmiar.

PROP:MaxWidth ustawia lub zwraca maksymalną szerokość okna, które może zmieniać swój rozmiar.

PROP:MinHeight ustawia lub zwraca minimalną wysokość okna, które może zmieniać swój rozmiar.

PROP:MinWidth ustawia lub zwraca minimalną szerokość okna, które może zmieniać swój rozmiar. Ustawia także minimalną szerokość zakładek TAB kontrolki SHEET.

Przykład:

```
WinView   WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL,SYSTEM,RESIZE
            LIST,AT(6,6,120,90),USE(?List),FORMAT('120L'),FROM(Q),IMM
          END
  CODE
  OPEN(WinView)
  WinView{PROPMaxHeight} = 200      ! ustawienie granic poza które użytkownik nie może
  WinView{PROPMaxWidth} = 320       ! zwiększać rozmiaru okna
  WinView{PROPMinHeight} = 90
  WinView{PROPMinWidth} = 120
  ACCEPT
  END
```

PROP:NextField

Właściwość tablicowa zwracająca numer następnej kontrolki w sekwencji okna lub raportu. Tylko-do-odczytu. Porządek, w jakim PROP:NextField zwraca numery pól jest niezdefiniowany. Właściwość PROP:NextField zwraca zero, gdy numer elementu tablicy jest ostatnią kontrolką w liście. Ta właściwość w prosty sposób pozwala na poruszanie się po wszystkich kontrolkach w oknie lub w raporcie, niezależnie od tego, czy posiadają one atrybut USE, czy nie.

Przykład:

```
WinView   WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            IMAGE(),AT(0,0,,),USE(?Image)
            BUTTON('Save Picture'),AT(80,180,60,20),USE(?SavePic)
            BUTTON('New Picture'),AT(160,180,60,20),USE(?NewPic)
            BUTTON('Last Picture'),AT(240,180,60,20),USE(?LastPic)
          END
ThisField SHORT(0)

  CODE
  OPEN(WinView)
  LOOP
    ThisField = WinView{PROP:NextField,ThisField}    ! przetwarzaj każdą kontrolkę
    IF ThisField
      ThisField{PROP:FontName} = 'Arial'             ! zmiana czcionki
      ThisField{PROP:FontSize} = 10
    ELSE
      BREAK                                          ! przerwij, gdy wykonane
  ..
  ACCEPT
  END
```

PROP:NextPageNo

Właściwość ustawiająca lub zwracająca numer następnej strony raportu.

Przykład:

```
CustRpt   REPORT,AT(1000,1000,6500,9000),THOUS
            HEADER
              STRING(@n3),USE(?Page),PAGENO
            END
Detail      DETAIL,AT(0,0,6500,1000)
              STRING,AT(10,10),USE(Fil:Field)
          ..
  CODE
  OPEN(File);SET(File)
  OPEN(CustRpt)
  LOOP
    NEXT(File)
    IF Fil:KeyField <> Sav:KeyField          ! wykryj grupowanie
      Sav:KeyField = Fil:KeyField            ! wykryj grupowanie
    ENDPAGE                                  ! wymuś grupowanie
    CustRpt{PROP:NextPageNo} = 1             ! każda grupowa zaczyna się od strony jeden
  END
  PRINT(Detail)
  END
```

PROP:NoHeight, PROP:NoWidth

Właściwość PROP:NoHeight jest atrybutem przełącznikowym określającym, czy okno lub kontrolka została ustawiona do swej domyślnej wysokości (ma pominięty parametr wysokości atrybutu AT). Ustawienie tej właściwości na TRUE jest równoważne ze zresetowaniem kontrolki do jej domyślnej wysokości określonej przez bibliotekę (czego nie możemy zrobić stosując PROP:Height).

Właściwość PROP:NoWidth jest atrybutem przełącznikowym określającym, czy okno lub kontrolka została ustawiona do swej domyślnej szerokości (ma pominięty parametr szerokości atrybutu AT). Ustawienie tej właściwości na TRUE jest równoważne ze zresetowaniem kontrolki do jej domyślnej szerokości określonej przez bibliotekę (czego nie możemy zrobić stosując PROP:Width).

Przykład:

```
WinView   WINDOW('View'),AT(0,0,,),MDI,MAX,HVSCROLL,SYSTEM,RESIZE
            LIST,AT(6,6,120,90),USE(?List),FORMAT('120L'),FROM(Q),IMM
          END
  CODE
  OPEN(WinView)
  IF WinView{PROP:NoHeight} = TRUE
    WinView{PROP:Height} = 200               ! ustaw wysokość
  END
  IF WinView{PROP:NoWidth} = TRUE
    WinView{PROP:Width} = 320                ! ustaw szerokość
  END
  ACCEPT
  END
```

PROP:NoTips

Wyłącza (gdy jest ustawione na 1) lub włącza (gdy jest ustawione na 0) wyświetlanie podpowiedzi w dymkach (atrybut TIP) dla całej aplikacji (SYSTEM), okna lub kontrolki.

Przykład:

```
WinView   APPLICATION('MyApp'),AT(0,0,320,200),MAX,HVSCROLL,SYSTEM
          END
  CODE
  OPEN(WinView)
  SYSTEM{PROP:NoTips} = 1 !Disable TIP display throughout entire application
  ACCEPT
  END
```

PROP:Parent

Zwraca kontrolkę nadrzędną dla kontrolki znajdującej się w jakiejś strukturze (np. OPTION lub GROUP bądź DETAIL, TOOLBAR, czy MENUBAR). Tylko-do-odczytu.

Przykład:

```
WinView         WINDOW('MyApp'),AT(0,0,320,200),MAX,HVSCROLL,SYSTEM
                END
OptionSelected  STRING(6)
?OptionControl  EQUATE(100)      ! numer ekwiwalentu pola dla CREATE
?Radio1         EQUATE(101)      ! numer ekwiwalentu pola dla CREATE
?Radio2         EQUATE(102)      ! numer ekwiwalentu pola dla CREATE

  CODE
  OPEN(WinView)
  CREATE(?OptionControl,CREATE:option)                  ! utworzenie kontrolki OPTION
  ?OptionControl{PROP:use} = OptionSelected
  ?OptionControl{PROP:Text} = 'Pick one'
  ?OptionControl{PROP:Boxed} = TRUE
  SETPOSITION(?OptionControl,10,10)
  CREATE(?Radio1,CREATE:radio,?OptionControl)           ! utworzenie kontrolki RADIO
  ?Radio1{PROP:Text} = 'First'
  SETPOSITION(?Radio1,12,20)
  CREATE(?Radio2,CREATE:radio,?Radio1{PROP:Parent})     ! utworzenie jeszcze jednej z tą samą
                                                        ! kontrolką nadrzędną
  ?Radio2{PROP:Text} = 'Second'
  SETPOSITION(?Radio2,12,30)
  UNHIDE(?OptionControl,?Radio2)                        ! wyświetlenie nowych kontrolek
  ACCEPT
  END
```

PROP:Pixels

Właściwość okna WINDOW, która przełącza układ miar ekranu pomiędzy jednostkami dialogowymi (DLU) i pikselami (niedostępne dla raportów). Po ustawieniu tej właściwości, wszystkie pozycjonowania ekranu (GETPOSITION, SETPOSITION, PROP:Xpos, PROP:Ypos, PROP:Width, czy PROP:Height) zwracają i wymagają określania współrzędnych w pikselach zamiast w jednostkach dialogowych (DLU).

Przykład:

```
WinView  WINDOW('MyApp'),AT(0,0,320,200),MAX,HVSCROLL,SYSTEM
         END
  CODE
  OPEN(WinView)
  WinView{PROP:Pixels} = 1              ! zmiana jednostki miary na piksele
  ACCEPT
  END
```

PROP:PrintMode

Właściwość SYSTEM lub IMAGE ustawiająca lub zwracająca tryb, w którym grafiki są drukowane w raporcie. Ustawienie na jeden (1) generuje grafiki do plików tymczasowych. Ustawienie na dwa (2 - domyślnie) drukuje grafiki. Ustawienie na trzy (3) – powoduje wykonanie obu operacji. Właściwość ta jest wykorzystywana wewnętrznie przez szablony Internet Connect.

Przykład:

```
sprawdź szablony Internet Connect
```

PROP:Progress

Można bezpośrednio zaktualizować wyświetlanie kontrolki PROGRESS poprzez przypisanie wartości (znajdującej się w zakresie wartości zdefiniowanym w atrybucie RANGE) do właściwości PROP:Progress kontrolki.

Przykład:

```
BackgroundProcess  PROCEDURE                    ! proces wsadowy przetwarzania w tle

Win     WINDOW('Batch Processing...'),AT(,,400,400),TIMER(1),MDI,CENTER
          PROGRESS,AT(100,140,200,20),USE(?ProgressBar),RANGE(0,200)
          BUTTON('Cancel'),AT(190,300,20,20),STD(STD:Close)
        END

  CODE
  OPEN(Win)
  OPEN(File)
  ?ProgressBar{PROP:rangehigh} = RECORDS(File)
  SET(File)                                     ! przygotowanie procesu wsadowego
  ACCEPT
    CASE EVENT()
    OF EVENT:CloseWindow
      BREAK
    OF EVENT:Timer                              ! przetwarzaj rekordy, gdy pozwala na to timer
      ProgressVariable += 3                     ! auto-aktualizacja pierwszego paska progresu
      LOOP 3 TIMES
        NEXT(File)
        IF ERRORCODE() THEN BREAK.
          ?ProgressBar{PROP:progress} = ?ProgressBar{PROP:progress} + 1
                                                ! ręczna aktualizacja drugiego paska progresu
                                                ! wykonanie kodu przetwarzania
  . . .
  CLOSE(File)
```

PROP:RejectCode

Właściwość kontrolki ENTRY, TEXT, COMBO lub SPIN zwracająca ostatnią odrzuconą wartość REJECTCODE. Właściwość PROP:RejectCode jest stała, podczas gdy procedura REJECTCODE zwraca tylko właściwą wartość podczas zdarzenia EVENT:Rejected.

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P1)
            ENTRY(@N8),AT(100,140,32,20),USE(E1)
            BUTTON('Ok'),AT(100,180,20,20),USE(?Ok)
            BUTTON('Cancel'),AT(200,180,20,20),USE(?Cancel)
          END

  CODE
  OPEN(MDIChild)
  ACCEPT
    CASE EVENT()
    OF EVENT:Completed
      BREAK
    END
    CASE FIELD()
    OF ?Ok
      CASE EVENT()
      OF EVENT:Accepted
        SELECT
      END
    OF ?E1
      CASE EVENT()
      OF EVENT:Accepted
        IF ?E1{PROP:RejectCode} <> 0          ! sprawdzenie, czy wpis nie został odrzucony
          SELECT(?)                           ! i umożliwienie użytkownikowi
          CYCLE                               ! jego ponownego wprowadzenia
        END
      END
    OF ?Cancel
      CASE EVENT()
      OF EVENT:Accepted
        BREAK
      END
    END
  END
```

PROP:ScreenText

Zwraca tekst wyświetlany w danej kontrolce na ekranie.

Przykład:

```
WinView  WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL
           SPIN(@n3),AT(0,0,320,180),USE(Fil:Field),RANGE(0,255)
         END

  CODE
  OPEN(WinView)
  ACCEPT
    CASE FIELD()
    OF ?Fil:Field
      CASE EVENT()
      OF EVENT:Rejected
        MESSAGE(?Fil:Field{PROP:ScreenText} & ' is not in the range 0-255')
        SELECT(?)
        CYCLE
      END
    END
  END
```

PROP:SelStart, PROP:Selected, PROP:SelEnd

Właściwość PROP:SelStart (nazywana również PROP:Selected) ustawia lub odczytuje początkowy (włącznie) znak bloku tekstu w kontrolce ENTRY lub TEXT. Kursor jest umieszczany na lewo od tego znaku i jest ustawiana właściwość PROP:SelEnd na wartość zero (0) w celu określenia, że blok nie jest zaznaczony. Identyfikuje to również bieżąco podświetloną wartość w kontrolce LIST lub COMBO (zazwyczaj kodowana w tym przypadku jako PROP:Selected).

Właściwość PROP:SelEnd ustawia lub odczytuje końcowy (włącznie) znak bloku tekstu w kontrolce ENTRY lub TEXT.

Przykład:

```
WinView  WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL
           ENTRY(@S30),AT(0,0,320,180),USE(Fil:Field),ALRT(F10Key)
           LIST,AT(6,6,120,90),USE(?List),FORMAT('120L'),FROM(Q),IMM
         END
  CODE
  OPEN(WinView)
  ACCEPT
    CASE ACCEPTED()
    OF ?List
      GET(Q,?List{PROP:Selected})                    ! pobranie podświetlonego elementu kolejki
    OF ?Fil:Field
      SETCLIPBOARD(Fil:Field[?Fil:Field{PROP:SelStart} : ?Fil:Field{PROP:SelEnd}])
      ! umieszczenie podświetlonego fragmentu tekstu w Schowku Windows
    END
  END
```

PROP:Size

Zwraca lub ustawia rozmiar pola BLOB. Przed przypisaniem danych do pola BLOB w oparciu o technikę fragmentowania, gdy pole BLOB nie zawiera jeszcze żadnych danych, musimy określić jego rozmiar stosując właściwość PROP:Size. Przed dopisaniem dodatkowych danych, które zwiększą ilość przechowywanych danych w polu BLOB (w oparciu o technikę fragmentowania), musimy zresetować jego rozmiar stosując PROP:Size.

Przykład:

```
Names     FILE,DRIVER('TopSpeed')
NbrKey      KEY(Names:Number)
Notes       BLOB                                ! może być większe niż 64K
Rec         RECORD
Name          STRING(20)
Number        SHORT
            . .
BlobSize      LONG
BlobBuffer1   STRING(65520),STATIC                ! łańcuch maksymalnego rozmiaru
BlobBuffer2   STRING(65520),STATIC                ! łańcuch maksymalnego rozmiaru

WinView       WINDOW('View BLOB Contents'),AT(0,0,320,200),SYSTEM
                TEXT,AT(0,0,320,180),USE(BlobBuffer1),VSCROLL
                TEXT,AT(0,190,320,180),USE(BlobBuffer2),VSCROLL,HIDE
              END
  CODE
  OPEN(Names)
  SET(Names)
  NEXT(Names)
  OPEN(WinView)
  BlobSize = Names.Notes{PROP:Size}               ! pobranie rozmiaru zawartości BLOB
  IF BlobSize > 65520
    BlobBuffer1 = Names.Notes[0 : 65519]
    BlobBuffer2 = Names.Notes[65520 : BlobSize - 1]
    WinView{PROP:Height} = 400
    UNHIDE(?BlobBuffer2)
  ELSE
    BlobBuffer1 = Names.Notes[0 : BlobSize - 1]
  END
  ACCEPT
  END
```

PROP:TabRows, PROP:NumTabs

Właściwość PROP:TabRows zwraca liczbę wierszy zakładek TAB w arkuszu SHEET. Tylko-do-odczytu.

Właściwość PROP:NumTabs zwraca liczbę zakładek TAB w arkuszu SHEET. Tylko-do-odczytu.

Przykład:

```
MDIChild    WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
              SHEET,AT(0,0,320,175),USE(SelectedTab)
                TAB('Tab One'),USE(?TabOne)
                  OPTION('Option 1'),USE(OptVar1),KEY(F10Key),HLP('Option1Help')
                    RADIO('Radio 1'),AT(20,0,20,20),USE(?R1)
                    RADIO('Radio 2'),AT(40,0,20,20),USE(?R2)
                  END
                  OPTION('Option 2'),USE(OptVar2),MSG('Option 2')
                    RADIO('Radio 3'),AT(60,0,20,20),USE(?R3)
                    RADIO('Radio 4'),AT(80,0,20,20),USE(?R4)
                  END
                END
                TAB('Tab Two'),USE(?TabTwo)
                  PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P1)
                  ENTRY(@S8),AT(100,140,32,20),USE(E1)
                  PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P2)
                  ENTRY(@S8),AT(100,240,32,20),USE(E2)
                END
                TAB('Tab Three'),USE(?TabThree)
                  OPTION('Option 3'),USE(OptVar3)
                    RADIO('Radio 1'),AT(20,0,20,20),USE(?R5)
                    RADIO('Radio 2'),AT(40,0,20,20),USE(?R6)
                  END
                  OPTION('Option 4'),USE(OptVar4)
                    RADIO('Radio 3'),AT(60,0,20,20),USE(?R7)
                    RADIO('Radio 4'),AT(80,0,20,20),USE(?R8)
                  END
                END
                TAB('Tab Four'),USE(?TabFour)
                  PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P3)
                  ENTRY(@S8),AT(100,140,32,20),USE(E3)
                  PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P4)
                  ENTRY(@S8),AT(100,240,32,20),USE(E4)
                END
              END
              BUTTON('Ok'),AT(100,180,20,20),USE(?Ok)
              BUTTON('Cancel'),AT(200,180,20,20),USE(?Cancel)
            END

  CODE
  OPEN(MDIChild)
  MESSAGE('Number of TABs: ' & ?SelectedTab{PROP:NumTabs})
  MESSAGE('Number of rows of TABs: ' & ?SelectedTab{PROP:TabRows})
  ACCEPT
  END
```

PROP:TempImage

Właściwość kontrolki IMAGE zwracająca nazwę pliku tworzonego dla grafiki. Przeznaczona do wewnętrznego stosowania przez szablony Internet Connect.

PROP:TempImageStatus

Właściwość kontrolki IMAGE określająca, czy PROP:TempImage utworzyło nowy bądź zastąpiło istniejący plik dla grafiki. Przeznaczona do wewnętrznego stosowania przez szablony Internet Connect.

PROP:TempPath

Właściwość tablicowa SYSTEM ustawiająca lub zwracająca ścieżkę do plików tymczasowych stron raportu lub ścieżkę do tymczasowych plików graficznych ustawioną przez PROP:PrintMode. Przeznaczona do wewnętrznego stosowania przez szablony Internet Connect.

PROP:TempPagePath

Właściwość SYSTEM ustawiająca lub zwracająca ścieżkę do tymczasowych plików stron raportu. Równoważne z {PROP:TempPath,1}. Przeznaczona do wewnętrznego stosowania przez szablony Internet Connect.

PROP:TempImagePath

Właściwość SYSTEM ustawiająca lub zwracająca ścieżkę do tymczasowych plików grafiki ustawioną przez PROP:PrintMode lub PROP:TempImage. Równoważne z {PROP:TempPath,2}. Przeznaczona do wewnętrznego stosowania przez szablony Internet Connect.

PROP:TempNameFunc

Właściwość kontrolki REPORT pozwalająca na określenie własnych nazw dla metaplików generowanych dla atrybutu PREVIEW, poprzez napisanie funkcji callback dostarczającej nazwy metaplików dla każdej strony raportu. Funkcja callback musi być procedurą PROCEDURE, do której jest przekazywany pojedynczy parametr SIGNED i która zwraca wartość STRING. By włączyć tę funkcję, należy przypisać jej adres ADDRESS do właściwości PROP:TempNameFunc. By ją wyłączyć, należy przypisać do wymienionej właściwości wartość (0).

Mechanizm raportu, gdy potrzebuje zapisać stronę raportu na dysku, wywołuje procedurę, przekazuje do niej numer strony i wykorzystuje wartość zwracaną przez procedurę jako nazwę metapliku (zarówno dla dysku, jak i kolejki QUEUE atrybutu PREVIEW). Funkcja callback musi utworzyć plik, by upewnić się, że przydzielona dla niego nazwa jest dostępna.

Gdy stosuje się właściwość PROP:TempNameFunc, właściwość PROP:FlushPreview zapisuje metapliki do drukarki, ale nie usuwa ich automatycznie (musimy je usuwać samodzielnie, gdy program nie będzie już z nich korzystał).

Przykład:

```
  MEMBER('MyApp')
  MAP
PageNames  PROCEDURE(SIGNED),STRING          ! prototyp funkcji callback
  END

MyReport   PROCEDURE
MyQueue      QUEUE                           ! kolejka podglądu
               STRING(64)
             END

Report       REPORT,PREVIEW(MyQueue)         ! deklaracja raportu
             END

  CODE
  OPEN(Report)
  Report{PROP:TempNameFunc} = ADDRESS(PageNames) ! przypisanie adresu do właściwości, tak że
                                                 ! mechanizm raportu wywołuje PageNames do
                                                 ! do pobrani nazwy stosowanej dla każdej strony
  ! tu następuje kod przetwarzania raportu
  Report{PROP:TempNameFunc} = 0                  ! przypisanie zera do właściwości w celu jej wyłączenia
  Report{PROP:FlushPreview} = TRUE               ! wysyła raport do drukarki
                                                 ! a pliki .WMF pozostają cały czs na dysku

PageNames  PROCEDURE(PageNumber)               ! funckcja callback dla nazw stron
NameVar      STRING(260),STATIC
PageFile     FILE,DRIVER('DOS'),NAME(NameVar),CREATE
Rec            RECORD
F1               LONG
               ..

  CODE
  NameVar = PATH() & '\PAGE' & FORMAT(PageNumber,@n04) & '.WMF'
  CREATE(PageFile)
  RETURN(NameVar)
```

PROP:Thread

Zwraca numer wątku okna. Nie jest to konieczne dla aktualnie wykonywanego wątku, jeśli zastosowało się SETTARGET do ustawienia wbudowanej zmiennej TARGET. Tylko-do-odczytu.

Przykład:

```
WinView         WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL,SYSTEM
                END
ToolboxThread   BYTE

  CODE
  OPEN(WinView)
  ToolboxThread = ToolboxWin{PROP:Thread}          ! pobranie numeru wątku okna
  ACCEPT
  END
```

PROP:Threading

Właściwość wbudowanej zmiennej SYSTEM, która, gdy jest ustawiona na zero (0), wyłącza aktywność MDI I przełącza aplikację w tryb SDI.

Przykład:

```
  PROGRAM
  ! deklaracje danych
  CODE
  IF SomeCondition = TRUE
    SYSTEM{PROP:Threading} = 0        ! włączenie interfejsu SDI
  END
```

PROP:TipDelay, PROP:TipDisplay

Właściwość PROP:TipDelay ustawia odstęp czasowy, po jakim będzie wyświetlona podpowiedź w dymku (atrybut TIP) dla SYSTEM (tylko 16-bitów).

Właściwość PROP:TipDisplay ustawia czas trwania wyświetlania podpowiedzi w dymku (atrybut TIP) dla SYSTEM (tylko 16-bitów).

Przykład:

```
WinView   APPLICATION('MyApp'),AT(0,0,320,200),MAX,HVSCROLL,SYSTEM
          END

  CODE
  OPEN(WinView)
  SYSTEM{PROP:TipDelay} = 50        ! opóźnienie wyświetlania TIP o 1/2 sekundy
  SYSTEM{PROP:TipDisplay} = 500     ! wyświetlanie TIP przez 5 sekund
  ACCEPT
  END
```

PROP:Touched

Wartość niezerowa oznacza, że dana w kontrolce ENTRY, TEXT, SPIN lub COMBO posiadającej aktywność wprowadzania, została zmieniona przez użytkownika od momentu ostatniego zdarzenia EVENT:Accepted. Jest ona automatycznie resetowana do zera za każdym razem, gdy kontrolka generuje zdarzenie EVENT:Accepted. Ustawienie tej właściwości (w EVENT:Selected) umożliwia upewnienie się, że zdarzenie EVENT:Accepted jest generowane do wymuszenia wykonania kodu sprawdzającego poprawność danych. W ten sposób przykrywa się standardowe zachowanie Windows – wciśnięcie po prostu klawisza TAB w celu przejścia do następnej kontrolki nie generuje automatycznie zdarzenia EVENT:Accepted.

Właściwość PROP:Touched może być także sprawdzana w celu określenia, czy zawartość pola BLOB zmieniła się od czasu jej wczytania z dysku.

Przykład:

```
WinView   WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            ENTRY(@S30),AT(0,0,320,180),USE(Fil:Field)
            BUTTON('Ok'),AT(100,180,20,20),USE(?Ok)
            BUTTON('Cancel'),AT(200,180,20,20),USE(?Cancel)
          END
SaveCancelPos   LONG,DIM(4)

  CODE
  OPEN(WinView)
  SaveCancelPos[1] = ?Cancel{PROP:Xpos}                          ! zachowanie obszaru przycisku Cancel
  SaveCancelPos[2] = ?Cancel{PROP:Xpos}+?Cancel{PROP:Width}
  SaveCancelPos[3] = ?Cancel{PROP:Ypos}
  SaveCancelPos[4] = ?Cancel{PROP:Ypos}+?Cancel{PROP:Height}
  ACCEPT
    CASE FIELD()
    OF ?Fil:Field
      CASE EVENT()
      OF EVENT:Selected
        ?Fil:Field{PROP:Touched} = 1                             ! wymuszenie generowania zdarzenia EVENT:Accepted
      OF EVENT:Accepted
        IF KEYCODE() = MouseLeft AND |                           ! wykrycie, czy użytkownik kliknął przycisk Cancel
          INRANGE(MOUSEX(),SaveCancelPos[1],SaveCancelPos[2]) AND |
          INRANGE(MOUSEY(),SaveCancelPos[3],SaveCancelPos[4])
          CYCLE                                                  ! użytkownik kliknął Cancel
        ELSE
          ! przetwazranie danych, wprowadzonych przez użytkownika lub do pola przy starcie
        END
      OF ?Ok
        CASE EVENT()
        OF EVENT:Accepted
          ! zapisanie danych na dysku
        END
      OF ?Cancel
        CASE EVENT()
        OF EVENT:Accepted
          ! brak zapisu danych na dysku
        END
      END
    END
```

PROP:Type

Zawiera typ kontrolki. Wartościami są ekwiwalenty CREATE:xxxx (wyszczególnione EQUATES.CLW). Tylko-do-odczytu.

Przykład:

```
MyField    STRING(1)
?MyField   EQUATE(100)
WinView    WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL
           END

  CODE
  OPEN(WinView)
  IF UserChoice = 'CheckField'
    CREATE(?MyField,CREATE:Check)
  ELSE
    CREATE(?MyField,CREATE:Entry)
  END
  ?MyField{PROP:Use} = MyField
  SETPOSITION(?MyField,10,10)
  IF ?MyField{PROP:Type} = CREATE:Check
    ?MyField{PROP:TrueValue} = 'T'
    ?MyField{PROP:FalseValue} = 'F'
  END
  ACCEPT
  END
```

PROP:UpsideDown

Przełącza jednocześnie oba atrybuty UP i DOWN w celu wyświetlenia odwróconego tekstu w zakładce TAB kontrolki SHEET.

Przykład:

```
WinView  WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL
           SHEET,AT(0,0,320,175),USE(SelectedTab),RIGHT,DOWN    ! zakładki po prawej, czytane w dół
             TAB('Tab One'),USE(?TabOne)
               PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P1)
               ENTRY(@S8),AT(100,140,32,20),USE(E1)
               PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P2)
               ENTRY(@S8),AT(100,240,32,20),USE(E2)
             END
             PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P3)
             ENTRY(@S8),AT(100,140,32,20),USE(E3)
             PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P4)
             ENTRY(@S8),AT(100,240,32,20),USE(E4)
           END
         END
         BUTTON('Ok'),AT(100,180,20,20),USE(?Ok)
         BUTTON('Cancel'),AT(200,180,20,20),USE(?Cancel)
       END

  CODE
  OPEN(WinView)
  ?SelectedTab{PROP:BELOW} = TRUE            ! zakładki wyświetlane na spodzie arkusza
  ?SelectedTab{PROP:UpsideDown} = TRUE       ! odwrócenie tekstu wyświetlanego w zakładkach
  ACCEPT
  END
```

PROP:VBXEvent, PROP:VBXEventArg

Właściwość PROP:VBXEvent zwraca nazwę zdarzenia VBX. Tylko-do-odczytu.

Właściwość PROP:VBXEventArg zwraca parametry zdarzenia VBX. Tablica.

Przykład:

```
WinView   WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            CUSTOM,USE(?Graph),CLASS('graph.vbx','graph'),'graphstyle'('2')
          END

  CODE
  OPEN(WinView)
  ACCEPT
    CASE EVENT()
    OF EVENT:VBXEvent
      IF ?Graph{PROP:VBXEvent} = 'FooEvent'                ! sprawdzenie nazwy zdarzenia
        ProcessFoo(?Graph{PROP:VBXEventArg,1},?Graph{PROP:VBXEventArg,2})
        ! pobranie pierwszego I drugiego parametru zdarzenia i przekazanie do przetworzenia przez procedurę
      END
    END
  END
```

PROP:Visible

Zwraca pusty łańcuch jeśli kontrolka nie jest widoczna z tego powodu, że jest ukryta lub że jest ukryta jej kontrolka nadrzędna (OPTION, GROUP, MENU, SHEET lub TAB) bądź też znajduje się w zakładce TAB, która nie jest aktualnie wybrana. Tylko-do-odczytu.

Przykład:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            SHEET,AT(0,0,320,175),USE(SelectedTab)
              TAB('Tab One'),USE(?TabOne)
                PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P1)
                ENTRY(@S8),AT(100,140,32,20),USE(E1)
                PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P2)
                ENTRY(@S8),AT(100,240,32,20),USE(E2)
              END
              TAB('Tab Two'),USE(?TabTwo)
                PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P3)
                ENTRY(@S8),AT(100,140,32,20),USE(E3)
                PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P4)
                ENTRY(@S8),AT(100,240,32,20),USE(E4)
              END
            END
            BUTTON('Ok'),AT(100,180,20,20),USE(?Ok)
            BUTTON('Cancel'),AT(200,180,20,20),USE(?Cancel)
          END
  CODE
  OPEN(MDIChild)
  ACCEPT
    CASE EVENT()
    OF EVENT:Completed
      BREAK
    END
    CASE FIELD()
    OF ?Ok
      CASE EVENT()
      OF EVENT:Accepted
        SELECT
      END
    OF ?E3
      CASE EVENT()
      OF EVENT:Accepted
        E3 = UPPER(E3)                                  ! konwersja wprowadzanych danych na duże litery
        IF ?E3{PROP:Visible} AND MDIChild{PROP:AcceptAll}
                                                        ! sprawdzenie widzialności w trybie AcceptAll
          DISPLAY(?E3)                                  ! i wyświetlenie danej wielkimi literami
        END
      END
    OF ?Cancel
      CASE EVENT()
      OF EVENT:Accepted
        BREAK
      END
    END
  END
```

PROP:VLBproc, PROP:VLBval

Właściwość PROP:VLBProc ustawia procedurę źródłową dla listy "Virtual List Box" kontrolki LIST lub COMBO nie posiadającej atrybutu FROM. Ta procedura dostarcza kontrolkę wraz z danymi do wyświetlenia.

Prototyp procedury musi otrzymywać trzy parametry:

```
VLBProc PROCEDURE(LONG,LONG,SHORT),STRING
```

Gdzie w roli pierwszego parametru typu LONG używa się albo SELF (oznaczające, że procedura jest metodą klasy CLASS) albo wartości ustawionej dla PROP:VLBval. W drugim parametrze LONG przekazuje się numer wiersza "virtual list box", którego dotyczy operacja. Występują trzy "specjalne" wartości dla tego parametru: -1 sprawdza liczbę rekordów do wyświetlenia w liście, -2 sprawdza liczbę pól w nominalnej kolejce Queue (dana i color/drzewo/ikona) do wyświetlenia w liście, -3 sprawdza, czy są jakieś zmiany wymagające wyświetlenia. Parametr SHORT określa numer kolumny "virtual list box", której dotyczy działanie.

Właściwość PROP:VLBVal ustawia obiekt źródłowy dla "Virtual List Box" kontrolki LIST lub COMBO nie posiadającej atrybutu FROM. Może to być dowolna, unikalna wartość 32-bitowa identyfikująca specyficzną listę, ale na ogół jest to wartość zwracana przez ADDRESS(SELF), gdzie procedura PROP:VLBProc jest metoda klasy CLASS.

Przykład:

```
    PROGRAM
    MAP
Main
    END

StripedListQ  QUEUE,TYPE
S               STRING(20)
              END
StripedList   CLASS,TYPE
Init            PROCEDURE(WINDOW w, SIGNED feq, StripedListQ Q)
VLBproc         PROCEDURE(LONG row, SHORT column),STRING,PRIVATE
                ! wymagany pierwszy parametr jest implikowany w metodzie klasy
Q               &StripedListQ,PRIVATE
ochanges        LONG,PRIVATE
              END

    CODE
    Main

StripedList.Init  PROCEDURE(WINDOW w, SIGNED feq, StripedListQ Q)
    CODE
    SELF.Q &= Q
    SELF.ochanges = CHANGES(Q)
    w $ feq{PROP:VLBval} = ADDRESS(SELF)              ! najpierw trzeba przypisać to
    w $ feq{PROP:VLBproc} = ADDRESS(SELF.VLBproc)     ! potem - to

StripedList.VLBproc  PROCEDURE(LONG row, SHORT col)  ! wymagany pierwszy parametr
nchanges    LONG
    CODE
    CASE row
    OF -1                                             ! ile wierszy?
      RETURN RECORDS(SELF.Q)
    OF -2                                             ! ile kolumn?
      RETURN 5                                        ! 1 dana, cztery pola koloru w "nominal Q"
    OF -3                                             ! zmieniło się
```

```
    nchanges = CHANGES(SELF.Q)
    IF nchanges <> SELF.ochanges THEN
      SELF.ochanges = nchanges
      RETURN 1
    ELSE
      RETURN 0
    END
  ELSE
    GET(SELF.Q, row)
    CASE col
    OF 1                                                    ! pole danej
      RETURN WHAT(SELF.Q,1)
    OF 3                                                    ! pole koloru tla
      RETURN CHOOSE(BAND(row,1), COLOR:none, 0c00000H)
    ELSE                                                    ! wszystkie pozostałe pola
      RETURN COLOR:None                                     ! stosuj kolor domyślny
    END
  END

Main   PROCEDURE
window    WINDOW('Caption'),AT(,,153,103),GRAY
            LIST,AT(33,12,80,80),USE(?List1),FORMAT('20L*')
          END

Q     QUEUE(StripedListQ)
      END
SL    StripedList
i     SIGNED

  CODE
  LOOP i = 1 TO 20
    Q.s = 'Line ' & i
    ADD(Q)
  END
  OPEN(window)
  SL.Init(window, ?list1, Q)
  ACCEPT
  END
```

PROP:VscrollPos

Zwraca pozycję suwaka paska pionowego. W oknie i w kontrolkach IMAGE oraz TEXT, które posiadają atrybut VSCROLL prawidłowym zakresem jest od 0 do 255. W kontrolkach LIST oraz COMBO, które posiadają atrybut VSCROLL prawidłowym zakresem jest od 0 do 100. Ustawienie to daje możliwość przewijania zawartości okna lub kontrolki w pionie (o ile nie występuje atrybut IMM w kontrolce LIST lub COMBO, wtedy porusza się tylko sam suwak).

Przykład:

```
Que    QUEUE
         STRING(50)
       END

WinView  WINDOW('View'),AT(0,0,320,200),MDI,SYSTEM
           LIST,AT(0,0,320,200),USE(?List),FROM(Que),IMM,VSCROLL
         END
  CODE
  OPEN(WinView)
  Fil:KeyField = 'A' ;DO BuildListQue
  ACCEPT
    CASE FIELD()
    OF ?List
      CASE EVENT()
      OF EVENT:ScrollDrag
        EXECUTE INT(?List{PROP:VscrollPos}/10) + 1
        Fil:KeyField = 'A'
        Fil:KeyField = 'C'
        Fil:KeyField = 'E'
        Fil:KeyField = 'G'
        Fil:KeyField = 'K'
        Fil:KeyField = 'M'
        Fil:KeyField = 'P'
        Fil:KeyField = 'S'
        Fil:KeyField = 'V'
        Fil:KeyField = 'Y'
      END
      DO BuildListQue
  ...
  FREE(Que)

BuildListQue  ROUTINE
  FREE(Queue)
  SET(Fil:SomeKey,Fil:SomeKey)                    ! ustawienie na wybrane pole klucza
  LOOP ?List{PROP:Items} TIMES                    ! przetwarzanie rekordów widocznych w liście
    NEXT(SomeFile) ;IF ERRORCODE() THEN BREAK.    ! przerwanie, jeśli koniec pliku
    Que = Fil:KeyField                            ! przypisanie pola do wyświetlanie
    ADD(Que)                                      ! i dodanie go do kolejki
  END
```

PROP:WndProc

Ustawia lub pobiera procedurę obsługi komunikatów okna (nie jego obszaru roboczego) lub określonej kontrolki do zastosowania w wywołaniu funkcji Windows API. Stosuje się na ogół, gdy wystąpi potrzeba śledzenia wszystkich komunikatów Windows.

Przykład:

```
  PROGRAM
  MAP
Main              PROCEDURE
SubClassFunc1     PROCEDURE(USHORT,SHORT,USHORT,LONG),LONG,PASCAL
SubClassFunc2     PROCEDURE(USHORT,SHORT,USHORT,LONG),LONG,PASCAL

  MODULE('Windows')                       ! biblioteka Win31 TopSpeed
CallWindowProc    PROCEDURE(LONG,UNSIGNED,SIGNED,UNSIGNED,LONG),LONG,PASCAL
  . .                                     ! koniec MAP i MODULE

SavedProc1        LONG
SavedProc2        LONG
WM_MOUSEMOVE      EQUATE(0200H)
PT                GROUP
X                   SHORT
Y                   SHORT
                  END
  CODE
  Main

Main        PROCEDURE
WinView       WINDOW('View'),AT(0,0,320,200),HVSCROLL,MAX,TIMER(1),STATUS
                STRING('X Pos'),AT(1,1,,),USE(?String1)
                STRING(@n3),AT(24,1,,),USE(PT:X)
                STRING('Y Pos'),AT(44,1,,),USE(?String2)
                STRING(@n3),AT(68,1,,),USE(PT:Y)
                BUTTON('Close'),AT(240,180,60,20),USE(?Close)
              END
  CODE
  OPEN(WinView)
  SavedProc1 = WinView{PROP:WndProc}                          ! zachowaj tę procedurę
  WinView{PROP:WndProc} = ADDRESS(SubClassFunc1)              ! wskaż procedurę zastępującą
  SavedProc2 = WinView{PROP:ClientWndProc}                    ! zachowaj tę procedurę
  WinView{PROP:ClientWndProc} = ADDRESS(SubClassFunc2) ! wskaż procedurę zastępującą
  ACCEPT
    CASE ACCEPTED()
    OF ?Close
      BREAK
  ..

SubClassFunc1   PROCEDURE(hWnd,wMsg,wParam,lParam)    ! procedura zastepująca
  CODE                                                ! do śledzenia ruchu myszki w
  IF wMsg = WM_MOUSEMOVE                              ! pasku stanu okna (tylko)
    PT.X = MOUSEX() ; PT.Y = MOUSEY()                 ! przypisanie pozycji myszki
  END
  RETURN(CallWindowProc(SavedProc1,hWnd,wMsg,wParam,lParam))
                                                      ! przekazanie sterowania do SavedProc1
SubClassFunc2   PROCEDURE(hWnd,wMsg,wParam,lParam)    ! procedura zastepująca
  CODE                                                ! do śledzenia ruchu myszki w
  IF wMsg = WM_MOUSEMOVE                              ! obszarze roboczym okna
    PT.X = MOUSEX() ; PT.Y = MOUSEY()                 ! przypisanie pozycji myszki
  END
  RETURN(CallWindowProc(SavedProc2,hWnd,wMsg,wParam,lParam))!Pass control to SavedProc2
```

Właściwości runtime widoku VIEW i pliku FILE

PROP:ConnectString

Właściwość pliku FILE stosującego sterownik ODBC, która zwraca łańcuch połączenia (standardowo przechowywany w atrybucie OWNER pliku) pozwalający na jego zestawienie. Jeśli atrybut OWNER zawiera tylko nazwę źródła danych, pojawia się okienko dialogowe logowania umożliwiające wprowadzenie pozostałych informacji koniecznych do zestawienia połączenia. Okno logowania pojawia się za każdym razem, gdy logujemy się do bazy danych. Dzięki tej właściwości, informacja w dialogu logowania może być wprowadzona tylko raz i zapamiętana w atrybucie OWNER, a następnie zwracana co pozwoli na wyeliminowanie dialogu logowania przy następnym logowaniu się do bazy danych.

Przykład:

```
OwnerString  STRING(20)
Customer     FILE,DRIVER('ODBC'),OWNER(OwnerString)
Record         RECORD
Name             STRING(20)
               . .
  CODE
  OwnerString = 'DataSourceName'
  OPEN(Customer)
  OwnerString = Customer{PROP:ConnectString}        ! pobierz pełny łańcuch połączenia
  MESSAGE(OwnerString)                              ! wyświetl go do późniejszego użycia
```

PROP:CurrentKey

Właściwość pliku FILE zwracająca referencje na bieżący klucz KEY lub INDEX używany w przetwarzaniu sekwencyjnym lub na bieżący klucz budowany za pomocą operacji BUILD lub PACK Tylko-do-odczytu. Poprawne zastosowanie wymaga użycia tej właściwości jako części źródłowej operacji przypisania referencyjnego lub w wyrażeniu logicznym porównującym zwracany rezultat z NULL. Zwraca NULL jeśli plik jest przetwarzany w porządku fizycznym.

Przykład:

```
KeyRef      &KEY
Customer    FILE,DRIVER('Clarion'), PRE(Cus)
NameKey       KEY(Cus:Name),DUP
Record        RECORD
Name            STRING(20)
              . .

  CODE
  OPEN(Customer)
  SET(Customer)
  KeyRef &= Customer{PROP:CurrentKey)               ! zwraca NULL
  IF Customer{PROP:CurrentKey) &= NULL              ! porównuje z NULL
    MESSAGE('SET to record order')
  END
  SET(Cus:NameKey)
  KeyRef &= Customer{PROP:CurrentKey)               ! zwraca referencję do Cus:NameKey
```

PROP:DriverLogsoutAlias

Właściwość pliku FILE określająca, czy sterownik pliku pozwala na użycie instrukcji LOGOUT stosującej jednocześnie plik i jego alias. Tylko-do-odczytu..

Przykład:

```
IF Customer{PROP:DriverLogsoutAlias} = ''          ! sprawdzenie, czy alias jest dopuszczalny w LOGOUT
  MESSAGE('Driver does not allow files i their aliases in LOGOUT')
END
```

PROP:FetchSize

Właściwość pliku FILE lub widoku VIEW ustawiająca lub pobierająca parametr *pagesize* dla ostatnio wykonanej instrukcji BUFFER.

Przykład:

```
CODE
OPEN(MyFile)
BUFFER(MyFile,10,5,2,300)      ! 10 rekordów na stronę, 5 stron z tyłu, 2 z przodu,
                               ! z 5-cio minutowym timeoutem
MyFile{PROP:FetchSize} = 1     ! zmiana współczynnika pobierania do jednego rekordu jednocześnie
```

PROP:File

Właściwość tablicowa widoku VIEW. Każdy element tablicy zwraca referencję do pliku numerowanego w widoku VIEW. Referencja ta może być stosowana jako część źródłowa operacji przypisania referencyjnego. Pliki są numerowane w widoku VIEW poczynając od 1 (podstawowy plik widoku VIEW) i kolejno dla każdego JOIN, w kolejności występowania w strukturze VIEW. Tylko-do-odczytu.

PROP:Files

Właściwość widoku VIEW zwracająca całkowitą liczbę plików tworzących widok. Równoważne z całkowitą liczbą struktur JOIN powiększoną o jeden (podstawowy plik widoku VIEW). Tylko-do-odczytu.

Przykład:

```
AView VIEW(BaseFile)                                                    ! plik 1
        JOIN(ParentFile,'BaseFile.parentID = ParentFile.ID')            ! plik 2
          JOIN(GrandParent.PrimaryKey, ParentFile.GrandParentID)        ! plik 3
          END
        END
        JOIN(OtherParent.PrimaryKey,BaseFile.OtherParentID)             ! plik 4
        END
      END ! AView{PROP:Files} zwraca 4

  ! AView{PROP:File,1} zwraca referencję do BaseFile
  ! AView{PROP:File,2} zwraca referencję do Parent
  ! AView{PROP:File,3} zwraca referencję do GrandParent
  ! AView{PROP:File,4} zwraca referencję do OtherParent

FilesQ   QUEUE
FileRef    &FILE
        END

  CODE
  LOOP X# = 1 TO AView{PROP:Files}          ! powtórz 4 razy
    FilesQ.FileRef &= AView{PROP:File,X#}   ! przypisz referencję dla każdego pliku widoku
    ADD(FilesQ)                             ! i dodaj go do kolejki
    ASSERT(~ERRORCODE())                    ! przyjmij, że brak błędu
    CLEAR(FilesQ)                           ! wyczyśc kolejkę dla  następnego przypisania
  END
```

PROP:GlobalHelp

Właściwość SYSTEM która, gdy jest ustawiona, powoduje wyłączenie automatycznego zamykania pliku pomocy. HLP w momencie zamknięcia okna, które go otworzyło. Powoduje to, że plik pomocy jest cały czas wyświetlany na ekranie, do momentu zamknięcia go przez użytkownika.

Przykład:

```
SYSTEM{PROP:GlobalHelp} = TRUE          ! wyłącz automatyczne zamykanie pliku pomocy
```

PROP:Held

Właściwość pliku FILE określająca, czy bieżący rekord jest zablokowany. Zwraca 1, jeśli tak, bądź pusty łańcuch (‘’), jeśli nie. Tylko-do-odczytu.

Przykład:

```
FileName    STRING(256)
Customer    FILE,DRIVER('Clarion')
Record        RECORD
Name            STRING(20)
              . .

  CODE
  OPEN(Customer)
  SET(Customer)
  LOOP
    HOLD(Customer,1)
    NEXT(Customer)
    IF ERRORCODE() THEN BREAK.
    IF Customer{PROP:Held} <> ''
      MESSAGE('Record Held')
    END
  END
```

PROP:Logout

Właściwość pliku FILE przypisująca lub zwracająca poziom priorytetu w transakcji. Właściwość PROP:Logout może zostać użyta do zbudowania listy plików w transakcji przed wydaniem polecenia LOGOUT(*seconds*), które zaczyna transakcję. Stosując właściwość PROP:Logout można włączyć do transakcji więcej plików, niż pozwala na to ograniczona liczba parametrów instrukcji LOGOUT. Jeśli instrukcja LOGOUT sama wymienia listę plików, wszystkie pliki ustawione wcześniej do transakcji za pośrednictwem PROP:Logout są z niej odrzucane; pod uwagę brane są jedynie pliki wymienione w instrukcji LOGOUT.

Poziom priorytetu określa kolejność, w jakiej pliki są rejestrowane w transakcji, pliki o niższych numerach są rejestrowane wcześniej. Jeśli dwa pliki posiadają ten sam poziom priorytetu, są rejestrowane w kolejności, w jakiej zostały włączone do listy. Przypisanie dodatniego poziomu priorytetu dołącza plik FILE do transakcji, przypisanie ujemnego poziomu priorytetu usuwa plik z transakcji, przypisanie zera (0) nie daje żadnego efektu. Sprawdzenie właściwości PROP:Logout pozwala na określenie poziomu priorytetu nadanego plikowi. Wartość zero (0) oznacza, że plik nie wchodzi do transakcji.

Próba zastosowania właściwości PROP:Logout do dołączenia do transakcji pliku stosującego odmienny sterownik pliku powoduje powstanie błędu ERRORCODE 48 - “Unable to log transaction.”

Przykład:

```
Customer    FILE,DRIVER('TopSpeed')
Record        RECORD
CustNumber      LONG
Name            STRING(20)
                . .

Orders      FILE,DRIVER('TopSpeed')
Record        RECORD
CustNumber      LONG
OrderNumber     LONG
OrderDate       LONG
                . .

Items       FILE,DRIVER('TopSpeed')
Record        RECORD
OrderNumber     LONG
ItemNumber      LONG
                . .

  CODE
  Customer{PROP:Logout} = 1        ! dodaj plik Customer do ramki transakcji i ustaw priorytet na 1
  Items{PROP:Logout} = 2           ! dodaj plik Items do ramki transakcji i ustaw priorytet na 2
  Orders{PROP:Logout} = 1          ! dodaj plik Orders do ramki transakcji i ustaw priorytet na 1
  X# = Items{PROP:Logout}          ! zwróć poziom priorytetu dla pliku Items (X# = 2)
  Customer{PROP:Logout} = -1       ! usuń plik Customer z ramki transakcji
  LOGOUT(1)                        ! rozpocznij transakcje i
                                   ! zaloguj pliki w nastepującej kolejności: Orders, Items
  COMMIT                           ! zakończ transakcję
```

PROP:Profile

Właściwość pliku FILE przełączająca rejestrowanie (profilowanie) wszystkich wywołań I/O i błędów zapisywanych przez sterownik pliku w specjalnym pliku tekstowym. Przypisanie do właściwości PROP:Profile nazwy pliku inicjuje profilowanie, podczas gdy przypisanie pustego łańcucha (‘’) wyłącza je. Sprawdzenie tej właściwości pozwala na określenie nazwy aktualnego pliku, w którym są rejestrowane operacje, jeśli rezultatem jest łańcuch pusty (‘’) – profilowanie jest wyłączone.

PROP:Log

Właściwość pliku FILE zapisująca łańcuch do bieżącego pliku rejestracji (przypisanego do PROP:Profile). Łańcuch ten jest umieszczany w oddzielnym wierszu w pliku. Tylko-do-zapisu.

Przykład:

```
FileName   STRING(256)
Customer   FILE,DRIVER('TopSpeed')
Record       RECORD
Name         STRING(20)
           . .

  CODE
  Customer{PROP:Profile} = 'CustLog.TXT'          ! włącz profilowanie, plik wyjściowy: CustLog.TXT
  OPEN(Customer)
  Filename = Customer{PROP:Profile}               ! pobierz nazwę bieżącego pliku rejestrowania
  Customer{PROP:Log} = CLIP(FileName) & ' ' & |
  FORMAT(TODAY(),@D2) & ' ' & |
  FORMAT(CLOCK(),@T1)
      ! zapisz wiersz tekstu do pliku rejestrowania
  SET(Customer)
  LOOP
    NEXT(Customer)                                ! wszystkie operacje I/O pliku są rejestrowane
    IF ERRORCODE() THEN BREAK.                    ! w pliku CustLog.TXT
  END
  Customer{PROP:Profile} = ''                     ! wyłącz profilowanie
```

PROP:ProgressEvents, PROP:Completed

PROP:ProgressEvents jest właściwością pliku FILE generującą zdarzenia dla aktualnie otwartego okna, podczas wykonywania operacji BUILD lub PACK. Tylko-do-zapisu. Właściwość ta zależy od sterownika pliku, dlatego należy dodatkowo sprawdzić dokumentację stosowanego sterownika, czy ja obsługuje.

Przypisanie wartości zero (0) wyłącza generowanie zdarzeń dla następnej operacji BUILD lub PACK, podczas gdy przypisanie dowolnej innej wartości (prawidłowy zakres to: od 1 do 100) włącza generowanie zdarzeń. Przypisanie wartości leżącej poza dopuszczalnym zakresem jest traktowane w sposób następujący: zamiast liczby ujemnej jest przyjmowana wartość jeden (1), dowolna wartość większa od sto (100) jest traktowana jako liczba sto (100). Im większa jest przypisana wartość, tym więcej zdarzeń jest generowanych I tym wolniej działa operacja BUILD lub PACK.

Generowanymi zdarzeniami są: EVENT:BuildFile, EVENT:BuildKey oraz EVENT:BuildDone. Nie jest właściwe: wykonywanie jakichkolwiek wywołań pliku FILE podczas odbudowy kluczy, poza sprawdzaniem jego właściwości, wywołanie NAME(*file*) lub CLOSE(*file*) (co przerywa proces i nie jest zalecane). Wydanie polecenia CYCLE w odpowiedzi na dowolne z wygenerowanych zdarzeń (za wyjątkiem EVENT:BuildDone) anuluje operację.

Do pobrania referencji do aktualnie odbudowywanego indeksu może być stosowana właściwość PROP:CurrentKey, wczytanie nazwy tego indeksu umożliwia właściwość PROP:Label.

Właściwość PROP:Completed pliku FILE określa procent wykonania operacji BUILD lub PACK, dla której włączono PROP:ProgressEvents. Właściwość zwraca zero (0) jeśli sterownik pliku nie jest w stanie określić, jaka część operacji BUILD lub PACK została już wykonana. Tylko-do-odczytu.

Przykład:

```
    PROGRAM
    MAP.
    INCLUDE('ERRORS.CLW')

Test    FILE,DRIVER('TOPSPEED','/FULLBUILD=ON'),CREATE,PRE(TEST)
K1        KEY(Test:Xval)
          RECORD
Xval        LONG
          ..

counter       LONG
CurrentKey    &KEY
cancelling    BYTE(FALSE)
BuildDone     BYTE(FALSE)
Completed     LONG(1)
CurEvent      LONG

window      WINDOW('Time Slicing Build Example'),AT(,,127,68),SYSTEM,GRAY
              STRING('Building'),AT(9,6),USE(?BuildStr)
              STRING(''),AT(39,6),USE(?Name)
              PROGRESS,USE(counter),AT(9,25,107,8),RANGE(0,100)
              BUTTON('&Cancel'),AT(82,45),USE(?Cancel),DISABLE
            END

  CODE
  OPEN(Test)
  IF ERRORCODE()
    CREATE(Test); OPEN(Test); STREAM(Test)
    LOOP 20000 TIMES
```

```
      Test.Xval = X#; X# += 1; APPEND(Test)
    END
    FLUSH(Test)
  END
  OPEN(window)
  ACCEPT
    CurEvent = EVENT()
    CASE CurEvent
    OF EVENT:OpenWindow
      Test{PROP:ProgressEvents} = 100                    ! włącz generowanie zdarzeń
      BUILD(Test)
      ENABLE(?Cancel)
    OF EVENT:Accepted
      IF ACCEPTED() = ?Cancel
        IF BuildDone THEN BREAK.
        IF MESSAGE('Cancelling build leaves file unusable. Cancel Anyway?','Warning', |
          ICON:Exclamation,BUTTON:Yes+BUTTON:No,BUTTON:No) = BUTTON:Yes
          Cancelling = TRUE
          ?BuildStr{PROP:Text} = 'Please Wait. Cancelling Build'
          ?Name{PROP:Text} = ''
          DISPLAY(?BuildStr,?Name)
        END
      END
    OF EVENT:BuildFile
    OROF EVENT:BuildKey                                  ! przetwarzaj zdarzenia BUILD
      IF Cancelling = TRUE; DO Done; CYCLE.
      IF CurEvent = EVENT:BuildKey
        CurrentKey &= Test{PROP:CurrentKey}              ! pobierz referencję do bieżacego klucza
        IF NOT (CurrentKey &= NULL)
          ?Name{PROP:Text} = CurrentKey{PROP:Label}  ! wyświetl nazwę klucza
        END
      ELSE
        ?Name{PROP:Text} = NAME(Test)
      END
      IF Completed <> 0; Completed = Test{PROP:Completed}.      ! pobierz procent wykonania
        IF Completed = 0
          counter += 10
          IF (counter>100) THEN counter = 0.
        ELSE
          counter = Completed
        END
        DISPLAY(?Name,?Counter)
      OF EVENT:BuildDone
        DO Done
      END
    END
    OPEN(Test)
    IF ERRORCODE() = BadKeyErr THEN MESSAGE(NAME(Test) & ' BUILD failed' ).

Done    ROUTINE
  BuildDone = TRUE
  ?Cancel{PROP:Text} = '&OK'
  CLOSE(Test)
```

PROP:SQLDriver

Właściwość pliku FILE zwracająca ‘1’ jeśli sterownik akceptuje SQL lub łańcuch pusty (‘’) – w przeciwnym wypadku. Tylko-do-odczytu.

Przykład:

```
Customer FILE,DRIVER('Clarion'),PRE(CUS)
Record       RECORD
Name           STRING(20)
             . .

SQLFlag  BYTE

  CODE
  IF Customer{PROP:SQLDriver} THEN SQLFlag = TRUE.
```

PROP:Text

Właściwość tablicowa pliku FILE ustawiająca lub zwracająca dane określonego pola MEMO. Kontrolki MEMO są numerowane ujemnie, z tego względu numer w elemencie tablicy musi być wartością ujemną.

Przykład:

```
MemoText  STRING(2000)
Customer  FILE,DRIVER('Clarion'),PRE(CUS)
Notes       MEMO(2000)
Record      RECORD
Name          STRING(20)
            . .

  CODE
  OPEN(Customer)
  SET(Customer)
  NEXT(Customer)
  ASSERT(~ERRORCODE())
  Memotext = Customer{PROP:Text,-1}
```

PROP:Value

Właściwość tablicowa pliku FILE, która ustawia lub zwraca daną zawartą w określonym polu MEMO (w przypadku pól pozostałych typów stosujemy procedurę WHAT). Element tablicy dla PROP:Value jest po prostu liczbą ujemną identyfikująca pole MEMO o numerze -n.

Przykład:

```
Text        STRING(2000)
Number      LONG
Customer    FILE,DRIVER('TopSpeed'),PRE(CUS)
Notes         MEMO(2000)
Record        RECORD
Number          LONG,DIM(20)
Name            STRING(20)
              . .

  CODE
  OPEN(Customer)
  SET(Customer)
  NEXT(Customer)
  ASSERT(~ERRORCODE())
  Text = Customer{PROP:Value,-1}              ! pobierz zawartość CUS:Notes
```

PROP:Watched

Właściwość pliku FILE określająca, czy bieżący rekord jest śledzony za pomocą WATCH. Zwraca 1 – jeśli tak lub łańcuch pusty (‘’) – jeśli nie. Tylko-do-odczytu.

Przykład:

```
FileName    STRING(256)
Customer    FILE,DRIVER('Clarion')
Record        RECORD
Name            STRING(20)
              . .

  CODE
  OPEN(Customer)
  SET(Customer)
  LOOP
    WATCH(Customer)
    NEXT(Customer)
    IF ERRORCODE() THEN BREAK.
    IF Customer{PROP:Watched} <> ''
      MESSAGE('Record watched')
    END
  END
```

Wbudowany SQL

PROP:SQL

Składnia właściwości może być wykorzystywana do wykonywania instrukcji języka SQL w kodzie programu. Służy do tego właściwość PROP:SQL określana dla konkretnego pliku FILE lub widoku VIEW. Dotyczy to oczywiście tylko plików posługujących się sterownikiem SQL (np. ODBC, Scalable SQL lub Oracle). Jest również możliwe sprawdzanie zawartości właściwości PROP:SQL w celu sprawdzenia ostatniego polecenia SQL wydanego przez sterownik pliku.

Możemy osadzać w kodzie dowolne instrukcje SQL obsługiwane przez serwer SQL, z którym współpracujemy. Jeśli wydamy polecenie SQL, którego rezultatem ma być zbiór rekordów (np. instrukcja SELECT), musimy wykonać NEXT(file) w celu wczytania zbioru wynikowego (po jednym wierszu) do bufora rekordu pliku. Deklaracja FILE rejestrująca zbiór wynikowy musi posiadać taką samą liczbę pól, jaką zwraca instrukcja SELECT. Procedury FILEERRORCODE() oraz FILEERROR() pozwalają na odczytanie dowolnego kodu i komunikatu błędu ustawionego przez serwer SQL, jeśli procedura ERRORCODE zwraca błąd o kodzie 90.

Przykład:

```
SQLFile{PROP:SQL} = 'SELECT field1,field2 FROM table1' |
  & 'WHERE field1 > (SELECT max(field1)'|
  & 'FROM table2'                                    ! zwraca zbiór wynikowy, z którego pobieramy jeden
                                                     ! wiersz jednocześnie za pomocą NEXT(SQLFile)
SQLFile{PROP:SQL} = 'CALL GetRowsBetween(2,8)'       ! wywołanie procedury składowanej
IF ERRORCODE() THEN STOP(FILEERROR()).               ! kontrola błędów
SQLFile{PROP:SQL} = 'CREATE INDEX ON table1 (field1, field2 DESC)'       ! brak zbioru wynikowego
SQLString = SQLFile{PROP:SQL}                        ! pobranie ostatniego polecenia SQL wydanego
                                                     ! przez sterownik pliku
```

PROP:SQLFilter

Właściwość PROP:SQLFilter może zostać wykorzystana do filtrowania widoków VIEW w oparciu o natywny kod SQL zamiast kodu Clarion. Można ją oczywiście stosować tylko dla plików posługujących się sterownikiem SQL (np. ODBC, Scalable SQL lub Oracle). Jeśli pierwszy znak wyrażenia PROP:SQLFilter jest znakiem plus (+), właściwość PROP:SQLFilter jest dołączana do istniejącego wyrażenia PROP i są one używane równocześnie. Pominięcie znaku plus zastępuje istniejące wyrażenie PROP:Filter wyrażeniem PROP:SQLFilter. Jeśli stosujemy PROP:SQLFilter, filtr SQL jest przekazywany bezpośrednio do serwera. Jako taki, nie może zawierać nazw procedur, zmiennych, ponieważ serwer SQL nic o nich nie wie.

Przykład:

```
View{PROP:SQLFilter} = 'DateField = TO_DATE("01-MAY-1995","DD-MON-YYYY")'
! zastępuje dowolne wyrażenie PROP:Filter
View{PROP:SQLFilter} = '+StrField LIKE "AD%"'
! dodane do wyrażenia PROP:Filter
```

DODATEK D – KODY BŁĘDÓW

Błędy działania aplikacji

Przechwytywalne błędy działania aplikacji

Poniższe błędy mogą być przechwytywane w kodzie aplikacji za pomocą funkcji ERRORCODE oraz ERROR. Każdy błąd posiada własny kod (zwracany jako rezultat funkcji ERRORCODE) i przypisany do niego komunikat tekstowy (zwracany przez funkcje ERROR) opisujący rodzaj problemu.

2 File Not Found

Plik nie istnieje we wskazanym katalogu.

3 Path Not Found

Określony katalog wskazany w ścieżce nie istnieje.

4 Too Many Open Files

Całkowita liczba dostępnych uchwytów plików jest już wykorzystywana. Sprawdź ustawienie FILES= w pliku CONFIG.SYS, bądź liczbę równolegle otwartych plików użytkownika lub sieci w środowisku sieciowym

5 Access Denied

Plik został już otwarty przez innego użytkownika w trybie wyłącznego dostępu, został pozostawiony w stanie zablokowania lub też nie posiadasz uprawnień sieciowych do jego otwarcia. Ten błąd może wystąpić również wtedy, gdy nie ma dostępnej odpowiedniej liczby wolnego miejsca na dysku.

6 Memory Corrupted

Wystąpił nieznany błąd pamięci.

7 Insufficient Memory

Pozostało zbyt mało wolnej pamięci na wykonanie danej operacji. Zamkniecie innych aplikacji może zwolnić niezbędną ilość pamięci. W Btrieve komunikat ten oznacza, że nie dysponujemy dostateczną ilością pamięci trybu rzeczywistego na załadowanie BTR32.EXE. W Windows 95 ładowanie WBTR32.EXE z poziomu WINSTART.BAT może rozwiązać ten problem.

15 Invalid Drive

Próba odczytu z nieistniejącego napędu dyskowego.

30 Entry Not Found

Operacja pobrania GET z kolejki QUEUE zakończyła się niepowodzeniem. Dla GET(Q,*key*) nie została odnaleziona pasująca wartość klucza *key*, dla GET(Q,*pointer*) pozycja *pointer* jest poza zakresem.

32 File Is Already Locked

Próba zablokowania LOCK pliku zakończyła się niepowodzeniem ponieważ inny użytkownik już zablokował ten plik.

33 Record Not Available

Zazwyczaj jest to próba odczytu za ostatnim lub przed pierwszym rekordem pliku za pomocą NEXT lub PREVIOUS. Może być także wysyłany przez PUT bądź DELETE, gdy nie został odczytany rekord przed wykonaniem takiej instrukcji

35 Record Not Found

Dla GET(File,*key*) nie została odnaleziona pasująca wartość pola klucza *key*.

36 Invalid Data File

Wystąpiło nieznane uszkodzenie pliku danych lub atrybut OWNER nie jest zgodny z hasłem szyfrowania pliku.

37 File Not Open

Próba wykonania operacji wymagającej, by plik był już otwarty zakończyła się niepowodzeniem, ponieważ plik nie jest otwarty.

38 Invalid Key File

Wystąpiło nieznane uszkodzenie klucza pliku.

40 Creates Duplicate Key

Próba dopisania ADD lub aktualizacji PUT rekordu z wartościami pól powodującymi powstanie duplikatu indeksu (rekord o takich wartościach już istnieje), na co nie pozwala definicja tego indeksu.

43 Record Is Already Held

Próba zablokowania HOLD rekordu zakończyła się niepowodzeniem, ponieważ został on już zablokowany przez innego użytkownika.

45 Invalid Filename

Nazwa pliku nie spełnia wymagań systemu DOS.

46 Key File Must Be Rebuilt

Wystąpiło nieznane uszkodzenie indeksu wymagające odbudowania indeksu za pomocą instrukcji BUILD.

47 Invalid Record Declaration

Struktura fizycznego pliku danych nie jest zgodna z deklaracją w pliku .EXE. powodem jest zazwyczaj zmiana definicji pliku w Data Dictionary i nie wykonanie operacji konwersji istniejących danych do nowego formatu.

48 Unable To Log Transaction

Logout transakcji lub plik pre-image nie mógł zostać zapisany na dysk. Zdarza się to zazwyczaj w przypadku braku dostatecznej ilości wolnego miejsca na dysku lub gdy użytkownik nie posiada odpowiednich uprawnień sieciowych do wykonania takiej operacji.

52 File Already Open

Próba otwarcia OPEN pliku, który już został otwarty przez danego użytkownika.

54 No Create Attribute

Próba wykonania procedury CREATE na pliku, którego deklaracja nie posiada atrybutu CREATE.

55 File Must Be Shared

Próba otwarcia w trybie wyłącznym pliku, który musi być współdzielony (szablony standardowe, nie jest więcej używany).

56 LOGOUT Already Active

Próba wykonania drugiej instrukcji LOGOUT podczas gdy transakcja jest nadal w fazie przetwarzania.

57 Invalid Memo File

Wystąpił nieznany błąd pliku memo. W przypadku plików formatu Clarion może to być spowodowane uszkodzeniem sygnatury pliku .MEM lub wskaźników do pliku memo w pliku danych znajdujących się w stanie „out of sync" (zazwyczaj spowodowane kopiowaniem pliku z jednej lokalizacji do innej i użycie niewłaściwych plików .MEM).

63 Exclusive Access Required

Próba wykonania BUILD(file), BUILD(key), EMPTY(file) lub PACK(file) została przeprowadzona, gdy plik nie został otwarty w trybie wyłącznym.

64 Sharing Violation

Próba wykonania działania na pliku wymagającego otwarcia pliku w trybie współdzielonym.

65 Unable To ROLLBACK Transaction

Próba cofnięcia ROLLBACK transakcji zakończyła się niepowodzeniem z nieznanych przyczyn.

73 Memo File Missing

Próba otwarcia OPEN pliku, który został zadeklarowany z polem MEMO, a dla którego nie wykryto istnienia pliku memo.

75 Invalid Field Type Descriptor

Deskryptor typu jest uszkodzony, przy użyciu nazwy *name* nie istniejącej w GET(Q,*name*), bądź też definicja pliku nie jest prawidłowa dla danego sterownika pliku. Na przykład, próba zdefiniowania pola LONG w pliku xBase bez wstawienia powiązanego z nim pola MEMO.

76 Invalid Index String

Nieprawidłowy łańcuch indeksowy *string* przekazany do BUILD(DynIndex,*string*).

77 Unable To Access Index

Próba wczytania rekordów w oparciu o indeks dynamiczny zakończyła się niepowodzeniem, ponieważ nie został on odnaleziony.

78 Invalid Number Of Parameters

Nie została przekazana właściwa liczba parametrów do procedury wywołanej w instrukcji EVALUATE.

79 Unsupported Data Type In File

Sterownik pliku wykrył w pliku pole zadeklarowane z typem danych, który nie jest obsługiwany przez system plików, dla którego jest przeznaczony sterownik.

80 Unsupported File Driver Function

Sterownik pliku wykrył instrukcję dostępu do pliku, która nie jest obsługiwana. Jest to często nieobsługiwana forma (odmienne parametry) instrukcji obsługiwanej przez sterownik.

81 Unknown Error Posted

Sterownik pliku wykrył pewne błędy w systemie plików, o których nie może pobrać dodatkowych informacji.

88 Invalid Key Length

Próba utworzenia CREATE indeksu KEY lub INDEX pliku formatu Clarion o więcej niż 245 znakach. Inne sterowniki plików też mogą zgłaszać ten błąd, gdy zostaną przekroczone obowiązujące dla nich ograniczenia nałożone na długość indeksów.

89 Record Changed By Another Station

Instrukcja WATCH wykryła, że rekord na dysku nie jest zgodny z jego oryginalną, wersją zachowaną przed dokonaniem operacji aktualizacji. Oznacza to, że rekord został zmieniony przez innego użytkownika sieci.

90 File Driver Error

Sterownik pliku wykrył pewne błędy raportowane przez system plików. Można zastosować procedury FILEERRORCODE i FILEERROR do dokładnego określenia, jakiego rodzaju są to błędy.

91 No Logout Active

Instrukcje COMMIT lub ROLLBACK zostały użyte poza ramką transakcji (nie została wykonana instrukcja LOGOUT).

92 BUILD in Progress

Została wydana instrukcja BUILD oraz ustawiona właściwość PROP:ProgressEvents do generowania zdarzeń. Instrukcja generująca ten błąd nie nadaje się do wykonania w trakcie trwania procesu BUILD.

93 BUILD Cancelled

Użytkownik przerwał indeksowanie BUILD. Ten rekord jest ustawiany, gdy wysłane zostanie zdarzenie EVENT:BuildDone.

800 Illegal Expression

Procedura EVALUATE wykryła błąd w składni wyrażenia do ewaluacji.

801 Variable Not Found

Procedura EVALUATE nie wykryła zmiennej użytej w wyrażeniu do ewaluacji. Konieczne jest wcześniejsze bindowanie wszystkich zmiennych stosowanych w wyrażeniu za pomocą instrukcji BIND.

Nieprzechwytywalne błędy działania aplikacji

Poniższe błędy występują w czasie działania programu i nie mogą być przechwytywane przez procedury ERRORCODE i ERROR.

ACCEPT loop requires a window

Pętla ACCEPT, z którą nie powiązano okna.

ENDPAGE must only be called for reports

Próba wykonania instrukcji ENDPAGE, gdy nie jest aktywny żaden raport REPORT.

Event posted to a report control

Próba wysłania POST zdarzenia do kontrolki należącej do struktury REPORT.

Metafile record too large in report

Plik .WMF jest zbyt duży, by mógł być wydrukowany w raporcie.

Mismatch with C4VBX.DLL detected

Pierwszy plik C4VBX.DLL znaleziony w ścieżce dostępu nie jest w tej samej wersji (zazwyczaj jest we wcześniejszej), co ten który posłużył do utworzenia .EXE.

PRINT must only be called for reports

Próba wydrukowania PRINT struktury, która nie jest częścią raportu REPORT.

Report is already open

Próba otwarcia OPEN raportu REPORT, który został już wcześniej otwarty i nie został jeszcze zamknięty.

Too many keystrokes PRESSed

Parametr przekazany do instrukcji PRESS zawiera zbyt wiele znaków.

Unable to complete operation (system is MODAL)

Próba wykonania nielegalnej akcji w programie, który już otworzył modalne MODAL okno lub przetwarza modalne zdarzenie.

Unable to create control (system is MODAL)

Próba utworzenia CREATE kontrolki w programie, który już otworzył modalne MODAL okno lub przetwarza modalne zdarzenie.

Unable to open APPLICATION (APPLICATION already active)

Próba otwarcia OPEN okna APPLICATION w programie, który już otworzył okno sterujące MDI.

Unable to open APPLICATION (system is MODAL)

Próba otwarcia OPEN okna APPLICATION w programie, który już otworzył modalne MODAL okno lub przetwarza modalne zdarzenie.

Unable to open APPLICATION

Zakończyła się niepowodzeniem próba otwarcia OPEN okna APPLICATION.

Unable to open MDI window (No APPLICATION active)

Próba otwarcia OPEN okna WINDOW z atrybutem MDI w programie, który nie otworzył jeszcze okna sterującego APPLICATION.

Unable to open MDI window (system is MODAL)

Próba otwarcia OPEN okna WINDOW z atrybutem MDI w programie, który już otworzył modalne MODAL okno lub przetwarza modalne zdarzenie.

Unable to open MDI window on APPLICATION's thread

Próba otwarcia OPEN okna WINDOW z atrybutem MDI w tym samym wątku wykonywalnym, co okno sterujące APPLICATION.

Unable to open MDI WINDOW

Zakończyła się niepowodzeniem próba otwarcia OPEN okna WINDOW z atrybutem MDI.

Unable to open WINDOW

Zakończyła się niepowodzeniem próba otwarcia OPEN okna WINDOW.

Unable to process ACCEPT (system is MODAL)

Próba wykonania nielegalnej operacji w programie, który już otworzył modalne MODAL okno lub przetwarza modalne zdarzenie.

Unexpected error opening printer device

Nieoczekiwany błąd, który wystąpił przy próbie otwarcia urządzenia drukarki.

VBX control is too complex

Kontrolka .VBX zawiera więcej niż 64 okienka dialogowe (ukryte lub widoczne). Limit ten dotyczy tylko trybu 16-bitowego.

Window is already open

Próba otwarcia OPEN okna WINDOW, które jest już otwarte

Window is not open

Próba wykonania akcji, która wymaga wcześniejszego otwarcia okna. Zazwyczaj jest to instrukcja przypisania właściwości.

Błędy kompilacji

Kompilator generuje komunikat błędu dokładnie w tym miejscu kodu źródłowego, w którym stwierdzi, że coś jest nie tak. Dlatego problem „leży" albo na prawo od tego miejsca, albo gdzieś w poprzedzającym go kodzie. W przypadku większości komunikatów błędów problem istnieje na prawo od wskazanego miejsca, jednak niektóre błędy są wynikiem pomyłek popełnionych dość daleko od niego i trzeba się wówczas nieco zabawić w detektywa.

Odpowiednia analiza komunikatu błędu kompilatora jest podstawą pomyślnego jego zlokalizowania. Główną tego przyczyną jest to, że pojedynczy błąd może spowodować efekt kaskadowy – długą listę błędów. Często znikają one wraz z jego poprawieniem. Wystarczy zatem, że poprawimy pierwszy błąd, a pozostałe nie będą już miały racji bytu. Jeśli natomiast uzyskamy kilka błędów wykazanych w miejscach od siebie odległych, oznacza to, że są one od siebie niezależne i należy poprawić je wszystkie przed ponowną kompilacją.

Błędy specyficzne

Przedstawione poniżej komunikaty błędów powstają wtedy, gdy kompilator napotka na problemy ze składnią i informuje o nich jednocześnie podpowiadając przyczyny. Niektóre z komunikatów zawierają wstawki „%V". Kompilator wstawia w ich miejsce etykiety związane z powstaniem błędu, co pomaga w jego poprawieniu.

! introduces a comment

Najpowszechniejszy błąd popełniany przez programistów C. Jeśli napiszesz IF A != 1 wtedy otrzymasz właśnie ten błąd.

Actual value parameter cannot be array

Przekazany parametr nie może być tablicą.

ADDRESS parameter ambiguous

ADDRESS(*MyLabel*), gdzie *MyLabel* jest etykietą zarówno procedury, jak i elementu danych.

All fields must be declared before JOINs

Wszystkie instrukcje PROJECT dla pliku muszą poprzedzać instrukcje JOIN w strukturze widoku VIEW.

Ambiguous label

Składnia kwalifikacji pola pozwala na różne interpretacje etykiety. Na przykład:

```
G     GROUP
S:T     SHORT                ! Referencjowany jako G:S:T
        END
G:S   GROUP
T       SHORT                ! Referencjowany jako G:S:T
        END

    CODE
      G:S:T = 7              ! O czym właściwie jest mowa?
```

Array too big

Tablice są ograniczone do 64K w trybie 16-bitowym.

Attribute parameter must be QUEUE, QUEUE field or constant string

Parametr musi być etykietą wcześniej zadeklarowanej struktury QUEUE, pola struktury QUEUE lub stałej łańcuchowej.

Attribute requires more parameters

Konieczne jest przesłanie wszystkich wymaganych parametrów atrybutu.

Attribute string must be constant

Parametr musi być stałą łańcuchową, a nie etykietą lub zmienną.

Attribute variable must be global

Parametr musi być zmienną zadeklarowaną w module PROGRAM jako dana globalna.

Attribute variable must have string type

Parametr musi być zmienną zadeklarowaną jako STRING, CSTRING lub PSTRING.

BREAK structure must enclose DETAIL

Musi wystąpić przynajmniej jedna struktura DETAIL w zagnieżdżonych strukturach BREAK (na najniższym poziomie).

Calling function as procedure

Ostrzeżenie, że procedura PROCEDURE zwracająca wartość i nie posiadająca atrybutu PROC jest wywoływana jako zwykła procedura PROCEDURE bez zachowania zwracanego rezultatu.

Cannot call procedure as function

Nie można wywołać procedury PROCEDURE nie zwracającej wartości w postaci źródła dla instrukcji przypisania lub parametru.

Cannot declare KEY in a VIEW

Deklaracja KEY w strukturze VIEW jest nieprawidłowa.

Cannot EXIT from here

Tylko podprogram ROUTINE może zawierać instrukcję EXIT.

Cannot GOTO into ROUTINE

Celem GOTO musi być etykieta instrukcji kodu wykonywalnego wewnątrz tej samej procedury lub podprogramu ROUTINE, nie może to być podprogram ROUTINE.

Cannot have default parameter here

Wartość domyślna może być stosowana tylko w nie pomijalnych parametrach o typach całkowitych przekazywanych przez wartość.

Cannot have initial values with OVER

Deklaracja zmiennej z atrybutem OVER nie może mieć parametru określającego wartość początkową.

Cannot have statement here

Ten błąd występuje, gdy kompilator uważa, że jest wykonana próba zadeklarowania etykiety kodu w sekcji danych globalnych.

Cannot initialize variable reference

Zmienna referencyjna nie może mieć wartości początkowej.

Cannot return CSTRING from CLARION function

Typ CSTRING nie jest prawidłowym typem dla rezultatu procedury PROCEDURE napisanej w Clarion (tylko dla funkcji napisanych w innych językach).

Cannot RETURN value from procedure

Tylko procedura PROCEDURE, w której prototypie określono, że zwraca wartość, może zawierać instrukcję RETURN z parametrem określającym tę wartość.

CLARION function cannot use RAW lub NAME

Te atrybuty nie są właściwe dla procedury PROCEDURE napisanej w Clarion (tylko dla funkcji napisanych w innych językach).

DECIMAL has too many places

Deklaracja DECIMAL lub PDECIMAL może mieć maksymalnie 30 pozycji na prawo od kropki dziesiętnej, a część dziesiętna musi być mniejsza, niż całkowita długość.

DECIMAL too long

Deklaracja DECIMAL lub PDECIMAL może mieć maksymalnie 31 cyfr.

Declaration not valid in FILE structure

Ta deklaracja danej nie może się znaleźć w strukturze FILE.

Declaration too big

Kompilator wykrył PSTRING > 255 lub MEMO > 64K w trybie 16-bitowym, itp.

DLL attribute requires EXTERNAL attribute

Atrybut DLL definiuje później atrybut EXTERNAL i jest wymagany w programach 32-bitowych.

Dynamic INDEX must be empty

Próba użycia drugiego parametru wywołania BUILD dla klucza KEY lub INDEX zadeklarowanego z polami składowymi.

Embedded OVER must name field in same structure

Parametr dla atrybutu OVER musi być etykietą wcześniej zadeklarowanej zmiennej w tej samej strukturze.

ENCRYPT attribute requires OWNER

Atrybut ENCRYPT i atrybut OWNER występują wspólnie.

Entity-parameter cannot be an array

Nie można przekazywać tablic jako parametrów egzemplarzy FILE, QUEUE itp.

Expected: %V

Jest to jeden z najczęstszych błędów. Kompilator oczekiwał, że znajdzie coś (jeden z elementów w liście substytutów %V) jako następny kod do kompilacji, ale zamiast tego znalazł inny kod, co spowodowało wygenerowanie błędu.

Expression cannot be picture

Nastąpiła próba zastosowania etykiety ekwiwalentu EQUATE w roli wzorca w miejscu, gdzie nie jest on właściwy.

Expression cannot have conditional type

Wyrażenie nie jest wartością numeryczną. Na przykład, *MyValue = A > B* jest nieprawidłowe.

Expression must be constant

Zmienne nie są właściwe dla tego wyrażenia.

Field equate label not defined: %V

Etykieta ekwiwalentu pola nie została wcześniej zadeklarowana.

Field not found

Zastosowanie składni kwalifikacji pól do odwołania się do pola nie będącego składnikiem struktury nadrzędnej. Na przykład, referencja *MyGroup.SomeField*, gdzie *SomeField* nie znajduje się w deklaracji *MyGroup*.

Field not found in parent FILE

Instrukcja JOIN musi deklarować wszystkie pola stanowiące powiązanie pomiędzy plikiem nadrzędnym i plikami podrzędnymi.

Field requires (more) subscripts

Nastąpiło odwołanie do tablicy wielowymiarowej bez określenia indeksu dla każdego z wymiarów.

FILE must have DRIVER attribute

Atrybut DRIVER jest wymagany do zadeklarowania systemu plików, zgodnie z którym będzie zapisywany plik danych.

FILE must have RECORD structure

Nie jest możliwe zadeklarowanie pliku FILE nie zawierającego struktury RECORD.

FILEs must have same DRIVER attribute

Wszystkie pliki wymienione w instrukcji LOGOUT muszą stosować ten sam system plików.

Function did not return a result

Ostrzeżenie, że implementacja procedury PROCEDURE prototypowanej do zwrotu wartości, nie zwraca rezultatu.

Function result is not of correct type

Instrukcja RETURN musi zwrócić wartość typu prototypowanego w strukturze MAP.

Group too big

Grupy GROUP są ograniczone do 64K w trybie 16-bitowym.

Ignoring EQUATE redefinition: %V

Ostrzeżenie, że ekwiwalent zostanie zignorowany. Jest to rzeczywisty błąd redefinicji etykiety, za wyjątkiem sytuacji, gdy ta definicja nie jest odrzucona.

Illegal array assignment

Przypisanie do tablicy musi stanowić referencję na pojedynczy element, nie na całą tablicę.

Illegal character

Nieprawidłowy znak leksykalny. Na przykład kod ASCII 255 w kodzie źródłowym.

Illegal data type: %V

Typ danych nie jest właściwy w strukturze, w której został użyty.

Illegal key component

Klucz KEY posiada składnik niewłaściwego typu.

Illegal nesting of window controls

Kontrolki okna, inne niż RADIO, zostały umieszczone w strukturze OPTION, bądź też kontrolki inne niż TAB zostały umieszczone bezpośrednio w strukturze SHEET.

Illegal parameter for LIKE

Nielegalny parametr dla deklaracji LIKE. Na przykład, *LIKE(7)*.

Illegal parameter type for STRING

Nieprawidłowy parametr dla deklaracji STRING. Na przykład, *STRING(MyVar)*, gdzie *MyVar* jest etykietą zmiennej a nie ekwiwalentem EQUATE.

Illegal reference assignment

Do zmiennej referencyjnej może być przypisana tylko zmienna referencyjna tego samego typu lub zmienna typu, do którego stanowi ona referencję.

Illegal return type lub attribute

Prototyp zawiera nieprawidłowy typ danych dla rezultatu (np. *CSTRING).

Illegal target for DO

Cel instrukcji DO musi być etykietą podprogramu ROUTINE.

Illegal target for GOTO

Cel instrukcji GOTO musi być etykietą instrukcji kodu wykonywalnego w tej samej procedurze lub podprogramie i nie może być etykieta podprogramu ROUTINE.

INCLUDE invalid, expected: %V

Parametr instrukcji INCLUDE musi być prawidłowym łańcuchem Clarion. W szczególności, konwersja typów nie jest prawidłowa, tak więc *INCLUDE('MyFile'&MyValue)* jest nieprawidłowe.

INCLUDE misplaced

INCLUDE nastąpiło po końcu wiersza lub po średniku (prawdopodobnie poprzedzanym prze odstęp).

INCLUDE nested too deep

Można zagnieżdżać INCLUDE tylko do 3 poziomów. Innymi słowy, można włączyć INCLUDE plik, który włącza INCLUDE inny plik, który z kolei włącza INCLUDE kolejny plik. Ostatni z dołączanych plików, nie może już dołączać następnych.

Incompatible assignment types

Próba przypisania danych o niezgodnych typach.

Incorrect procedure profile

Próba przekazania procedury o nieprawidłowym prototypie jako parametru proceduralnego.

Indices must be constant

Próba zastosowania, w roli zmiennej USE, elementu tablicy występującego w postaci zmiennej.

Indistinguishable new prototype: %V

Prototyp, którego kompilator nie może w sposób unikalny odróżnić od poprzedniego prototypu korzystając z reguł przeciążania procedur..

Integer expression expected

Wyrażenie musi być ewaluowane jako liczba całkowita.

Invalid BREAK statement

Próba grupowanie BREAK w oparciu o etykietę nie będącą etykietą LOOP lub leżącą poza strukturą LOOP bądź ACCEPT.

Invalid CYCLE statement

Próba użycia instrukcji CYCLE w oparciu o etykietę nie będącą etykietą LOOP lub leżącą poza strukturą LOOP bądź ACCEPT.

Invalid data declaration attribute

Niewłaściwy atrybut w deklaracji danej.

Invalid data type for value parameter

Typ danych prototypowany w MAP nie może być przekazany poprzez wartość, a musi być przekazany poprzez adres. Na przykład, by przesłać parametr CSTRING do procedury Clarion, musi on być prototypowany jako *CSTRING.

Invalid FILE attribute

Niewłaściwy atrybut w deklaracji FILE.

Invalid first parameter of ADD

Pierwszy parametr instrukcji nie jest prawidłowy.

Invalid first parameter of FREE

Pierwszy parametr instrukcji nie jest prawidłowy.

Invalid first parameter of NEXT

Pierwszy parametr instrukcji nie jest prawidłowy.

Invalid first parameter of PUT

Pierwszy parametr instrukcji nie jest prawidłowy.

Invalid GROUP/QUEUE/RECORD attribute

Niewłaściwy atrybut w deklaracji GROUP, QUEUE lub RECORD.

Invalid KEY/INDEX attribute

Niewłaściwy atrybut w deklaracji KEY lub INDEX.

Invalid label

Etykieta zawierająca znaki inne, niż litery, liczby, znaki podkreślenia (_), znaki dwukropka (:), bądź nie zaczynająca się literą lub znakiem podkreślenia.

Invalid LOOP variable

Próba użycia nielegalnego typu danych (DATE, TIME, STRING, itp.) w roli zmiennej LOOP.

Invalid MEMBER statement

Parametr instrukcji MEMBER nie jest stałą łańcuchową lub nie wskazuje modułu PROGRAM dla bieżącego projektu.

Invalid method invocation syntax

Próba zastosowania składni {} dla przywołania metody dla BLOB lub FILE.

Invalid number

Wymagana jest liczba, np. wewnątrz notacji powtarzania znaków ({}) w stałej łańcuchowej.

Invalid OMIT expression

Parametr instrukcji OMIT nie jest prawidłowy.

Invalid parameters for attribute

Nieprawidłowy parametr dla danego atrybutu.

Invalid picture token

Wzorzec zawiera niedopuszczalne znaki.

Invalid printer control token

Instrukcja PRINT zawiera niedopuszczalne znaki sterujące.

Invalid QUEUE/RECORD attribute

Niewłaściwy atrybut w deklaracji QUEUE lub RECORD.

Invalid SIZE parameter

SIZE(Junk+SomeMoreJunk)

Invalid string (misused <...> lub {...})

Stała łańcuchowa zawiera pojedynczy nawias otwierający (<lub {) bez odpowiedniego nawiasu zamykającego (> lub }). Jeśli taki znak ma być po prostu częścią łańcucha, wstawiamy go dwukrotnie (<< lub {{).

Invalid structure as first parameter

Pierwszy parametr instrukcji jest niewłaściwy.

Invalid structure within property syntax

Niewłaściwa struktura w instrukcji przypisania właściwości.

Invalid USE attribute parameter

Niewłaściwy parametr atrybutu USE.

Invalid use of PRIVATE data

Próba dostępu do prywatnego PRIVATE elementu danych spoza modułu klasy CLASS.

Invalid use of PRIVATE procedure

Próba wywołania prywatnej PRIVATE metody spoza modułu klasy CLASS.

Invalid variable data parameter type

Przy przekazywaniu parametru poprzez adres, należy przekazać daną o typie zgodnym z prototypem w strukturze MAP.

Invalid WINDOW control

Niewłaściwa kontrolka dla struktury WINDOW.

ISL error: %V

Ten błąd oznacza, że trzeba się skontaktować z Technical Support.

KEY must have components

Nie można deklarować klucza KEY bez określenia jego pól składowych.

Label duplicated, second used: %V

Etykieta ekwiwalentu pola jest użyta wielokrotnie w ramach tego samego modułu i tylko ostatnie jej wystąpienie jest uwzględnione w liście etykiet, które mogą być użyte w sekcji kodu wykonywalnego. Można to naprawić za pomocą trzeciego parametru atrybutu USE.

Label in prototype not defined: %V

Zastosowanie prototypu, którego jeden z typów danych nie został jeszcze zdefiniowany.

Label not defined: %V

Etykieta nie została wcześniej zdefiniowana.

Mis-placed string slice operator

Fragment łańcucha, który nie jest ostatnim indeksem tablicy. Na przykład, *MyStringArray[3:4,5]*.

Missing procedure definition: %V

Procedura nie została prototypowana w strukturze MAP.

Missing virtual function

Błąd kompilatora.

Must be dimensioned variable

To musi być tablica.

Must be field of a FILE lub VIEW

To musi być pole należące do struktury FILE lub VIEW.

Must be FILE lub KEY

Parametr JOIN nie jest plikiem FILE lub kluczem KEY.

Must be reference variable

Zwolnienie za pomocą DISPOSE dotyczy tylko zmiennych referencyjnych.

Must be variable

To musi być etykieta wcześniej zadeklarowanej zmiennej.

Must have constant string parameter

Parametr musi być stałą łańcuchową, nie etykietą zmiennej.

Must RETURN value from function

Procedura PROCEDURE, w której prototypie określono, że zwraca wartość, musi zawierać instrukcję RETURN z parametrem określającym tę wartość.

Must specify DECIMAL size

Deklaracja DECIMAL lub PDECIMAL musi określać maksymalna liczbę przechowywanych cyfr.

Must specify identifier

Identyfikator jest wymagany, ale nie został dostarczony.

Must specify print-structure

Instrukcja PRINT może drukować tylko strukturę raportu REPORT.

No matching prototype available

Próba zdefiniowania procedury, dla której nie ma prototypu w MAP lub CLASS.

Not valid inside structure

Typ danych jest niewłaściwy dla struktury, w której został użyty.

OMIT cannot be nested

Nie można zagnieżdżać dyrektyw OMIT. Znajdujemy się w OMIT (lub COMPILE), które *nie* pomija kodu, a kompilator wykrywa kolejne OMIT.

OMIT misplaced

OMIT powinno nastąpić w nowym wierszu lub średnik (prawdopodobnie poprzedzony spacją).

OMIT not terminated: %V

Odpowiedni parametr OMIT nie został odnaleziony przed końcem modułu kodu źródłowego.

Order is MENUBAR, TOOLBAR, Controls

Struktura MENUBAR musi wystąpić przed TOOLBAR, a struktura TOOLBAR musi wystąpić przed kontrolkami w oknie WINDOW lub APPLICATION.

OVER must name variable

Parametr atrybutu OVER musi być etykietą wcześniej zadeklarowanej zmiennej.

OVER must not be larger than target variable

Parametr atrybutu OVER musi być etykietą wcześniej zadeklarowanej zmiennej, która jest większa lub równa rozmiarowi zmiennej, którą jest na nią nakładana.

OVER not allowed with STATIC lub THREAD

Deklaracja zmiennej z atrybutem OVER nie może dodatkowo posiadać atrybutów STATIC lub THREAD (muszą występować w deklaracji źródłowej).

Parameter cannot be omitted

Wywołanie procedury musi przekazać wszystkie parametry, które nie zostały określone jako pomijalne.

Parameter kind does not match

Podczas przekazywania parametru poprzez adres, należy przekazać ten sam typ danych, co w prototypie umieszczonym w strukturze MAP.

Parameter must be picture

To musi być wzorzec wyświetlania.

Parameter must be procedure label

To musi być etykieta procedury.

Parameter must be report DETAIL label

Instrukcja PRINT może tylko drukować strukturę DETAIL raportu REPORT.

Parameters must have labels

Próba zdefiniowania procedury bez użycia etykiet parametrów.

Parameter type label ambiguous (CODE lub DATA)

Występuje procedura PROCEDURE i deklaracja danych o takiej samej nazwie, nie będzie możliwe użycie tej nazwy w prototypie procedury.

PROCEDURE cannot have return type

Jeśli procedura została zadeklarowana bez typu rezultatu w strukturze MAP, musi zostać utworzona jako procedura, a nie funkcja.

Procedure doesn't belong to module: %V

Próba zdefiniowania procedury posiadający prototyp mówiący, że znajduje się ona w innym module.

Procedure in parent CLASS has VIRTUAL mismatch

Metody wirtualne wymagają atrybutu VIRTUAL w prototypie klasy nadrzędnej i klasy dziedziczących.

Prototype is: %V

Próba zdefiniowania procedury z błędnym prototypem.

QUEUE/RECORD not valid in GROUP

Struktura GROUP nie może zawierać struktury QUEUE lub RECORD.

Redefining system intrinsic: %V

Ostrzeżenie, że procedura (stanowiąca część kodu źródłowego) ma tę samą nazwę, co procedura biblioteki uruchomieniowej Clariona i że będzie ona wywoływana zamiast procedury bibliotecznej.

Routine label duplicated

Etykieta instrukcji ROUTINE została już wcześniej użyta w innej instrukcji.

Routine not defined: %V

Podprogram ROUTINE nie istnieje.

SECTION duplicated: %V

Sekcja SECTION występuje dwa razy w dołączanym (INCLUDE) pliku.

SECTION not found: %V

Sekcja SECTION nie występuje w dołączanym (INCLUDE) pliku.

Statement label duplicated

Dwie linie kodu wykonywalnego posiadają tę samą etykietę

Statement must have label

Instrukcja (np. ROUTINE lub PROCEDURE) musi posiadać etykietę.

String not terminated

Stała łańcuchowa bez kończącego apostrofu (’).

Subscript out of range

Próba adresowania elementu tablicy leżącego poza poprawnymi wymiarami określonymi w deklaracji.

Too few indices

Jest to referencja do tablicy wielowymiarowej i musimy określić indeks dla każdego wymiaru.

Too few parameters

Procedura musi przekazać wszystkie parametry, które nie zostały określone jako pomijalne.

Too many indices

Odwołanie do elementu tablicy, w którym określono zbyt dużo wymiarów

Too many parameters

Wywołanie procedury nie może przekazywać więcej parametrów, niż ma ich w prototypie.

Unable to verify validity of OVER attribute

Ostrzeżenie, że została zadeklarowana zmienna nałożona OVER na przekazany parametr i że typy danych mogą być niezgodne.

Unknown attribute: %V

Atrybut nie jest częścią języka Clarion.

Unknown function label

Procedura PROCEDURE nie była wcześniej prototypowana w strukturze MAP.

Unknown identifier

Etykieta nie była wcześniej zadeklarowana.

Unknown identifier: %V

Identyfikator nie był wcześniej zadeklarowany.

Unknown key component: %V

Składnik klucza nie występuje w strukturze FILE.

Unknown procedure label

Procedura PROCEDURE nie była wcześniej prototypowana w strukturze MAP.

UNTIL/WHILE illegal here

Próba użycia UNTIL lub WHILE do zakończenia pętli LOOP, która została już zakończona.

Value-parameter cannot be an array

Nie można przekazać tablicy jako parametru w postaci wartości.

Value requires (more) subscripts

Jest to referencja do tablicy wielowymiarowej i musimy określić indeks dla każdego wymiaru.

Variable expected

To musi być etykieta wcześniej zadeklarowanej zmiennej.

Variable-size must be constant

Deklaracja zmiennej musi zawierać wyrażenie stałe określające jej rozmiar.

VIRTUAL illegal outside of CLASS structure

Atrybut VIRTUAL może być stosowany dla prototypów struktury CLASS, a nie MAP.

Wrong number of parameters

Procedura musi przekazać wszystkie parametry, które nie zostały określone jako pomijalne.

Wrong number of subscripts

Próba dostępu do tablicy wielowymiarowej bez określenia indeksu dla każdego wymiaru. Na przykład:

```
MyShort SHORT,DIM(8,2) !Two-dimensional array
CODE
MyValue = MyShort[7] !Wrong number of subscripts error
```

Błędy nieznane

Poniżej wymieniono błędy, które w zasadzie nigdy nie powinny wystąpić. Jeśli jednak, służą one do określenia tropu prowadzącego *twórców kompilatora TopSpeed* do przyczyny powstania takiego błędu. W takiej sytuacji należy niezwłocznie zgłosić do TopSpeed powstanie błędu wraz plikiem źródłowym, który spowodował jego wygenerowanie

Inconsistent scanner initialization

Unknown operator

Unknown expression type

Unknown expression kind

Unknown variable context

Unknown parameter kind

Unknown assignment operator

Unknown variable type

Unknown case type

Unknown equate type

Unknown string kind

Unknown picture type

Unknown descriptor type

Unknown initializer type

Unknown designator kind

Unknown structure field

Unknown formal entity

Type descriptor not static

Unknown clear type

Unknown simple formal type

Out of attribute space

Unknown label/routine

Unknown special identifier

Value not static

Unknown static label

Unknown screen structure kind

Corrupt pragma string

Old symbol non-NIL

Not implemented yet

String not CCST

DODATEK E – INSTRUKCJE WCZEŚNIEJSZYCH WERSJI

Wszystkie instrukcje wymienione w tym Dodatku zostały zachowane w celu zapewnienia kompatybilności z poprzednimi wersjami Clariona. Wszystkie one będą stopniowo usuwane z kolejnych wersji Clariona, tak więc nie zaleca się ich stosowania.

BOF (określenie początku pliku)

BOF(*file*)

BOF Określa początek pliku FILE podczas operacji przetwarzania sekwencyjnego.

file Etykieta deklaracji pliku FILE.

Funkcja **BOF** daje w rezultacie wartość niezerową (prawda), gdy rekord odczytany za pomocą instrukcji PREVIOUS lub przekazany przez SKIP jest pierwszym rekordem w pliku (znajdującym się w określonej sekwencji). W przeciwnym wypadku wartością rezultatu jest zero (fałsz).

Funkcja BOF nie jest obsługiwana przez wszystkie sterowniki plików I jest przy tym bardzo nieefektywna; z tego powodu nie zaleca się jej stosowania. Zamiast niej powinno się używać funkcji ERRORCODE() po każdym odczycie z dysku w celu określenia, czy nie miała miejsca próba odczytu po minięciu początku pliku.

Typ rezultatu: LONG

Przykład:

```
! Nie zalecane, ale nadal wspierane w celu zachowania kompatybilności wstecz:
SET(Trn:DateKey)                          ! Koniec/Początek pliku w sekwencji wg klucza
LOOP UNTIL BOF(Trans)                     ! Przetwarzanie pliku wstecz
  PREVIOUS(Trans)                         ! odczyt poprzedniego rekordu
  IF ERRORCODE() THEN STOP(ERROR()).
  DO LastInFirstOut                       ! wywołanie podprogramu
END

! Zalecane jak o najbardziej efektywne dla wszystkich formatów plików:
SET(Trn:DateKey)                          ! Koniec/Początek pliku w sekwencji wg klucza
LOOP                                      ! Przetwarzanie pliku wstecz
  PREVIOUS(Trans)                         ! odczyt rekordu z sekwencji
  IF ERRORCODE() THEN BREAK.              ! przerwanie pętli przy próbie odczytu przed początkiem pliku
  DO LastInFirstOut                       ! wywołanie podprogramu
END
```

Porównaj: ERRORCODE

EOF (określenie końca pliku)

EOF(*file*)

EOF Określa początek pliku FILE podczas operacji przetwarzania sekwencyjnego.

file Etykieta deklaracji pliku FILE.

Funkcja **EOF** daje w rezultacie wartość niezerową (prawda), gdy rekord odczytany za pomocą instrukcji NEXT lub przekazany przez SKIP jest pierwszym rekordem w pliku (znajdującym się w określonej sekwencji). W przeciwnym wypadku wartością rezultatu jest zero (fałsz).

Funkcja EOF nie jest obsługiwana przez wszystkie sterowniki plików i jest przy tym bardzo nieefektywna; z tego powodu nie zaleca się jej stosowania. Zamiast niej powinno się używać funkcji ERRORCODE() po każdym odczycie z dysku w celu określenia, czy nie miała miejsca próba odczytu po minięciu początku pliku.

Typ rezultatu: LONG

Przykład:

```
! Nie zalecane, ale nadal wspierane w celu zachowania kompatybilności wstecz:
SET(Trn:DateKey)                          ! Początek pliku wg klucza
LOOP UNTIL EOF(Trans)                     ! Przetwarzanie wszystkich rekordów
  NEXT(Trans)                             ! odczyt rekordu z sekwencji
  IF ERRORCODE() THEN STOP(ERROR()).
  DO LastInFirstOut                       ! wywołanie podprogramu
END

! Zalecane jak o najbardziej efektywne dla wszystkich formatów plików:
SET(Trn:DateKey)                          ! Początek pliku wg klucza
LOOP                                      ! Przetwarzanie wszystkich rekordów
  NEXT(Trans)                             ! odczyt rekordu z sekwencji
  IF ERRORCODE() THEN BREAK.              ! przerwanie pętli przy próbie odczytu przed początkiem pliku
  DO LastInFirstOut                       ! wywołanie podprogramu
END
```

Porównaj: ERRORCODE

FUNCTION (definicja funkcji)

label **FUNCTION** [(*parameter list*)]
local data
CODE
statements
RETURN(*value*)

FUNCTION jest instrukcją, która jednorazowo definiuje procedurę prototypowaną z rezultatem (określaną jako funkcja w innych językach programowania). Słowo kluczowe FUNCTION zostało przez instrukcję i jest teraz synonimem PROCEDURE.

Przykład:

```
    PROGRAM
    MAP
FullName    FUNCTION(STRING Last,STRING First,<STRING Init>),STRING
                                                    ! prototyp funkcji z parametrami
DayString   FUNCTION,STRING                         ! prototyp funkcji bez parametrów
    END

TodayString   STRING(3)

    CODE
    TodayString = DayString()                       ! wywołanie funkcji bez parametrów
                                                    ! nawiasy () są wymagane dla funkcji

FullName  FUNCTION(STRING Last, STRING First,STRING Init)   ! pełna nazwa funkcji
    CODE                                            ! początek sekcji kodu
    IF OMITTED(3) OR Init = ''                      ! jeśli brak środkowego inicjału
      RETURN(CLIP(First) & ' ' & Last)              ! zwróć nazwisko
    ELSE                                            ! w przeciwnym wypadku
      RETURN(CLIP(First) & ' ' & Init & '. ' & Last)    ! zwróć pełne nazwisko
    END

DayString   FUNCTION                                ! funkcja
ReturnString  STRING(9),AUTO                        ! niezainicjowana zmienna stosu lokalnego
    CODE                                            ! początek sekcji kodu
    RETURN(CHOOSE(TODAY()%7)+1,'Sun','Mon','Tue','Wed','Thu','Fri','Sat'))
```

Porównaj: PROCEDURE

POINTER (zwraca względną pozycję rekordu w pliku)

```
POINTER( | file | )
         | key  |
```

POINTER Zwraca w rezultacie względną pozycję rekordu w pliku.

file Etykieta deklaracji FILE. Określa fizyczny porządek dla rekordów w pliku.

key Etykieta klucza KEY lub INDEX. Określa logiczny (wg wybranego klucza) porządek dla rekordów w pliku.

POINTER zwraca względną pozycję rekordu w pliku (w porządku fizycznym – *file* lub logicznym - *key*), do którego ostatnio realizowany był dostęp.

Wartość zwracana przez funkcję POINTER zależy od sterownika (stosowanego formatu pliku). Może to być numer rekordu, względna pozycja (bajtowa) lub inny rodzaj pozycji określający położenie rekordu w pliku.

Funkcja POINTER nie jest dostępna dla wszystkich formatów plików danych. Z tego powodu powinniśmy ją stosować tylko wtedy, gdy mamy pewność, że wybrany format pliku pozwala na jej użycie i że nie zmienimy go w przyszłości. Preferowaną metodą określania pozycji rekordu, która pozwala na pracę ze wszystkimi formatami plików jest stosowanie funkcji POSITION w połączeniu z procedurami RESET i REGET.

Typ rezultatu: LONG

Przykład:

```
SavePtr# = POINTER(Customer)                    ! zapisanie wskaźnika pliku
```

Porównaj: POSITION

SHARE (otwiera plik danych w trybie współdzielenia)

SHARE(*file* [, *access mode*])

SHARE Otwiera plik FILE do przetwarzania.

file Etykieta deklaracji FILE.

access mode Stała numeryczna, zmienna lub wyrażenie określające uprawnienia dostępu dla użytkownika otwierającego plik jako pierwszy i wszystkich pozostałych użytkowników. Jeśli parametr ten zostanie pominięty, przyjmuje się dla niego domyślna wartość 42h (Odczyt/Zapis, Nic_nie_zabraniaj).

Instrukcja **SHARE** otwiera plik FILE do przetwarzania i ustawia prawa dostępu *access mode*. Instrukcja SHARE działa dokładnie tak samo, jak instrukcja OPEN, z tą różnicą, że posiada inną domyślną wartość dla *access mode*.

Parametr *access mode* stanowi mapę bitową, która określa prawa dostępu dla użytkownika otwierającego dany plik jako pierwszy i dla wszystkich pozostałych użytkowników.

Wartości dla poszczególnych poziomów dostępu są następujące:

	Dzi.	Hex.	Rodzaj dostępu
Użytkownik:	0	0h	Tylko odczyt
	1	1h	Tylko zapis
	2	2h	Odczyt/Zapis
Pozostali użytkownicy:	0	0h	Dowolny dostęp (tryb zgodny z FCB)
	16	10h	Zabroniony dostęp
	32	20h	Zabroniony zapis
	48	30h	Zabroniony odczyt
	64	40h	Nic nie zabraniaj

Komunikaty błędów: Ten sam zakres błędów, co w przypadku funkcji OPEN

Przykład:

```
ReadOnly    EQUATE(0)                ! ekwiwalenty trybu dostępu
WriteOnly   EQUATE(1)
ReadWrite   EQUATE(2)
DenyAll     EQUATE(10h)
DenyWrite   EQUATE(20h)
DenyRead    EQUATE(30h)
DenyNone    EQUATE(40h)

  CODE
  SHARE(Master,ReadOnly+DenyWrite)     ! otwarcie w trybie tylko do odczytu
```

Porównaj: OPEN