Dual Interface

Dual Interface
`IDispatch`

`IDispatch` po `IUnknown` jest drugim najczęściej używanym interfejsem standardowym w obiektach COM. Pozwala on na wywoływanie funkcji pośrednio poprzez jej identyfikator i struktury danych zawierające argumenty funkcji. Pozwala to klientom na korzystanie z komponentu (wywoływanie jego metod) bez konieczności znajomości struktury interfejsu podczas kompilacji.
Część języków (np. Visual Basic) pozwala na korzystanie z komponentu wyłącznie przez interfejs dualny.

Dual Interface
`IDispatch`

Interfejs dualny wymaga, aby argumenty metod były zgodne z typami OLE, dlatego dla interfejsów z dualnym interfejsem wymagany jest atrybut `oleautomation`.

Dodatkowo dla każdej metody interfejsu dualnego definiuje się unikalny identyfikator poprzez atrybut `id()`.

Dual Interface
IDispatch

```
[
  object,
  dual,
  oleautomation,
  uuid(C04E9202-BAFA-45e2-9F07-942D7CF76361),
  helpstring("IStopwatch2 Interface"),
  pointer_default(unique)
]
interface IStopwatch2 : IDispatch
{
 [id(1), helpstring("Starts the timer")]
   HRESULT Start();
 [id(DISPID_VALUE), helpstring("Elapsed time in seconds Start call")]
   HRESULT ElapsedTime([out, retval] float* Time);
 [id(2), propget, helpstring("Returns/sets the overhead time.")]
   HRESULT Overhead([out, retval] float *pVal);
 [id(2),propput, helpstring("Returns/sets the overhead time.")]
   HRESULT Overhead([in] float newVal);
};
```

Identyfikator `DISPID_VALUE` używany jest np. w Visual Basic. Definiuje on wybraną metodę jako domyślną, gdy podczas wywołania obiektu nie sprecyzuje się metody.

Dual Interface
IDispatch

```
library TIMERSLib
{

  importlib("stdole32.tlb");
  importlib("stdole2.tlb");

  [
    uuid(83DC3C46-1259-4f95-A2D1-CD11A8819E2E),
    version(2.0),
    helpstring("Stopwatch Component")
  ]
  coclass Stopwatch
  {
    [default] dispinterface IStopwatch2;
    interface IStopwatch;
  };
};
```

Dual Interface
IDispatch

```
interface IDispatch : IUnknown {
    HRESULT GetTypeInfoCount( [out] UINT * pctinfo );

    HRESULT GetTypeInfo(    [in] UINT iTInfo,
                            [in] LCID lcid,
                            [out] ITypeInfo ** ppTInfo );

    HRESULT GetIDsOfNames( [in] REFIID riid,
                            [in, size_is(cNames)] LPOLESTR * rgszNames,
                            [in] UINT cNames,
                            [in] LCID lcid,
                            [out, size_is(cNames)] DISPID * rgDispId );

    HRESULT Invoke(         [in] DISPID dispIdMember,
                            [in] REFIID riid,
                            [in] LCID lcid,
                            [in] WORD wFlags,
                            [in, out] DISPPARAMS * pDispParams,
                            [out] VARIANT * pVarResult,
                            [out] EXCEPINFO * pExcepInfo,
                            [out] UINT * puArgErr );
}
```

Dual Interface
IDispatch: GetIDsOfNames

```
HRESULT GetIDsOfNames( [in] REFIID riid,
                       [in, size_is(cNames)] LPOLESTR * rgszNames,
                       [in] UINT cNames,
                       [in] LCID lcid,
                       [out, size_is(cNames)] DISPID * rgDispId );
```

`GetIDsOfNames` jest używana do pobrania identyfikatora metody o nazwie przekazywanej jako pierwszy element wektora `rgszNames`. Kolejne elementy wektora (opcjonalne) precyzują nazwy argumentów poszukiwanej metody, dla których zwrócone zostaną identyfikatory. Pozwala to na przekazywanie argumentów według identyfikatorów, a nie według kolejności występowania parametrów.

`cNames` definiuje rozmiar wektorów `rgszNames` i `rgDispId`.

`lcid` określa identyfikator języka użytego do tworzenia nazw (zwykle `LOCALE_SYSTEM_DEFAULT`).

`rgDispId` wskazuje na tablicę, w której metoda umieści identyfikatory.

`riid` przewidziane jest do późniejszych rozszerzeń, musi być `IID_NULL`.

Dual Interface
IDispatch: GetTypeInfoCount

```
HRESULT GetTypeInfoCount( [out] UINT * pctinfo );
```

Metoda `GetTypeInfoCount` zwraca liczbę zestawów informacji o interfejsach dostępnych przez `GetTypeInfo`. Obecnie możliwe są wartości 0 (brak informacji) lub dla projektów ATL: - 1.

Dual Interface
IDispatch: GetTypeInfo

```
HRESULT GetTypeInfo(    [in] UINT iTInfo,
                        [in] LCID lcid,
                        [out] ITypeInfo ** ppTInfo );
```

Metoda `GetTypeInfo` używana jest przez narzędzia pozwalające na analizę struktury interfejsu (nazwy metod, typy argumentów itp.). Zwraca ona wskaźnik do obiektu typu `ITypeInfo`, za pomocą którego można odczytać szczegółowe informacje o interfejsie.
`iTInfo` wskazuje na typ żądanej informacji (0 aby uzyskać informacje od interfejsu `IDispatch`).
`lcid` określa identyfikator języka użytego do tworzenia nazw (zwykle `LOCALE_SYSTEM_DEFAULT`).

Dual Interface
IDispatch: Invoke

```
HRESULT Invoke(         [in] DISPID dispIdMember,
                        [in] REFIID riid,
                        [in] LCID lcid,
                        [in] WORD wFlags,
                        [in, out] DISPPARAMS * pDispParams,
                        [out] VARIANT * pVarResult,
                        [out] EXCEPINFO * pExcepInfo,
                        [out] UINT * puArgErr );
```

Metoda `Invoke` używana jest do wywołania wybranej metody interfejsu.
`dispIdMember` jest identyfikatorem metody zwróconym przez `GetIDsOfNames`().
`riid` musi być `IID_NULL` (zarezerwowane do przyszłego użytku).
`lcid` określa identyfikator języka.
`wFlags` identyfikuje rodzaj wywoływanej metody:

- `DISPATCH_METHOD`
- `DISPATCH_PROPERTYGET`
- `DISPATCH_PROPERTYPUT`
- `DISPATCH_PROPERTYPUTREF`

Dual Interface
IDispatch: Invoke

```
HRESULT Invoke(         [in] DISPID dispIdMember,
                        [in] REFIID riid,
                        [in] LCID lcid,
                        [in] WORD wFlags,
                        [in, out] DISPPARAMS * pDispParams,
                        [out] VARIANT * pVarResult,
                        [out] EXCEPINFO * pExcepInfo,
                        [out] UINT * puArgErr );
```

`pDispParams` jest strukturą, używaną do przekazywania argumentów (musi wskazywać na istniejącą strukturę, nawet jeśli nie przekazujemy żadnych argumentów):

```
typedef struct FARSTRUCT tagDISPPARAMS{
    VARIANT *rgvarg;              // Array of arguments.
    DISPID *rgdispidNamedArgs;    // Dispatch IDs of named arguments
                                  // (NULL if by order).
    unsigned int cArgs;           // Number of arguments.
    unsigned int cNamedArgs;      // Number of named arguments.
} DISPPARAMS;
```

Dla listy argumentów uporządkowanej wg kolejności należy pamiętać, że uporządkowane są one w odwrotnym porządku (pierwszy element `rgvarg` odpowiada ostatniemu argumentowi).

Dual Interface
IDispatch: Invoke

```
HRESULT Invoke(         [in] DISPID dispIdMember,
                        [in] REFIID riid,
                        [in] LCID lcid,
                        [in] WORD wFlags,
                        [in, out] DISPPARAMS * pDispParams,
                        [out] VARIANT * pVarResult,
                        [out] EXCEPINFO * pExcepInfo,
                        [out] UINT * puArgErr );
```

`pVarResult` jest zmienną, w której zapamiętany zostanie rezultat wywołania funkcji (określony atrybutem `retval`). Może być NULL jeśli nie spodziewamy się żadnego rezultatu. Argument ten jest ignorowany gdy wywołanie jest typu `DISPATCH_PROPERTYPUT` lub `DISPATCH_PROPERTYPUTREF`.

`pExceptInfo` jest strukturą, w której zapamiętana zostanie informacja o błędzie operacji gdy `Invoke` zwróci `DISP_E_EXCEPTION`. Parametr ten może być NULL.

`puArgErr` wskazuje na indeks parametru w `rgvarg` struktury `pDispParams` dla którego wystąpił błąd gdy `Invoke` zwrócił `DISP_E_TYPEMISMATCH` lub `DISP_E_PARAMNOTFOUND`. Parametr ten może być NULL.

Dual Interface
Przykład klienta

```
float nElapsedTimeLateBound;
HRESULT hr;

// Declare an enumeration corresponding to the FunctionNames array.
enum StopwatchMethods {
  StopwatchMethodsStart,
  StopwatchMethodsElapsedTime,
  StopwatchMethodsOverhead,
  StopwatchMethodsTotal
};

// IDispatch stuff for calling the Stopwatch Object
OLECHAR FAR* FunctionNames[StopwatchMethodsTotal]
  = {L"Start", L"ElapsedTime", L"Overhead"};
DISPID DispIds[StopwatchMethodsTotal];
variant_t varElapsedTime;
variant_t varOverhead;

// Create an array to store the method params.  Since we never have more
// than one param this array is for one item only
variant_t varParms[1];
DISPPARAMS dpParms = { NULL, NULL, 0, 0 };
```

Dual Interface
Przykład klienta

```
// A vtable pointer to the Stopwatch object
IStopwatch2Ptr pStopwatch2(IID_IStopwatch);
// Create a new Stopwatch object which will be timed
IDispatchPtr pDispStopwatch;
// Get the dispatch interface
pStopwatch.QueryInterface( IID_IDispatch, &pDispStopwatch );
// Get "Start" method ID and store in the DispIds array for future use
if (FAILED(hr = pDispStopwatch->GetIDsOfNames(
      IID_NULL,
      &FunctionNames[StopwatchMethodsStart],
      1,
      LOCALE_SYSTEM_DEFAULT,
      &DispIds[StopwatchMethodsStart])))
   com_raise_error(hr);
// Invoke "Start" method
if (FAILED(hr = pDispStopwatch->Invoke(
      DispIds[StopwatchMethodsStart],
      IID_NULL,
      LOCALE_SYSTEM_DEFAULT,
      DISPATCH_METHOD,
      &dpParms,
      NULL,
      NULL,
      NULL)))
   com_raise_error(hr);
```

Dual Interface
Przykład klienta

```
// Get "ElapsedTime" method ID and store in the DispIds array
if (FAILED(hr = pDispStopwatch->GetIDsOfNames(
      IID_NULL,
      &FunctionNames[StopwatchMethodsElapsedTime],
      1,
      LOCALE_SYSTEM_DEFAULT,
      &DispIds[StopwatchMethodsElapsedTime])))
   _com_raise_error(hr);
// Invoke "ElapsedTime" method
if (FAILED(hr = pDispStopwatch->Invoke(
      DispIds[StopwatchMethodsElapsedTime],
      IID_NULL,
      LOCALE_SYSTEM_DEFAULT,
      DISPATCH_METHOD,
      &dpParms,
      &varElapsedTime,
      NULL,
      NULL)))
   _com_raise_error(hr);

nElapsedTimeLateBound = varElapsedTime;

std::cout << "The late bound elapsed time including a query"<<
             " for disp ids" is << nElapsedTimeLateBound << std::endl;
```