



Poznaj oprogramowanie ERP,
które pracuje na Twój sukces
przez 365 dni w roku!

**Systemowy rozwój
Twojej firmy.**

Business Intelligence

Handel i magazyn

Sprzedaż i CRM

Finanse i księgowość

Kadry Płace i HR

Procesy

Projekty i usługi

Produkcja

Spis treści

Spis treści	2
Biblioteki Soneta	3
Biblioteki Soneta w dodatkach	3
Biblioteki Soneta w aplikacji lub serwisie	3
Loader	4
Zależności w projekcie	5
Gdzie loader szuka bibliotek	7

Biblioteki Soneta

Biblioteki Soneta w dodatkach

Większość rozszerzeń funkcjonalnych do systemu enova365 tworzonych jest w formie dodatków DLL. W takim przypadku system enova365 (program SonetaExplorer lub SonetaServer) odpowiedzialny jest za wczytanie wszystkich potrzebnych bibliotek firmy Soneta, a także za wczytanie bibliotek dodatków. Standardowo utworzony dodatek DLL **nie musi i nie powinien** w ogóle zajmować się wczytywaniem bibliotek Soneta. Framework enova365 zadba o to, by w momencie wykorzystania dowolnej funkcjonalności dodatku, wczytane były wszystkie potrzebne biblioteki. Wczytanie niezbędnych DLL-ek odbywa się przy starcie aplikacji. Wczytanie dodatkowych komponentów odbywa się dynamicznie.

W przypadku dodatków korzystających z zewnętrznych bibliotek, programista musi zadbać w projekcie o odpowiednie zależności (referencje). Dodatkowe biblioteki zostaną wczytane albo z katalogu dodatków (tak samo jak DLL samego dodatku), albo zostaną doczytane dynamicznie, kiedy program wraz z dodatkiem odwoła się do kodu wymagającego zewnętrznej biblioteki. Przykład dynamicznego doczytywania zewnętrznych bibliotek został opisany [tutaj](#).

Nie mniej **dopóki mamy do czynienia z typowym dodatkiem do enova365, ręczne wczytywanie bibliotek (ani bibliotek Soneta ani żadnych bibliotek zewnętrznych) nie jest potrzebne.**

Biblioteki Soneta w aplikacji lub serwisie

Inaczej jest w przypadku, gdy stworzymy samodzielną aplikację lub serwis, który korzysta z bibliotek Soneta. W takim przypadku to aplikacja odpowiedzialna jest za to, by wczytać biblioteki. Bibliotek należących do enova365 nie należy traktować jako samodzielnych komponentów, które zostaną doczytane dynamicznie w razie potrzeby. Należy je traktować raczej jako kompletny framework, którego komponenty powinny być wczytane zbiorczo i w odpowiednim porządku, z odpowiednich lokalizacji.

Loader

Ze względu na złożony system zależności między bibliotekami framework-u, do wczytywania bibliotek Soneta stworzony został dedykowany mechanizm, zaimplementowany w klasie Loader.

UWAGA: Nie wolno wczytywać bibliotek Soneta za pomocą własnych rozwiązań korzystających wprost z `System.Reflection.Assembly.Load()`. Począwszy od wersji 2006 system enova365 wykonuje sprawdzenie, czy biblioteki zostały wczytane prawidłowo, tzn. czy użyta została klasa Loader. enova365, zarówno w postaci okienkowej jak i serwerowej, również korzysta z loadera podczas inicjalizacji.

Wczytanie bibliotek framework-u wymaga jedynie utworzenia instancji klasy Loader i wywołania metody `Load()`:

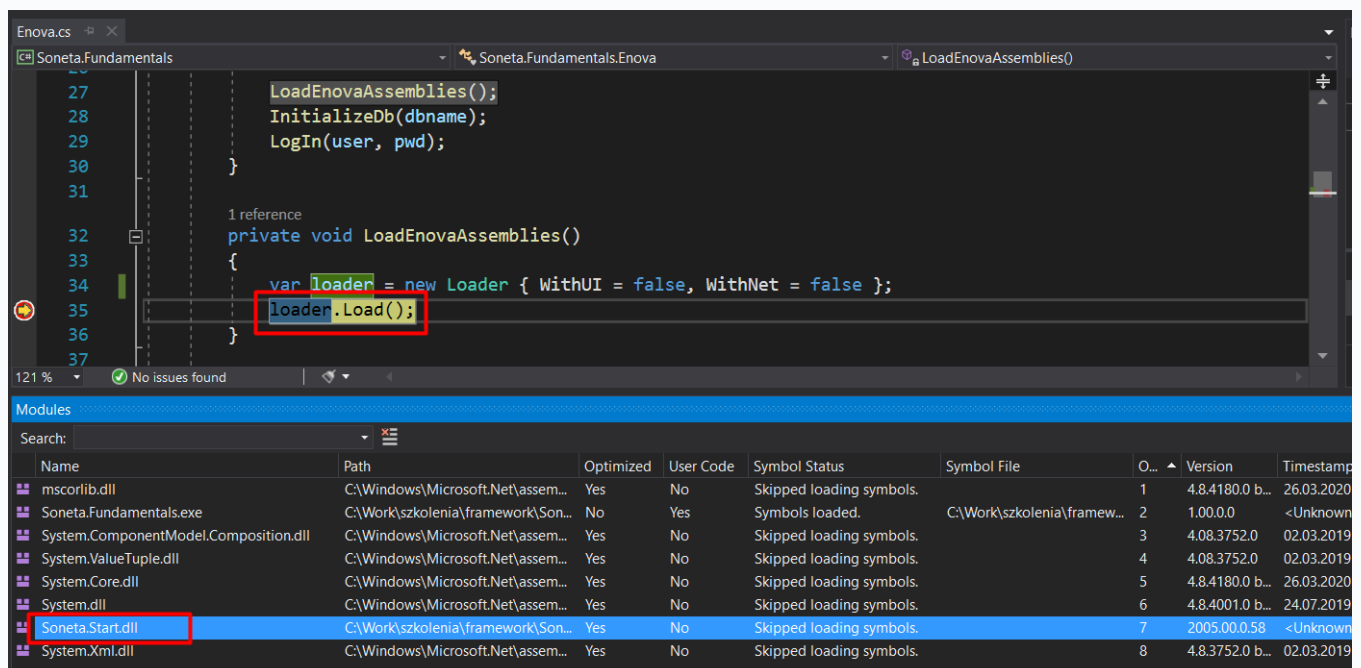
```
var loader = new Loader { WithUI = true, WithNet = false };
loader.Load();
```

Do konstruktora można przekazać opcje. Najważniejsze z nich to

- `WithNet` - kontroluje wczytywanie bibliotek potrzebnych jedynie dla wersji serwerowej; `false` - framework zostanie przygotowany do pracy tak jak ma to miejsce w wersji okienkowej, `true` - framework zostanie przygotowany do pracy tak, jak w wersji serwerowej
- `WithUI` - kontroluje wczytywanie bibliotek powiązanych z UI, a także elementów dot. wydruków

Należy upewnić się, że w momencie wywołania `loader.Load()` biblioteki Soneta nie zostały wczytane wcześniej, przez standardowy mechanizm dynamicznego ładowania bibliotek w środowisku .NET. Użycie klasy Loader powinno zostać tak opakowane, by w tym samym kodzie nie było deklaracji do typów, ani odwołań do klas i metod pochodzących z bibliotek. Jeśli takie odwołania znalazłyby się w tym samym kodzie, w którym uruchamiany jest Loader, środowisko .NET może załadować biblioteki Soneta przed wywołaniem **loader.Load()**.

Aby sprawdzić, czy loader został prawidłowo użyty, można postawić *breakpoint* na wywołaniu metody **Load()** i w oknie *Debug -> Windows -> Modules* sprawdzić, czy biblioteki Soneta nie zostały wczytane wcześniej. W momencie wywołania metody **Load()** jedyną wczytaną automatycznie biblioteką ma być **Soneta.Start.dll**.



The screenshot shows the Visual Studio IDE with the `Enova.cs` file open. The code defines a `LoadEnovaAssemblies()` method that initializes the database and logs in a user. A `private void LoadEnovaAssemblies()` method is also shown, which creates a `Loader` instance and calls `loader.Load()`. The `Modules` window is open at the bottom, displaying a list of loaded modules. The `Soneta.Start.dll` module is highlighted, indicating it is the only one loaded at this point.

Name	Path	Optimized	User Code	Symbol Status	Symbol File	O...	Version	Timestamp
mscorlib.dll	C:\Windows\Microsoft.Net\assem...	Yes	No	Skipped loading symbols.		1	4.8.4180.0 b...	26.03.2020
Soneta.Fundamentals.exe	C:\Work\szkolenia\framework\Son...	No	Yes	Symbols loaded.	C:\Work\szkolenia\framew...	2	1.00.0.0	<Unknown
System.ComponentModel.Composition.dll	C:\Windows\Microsoft.Net\assem...	Yes	No	Skipped loading symbols.		3	4.08.3752.0	02.03.2019
System.ValueTuple.dll	C:\Windows\Microsoft.Net\assem...	Yes	No	Skipped loading symbols.		4	4.08.3752.0	02.03.2019
System.Core.dll	C:\Windows\Microsoft.Net\assem...	Yes	No	Skipped loading symbols.		5	4.8.4180.0 b...	26.03.2020
System.dll	C:\Windows\Microsoft.Net\assem...	Yes	No	Skipped loading symbols.		6	4.8.4001.0 b...	24.07.2019
Soneta.Start.dll	C:\Work\szkolenia\framework\Son...	Yes	No	Skipped loading symbols.		7	2005.00.0.58	<Unknown
System.Xml.dll	C:\Windows\Microsoft.Net\assem...	Yes	No	Skipped loading symbols.		8	4.8.3752.0 b...	02.03.2019

Użycie loader-a powinno być pierwszą operacją aplikacji, przed jakimkolwiek użyciem funkcjonalności pochodzących z bibliotek. Jednak użycie loadera przed innymi operacjami **nie wystarczy** do zapewnienia poprawnej kolejności ładowania bibliotek. W poniższym fragmencie kodu wywołanie loadera jest wykonane przed inicjalizacją zmiennej typu Database. A mimo to, w momencie użycia loadera już załadowane są biblioteki, które powinien wczytać dopiero loader:

```

30 }
31
32 1reference
33 private void LoadEnovaAssemblies()
34 {
35     var loader = new Loader { WithUI = false, WithNet = false };
36     loader.Load();
37     //db = BusApplication.Instance["demo"]; // źle, powoduje przedwczesne ładowanie Soneta.*.dll
38     db = null; // ditto
39 }
40
41 private Database db;

```

Name	Path	Optimized	User Code	Symbol Status	Symbol File	O...	Version	Timestamp
mscorlib.dll	C:\Windows\Microsoft.Net\assem...	Yes	No	Skipped loading symbols.		1	4.8.4180.0 b...	26.03.2020
Soneta.Fundamentals.exe	C:\Work\szkolenia\framework\Son...	No	Yes	Symbols loaded.	C:\Work\szkolenia\framew...	2	1.00.0.0	<Unknown
System.ComponentModel.Composition.dll	C:\Windows\Microsoft.Net\assem...	Yes	No	Skipped loading symbols.		3	4.08.3752.0	02.03.2019
System.ValueTuple.dll	C:\Windows\Microsoft.Net\assem...	Yes	No	Skipped loading symbols.		4	4.08.3752.0	02.03.2019
System.Core.dll	C:\Windows\Microsoft.Net\assem...	Yes	No	Skipped loading symbols.		5	4.8.4180.0 b...	26.03.2020
System.dll	C:\Windows\Microsoft.Net\assem...	Yes	No	Skipped loading symbols.		6	4.8.4001.0 b...	24.07.2019
Soneta.Start.dll	C:\Work\szkolenia\framework\Son...	Yes	No	Skipped loading symbols.		7	2005.00.0.58	<Unknown
Soneta.Business.dll	C:\Work\szkolenia\framework\Son...	Yes	No	Skipped loading symbols.		8	2005.00.0.58	<Unknown
Soneta.Types.dll	C:\Work\szkolenia\framework\Son...	Yes	No	Skipped loading symbols.		9	2005.00.0.58	<Unknown
System.Xml.dll	C:\Windows\Microsoft.Net\assem...	Yes	No	Skipped loading symbols.		10	4.8.3752.0 b...	02.03.2019

Uwaga!

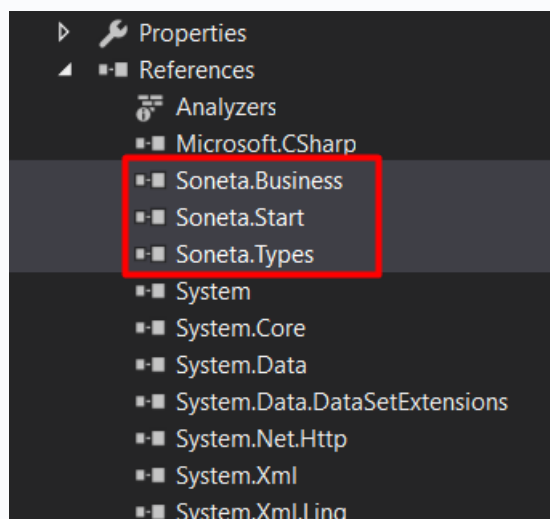
Należy podkreślić, że błędem jest tutaj nie kolejność operacji, bo użycie loadera **jest** w tym przykładzie wykonane jako pierwsze. Problemem jest **sam fakt użycia w tej samej metodzie / klasie odwołania do typu pochodzącego z bibliotek Soneta**. Inaczej mówiąc, użycie loadera zostało nieumiejętnie opakowane. Kompilator C# wygeneruje z takiej metody kod, który spowoduje wczytanie bibliotek potrzebnych do jego wykonania zanim loader zostanie faktycznie uruchomiony.

W .NET nie mamy ścisłej kontroli nad tym, kiedy środowisko wczyta DLL-ki. W omawianym przykładzie samo użycie zmiennej Database db spowodowało przedwczesne wczytanie **Soneta.business.dll** oraz **Soneta.types.dll**. Najlepszą praktyką jest użycie loadera w odizolowanej klasie, w której nie ma odwołań do innych typów, klas, metod framework-u enova365.

Ładowanie bibliotek przez loader odbywa się tylko raz. Wielokrotne wywołanie metody loader.Load() czy tworzenie wielu instancji loadera nie ma sensu. Statyczne property Loader.IsLoaded zawiera informację, czy biblioteki zostały załadowane. Metoda Load() kończy działanie, jeśli flaga ma wartość true. Flaga ta jest **tylko do odczytu**. W wyjątkowych sytuacjach można użyć metody loader.ResetLoaded() to zresetowania stanu tej flagi. Powoduje to tylko zmianę wartości IsLoaded na false. Nie ma możliwości wyładowania wczytanych bibliotek.

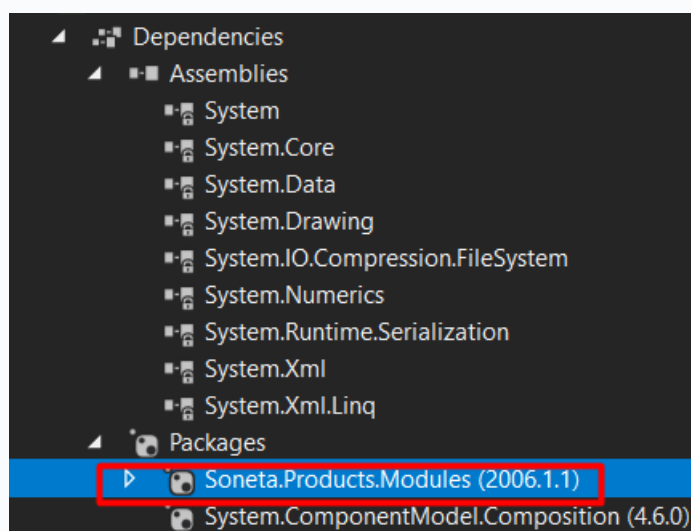
Zależności w projekcie

Należy zwrócić uwagę na sposób, w jaki ustawione są zależności projektu od bibliotek Soneta. Częstym błędem jest ręczne ustawianie zależności od wybranych bibliotek, jak na poniższej ilustracji:



Mimo iż instrukcja `Loader.Load()` może wykonać się bez przeszkód, brak odpowiednich zależności może powodować problemy w nieoczekiwanych miejscach, np. błąd trakcie wykonywania instrukcji `database.Login()`.

Prawidłowym sposobem jest wprowadzenie zależności od paczki nuget, np. [Soneta Product Modules](#):



W pliku projektu (*.csproj) zależność ta opisana jest w ten sposób:

```
<PackageReference Include="Soneta.Products.Modules" Version="2110.0.0" />
```

Oba wpisy w XML dodawane są automatycznie, nie ma potrzeby ręcznej modyfikacji. Przy aktualizacji dodatku do nowych wersji enova365, zazwyczaj wystarcza zmiana atrybutu `Version`, odczekanie aż kompilator dociągnie stosowne paczki nuget i przekompilowanie kodu.

W przypadku dodatków w formie EXE zależności w projekcie powinny być opisane w specyficzny sposób:

```
<PackageReference Include="Soneta.Products.Modules" Version="2012.4.5"
```

```
  PrivateAssets="all" ExcludeAssets="runtime"/>
```

```
<PackageReference Include="Soneta.Start" Version="2012.4.5"/>
```

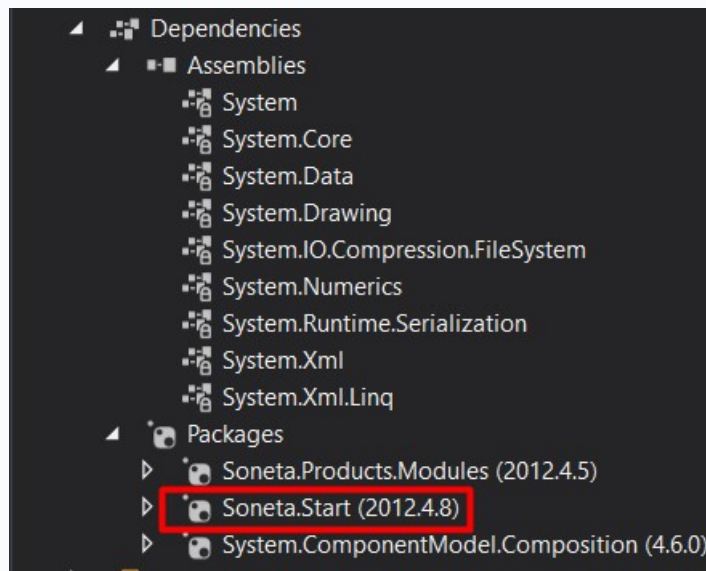
Jeśli pominiemy dodatkowe atrybuty `PrivateAssets` oraz `ExcludeAssets` z wpisu dotyczącego odwołania do `Soneta.Products.Modules`, kompilator skopiuje wszystkie DLL-ki z tego pakietu do katalogu wyjściowego, razem z docelowym plikiem wykonywalnym. Podczas uruchamiania tego pliku zostanie zapewne uruchomiony loader, który wczyta DLL-ki z tej właśnie lokalizacji. Takie rozwiązanie może wydawać się poprawne, ale niesie ryzyko trudnych w diagnostyce błędów systemu. Na przykład, niektóre biblioteki Soneta mogą zależeć od innych bibliotek, których nie ma w pakiecie `Soneta.Product.Modules` (choćby wszystkie biblioteki UI). Tym samym nie będzie ich wśród bibliotek skopiowanych do katalogu wyjściowego podczas kompilacji dodatku EXE. Brak takiej zależności może spowodować błędy przy podstawowych operacjach, takich jak logowanie do bazy danych.

```
db = BusApplication.Instance[nazwa_bazy];
```

```
/* w tym miejscu prawdopodobny wyjątek przy nieprawidłowych ustawieniach csproj */
```

Prawidłową konfiguracją jest wczytywanie bibliotek Soneta z **katalogu instalacyjnego enova365**. Dodatek do enova365, także w formie EXE, powinien pracować z **zainstalowaną wersją enova365**, a nie z zestawem bibliotek ściągniętych / skopiowanych z paczki nuget lub katalogu enova. W tym kontekście, program EXE korzystający z bibliotek enova365 nie jest w pełni samodzielny, pozostaje **dodatkiem** do systemu.

Wyjątkiem od tej zasady jest biblioteka `Soneta.Start.dll`, która musi zostać wczytana przez .NET, żeby dodatek w ogóle mógł skorzystać z klasy `Loader` (w tej bibliotece znajduje się m.in. implementacja `loader-a`). Stąd też drugi wpis w csproj dodający zależność od `Soneta.Start`.



W odróżnieniu od reszty bibliotek, ta biblioteka, jako jedyna, musi zostać skopiowana do katalogu wyjściowego wraz z plikiem EXE. Zostanie wczytana z tego katalogu zaraz po uruchomieniu dodatku. Pozostałe biblioteki powinny zostać wczytane z katalogu instalacyjnego enova365, i tym się zajmuje omówiony wyżej loader.

W przypadku tworzenia standardowych dodatków przy użyciu szablonów [Soneta Platform Developer](#), umożliwiającego korzystanie z [Soneta SDK](#) w Visual Studio, wszystkie zależności w plikach projektu są zadane prawidłowo i nie ma potrzeby ręcznej konfiguracji dla dodatków DLL. Dodatki w formie EXE wymagają opisanych modyfikacji.

Gdzie loader szuka bibliotek

Kiedy Loader potrzebuje ścieżki do katalogu z bibliotekami, wylicza ją wg następującej logiki:

1. Wśród wczytanych assemblies wyszukiwana jest biblioteka **Soneta.business.dll**. Jeśli taka zostanie znaleziona, katalog z którego została wczytana jest traktowany jako ścieżka do wczytywania wszystkich pozostałych. W momencie wywołania metody Load() biblioteka zazwyczaj nie jest jeszcze wczytana, więc algorytm przejdzie do drugiego kroku.
2. Sprawdzany jest katalog, z którego została uruchomiona aplikacja, czyli ten sam katalog, w którym znajduje się program (AppDomain.CurrentDomain.BaseDirectory). Jeśli w tym katalogu znajdują się biblioteki Soneta (**Soneta.business.dll**), to katalog jest traktowany jako ścieżka wszystkich assemblies Soneta.
3. W ostatnim kroku wczytywany jest plik **Modules.xml** znajdujący się w **%USERPROFILE%\AppData\Local\Soneta\Modules.xml**, np.:

```
<LoadSettings>
  <Installations>
    <Installation>C:\enova365server\2001\enova365Server</Installation>
    <Installation>C:\Program Files (x86)\Soneta\enova365 2002.1.1.17903</Installation>
    <Installation>C:\Program Files (x86)\Soneta\enova365 2003.2.2.18000</Installation>
    <Installation>C:\Program Files (x86)\Soneta\enova365 2004.0.2.18046</Installation>
    <Installation>C:\Program Files (x86)\Soneta\enova365 2005.2.2.18141</Installation>
    <Installation>C:\Program Files (x86)\Soneta\enova365 2006.0.0.682</Installation>
    ...
    <Installation>C:\Program Files (x86)\Soneta\enova365 2110.0.0.777</Installation>
  </Installations>
</LoadSettings>
```

Ten plik aktualizuje się sam wraz z kolejnymi instalowanymi wersjami enova. Plik czytany jest od ostatniego wpisu. W każdej ze ścieżek sprawdzana jest obecność bibliotek Soneta (**Soneta.business.dll**). Pierwsza taka ścieżka traktowana jest jako katalog bibliotek Soneta.

Jeśli nie uda się znaleźć katalogu bibliotek, pojawia się błąd Nie znaleziono instalacji programu enova365.

enova 365

dla biznesu

Soneta Sp.z o.o.
ul. Wadowicka 8A, 30-415 Kraków, tel. +48 12 349 28 00,
e-mail: kontakt@enova.pl,