.NET - Technical notes and limitations

On this page:

Summary: This section provides more detail about the support for specific .NET technologies and the way in which the they are supported.

ASP.NET web applications

Unused files inside of web site application folders are ignored during the analysis.

ASP.NET MVC Razor

ASP.NET LMVC Razor is supported through the HTML5 and JavaScript extension. Please see .NET - ASP.NET MVC Razor support for more information.

ASP classic

See ASP for more information.

.NET Core

.NET Core is supported when using **CAST AIP 8.3.5** and the **.NET Analyzer extension 1.1.0**.

Silverlight

Silverlight is supported through a Silverlight extension, however, .NET projects are only recognized as Silverlight projects if their CSPROJ file contains the following XML tag:

```
<TargetFrameworkIdentifier>Silverlight</TargetFrameworkIdentifier>
```

If a project is a Silverlight project but it does not contain this tag, then during the packaging process with the CAST Delivery Manager Tool alerts may be generated indicating missing .NET libraries and assemblies. These alerts may persist even though the required files are present in one or more .NET packages. To avoid this issue, ensure that the required XML tag is present in the source code and then repackage your packages.

Language Integrated Query (LINQ)

Current support of **LINQ** is limited to: **LINQ to Objects**, **LINQ to DataSets** and **LINQ to SQL** providers only. No other provider is supported.

LINQ to Objects

With regard to LINQ to Objects, in the following example:

- A link will be created to the object "customer" (if it is not declared as a local variable)
- A link will be created to the field "city"

```
var queryLondonCustomers = from cust in customers
        where cust.City == "London"
        select cust;
```

Note that currently the following is NOT supported:

- Call to LINQ extension methods, such as the Where method, for example:

```
public static IEnumerable<TSource> Where<TSource>(
        this IEnumerable<TSource> source,
        Func<TSource,bool> predicate
)
```

- Where a LINQ statement implicitly calls a LINQ extension method - no Access link is created for these calls

LINQ to DataSets

With regard to LINQ to DataSets, in the following example:

- A **Use Select** link will be created between the "myMethod" containing the declaration of the table of the dataset and the "Product" database table:

```
// Fill the DataSet.
public void myMethod()
{
        DataSet ds = new DataSet();
        ds.Locale = CultureInfo.InvariantCulture;
        FillDataSet(ds);
        DataTable products = ds.Tables["Product"];
        IEnumerable<string> query =
            from product in products.AsEnumerable()
            select product.Field<string>("Name");
        Console.WriteLine("Product Names:");
        foreach (string productName in query)
                {
                    Console.WriteLine(productName);
                }
}
```

LINQ to SQL

With regard to LINQ to SQL, in the following example:

- A **Use Insert** link will be created from the method "InsertStudent" to the table "Student"
- A **Use Delete** link will be created from the method "RemoveStudent" to the table "Student"
- A **Use Select** link and a **Use Update** link will be created from the method "ChangeTitle" to the table "Course"

```
void InsertStudent(string firstName, string lastName, SchoolDataContext db)
{
    Console.WriteLine("\nAdd a new student to the Person table");

    // Create a new Student.
    Student student = new Student
    {
        FirstName = firstName,
        LastName = lastName,
        EnrollmentDate = DateTime.Now
    };

    // Add the new object to Students.
    db.Students.InsertOnSubmit(student);

    // Submit the change to the database.
    db.SubmitChanges();
}

void RemoveStudent(string firstName, string lastName, SchoolDataContext db)
{
    Student student = new Student
    {
        FirstName = firstName,
        LastName = lastName
    };

    // Deletes the student
    db.Students.DeleteOnSubmit(student);

    // Submit the change to the database.
    db.SubmitChanges();
}

void ChangeTitle(int id, string newTitle, SchoolDataContext db)
{
    // Retrieve the course from database
    var course =
        (from c in db.Courses
        where c.CourseID == id
        select c).Single();

    // Change the title
    course.Title = newTitle;

    // Submit the change to the database.
    db.SubmitChanges();
}
```

.NET WebServices

- **ASP.NET WebServices** are supported "out of the box" by the .NET analyzer as standard C#/VB.NET projects.
- **WCF WebServices** are supported through the separate installation of a CAST AIP extension - see WCF for more information.

Links from web front end > WebService > back-end database

If the entire transaction from a **web front end** > **WebService** > **back-end database** needs to be resolved, then further configuration is required to support this, as outlined below:

> Please note that if the web front end is an ASP.NET "website" (i.e. no project file (.csproj or .vbproj) exists), then no links will be created from this website to the WebServices. This is a limitation in the .NET analyzer.

Scenario 1

If your web front end is written in **ASP.NET** and you use **ASP.NET WebServices**, then:

- The .NET analyzer will handle the analysis of both the **front end web application** and the **WebServices**
- The Web Services Linker extension will resolve the full transaction from web application to database. This extension is installed automatically as a dependency with the .NET Analyzer extension and there is no further configuration to do.

In this scenario, the following will be resolved:

- In C# and VB.NET projects the web application counter part of ASP.NET WebServices technology is called a "Web Reference" object. Links will be created from the Web Reference objects to ASP.NET WebServices. ASP.NET WebServices may also be targeted by some other technologies.
- A call link will be created from the proxy method (a C# or VB.NET method having the attribute System.Web.Services.Protocols.SoapDocumentMethodAttribute) to an object typed ".NET SOAP service reference". These kind of objects are created for each proxy method, their name is the URL of the targeted web method.
- For the database side, an object typed ".NET SOAP operation" will be created, from which is drawn a call link to the web method (a C# or VB.NET method having the attribute System.Web.Services.WebMethodAttribute). An object ".NET SOAP operation" is created for each web method, their name is the URL of the web method.
- The Web Services Linker extension will create a call link from ".NET SOAP service reference" objects to their matching database counterpart ".NET SOAP operation" object.

Scenario 2

If your web front end is written in **ASP.NET** and you use **WCF WebServices**, then:

- The .NET analyzer will handle the analysis of the **front end web application only**
- You will need to install, separately, the WCF Support for C# and VB.NET Extension to handle the **WCF WebServices** - doing so will mean you will automatically get the **WBS Linker Extension** as a dependency to the **WCF Support for C# and VB.NET Extension** to resolve the full transaction from web application to database.

Scenario 3

If your web front end is written in some other language (such as **HTML5/AngularJS**) and you use **ASP.NET WebServices**, then:

- You will need to install the appropriate extension to handle the web front end code
- The .NET analyzer will handle the analysis of the **WebServices**
- The Web Services Linker extension will resolve the full transaction from web application to database. This extension is installed automatically as a dependency with the .NET Analyzer extension and there is no further configuration to do.

Scenario 4

If your web front end is written in some other language (such as **HTML5/AngularJS**) and you use **WCF WebServices**, then:

- You will need to install the appropriate extension to handle the web front end code
- You will need to install, separately, the WCF Support for C# and VB.NET Extension to handle the **WCF WebServices** - doing so will mean you will automatically get the **WBS Linker Extension** as a dependency to the **WCF Support for C# and VB.NET Extension** to resolve the full transaction from web application to database.

.datasource files

Any **.datasource** files that are referenced in the parent .csproj/.vbproj file will be listed as "**missing source files**" when delivering source code with the CAST Delivery Manager Tool. .datasource files are ignored by the DMT and are never delivered. These missing source file messages can therefore be ignored.

Generated code

The .NET analyzer handles auto generated code like all other CAST AIP analyzers:

- Auto generated code is analyzed to help understand the entire code being analyzed
- Objects are created from the code and saved in the CAST Analysis Service schema (to help trace transactions for example) and these objects are marked as being "generated"
- Any Quality Rule violations that are caused by these "generated" objects do not contribute to grade calculations
- "Generated" objects are excluded from any aggregated metrics (for example Lines of Code (LOC))

The .NET analyzer determines whether code is auto generated using a combination of the following factors:

Files that end with the following:

- "_CastGenerated.cs"
- "_CastGenerated.vb"
- ".Designer.cs"
- ".Designer.vb"
- "Reference.cs"
- "Reference.vb"

Note that to be considered "generated", files that end with **"Reference.cs"** and **"Reference.vb**" MUST ALSO be located in one of the following folders:

- "Web References"
- "Service References"

Symbols marked with one of the following attributes:

- "System.CodeDom.Compiler.GeneratedCodeAttribute"
- "System.Diagnostics.DebuggerNonUserCodeAttribute"

The following object types:

- Accessors for auto-generated properties
- Implicitly generated functions (constructor by default)
- Anonymous types
- Instances of generic types or methods
- Objects with no bookmark in the source code

Files that contain top level comments that include the following:

- "<autogenerated>"
- "<auto-generated>"

COM objects

The CAST .NET analyzer does not support the resolution of COM objects (such as ADODB) referenced in .NET projects.

Miscellaneous

- If a .NET project (either C# or VB.NET) depends on an assembly (or a project) that declares a class with the same name full name (i.e. same name, in the same namespace) as a class in said project, then that class is not saved to the Analysis Service.
- XML code embedded into VB is not taken into account. The code will be analyzed, but its contents will be ignored.
- Note that **.exe** and **.netmodules** dependencies are not supported.
- If a DLL file is placed in the "bin" folder of a .NET project, but its exact location is not defined in the .csproj or .vbproj project file, the CAST Delivery Manager Tool may not be able to identify the reference during the package action. To resolve this issue, you may need to create an additional package using the **Automated extraction or required .NET assemblies** option specifically to package the DLLs in the "bin folder.
- Please avoid **accidentally duplicating source code** as it might considerably increase analysis time during the "linker" phase. When CAST schemas are hosted on an Oracle Server, the analysis might not complete at all. Source code can easily get duplicated by accident when upgrading Visual Studio. Indeed, Visual Studio automatically creates a backup folder inside the project folder. This backup folder contains a copy of the project file (.csproj, .vbproj) which will lead to duplicate analysis of the source code if this folder is delivered via the CAST Delivery Manager Tool - you should therefore ensure that the backup folder is excluded from delivery.