CodePlex.Diagnostics

Version 2.0.0.4

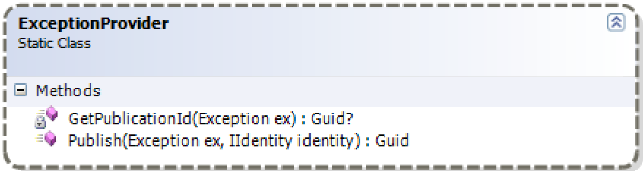
Introduction

Welcome to the second major release of the `CodePlex.Diagnostics` framework, designed to enable exception publication and logging through components based upon the ASP.NET 2.0 provider design pattern.

Using this framework you can write your applications and defer the decision of which exception providers will be used to publish exceptions until deployment time. In this release, the `SqlExceptionProvider` publishes the exceptions to a SQL Server 2005 / SQL Server 2008 database. The database T-SQL creation script is included within the `CodePlex.Diagnostics` Visual Studio 2008 solution.

Use of SQL Server 2005 or SQL Server 2008, as opposed to previous releases, is important because it allows the exceptions to be serialized using the `SoapFormatter` class. The Soap encoded exceptions are then stored within the database using the `Xml` column type. Columns are included within the `Exception` table for most properties within the base `System.Exception` type but the serialized exceptions are there to allow a developer to browse through any additional information provided by exceptions that are either directly or indirectly derived from the `System.Exception` base type. In most cases the properties within `System.Exception` have corresponding columns within the `Exception` table; the `TargetSite` property however has a corresponding `TargetSite` table.

Central to the `CodePlex.Diagnostics` framework is the static `ExceptionProvider` class shown in the following Visual Studio 2008 class diagram. As you can see, the `ExceptionProvider` contains two methods, one of which is public and requires two arguments. The first argument is the exception to be published and the second argument is the `IIdentity` representing the user on whose behalf the code was executing. Within a Windows Forms application this is usually the `WindowsIdentity` of the current user, however within an ASP.NET application an `IIdentity` representing the user who is authenticated to the Web site should be provided.

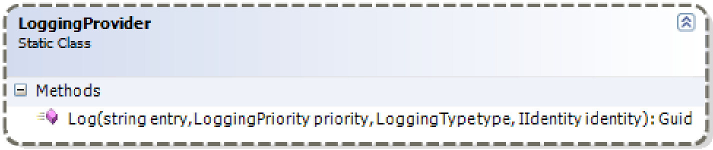


ExceptionProvider
Static Class

⊟ Methods
GetPublicationId(Exception ex) : Guid?
Publish(Exception ex, IIdentity identity) : Guid

Figure 1. `ExceptionProvider` class.

Another central type within the `CodePlex.Diagnostics` framework is the static `LoggingProvider` class, shown in the next Visual Studio 2008 class diagram, which provides logging capabilities. As with the publication of exceptions, the `LoggingProvider` class is designed to allow the decision of where log entries should be persisted to be deferred until deployment time. The default `SqlLoggingProvider` class persists log entries within the same SQL Server 2005 or SQL Server 2008 database.



LoggingProvider
Static Class

⊟ Methods
Log(string entry,LoggingPriority priority,LoggingTypetype, IIdentity identity): Guid

Figure 2. `LoggingProvider` class.

When using the static **LoggingProvider** class, the log entry is simply a **System.String**, however you must also determine the priority of the entry and the type of information the entry provides. The **LoggingPriority** enumeration allows you to indicate that the entry is of **Low**, **Medium**, **High**, or **Critical** priority. Additionally, using the **LoggingType** enumeration, the log entry can be described as **Information**, **Warning**'s or an **Error**. The fourth argument that the **Log** method expects is the **IIdentity** which, as with the **ExceptionProvider**, represents the user on whose behalf the code was executing.

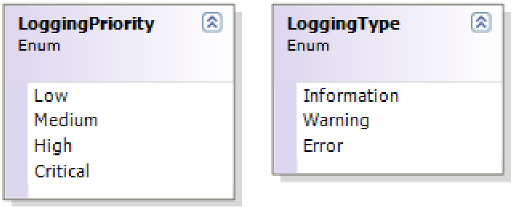


Figure 3. **LoggingPriority** and **LoggingType** enumerations.

Default Providers

The **CodePlex.Diagnostics** framework contains default exception and logging provider classes although you are free to construct other exception and logging providers, and then configure the framework accordingly. The default exception and logging providers are the **SqlExceptionProvider** and **SqlLoggingProvider** classes, these providers, as mentioned previously, store the exceptions and log entries within a SQL Server 2005 or SQL Server 2008 database. Figure 4 provides a Visual Studio 2008 class diagram depicting these two classes.

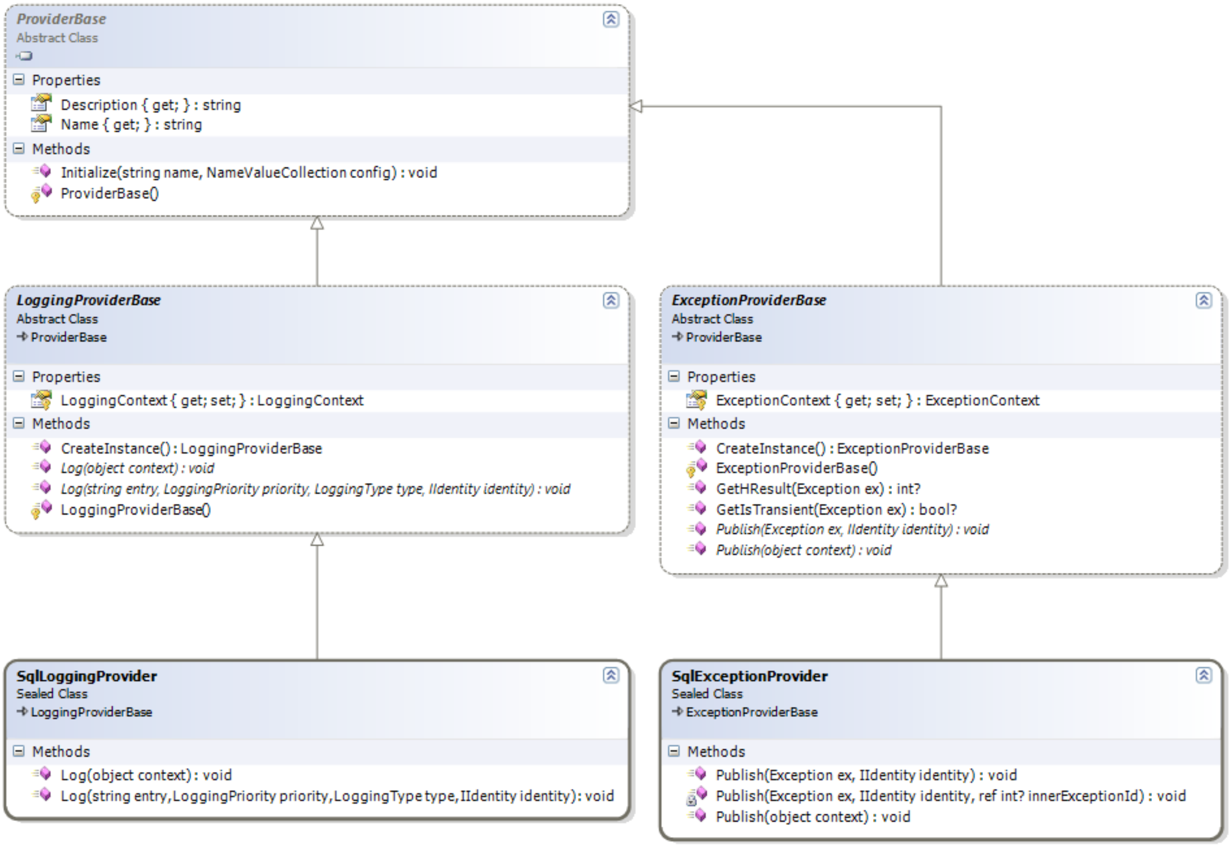

ProviderBase
Abstract Class
Properties
Description { get; } : string
Name { get; } : string
Methods
Initialize(string name, NameValueCollection config) : void
ProviderBase()
LoggingProviderBase
Abstract Class
ProviderBase
Properties
LoggingContext { get; set; } : LoggingContext
Methods
CreateInstance() : LoggingProviderBase
Log(object context) : void
Log(string entry, LoggingPriority priority, LoggingType type, IIdentity identity) : void
LoggingProviderBase()
ExceptionProviderBase
Abstract Class
ProviderBase
Properties
ExceptionContext { get; set; } : ExceptionContext
Methods
CreateInstance() : ExceptionProviderBase
ExceptionProviderBase()
GetHResult(Exception ex) : int?
GetIsTransient(Exception ex) : bool?
Publish(Exception ex, IIdentity identity) : void
Publish(object context) : void
SqlLoggingProvider
Sealed Class
LoggingProviderBase
Methods
Log(object context) : void
Log(string entry,LoggingPriority priority,LoggingType type,IIdentity identity) : void
SqlExceptionProvider
Sealed Class
ExceptionProviderBase
Methods
Publish(Exception ex, IIdentity identity) : void
Publish(Exception ex, IIdentity identity, ref int? innerExceptionId) : void
Publish(object context) : void

Figure 4. **SqlExceptionProvider** and **SqlLoggingProvider** types.

Within the `System.Configuration` assembly resides the `System.Configuration.Provider` namespace which is where you'll find the `ProviderBase` abstract base class shown in figure 4. The `ProviderBase` class forms the foundation for the ASP.NET 2.0 Provider design pattern, upon which this frameworks design is based. Subsequently, the `LoggingProviderBase` and `ExceptionProviderBase` abstract classes allow you to define specific providers such as the aforementioned default providers.

Configuration

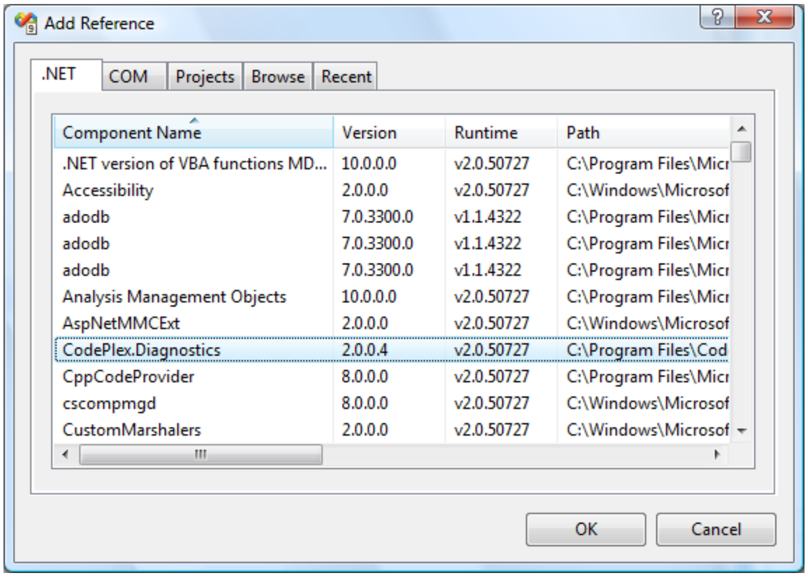
Configuration of the `CodePlex.Diagnostics` framework is achieved using the application domain's configuration file (e.g. `App.config` or `Web.config`). Some additional sections, within which, are required to allow the framework to determine the correct exception and logging providers at runtime. For an example of the Xml configuration sections required by the framework you should examine the `App.Config` from the unit test project `CodePlex.Diagnostics.External.UnitTests`.

Although these additional Xml sections are required within the `App.Config` or `Web.Config` for the framework to determine the correct providers, it is also permissible to omit them entirely. Essentially, the framework is designed to operate in a "*fire and forget*" mode which implies that if the publication of an exception or log entry fails then the calling code itself will be unaware of this failure. This is by design, because at no time should the use of this library inhibit the calling code itself.

Assuming you have access to the Visual Studio 2005 Team Developer or Visual Studio 2005 Team Suite editions then it is suggested that you use the various unit test projects included within the `CodePlex.Diagnostics` solution. Once Visual Studio 2008 is released the Professional edition will also contain support for unit testing.

These unit test projects show the intended use of the various types defined within the library and the `App.config` from the unit test project is a good starting point for using the framework within your own projects. If you have other versions of Visual Studio 2008 (including the express editions), you'll still be able to use the framework although some of the projects within the solution might not function due to the limitations within the particular edition you are using.

Once you've installed the `CodePlex.Diagnostics` library, using the setup program that is included with the solution, the `CodePlex.Diagnostics` library will appear as an option within the Visual Studio 2008 "Add Reference" dialog (see figure 5).



Component Name	Version	Runtime	Path
.NET version of VBA functions MD...	10.0.0.0	v2.0.50727	C:\Program Files\Micr
Accessibility	2.0.0.0	v2.0.50727	C:\Windows\Microsof
adodb	7.0.3300.0	v1.1.4322	C:\Program Files\Micr
adodb	7.0.3300.0	v1.1.4322	C:\Program Files\Micr
adodb	7.0.3300.0	v1.1.4322	C:\Program Files\Micr
Analysis Management Objects	10.0.0.0	v2.0.50727	C:\Program Files\Micr
AspNetMMCExt	2.0.0.0	v2.0.50727	C:\Windows\Microsof
CodePlex.Diagnostics	2.0.0.4	v2.0.50727	C:\Program Files\Cod
CppCodeProvider	8.0.0.0	v2.0.50727	C:\Program Files\Micr
cscompmgd	8.0.0.0	v2.0.50727	C:\Windows\Microsof
CustomMarshalers	2.0.0.0	v2.0.50727	C:\Windows\Microsof

Figure 5. Visual Studio 2008 *Add Reference* dialog showing the `CodePlex.Diagnostics` library.

Assuming you create a C# Console Application and add a reference to the **`CodePlex.Diagnostics`** library, then you can then use the following code to test the publication of exceptions and logging events to the supplied **`CodePlex.Diagnostics`** database. If you haven't done so already, run the database creation script that is included within the source code for the library within SQL Server 2005 or SQL Server 2008 Management Studio.

```
using System;

using System.Security;
using System.Security.Principal;

using CodePlex.Diagnostics;
using CodePlex.Diagnostics.Providers;

namespace CodePlex.Diagnostics.Samples
{
    public static class Program
    {
        #region private static void Main(string[] args)

        /// <summary>
        /// Program entry point.
        /// </summary>
        /// <param name="args">
        /// An array of <see cref="T:System.String"/> containing the command line arguments.
        /// </param>
        [STAThread]
        private static void Main(string[] args)
        {
            try
            {
                for (int index = 0; index < 10; index++)
                {
                    index /= index;
                }
            }
            catch (DivideByZeroException ex)
            {
                IIdentity identity = WindowsIdentity.GetCurrent() as IIdentity;
                ExceptionProvider.Publish(ex, identity);
            }

            Console.ReadLine();
        }

        #endregion
    }
}
```

Listing 1. Sample C# Console Application to publish an exception using the static **`ExceptionProvider`** class.

You'll also need to add the appropriate Xml nodes to the **`App.config`** which can be copied from the **`App.config`** within the **`CodePlex.Diagnostics.External.UnitTests`** project, remembering to change the SQL connection string to reference the server upon which you installed the **`CodePlex.Diagnostics`** database.

Using SQL Server 2005 or SQL Server 2008 Management Studio (or other similar tools), you'll observe the data that was collected when the **`DivideByZeroException`** was caught and subsequently published to the **`CodePlex.Diagnostics`** database using the **`ExceptionProvider`** class.

```
USE [CodePlex.Diagnostics]

SELECT * FROM Exception
SELECT * FROM LogEntry
SELECT * FROM AppDomain                  -- new in build 1.0.0.8
SELECT * FROM Assembly                   -- new in build 1.0.0.8
SELECT * FROM GraphicsProcessor          -- new in build 1.0.0.15
SELECT * FROM Machine                    -- new in build 1.0.0.15
SELECT * FROM MachineGraphicsProcessor   -- new in build 1.0.0.15
SELECT * FROM MachineProcessor           -- new in build 1.0.0.15
SELECT * FROM Processor                  -- new in build 1.0.0.15
SELECT * FROM TargetSite
SELECT * FROM Thread                    -- new in build 1.0.0.8
SELECT * FROM WorkItem                   -- new in build 1.0.0.60
SELECT * FROM TeamFoundationServer       -- new in build 1.0.0.60
```

Listing 2. Simple T-SQL select statements to show the content of the various tables used by the framework.

If all went according to plan then you should see something such as the screenshot of SQL Server 2008 Management Studio found in figure 8 which is on the last page of this document, although possibly containing only a single exception. In build 1.0.0.60, new **WorkItem** and **TeamFoundationServer** tables were added to the database to enable exceptions and log entries to be promoted to Team Foundation Server work items. These tables are merely place holders for future functionality which will be enabled within the forthcoming **Diagnostics Studio** smart-client application.

PublishedException

The **CodePlex.Diagnostics** framework includes three new **Exception** types that can be used to either control the behavior of the framework or add additional meta-data regarding the exceptions that were caught and subsequently published. The first of these new **Exception** types is the **PublishedException** class which is shown along with the **ThreadException** and **UnhandledException** in figure 7.

The code fragment in listing 3 shows how the **DivideByZeroException** can be thrown again as the inner-exception of a **PublishedException**, ensuring that the original exception is published only once by the **ExceptionProvider**. Subsequent calls to the class **ExceptionProvider**, further up the call stack, will ignore an exception that has already been published.

```
catch (DivideByZeroException ex)
{
    IIdentity identity = WindowsIdentity.GetCurrent() as IIdentity;

    Guid publicationId = ExceptionProvider.Publish(ex, identity);

    throw new PublishedException(publicationId, ex);
}
```

Listing 3. Using the **PublishedException** class to indicate an exception has already been published.

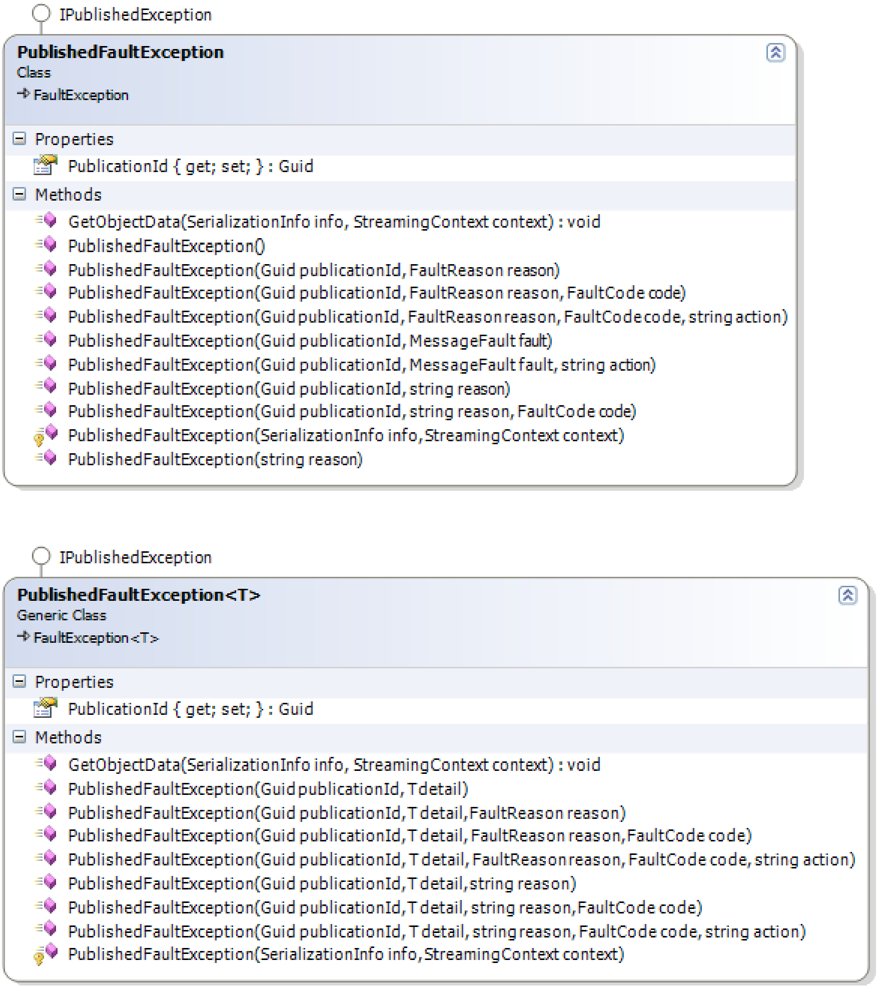
PublishedFaultException

When Microsoft released version 3.0 of the .NET Framework one of the major new additions to the framework was the Windows Communication Foundation (WCF) which provides architects and developers with a framework for building next generation Web services. Within the **`System.ServiceModel`** namespace is the **`FaultException`** type which is the type of exception that clients receive from WCF services. For more information on WCF and exception handling within WCF see Programming WCF Services 2nd Edition by Juval Löwy.

Within version 2.0.0.4 of **`CodePlex.Diagnostics`** the **`PublishedFaultException`** was added to the framework to allow exceptions to be thrown from the service layer indicating to code within the client that the original exception has been successfully published. Also beginning with version 2.0.0.4 the **`IPublishedException`** interface has been added to the framework and can be seen within figure 6 below.

PublishedFaultException<T>

Within the **`System.ServiceModel`** namespace there is also a generic **`FaultException<T>`** which derives from the aforementioned **`FaultException`** type. As such there is also a generic **`PublishedFaultException<T>`** within the CodePlex.Diagnostics framework which derives from the WCF **`FaultException<T>`** class.



IPublishedException

PublishedFaultException
Class
→ FaultException

Properties
- PublicationId { get; set; } : Guid

Methods
- GetObjectData(SerializationInfo info, StreamingContext context) : void
- PublishedFaultException()
- PublishedFaultException(Guid publicationId, FaultReason reason)
- PublishedFaultException(Guid publicationId, FaultReason reason, FaultCode code)
- PublishedFaultException(Guid publicationId, FaultReason reason, FaultCode code, string action)
- PublishedFaultException(Guid publicationId, MessageFault fault)
- PublishedFaultException(Guid publicationId, MessageFault fault, string action)
- PublishedFaultException(Guid publicationId, string reason)
- PublishedFaultException(Guid publicationId, string reason, FaultCode code)
- PublishedFaultException(SerializationInfo info, StreamingContext context)
- PublishedFaultException(string reason)

IPublishedException

PublishedFaultException<T>
Generic Class
→ FaultException<T>

Properties
- PublicationId { get; set; } : Guid

Methods
- GetObjectData(SerializationInfo info, StreamingContext context) : void
- PublishedFaultException(Guid publicationId, T detail)
- PublishedFaultException(Guid publicationId, T detail, FaultReason reason)
- PublishedFaultException(Guid publicationId, T detail, FaultReason reason, FaultCode code)
- PublishedFaultException(Guid publicationId, T detail, FaultReason reason, FaultCode code, string action)
- PublishedFaultException(Guid publicationId, T detail, string reason)
- PublishedFaultException(Guid publicationId, T detail, string reason, FaultCode code)
- PublishedFaultException(Guid publicationId, T detail, string reason, FaultCode code, string action)
- PublishedFaultException(SerializationInfo info, StreamingContext context)

Figure 6. PublishedFaultException and PublishedFaultException‹T›.

ThreadException

In addition to the aforementioned **`PublishedException`**, the framework also contains the new `ThreadException` class, which indicates that the exception was caught within an event handler for the **`ThreadException`** event. The **`ThreadException`** event is located within the **`Application`** class from the **`System.Windows.Forms`** namespace.

UnhandledException

Finally, the framework contains the new **`UnhandledException`** class, which indicates that an exception was caught within an event handler for the **`UnhandledException`** event. The **`ThreadException`** event is located within the **`AppDomain`** class.

```
AppDomain.CurrentDomain.UnhandledException += delegate(object sender,
                                                UnhandledExceptionEventArgs e)

   {
       Exception ex = e.ExceptionObject as Exception;

       IIdentity identity = WindowsIdentity.GetCurrent() as IIdentity;

       ExceptionProvider.Publish(new UnhandledException(ex), identity);
   };
```

Listing 4. Using the **`UnhandledException`** class to indicate an exception was unhandled.

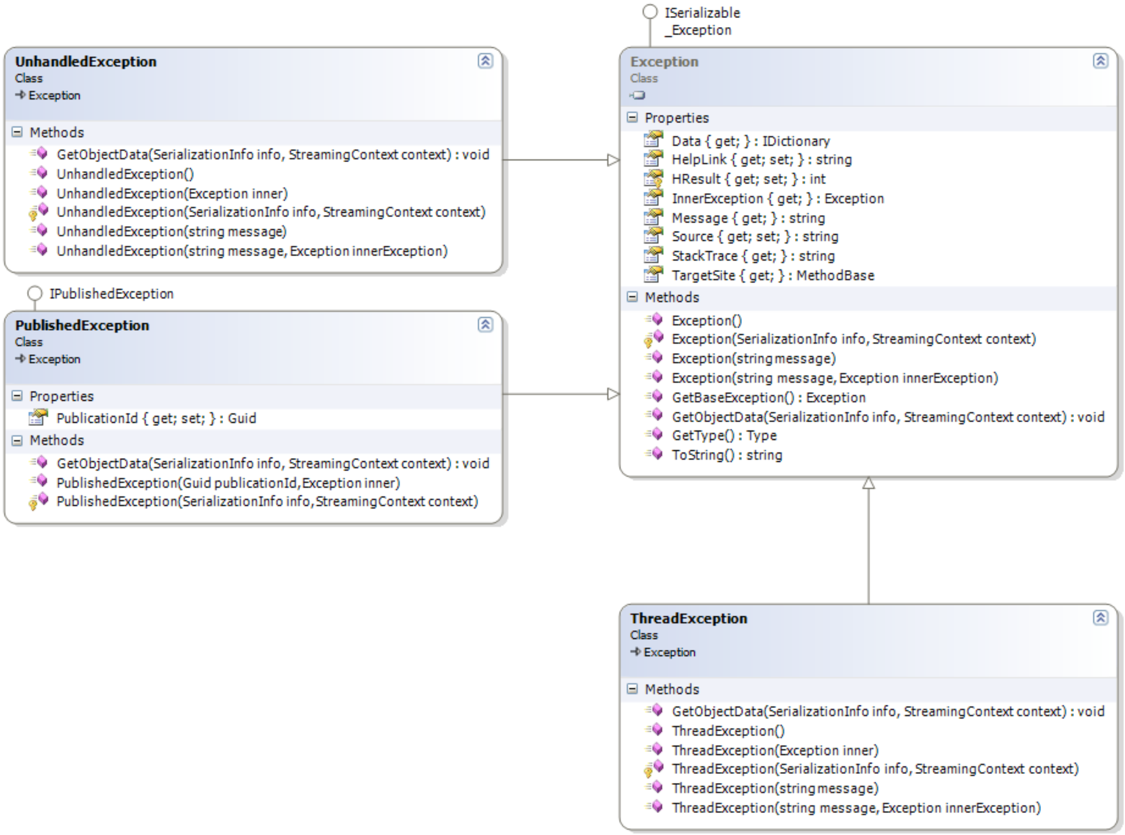

ISerializable
_Exception
UnhandledException
Class
Exception
Methods
GetObjectData(SerializationInfo info, StreamingContext context) : void
UnhandledException()
UnhandledException(Exception inner)
UnhandledException(SerializationInfo info, StreamingContext context)
UnhandledException(string message)
UnhandledException(string message, Exception innerException)
IPublishedException
PublishedException
Class
Exception
Properties
PublicationId { get; set; } : Guid
Methods
GetObjectData(SerializationInfo info, StreamingContext context) : void
PublishedException(Guid publicationId, Exception inner)
PublishedException(SerializationInfo info, StreamingContext context)
Exception
Class
Properties
Data { get; } : IDictionary
HelpLink { get; set; } : string
HResult { get; set; } : int
InnerException { get; } : Exception
Message { get; } : string
Source { get; set; } : string
StackTrace { get; } : string
TargetSite { get; } : MethodBase
Methods
Exception()
Exception(SerializationInfo info, StreamingContext context)
Exception(string message)
Exception(string message, Exception innerException)
GetBaseException() : Exception
GetObjectData(SerializationInfo info, StreamingContext context) : void
GetType() : Type
ToString() : string
ThreadException
Class
Exception
Methods
GetObjectData(SerializationInfo info, StreamingContext context) : void
ThreadException()
ThreadException(Exception inner)
ThreadException(SerializationInfo info, StreamingContext context)
ThreadException(string message)
ThreadException(string message, Exception innerException)

Figure 7. PublishedException, ThreadException and UnhandledException

Limitations

In this release, the internal **`ExceptionSerializer`** and **`IIdentitySerializer`** classes, which are responsible for creating the Soap encoded Xml which is stored within the **`Exception`** and **`LogEntry`** tables, are unable to serialize objects if either of the following conditions is true:

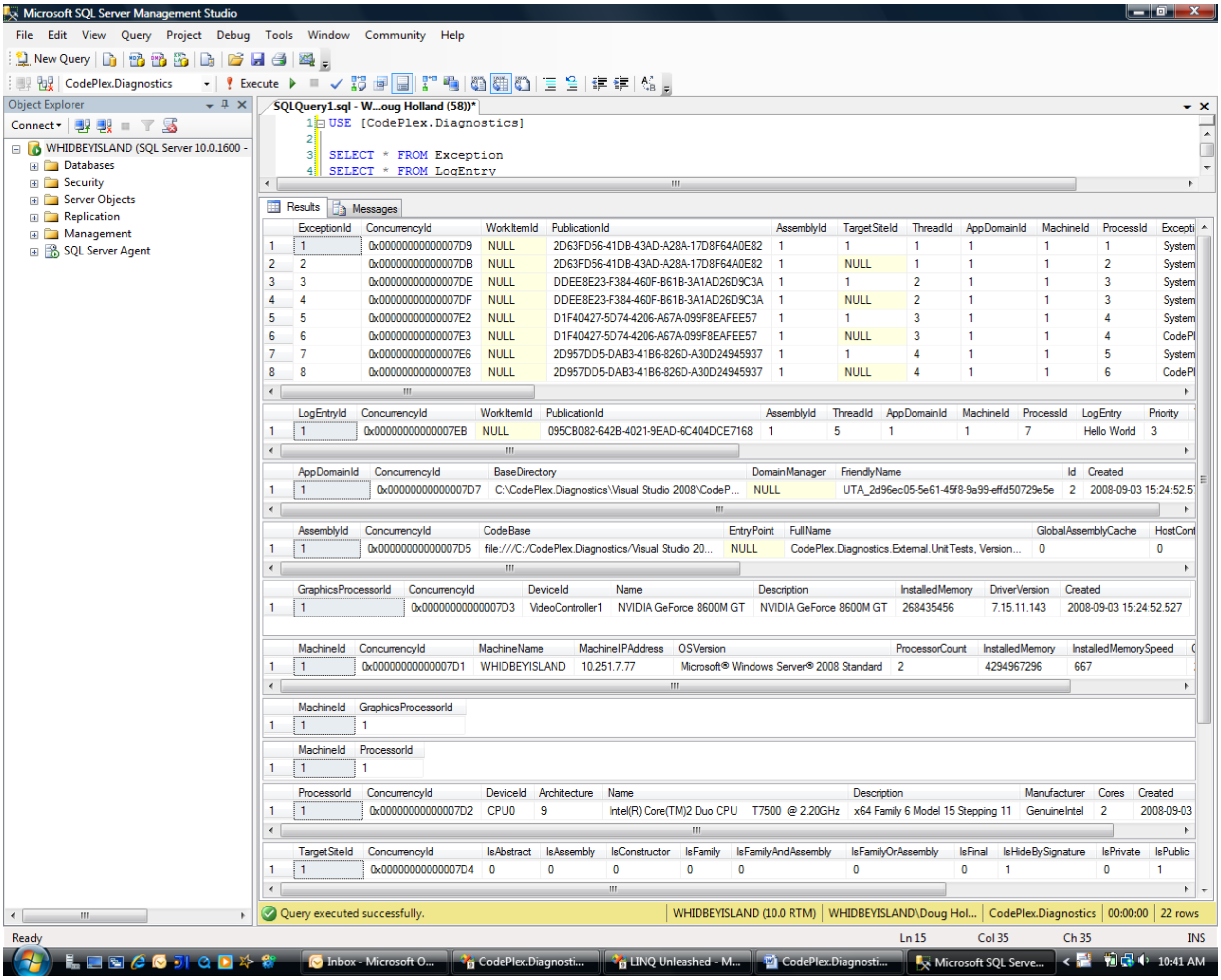
1. Supplied **`Exception`** | **`IIdentity`** is not serializable which is determined by checking the **`IsSerializable`** property of the .NET Framework **`Type`** class.
2. Supplied **`Exception`** | **`IIdentity`** is generic, which determined by checking the **`IsGeneric`** property of the .NET Framework **`Type`** class.

In both of these instances, the appropriate column within either the **`Exception`** or **`LogEntry`** tables will contain `NULL within the` **`SerializedIIdentityXml`** `column`. In previous releases, any generic **`Exception`** type would simply not be published and this was fixed in build 1.0.0.60.

The above limitations are in actual fact limitations of the serialization capabilities of the **`SoapFormatter`** class which is located within the **`System.Runtime.Serialization.Formatters.Soap`** namespace.

Other serialization alternatives are presently being explored although the **`XmlSerializer`** class is unable to handle serialization of objects implementing the **`IDictionary`** interface. When serializing types inheriting from the **`System.Exception`** base class this is a major problem because the **`Data`** property is itself an **`IDictionary`**.

Alternatively, the new serialization capabilities within the .NET Framework 3.0, specifically within the Windows Communication Foundation, require you to specify the known types using the **`KnownTypeAttribute`**. This also is not appropriate given we cannot at compile time know all of the known types for which serialization may be required.



```
USE [CodePlex.Diagnostics]

SELECT * FROM Exception
SELECT * FROM LogEntry
```

	ExceptionId	ConcurrencyId	WorkItemId	PublicationId	AssemblyId	TargetSiteId	ThreadId	AppDomainId	MachineId	ProcessId	Excepti
1	1	0x00000000000007D9	NULL	2D63FD56-41DB-43AD-A28A-17D8F64A0E82	1	1	1	1	1	1	System
2	2	0x00000000000007DB	NULL	2D63FD56-41DB-43AD-A28A-17D8F64A0E82	1	NULL	1	1	1	2	System
3	3	0x00000000000007DE	NULL	DDEE8E23-F384-460F-B61B-3A1AD26D9C3A	1	1	2	1	1	3	System
4	4	0x00000000000007DF	NULL	DDEE8E23-F384-460F-B61B-3A1AD26D9C3A	1	NULL	2	1	1	3	System
5	5	0x00000000000007E2	NULL	D1F40427-5D74-4206-A67A-099F8EAFEE57	1	1	3	1	1	4	System
6	6	0x00000000000007E3	NULL	D1F40427-5D74-4206-A67A-099F8EAFEE57	1	NULL	3	1	1	4	CodePl
7	7	0x00000000000007E6	NULL	2D957DD5-DAB3-41B6-826D-A30D24945937	1	1	4	1	1	5	System
8	8	0x00000000000007E8	NULL	2D957DD5-DAB3-41B6-826D-A30D24945937	1	NULL	4	1	1	6	CodePl

	LogEntryId	ConcurrencyId	WorkItemId	PublicationId	AssemblyId	ThreadId	AppDomainId	MachineId	ProcessId	LogEntry	Priority
1	1	0x00000000000007EB	NULL	095CB082-642B-4021-9EAD-6C404DCE7168	1	5	1	1	7	Hello World	3

	AppDomainId	ConcurrencyId	BaseDirectory	DomainManager	FriendlyName	Id	Created
1	1	0x00000000000007D7	C:\CodePlex.Diagnostics\Visual Studio 2008\CodeP...	NULL	UTA_2d96ec05-5e61-45f8-9a99-effd50729e5e	2	2008-09-03 15:24:52.5

	AssemblyId	ConcurrencyId	CodeBase	EntryPoint	FullName	GlobalAssemblyCache	HostCont
1	1	0x00000000000007D5	file:///C:/CodePlex.Diagnostics/Visual Studio 20...	NULL	CodePlex.Diagnostics.External.UnitTests, Version...	0	0

	GraphicsProcessorId	ConcurrencyId	DeviceId	Name	Description	InstalledMemory	DriverVersion	Created
1	1	0x00000000000007D3	VideoController1	NVIDIA GeForce 8600M GT	NVIDIA GeForce 8600M GT	268435456	7.15.11.143	2008-09-03 15:24:52.527

	MachineId	ConcurrencyId	MachineName	MachineIPAddress	OSVersion	ProcessorCount	InstalledMemory	InstalledMemorySpeed
1	1	0x00000000000007D1	WHIDBEYISLAND	10.251.7.77	Microsoft® Windows Server® 2008 Standard	2	4294967296	667

	MachineId	GraphicsProcessorId
1	1	1

	MachineId	ProcessorId
1	1	1

	ProcessorId	ConcurrencyId	DeviceId	Architecture	Name	Description	Manufacturer	Cores	Created
1	1	0x00000000000007D2	CPU0	9	Intel(R) Core(TM)2 Duo CPU T7500 @ 2.20GHz	x64 Family 6 Model 15 Stepping 11	GenuineIntel	2	2008-09-03

	TargetSiteId	ConcurrencyId	IsAbstract	IsAssembly	IsConstructor	IsFamily	IsFamilyAndAssembly	IsFamilyOrAssembly	IsFinal	IsHideBySignature	IsPrivate	IsPublic
1	1	0x00000000000007D4	0	0	0	0	0	0	0	1	0	1

```
Query executed successfully.    WHIDBEYISLAND (10.0 RTM) | WHIDBEYISLAND\Doug Hol... | CodePlex.Diagnostics | 00:00:00 | 22 rows
```

Figure 8. SQL Server 2008 Management Studio showing the result of running the T-SQL in listing 2.