Elaboração de Arquiteturas de Referência - FASE 2.CR



Arquitetura e Prova de Conceito do Framework para Aplicações .Net

Preparado para

**Secretaria da Fazenda do Estado de São Paulo**

16/05/2022

Versão 1.1 Final

Preparado por

**Hamilton de Paula; Carlos Dietrich; Diego Baroffio**

hapaula@microsoft.com; carlos.dietrich@microsoft.com; diego.baroffio@microsoft.com

Associate Enterprise Architects

Contribuidores

**Gleisson Bezerra**

Histórico de Atualizações e Revisões

Registro de Atualizações

|  |  |  |  |
| --- | --- | --- | --- |
| Data | Autor | Versão | Descrição |
| 31/10/2013 | Carlos Dietrich  Hamilton de Paula  Diego Baroffio | 1.0 | Versão entrega Fase 2. |
| 22/11/2013 | Carlos Dietrich  Hamilton de Paula  Diego Baroffio | 1.1 | Revisão geral. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Revisões

|  |  |  |  |
| --- | --- | --- | --- |
| Nome | Versão Aprovada | Posição | Data |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Conteúdo

[1 Objetivo 1](#_Toc372877193)

[2 Público Alvo 2](#_Toc372877194)

[3 Requisitos de Ambiente 3](#_Toc372877195)

[4 Siglas 4](#_Toc372877196)

[5 Framework para Desenvolvimento de Sistemas 5](#_Toc372877197)

[5.1 Benefícios 5](#_Toc372877198)

[5.1.1 Para o time de desenvolvimento: 5](#_Toc372877199)

[5.1.2 Para a área de negócios: 6](#_Toc372877200)

[6 Visão Geral de Arquitetura 7](#_Toc372877201)

[6.1 Core 9](#_Toc372877202)

[6.1.1 Business 9](#_Toc372877203)

[6.1.2 Common 9](#_Toc372877204)

[6.1.3 Data Access 9](#_Toc372877205)

[6.2 Util 10](#_Toc372877206)

[6.2.1 Audit 10](#_Toc372877207)

[6.2.2 Cache 11](#_Toc372877208)

[6.2.3 Config 11](#_Toc372877209)

[6.2.4 Exception Handling 12](#_Toc372877210)

[6.2.5 Log 12](#_Toc372877211)

[6.2.6 Security 13](#_Toc372877212)

[7 Desenho Físico 15](#_Toc372877213)

[7.1 Core 15](#_Toc372877214)

[7.1.1 Common 15](#_Toc372877215)

[7.1.2 Business 20](#_Toc372877216)

[7.1.3 Data Access 21](#_Toc372877217)

[7.2 Util 23](#_Toc372877218)

[7.2.1 Audit 24](#_Toc372877219)

[7.2.2 Cache 29](#_Toc372877220)

[7.2.3 Config 31](#_Toc372877221)

[7.2.4 Exception Handling 34](#_Toc372877222)

[7.2.5 Log 38](#_Toc372877223)

[7.2.6 Security 41](#_Toc372877224)

[7.3 Providers 43](#_Toc372877225)

[7.3.1 Audit 44](#_Toc372877226)

[7.3.2 Cache 50](#_Toc372877227)

[7.3.3 Config 51](#_Toc372877228)

[7.3.4 Data Access 52](#_Toc372877229)

[7.3.5 Log 63](#_Toc372877230)

[7.3.6 Security 68](#_Toc372877231)

[8 Guia de Uso 70](#_Toc372877232)

[8.1 Instalação 70](#_Toc372877233)

[8.1.1 Providers 71](#_Toc372877234)

[8.2 Exemplos – How-to 71](#_Toc372877235)

[8.2.1 Audit 71](#_Toc372877236)

[8.2.2 Cache 79](#_Toc372877237)

[8.2.3 Config 82](#_Toc372877238)

[8.2.4 Exception Handling 84](#_Toc372877239)

[8.2.5 Log 88](#_Toc372877240)

[8.2.6 Security 95](#_Toc372877241)

[8.2.7 Data Access 97](#_Toc372877242)

[8.2.8 Business 100](#_Toc372877243)

[8.3 Teste Integrado 103](#_Toc372877244)

[9 Governança 110](#_Toc372877245)

[9.1 Versionamento 110](#_Toc372877246)

[10 Referências 111](#_Toc372877247)

Índice de Figuras

[Figura 1 Distribuição lógica dos componentes de uma aplicação 7](#_Toc372877149)

[Figura 2 Dependência entre camadas e framework 8](#_Toc372877150)

[Figura 3 Principais componentes do bloco Sefaz.Fw.Common 15](#_Toc372877151)

[Figura 4 Sefaz.Fw.Common – Fábrica de providers 16](#_Toc372877152)

[Figura 5 Sefaz.Fw.Common – ConfigManager 17](#_Toc372877153)

[Figura 6 Classes para manipulação de entidades 18](#_Toc372877154)

[Figura 7 Classes base para tratamento de exceção 18](#_Toc372877155)

[Figura 8 Principais componentes da camada Sefaz.Fw.Business 20](#_Toc372877156)

[Figura 9 Principais componentes da camada Sefaz.Fw.Data 21](#_Toc372877157)

[Figura 10 Arquitetura física do bloco Audit 25](#_Toc372877158)

[Figura 11 Principais componentes do bloco Cache 30](#_Toc372877159)

[Figura 12 Estrutura física do bloco Config 32](#_Toc372877160)

[Figura 13 Estrutura física do bloco Exception Handling 35](#_Toc372877161)

[Figura 14 Arquitetura física do bloco Log 38](#_Toc372877162)

[Figura 15 Arquitetura física do bloco Security 41](#_Toc372877163)

[Figura 16 Arquitetura física do provider LocalSQLCompactProvider (Audit) 44](#_Toc372877164)

[Figura 17 Arquitetura física do provider SQLServerProvider (Audit) 47](#_Toc372877165)

[Figura 18 Arquitetura física do provider ServiceProvider (Audit) 49](#_Toc372877166)

[Figura 19 Arquitetura física do provider MemoryCacheProvider (Cache) 50](#_Toc372877167)

[Figura 20 Arquitetura física do provider SQLServerProvider (Config) 51](#_Toc372877168)

[Figura 21 Arquitetura física do provider ADOProvider (Data Access) 53](#_Toc372877169)

[Figura 22 Classes para mapeamento usado pelo provider ADOProvider (Data Access) 56](#_Toc372877170)

[Figura 23 Arquitetura física do provider SQLServerProvider (Data Access) 58](#_Toc372877171)

[Figura 24 Classes para mapeamento usado pelo provider SQLServerProvider (Data Access) 60](#_Toc372877172)

[Figura 25 Arquitetura física do provider EFProvider (Data Access) 62](#_Toc372877173)

[Figura 26 Arquitetura física do provider SQLServerProvider (Log) 64](#_Toc372877174)

[Figura 27 Arquitetura física do provider FileLogProvider 66](#_Toc372877175)

[Figura 28 Arquitetura física do provider ConsoleOutputProvider 68](#_Toc372877176)

[Figura 29 Arquitetura física do CertificateProvider 69](#_Toc372877177)

[Figura 30 Arquitetura física do SignatureProvider 70](#_Toc372877178)

[Figura 31 Referências para o uso do bloco Audit 72](#_Toc372877179)

[Figura 32 WCF hospedado no IIS 78](#_Toc372877180)

[Figura 33 Eventos de auditoria armazenados em banco 79](#_Toc372877181)

[Figura 34 Referências para o uso do bloco Cache 79](#_Toc372877182)

[Figura 35 Referências para o uso do bloco Config 82](#_Toc372877183)

[Figura 36 Objeto de configuração armazenado no banco de dados 84](#_Toc372877184)

[Figura 37 Referências para o uso do bloco ExceptionHandling 84](#_Toc372877185)

[Figura 38 Referências para o uso do bloco Log 88](#_Toc372877186)

[Figura 39 WCF hospedado no IIS 94](#_Toc372877187)

[Figura 40 Log armazenado em arquivo 95](#_Toc372877188)

[Figura 41 Referências para uso do bloco Security 95](#_Toc372877189)

[Figura 42 Referências para o uso do bloco Data Access 98](#_Toc372877190)

[Figura 43 Referências para o uso do bloco Business 100](#_Toc372877191)

[Figura 44 Arquitetura física do projeto de teste integrado 104](#_Toc372877192)

Índice de Tabelas

[Tabela 1 Requisitos de Ambiente 3](#_Toc372877145)

[Tabela 2 Siglas utilizadas 4](#_Toc372877146)

[Tabela 3 Componentes do framework 8](#_Toc372877147)

[Tabela 4 Nomes default das seções de configuração 24](#_Toc372877148)

Índice de Listagem

[Listagem 1 Detalhe do arquivo de configuração do bloco Data Access 23](#_Toc372877096)

[Listagem 2 Detalhe do arquivo de configuração do bloco Audit 29](#_Toc372877097)

[Listagem 3 Detalhe do arquivo de configuração do bloco de Cache 31](#_Toc372877098)

[Listagem 4 Detalhe do arquivo de configuração do bloco de Config. 34](#_Toc372877099)

[Listagem 5 Uso do método de tratamento de exceção - BPBase 34](#_Toc372877100)

[Listagem 6 Uso do método de tratamento de exceção – Uso direto do bloco 34](#_Toc372877101)

[Listagem 7 Detalhe do arquivo de configuração do bloco Log 40](#_Toc372877102)

[Listagem 8 Arquivo de configuração do bloco de Security 43](#_Toc372877103)

[Listagem 9 Arquivo de configuração do provider LocalSQLCompactProvider (Audit) 46](#_Toc372877104)

[Listagem 10 Arquivo de configuração do provider SQLServerProvider (Audit) 48](#_Toc372877105)

[Listagem 11 Arquivo de configuração do provider ServiceProvider (Audit) 50](#_Toc372877106)

[Listagem 12 Arquivo de configuração do provider SQLServerProvider (Config) 52](#_Toc372877107)

[Listagem 13 Arquivo de configuração do provider ADOProvider (Data Access) 55](#_Toc372877108)

[Listagem 14 Arquivo de configuração do provider SQLServerProvider (Data Access) 60](#_Toc372877109)

[Listagem 15 Arquivo de configuração do provider EFProvider (Data Access) 63](#_Toc372877110)

[Listagem 16 Arquivo de configuração do provider SQLServerProvider (Log) 65](#_Toc372877111)

[Listagem 17 Arquitetura física do provider ServiceProvider (Log) 65](#_Toc372877112)

[Listagem 18 Arquivo de configuração do provider FileLogProvider 67](#_Toc372877113)

[Listagem 19 Classe AuditExample 73](#_Toc372877114)

[Listagem 20 Configuração do bloco Audit 74](#_Toc372877115)

[Listagem 21 Configuração do ServiceProvider 74](#_Toc372877116)

[Listagem 22 Configuração do WCF para o ServiceProvider 75](#_Toc372877117)

[Listagem 23 Configuração do serviço para o bloco Audit 75](#_Toc372877118)

[Listagem 24 Configuração do WCF para hospedagem no executável 76](#_Toc372877119)

[Listagem 25 Iniciando o serviço WCF hospedado no executável 77](#_Toc372877120)

[Listagem 26 Configuração do WCF para hospedagem no IIS 77](#_Toc372877121)

[Listagem 27 Exemplo de uso do bloco Audit 79](#_Toc372877122)

[Listagem 28 Configuração do bloco Cache 80](#_Toc372877123)

[Listagem 29 Exemplo de uso do bloco Cache 81](#_Toc372877124)

[Listagem 30 Configuração do bloco Config 83](#_Toc372877125)

[Listagem 31 Exemplo de uso do bloco Config 84](#_Toc372877126)

[Listagem 32 Configuração do bloco ExceptionHandling 85](#_Toc372877127)

[Listagem 33 Exemplo de uso do bloco ExceptionHandling 87](#_Toc372877128)

[Listagem 34 Classe LogExample 89](#_Toc372877129)

[Listagem 35 Configuração do bloco Log 89](#_Toc372877130)

[Listagem 36 Configuração do ServiceProvider 90](#_Toc372877131)

[Listagem 37 Configuração do WCF para o ServiceProvider 90](#_Toc372877132)

[Listagem 38 Configuração do serviço para o bloco Log 91](#_Toc372877133)

[Listagem 39 Configuração do WCF para hospedagem no executável 92](#_Toc372877134)

[Listagem 40 Iniciando o serviço WCF hospedado no executável 92](#_Toc372877135)

[Listagem 41 Configuração do WCF para hospedagem no IIS 93](#_Toc372877136)

[Listagem 42 Exemplo de uso do bloco Log 94](#_Toc372877137)

[Listagem 43 Configuração do bloco Security 96](#_Toc372877138)

[Listagem 44 Exemplo de uso do bloco Security 97](#_Toc372877139)

[Listagem 45 Classe DAOrder 98](#_Toc372877140)

[Listagem 46 Configuração do bloco Data Access 99](#_Toc372877141)

[Listagem 47 Exemplo de uso do bloco Data Access 100](#_Toc372877142)

[Listagem 48 Classe EMOrder 101](#_Toc372877143)

[Listagem 49 Classe BPOrder 103](#_Toc372877144)

1. Objetivo

Este documento é parte integrante do projeto de **Elaboração de Arquiteturas de Software – FASE 2**, desenvolvido segundo o Escopo de Trabalho e Formato de Entregas especificados no documento de **Visão Escopo** (SEFAZ - ITAP Arquiteturas de Software - FASE 2.CR - Visão e Escopo v2.0.docx).

O projeto visa estabelecer uma referência para o desenvolvimento padronizado de aplicações, permitindo a adoção de práticas para aumentar a segurança, a disponibilidade, a escalabilidade e o desempenho dos sistemas da instituição.

Nas fases anteriores do projeto, diversos documentos foram gerados com a intenção de guiar os arquitetos e desenvolvedores da **SEFAZ** no desenho e codificação de aplicações, tais como:

* SEFAZ - SP - Guia de Desenvolvimento para Aplicações Web
* SEFAZ - SP - Guia de Desenvolvimento para Serviços Web
* SEFAZ - SP - Guia de Desenvolvimento para Processamento Batch
* SEFAZ - SP - Guia de Desenvolvimento para SQL Server
* SEFAZ - SP - Guia de Desenvolvimento para Plataforma .Net
* SEFAZ - SP - Guia de Desenvolvimento ORM

A necessidade de um framework foi identificada nas interações com o time de desenvolvimento da **SEFAZ**, e seu objetivo é auxiliar os desenvolvedores em tarefas comumente executadas, proporcionando benefícios em produtividade, reuso, qualidade e segurança.

Este documento apresenta a arquitetura recomendada para um framework de desenvolvimento que sirva de alicerce para as aplicações .NET da **SEFAZ**, oferecendo uma forma eficiente e padrão para a execução de tarefas comuns.

O código fonte fornecido deve ser tratado como uma prova de conceito e não pode ser utilizado para criar aplicativos em produção. Ele pode no entanto, ser utilizado como ponto de partida para o desenvolvimento de um framework definitivo para a **SEFAZ**.

1. Público Alvo

Este guia deve ser utilizado como referência por arquitetos e desenvolvedores especializados em projeto e codificação de aplicações corporativas distribuídas usando C# e plataforma Microsoft .NET 4.5.

1. Requisitos de Ambiente

Este tópico apresenta os requisitos de ambiente necessários ao acompanhamento deste documento.

|  |  |
| --- | --- |
| Recurso | Descrição |
| Internet Information Server | Necessário para hospedagem de serviços WCF dos blocos **Audit** e **Log**. |
| Visual Studio 2012 Ultimate | Usado para o desenvolvimento dos exemplos e controles citados no documento. |
| .NET Framework 4.5 | Usado para o desenvolvimento dos componentes utilizados neste documento. |
| SQL Server | Necessáriopara os *providers* de persistência dos blocos **Audit**, **Log** e **Config**. |
| SQL Server Compact Edition | Necessário para o *provider* de persistência local do bloco **Audit**. |

Tabela 1 Requisitos de Ambiente

1. Siglas

Este tópico apresenta as siglas mais usadas neste documento.

|  |  |
| --- | --- |
| Sigla | Descrição |
| IIS | Internet Information Server |
| CRUD | Operações de CRUD -> Create, Read, Update e Delete |
| GAC | Global Assembly Cache |
| W3C | World Wide Web Consortium |
| WCF | Windows Communication Foundation |

Tabela 2 Siglas utilizadas

1. Framework para Desenvolvimento de Sistemas

No âmbito da engenharia de software, denomina-se como *framework*, um conjunto de bibliotecas, rotinas ou procedimentos que realizam tarefas de uso comum, padrão e de maneira repetitiva.

Exemplos de tarefas ou rotinas que podem fazer parte de um *framework*:

* Acesso a dados;
* Auditoria;
* Cache;
* Criptografia;
* Configuração;
* Log e trace;
* Tratamento de exceções.
  1. Benefícios
     1. Para o time de desenvolvimento:

O uso de um *framework* oferece diversos benefícios para o time de desenvolvimento, tais como:

* Padronização: Todos os integrantes do time que precisam utilizar determinada funcionalidade contida no *framework* o fazem sempre da mesma maneira.
* Treinamento: É possível oferecer treinamento de maneira consistente e padronizada, diminuindo a curva de aprendizado dos recém-contratados.
* Foco no negócio: O time tem mais tempo para focar no que interessa: nas funcionalidades de negócio, e não na tecnologia.
* Produtividade: O time faz uso de rotinas prontas, sem perder tempo com desenvolvimento que não seja específico do negócio.
* Reuso: Possibilita o reuso evitando desperdício de tempo.
* Menor sujeição a erro: As rotinas disponibilizadas já foram devidamente testadas e aprovadas.
* Performance: O código disponibilizado foi previamente otimizado.
* Manutenção: A manutenção de código é facilitada porque ele segue um padrão de implementação.
* Evolução: Ao evoluir a implementação do *framework*, todas as aplicações que o utilizam são beneficiadas imediatamente, sem a necessidade de manutenção.
  + 1. Para a área de negócios:

Além de beneficiar o time de desenvolvimento, a área de negócios obtém os seguintes benefícios:

* Tempo de desenvolvimento: respostas mais rápidas no desenvolvimento e entrega de soluções.
* Redução de custo: como o time de desenvolvimento gasta menos tempo com tecnologia, tem mais tempo para focar no negócio, entregando mais funcionalidades em menor tempo.
* Melhor qualidade: o produto entregue possui qualidade superior.
* Melhor resiliência e suporte: o produto entregue é mais estável e seu suporte é mais facilitado.

A inexistência de um *framework*, ou a não obrigatoriedade de seu uso, pode trazer riscos como:

* Código ineficiente: a funcionalidade pode ter sido desenvolvida de maneira imprópria, sem a performance adequada para atender as necessidades de negócio ou aos requisitos do servidor onde ela será instalada;
* Código inseguro: a funcionalidade pode ter sido desenvolvida sem ter levado em consideração aspectos importantes de segurança;
* Código não padrão: gerando dificuldade para manter, evoluir e suportar;
* Treinamento: sem um procedimento padrão, não é possível oferecer treinamento, aumentando a curva de aprendizado de recém-contratados.

1. Visão Geral de Arquitetura

A definição da arquitetura do *framework* acompanha a arquitetura para as aplicações distribuídas proposta no guia “**SEFAZ - SP - Guia de Desenvolvimento para Plataforma .Net**”, uma vez que seu principal objetivo é facilitar a adoção das recomendações ali expostas.

A distribuição lógica dos blocos do *framework* também leva em consideração a separação de responsabilidades de cada camada/bloco, assim como encapsulamento e desacoplamento de tecnologias usadas.

A seguir está a recomendação de distribuição lógica dos componentes de uma aplicação:

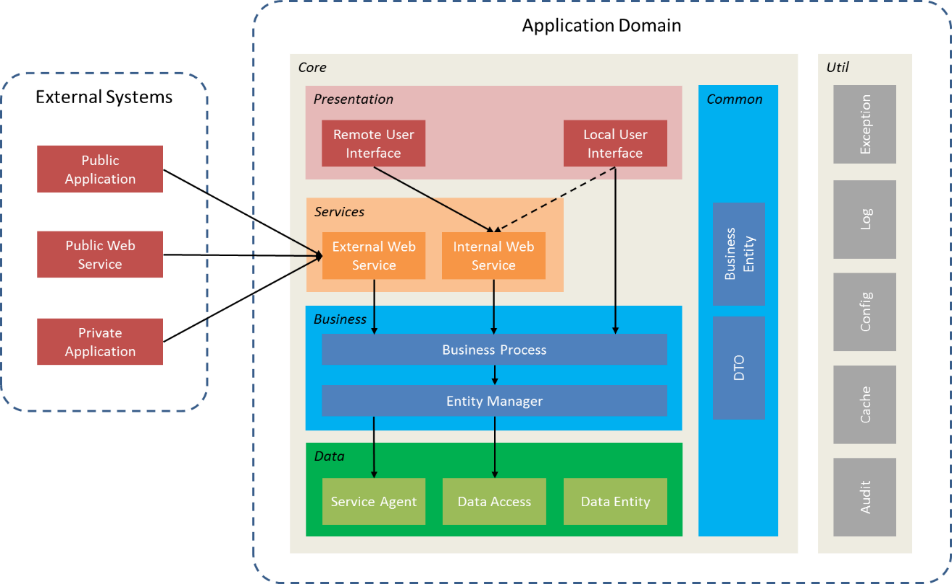


Figura 1 Distribuição lógica dos componentes de uma aplicação

Os componentes de uma aplicação distribuída foram agrupados da seguinte maneira:

* ***Core*** (Núcleo) – contém os blocos sempre presentes em uma aplicação. Possui as seguintes camadas: ***Presentation****,* ***Services***, ***Business*** e ***Data***. Para a integração de todas as camadas existem os blocos do ***Common***.
* ***Util*** (Utilitários) – contém os blocos complementares e de uso opcional.

No caso do *framework*, existe ainda mais um grupo:

* ***Providers*** – contém blocos com diferentes implementações de *providers*.

O *framework* é composto pelos seguintes blocos:

|  |  |  |
| --- | --- | --- |
| **Core** | **Util** | **Providers** |
| * + Common   + Data Access   + Business | * + Cache   + Config   + Auditoria   + Log   + Exception Handling   + Security | * + Data access   + Cache   + Config   + Auditoria   + Log   + Certificado |

Tabela 3 Componentes do framework

Fisicamente, cada camada possui seu *assembly* correspondente. Os utilitários são uma exceção, por serem opcionais, cada bloco possui seu *assembly* correspondente.

A figura a seguir mostra a integração da aplicação com o *framework* e suas dependências:

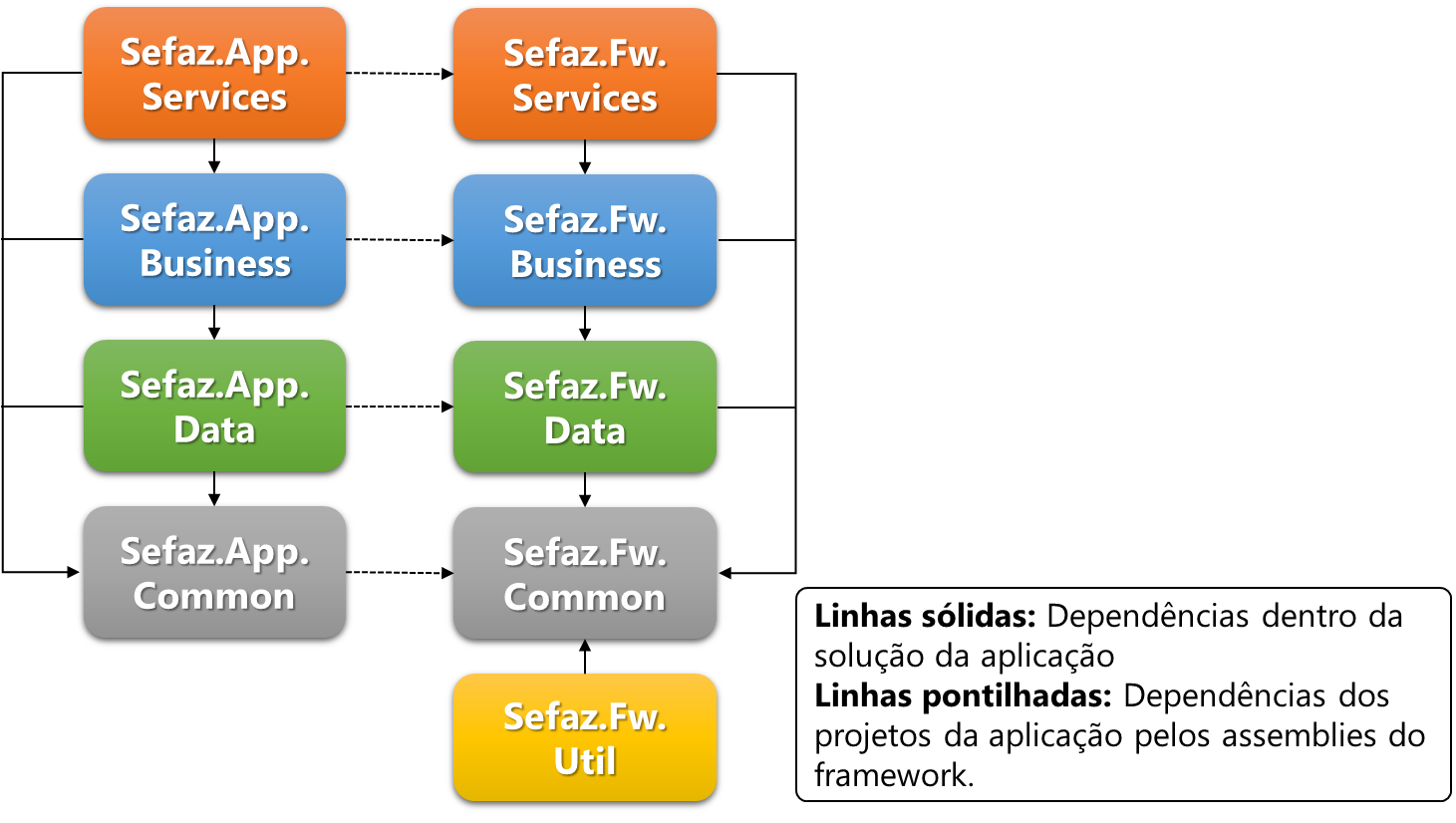


Figura 2 Dependência entre camadas e framework

* 1. Core
     1. Business

Bloco de componentes que fornece classes base para a camada de negócio da aplicação.

Características:

* Oferece métodos básicos para CRUD para as classes **Entity Manager**;
* Oferece método unificado para tratamento de exceção nas classes **Business Process**. Utiliza o bloco **Exception Handling**.
  + 1. Common

Blocos de componentes compartilhados entre as camadas. Utilizados normalmente para definir estruturas de dados, interfaces e utilitários usados por mais de uma camada.

Entre as funcionalidades previstas podem ser destacadas:

* Provê a definição das principais exceções usadas pelo *framework* e pela aplicação;
* Classes para manipulação de entidades;
* Classe para armazenamento das seções dos arquivos de configuração;
* Fábrica abstrata para os *providers* do camada **Util**.
  + 1. Data Access

Classes base responsáveis por todo acesso aos dados da aplicação. Estas classes devem acessar apenas os dados que são mantidos pela aplicação. Dados mantidos por outras aplicações devem ser acessados por **Service Agents**.

Provêm automaticamente operações de inclusão, alteração, exclusão e leitura de sua respectiva **Business Entity** associada. Para executar estas operações, uma classe **Data Access** utiliza um **Data Access Provider**.

Características:

* Encapsulam a tecnologia de persistência.
* Cada classe **Data Access** atende uma única **Business Entity**. Isto permite que as operações de CRUD sejam automáticas e sigam uma interface única padrão.
* Oferece métodos básicos para CRUD usando **ORM**;
* Oferece métodos básicos para CRUD usando SQL customizado ou *Stored Procedures*;
* Oferece mecanismo genérico para filtro, ordenação e paginação dos resultados;
* Provê mecanismo extensível para acesso à tecnologia de persistência, através de interfaces e *providers* que permitam o uso de diferentes implementações, sem a necessidade de alteração das aplicações que utilizam esse bloco.
  1. Util
     1. Audit

Este bloco tem o objetivo de criar uma mecanismo reutilizável que gera rastros de auditoria. Possui dois estágios de persistência local (temporário) e remoto (definitivo).

Utilizando *providers,* permite a persistência em diversos tipos de repositório.

Características:

* A chamada ao método de auditoria é feita explicitamente pelo programador;
* Suporta a gravação dos seguintes dados:
  + Aplicação (texto)
  + Usuário (texto)
  + IP (texto)
  + Timestamp (criação do registro de auditoria)
  + Timestamp (gravado pelo banco)
  + Categoria (texto)
  + Operação Executada [Caso de Uso] (texto)
  + Identificador (guid)
  + Identificador de correlação (guid)
  + Dados\_Texto (texto)
  + Dados\_XML
  + Dados\_XML\_Type (texto - tipo utilizado na serialização)
  + Hash (inteiro - utilizado como medida de segurança)
* A manipulação e armazenamento dos eventos de auditoria são feitos de modo a não expor informações sigilosas, fazendo criptografia dos dados armazenados localmente.
* A informação gravada na base de auditoria é protegida contra alteração;
* Não é ponto único de falha ou gargalo, possui mecanismo que garante a criação do registro de auditoria mesmo que atrasado;
* Suporta pedidos de log oriundos de uma camada de apresentação remota;
* Provê mecanismo extensível para a persistência das entradas de auditoria, através de interfaces e *providers* que permitam o uso de diferentes implementações, sem a necessidade de alteração das aplicações que utilizam este bloco.
  + 1. Cache

Este bloco tem como objetivo criar uma camada de abstração com funcionalidades básicas de cache.

Características:

* Oferece métodos para leitura e atualização de dados em cache;
* Os dados no cache podem expirar após um determinado tempo a partir da data de inclusão no cache, ou a partir da data do último acesso;
* A atualização dos dados do cache quando estes forem modificados nos seus repositórios originais é de responsabilidade da aplicação que utiliza este bloco;
* Provê mecanismo extensível à persistência dos valores em cache, através de interfaces e *providers* que permitem o uso de diferentes implementações, sem a necessidade de alteração das aplicações que utilizam este bloco.
  + 1. Config

Este bloco tem como objetivo sugerir um padrão para armazenamento de parâmetros da aplicação, similar aos arquivos .config, porém, gravando e recuperando um objeto com dados de configuração de uma tabela de banco de dados. Após a leitura do objeto, é mantida uma única cópia em memória (cache) e compartilhada pela aplicação (*singleton*).

A *connection string* e o tempo de permanência em cache são informados na configuração da aplicação.

Características:

* Oferece métodos para a leitura e gravação de valores de configuração na base de dados.
* Cada bloco de configuração é representado por uma classe serializável na aplicação.
* Ao ler a configuração da base de dados para a memória, esta informação fica armazenada em cache por um tempo determinado pela aplicação, utilizando o bloco de **Cache**;
* Provê mecanismo extensível de persistência dos valores através de interfaces e *providers* que permitam o uso de diferentes implementações, sem a necessidade de alteração das aplicações que utilizam este bloco.
  + 1. Exception Handling

Bloco para tratamento de exceção. Fornece as principais funcionalidades para tratar exceções, tais como: tratar as exceções por tipo; logar a exceção; traduzir a exceção e sanitizar a exceção.

Características:

* Faz o log da exceção e suas *inner exceptions* utilizando o bloco de **Log**;
* Suporta o *rethrow* ou a supressão exceção capturada;
* O *rethrow* pode ser feito com uma exceção sanitizada. Trata-se de uma exceção pré-definida no bloco **Common**, que possui uma mensagem simples de texto informada pelo programador;
* A chamada ao método de tratamento é feita explicitamente pelo programador.
* Provê mecanismo extensível para tratamento de exceções, utilizando interfaces e *providers* que permitem o uso de diferentes implementações, sem a necessidade de alteração das aplicações que utilizam este bloco.
  + 1. Log

Bloco para manipulação de logs. Permite persistir mensagens de log, tais como: **exceções e**, registros de **trilhas**..

Permite a persistência em diversos tipos de repositórios, por ex: **event log**; **banco de dados**; **arquivo texto**; envio de **e-mail**.

Características:

* A chamada ao método de log é feita explicitamente pelo programador;
* Suporta a gravação dos seguintes dados:
  + Timestamp
  + Criticidade (inteiro)
  + Verbosidade
  + Dados\_Resumo (texto – varchar(500))
  + Dados\_Longo (texto – ntext)
* Suporta níveis de verbosidade (verboso, informação, alerta ou erro);
* Não é ponto único de falha ou gargalo;
* Suporta pedidos de log oriundos de uma camada de apresentação remota;
* Possui métodos independentes para log e trace com assinaturas especializadas para cada finalidade;
* Provê mecanismo extensível para a persistência das entradas de log, através de interfaces e *providers* que permitam o uso de diferentes implementações, sem a necessidade de alteração das aplicações que utilizam este bloco.
  + 1. Security

Bloco que oferece funcionalidades para manipulação de certificados digitais.

Características:

* Oferece método para validar um certificado digital. Verifica minimamente: data validade, cadeia e revogação;
* Oferece métodos para extrair conteúdo do certificado ou propriedades específicas (informar o identificador da propriedade e retornar o valor);
* Oferece método para encontrar um certificado na **CertStore**;
* Oferece método para criar um certificado através de texto serializado;
* Oferece método assinar digitalmente e obter conteúdo assinado;
* Oferece método para validar assinatura digital, de arquivos binários e arquivos XML;
* Provê mecanismo extensível para as funcionalidades de manipulação de certificados digitais. Essas funcionalidades utilizam interfaces e *providers* que permitem o uso de diferentes implementações, sem a necessidade de alteração das aplicações que utilizam este bloco.

1. Desenho Físico

Detalhamento da implementação, descrição das interfaces, dependência dos componentes e distribuição física dos *assemblies*.

Para ter um melhor entendimento do funcionamento dos vários componentes do framework **Sefaz.Fw**, recomenda-se ler este documento acompanhado do seu código fonte.

* 1. Core
     1. Common

Bloco que possui recursos de uso comum do **Sefaz.Fw**. Pode ser utilizado por todas as camadas, portanto não possui dependências de outros blocos, com o objetivo de evitar referências circulares.

A seguir estão os principais componentes do bloco **Common**:

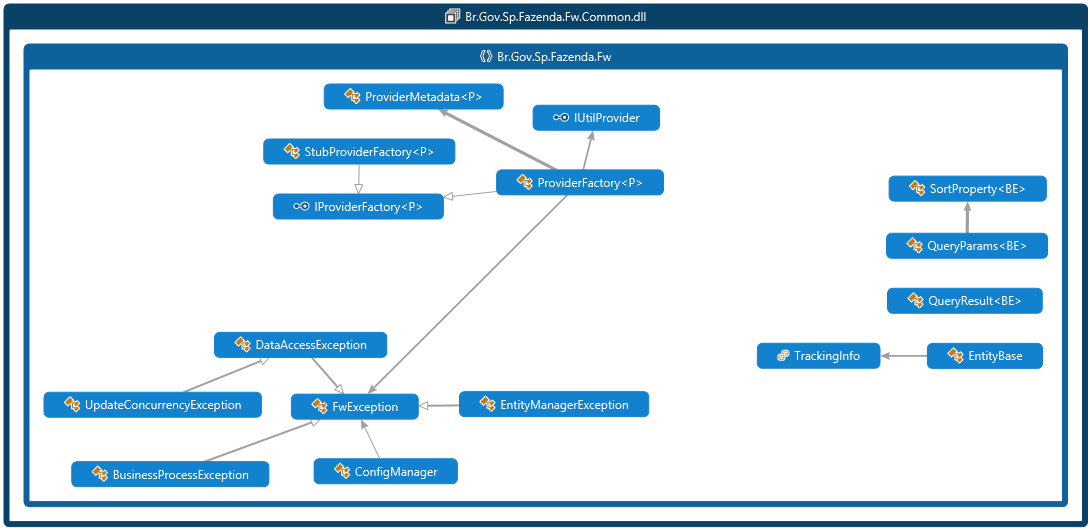


Figura 3 Principais componentes do bloco Sefaz.Fw.Common

#### Fábrica de providers

A fábrica de *providers* é um conjunto de classes com a responsabilidade de criar instâncias dos *providers* que serão utilizados pelos blocos **Data Access** e **Util**. A fábrica fornece um mecanismo onde o código responsável pela criação dos objetos está separado em uma classe específica que pode ser substituída em tempo de execução, dependendo das necessidades de negócio. Ela se mostra necessário em algumas situações, como por exemplo, na substituição de um *provider* “real” por uma implementação “falsa”.

Apesar de estarem disponíveis para uso externo as funcionalidades da fábrica foram criadas para atender às necessidades do próprio **Sefaz.Fw**.

Para um mecanismo mais robusto de injeção de dependências recomenda-se o uso do **Unity** disponível na **Enterprise Library**: <http://msdn.microsoft.com/en-us/library/dn169621.aspx>

Componentes da Fábrica de *Providers*:

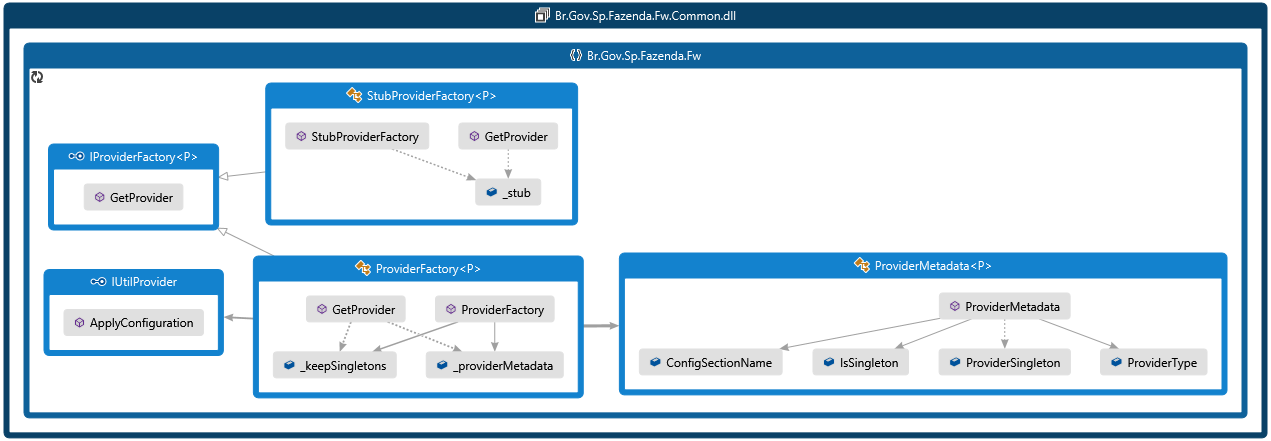


Figura 4 Sefaz.Fw.Common – Fábrica de providers

* ProviderFactory<P>: Classe responsável por criar *providers* utilizando uma *string* contendo o *strong name* da classe do *provider* e o nome de uma seção de configuração existente no app.config ou web.config
* ProviderMetadata<P>: Classe de suporte para a ProviderFactory. Armazena as propriedades que permitem a construção do *provider* e se for o caso o seu *singleton*.
* StubProviderFactory<P>: Classe responsável por retornar *stubs* de *providers*. Utilizada para a confecção de testes unitários onde se deseja substituir a implementação de um *provider* por um *stub*. Sempre que solicitado, retorna o *stub* que é informado na construção da fábrica;
* IProviderFactory<P>: Interface genérica que define o método responsável por retornar instância pronta para uso do *provider* do tipo “P”. Utilizada pelas fábricas ProviderFactory e StubProviderFactory;
* **IUtilProvider**: Utilizada para compor a interface dos *providers*. Como são criados em tempo de execução, as classes dos *providers* precisam de um construtor sem parâmetros. Essa classe define uma maneira padrão de finalizar a construção de um *provider*.

Cache de parâmetros de configuração

Mecanismo responsável por fazer o cache de seções de arquivos de configuração para evitar múltiplas leituras nos **arquivos .config**.

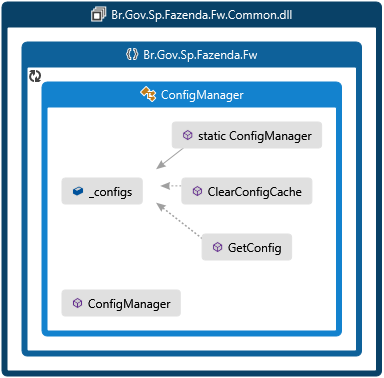


Figura 5 Sefaz.Fw.Common – ConfigManager

* **ConfigManager:** classe responsável pelo cache das seções dos arquivos de configuração.

#### Manipulação de entidades

Classes que fornecem funcionalidades prontas para a manipulação de entidades da aplicação.

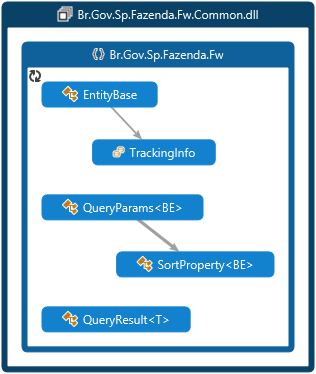


Figura 6 Classes para manipulação de entidades

* **EntityBase:** classe que pode ser utilizada como base para as entidades do sistemas. Se utilizada fornece eventos fazem o controle de alteração das propriedades da entidade;
* **TrackingInfo:** *enumeration* que contém os estados de alteração das entidades;
* **QueryParams:** classe que define parâmetros para a recuperação de listas de entidades pela camada de dados, como: filtros, paginação, ordenação e entidades associadas;
* **SortProperty:** classe utilizada pela **QueryParams** para definir o sentido da ordenação;
* **QueryResult:** classe que contém o resultado de uma busca de entidades. Contém a lista de entidades e metadados da busca.

#### Exceptions

Classes que com exceções especializadas para uso por vários componentes do *framework*.

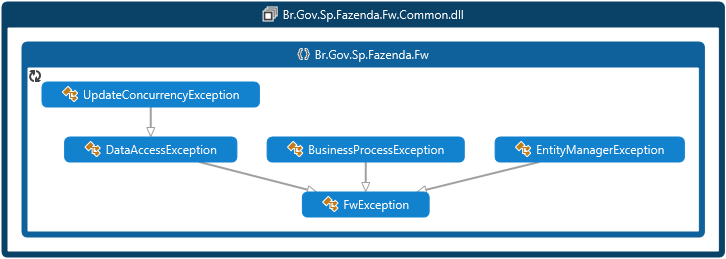


Figura 7 Classes base para tratamento de exceção

* **FwException:** classe base para todas as exceções do **Sefaz.Fw**;
* **DataAccessException:** classe base para as exceções dos DA’s definidos na camada **Data**;
* **UpdateConcurrencyExecption:** exceção especializada para indicar um falha de concorrência em uma operação de *updade* ou *delete*. Herda de **DataAccessException**;
* **BusinessProcessException:** classe base para as exceções dos BP’s definidos na camada **Business**;
* **EntityManagerException:** classe base para as exceções dos EM’s definidos na camada **Business**;

#### Service

Esse namespace oferece um conjunto de classes para tratar a serialização de objetos com referência circular em WCF.

Imagine um objeto Pedido que tem uma coleção de objetos ItemPedido e cada ItemPedido tem uma propriedade que aponta para Pedido. Isso é um exemplo de Referência Circular que a serialização comum do WCF não trata, disparando uma exceção como: *“The formatter threw an exception while trying to deserialize the message...”*

Para solucionar esse problema, temos que alterar a forma com que o serializador trabalha, ligando a propriedade preserveObjectReference do objeto DataContractSerializer. Isso foi devidamente encapsulado em um behavior do WCF no namespace Service.

Você deve decorar a operação problemática do ***contrato*** de seu serviço WCF com o atributo ReferencePreservingDataContractFormat conforme ilustrado abaixo:

[ServiceContract(Name = "IFeriadoService", Namespace = "http://www.fazenda.sp.gov.br/Feriado/2013/01")]

public interface IFeriadoService

{

[OperationContract(Name = "ObterFeriados")]

[ReferencePreservingDataContractFormat]

PagedList<BEFeriado> ObterFeriados(int pagina, int tamanhoPagina);

* + 1. Business

Camada que possui classes base que possuem funcionalidades prontas que servem de apoio para a criação da camada **Business** da aplicação.

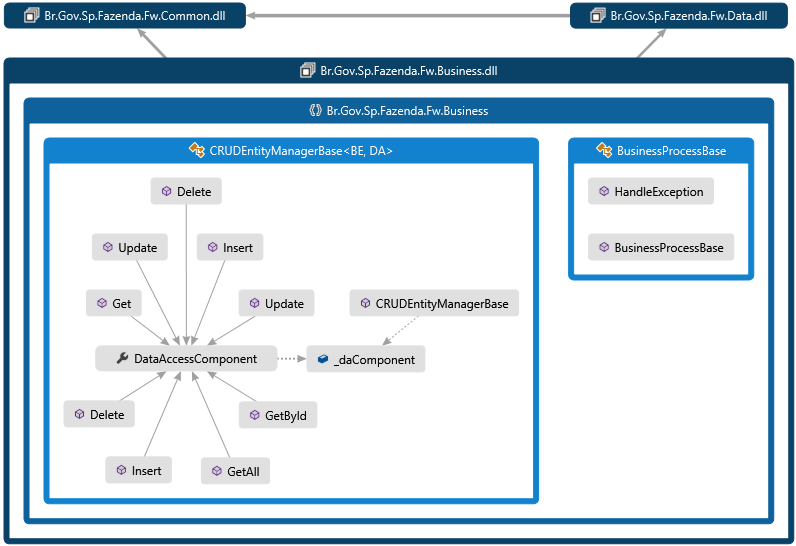


Figura 8 Principais componentes da camada Sefaz.Fw.Business

* **BusinessProcessBase:** classe que pode ser usada como base para as classes BP da aplicação e fornece um método abstrato para o tratamento de exceções.
* **CRUDEntityManagerBase<BE, DA>:** classe que pode ser usada como base para as classes EM da aplicação e fornece métodos prontos que executam operações de CRUD.
  + **BE:** **Business Entity** que se deseja executar as operações de CRUD;
  + **DA:** **Data Access** responsável por efetivar as operações de CRUD.
  + **Insert:** usando o DA, insere um ou uma lista de BE’s no repositório;
  + **Update:** usando o DA, atualiza um ou uma lista de BE‘s no repositório;
  + **Delete:** usando o DA, exclui um ou uma lista de BE’s do repositório;
  + **GetById:** usando o DA, retorna um BE do repositório fazendo uma busca pelo seu Id;
  + **Get:** usando o DA, retorna um objeto **QueryResult**com o resultado de uma busca filtrada usando um objeto **QueryParams**;
  + **GetAll**: usando o DA, retorna todos os BE’s do repositório;
    1. Data Access

Camada que possui as classes que executam acesso aos dados. As classes dessa camada são abstratas, portanto possuem funcionalidades prontas, porém devem ser estendidas de maneira a criar todas as operações necessárias que interagem com o banco de dados. Suas operações são efetivadas através do uso de *providers*.

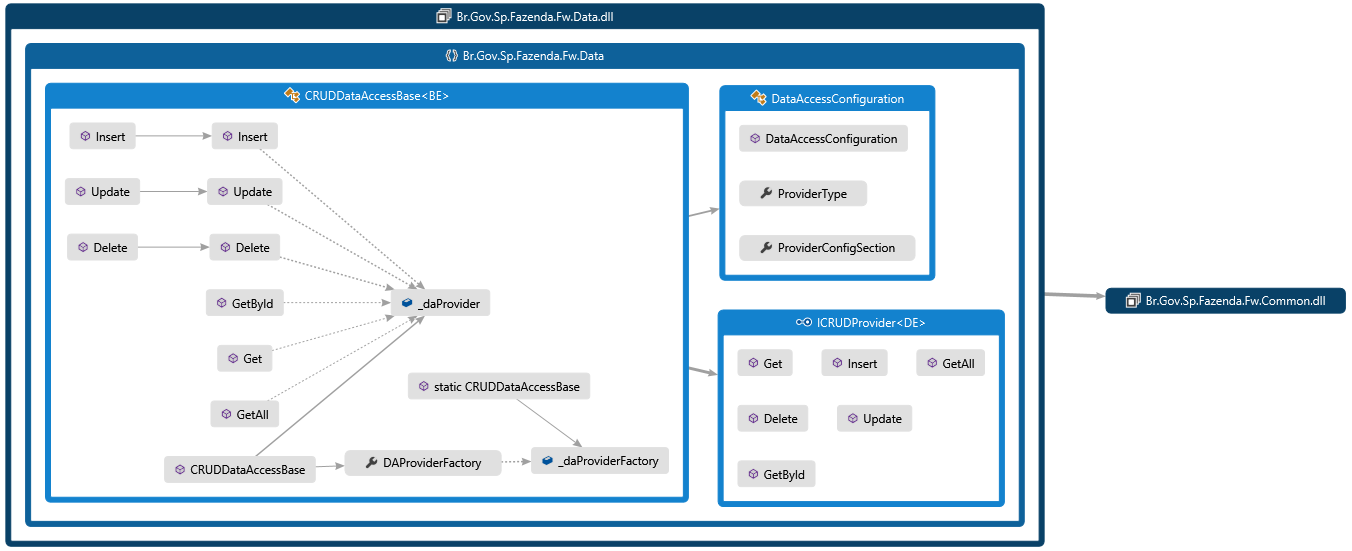


Figura 9 Principais componentes da camada Sefaz.Fw.Data

* **CRUDDataAccessBase<BE>**: classe que dá suporte a execução de operações de CRUD, repassando as chamadas dos métodos abaixo para *providers* responsáveis pela comunicação com os repositórios.
  + **Insert:** insere uma ou uma lista de entidades no repositório;
  + **Update:** atualiza uma ou uma lista de entidades no repositório;
  + **Delete:** exclui uma ou uma lista de entidades do repositório;
  + **GetById:** retorna uma entidade do repositório fazendo uma busca pelo seu Id;
  + **Get:** retorna um objeto **QueryResult**com o resultado de uma busca filtrada usando um objeto **QueryParams**;
  + **GetAll**: retorna todas as entidades do repositório;
* **ICRUDProvider<DE>**: interface que define os métodos para o *providers* que fazem operações de CRUD;
  + **DE:** Entidade a ser manipulada pelo *provider* de dados;
  + As funções dos demais métodos estão descritas acima.
* **DAProviderFactory**: propriedade que armazena a fábrica que constrói o *provider* de acesso a dados;
* **DataAccessConfiguration**: classe responsável por definir a configuração do bloco de acesso a dados
  + **ProviderType**: propriedade do tipo *string* que define a classe e *assembly* do *provider* de acesso à dados. Deve ser informado no seguinte formato: “nome\_da\_classe\_com\_namespace, strong\_name\_do\_assembly”;
  + **ProviderConfigSection**: propriedade do tipo *string* que define o nome da seção de configuração usada para inicializar o *ProviderType*.

Exemplo do arquivo de configuração:

<configSections>

<section name="dataAccessConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Data.DataAccessConfiguration, Br.Gov.Sp.Fazenda.Fw.Data, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

<section name="dataEFProviderConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Data.EFProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Data.EFProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

<section name="dataADOProviderConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Data.ADOProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Data.ADOProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

<section name="dataSQLServerProviderConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Data.SQLServerProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Data.SQLServerProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

</configSections>

<connectionStrings>

<add name="EFNorthwind" providerName="System.Data.EntityClient" connectionString="metadata=res://\*/Model.Northwind.csdl|res://\*/Model.Northwind.ssdl|res://\*/Model.Northwind.msl;provider=System.Data.SqlClient;provider connection string=&quot;Data Source=(LocalDB)\v11.0;AttachDbFilename=E:\NORTHWND.MDF;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework&quot;" />

<add name="ADONorthwind" providerName="System.Data.SqlClient" connectionString="Data Source=(LocalDB)\v11.0;AttachDbFilename=E:\NORTHWND.MDF;integrated security=True;MultipleActiveResultSets=True" />

</connectionStrings>

<dataAccessConfiguration providerType="Br.Gov.Sp.Fazenda.Fw.Data.ADOProvider`1, Br.Gov.Sp.Fazenda.Fw.Data.ADOProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d" providerConfigSection="" />

<dataEFProviderConfiguration connectionStringName="EFNorthwind" />

<dataADOProviderConfiguration connectionStringName="ADONorthwind" />

<dataSQLServerProviderConfiguration connectionStringName="ADONorthwind" generateCommands="true" />

<system.data>

<DbProviderFactories>

<add name="SqlClient Data Provider" invariant="System.Data.SqlClient" description=".Net Framework Data Provider for SqlServer" type="System.Data.SqlClient.SqlClientFactory, System.Data, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />

</DbProviderFactories>

</system.data>

Listagem 1 Detalhe do arquivo de configuração do bloco Data Access

* 1. Util

Classes que oferecem funcionalidades adicionais ao **Sefaz.Fw**. Não fazem parte do ***Core*** pois são utilizadas somente quando necessário. Com exceção do bloco **ExceptionHandling**, todos os outros usam classes externas que utilizam uma interface padrão (ou *providers*) para executar as suas tarefas.

O uso dos *providers* permite estender as funcionalidades do **Sefaz.Fw**, uma vez que a implementação tem dependência apenas com a interface de um *provider* e a não com a sua implementação.

As configurações dos blocos ***Util*** e *providers* são definidas nos arquivos de configuração da aplicação (web.config ou app.config), em seções de configuração criadas especialmente para esse fim. Para simplificar os arquivo .config, foram criados nomes default para cada uma dessas seções. Caso nenhum nome seja informado na criação das instâncias, serão utilizados os valores conforme a tabela abaixo:

|  |  |  |  |
| --- | --- | --- | --- |
| Bloco | Nome default | Provider | Nome default |
| Audit | auditConfiguration | LocalSQLCompactProvider | auditLocalSQLCompactProviderConfiguration |
| ServiceProvider | auditServiceProviderConfiguration |
| SQLServerProvider | auditSQLServerProviderConfiguration |
| Cache | cacheConfiguration | MemoryCacheProvider | NA |
| Config | cfgConfiguration | SQLServerProvider | cfgSQLServerProviderConfiguration |
| Data | dataAccessConfiguration | ADOProvider | dataADOProviderConfiguration |
| EFProvider | SQLServerProvider |
| Exception Handling | exceptionConfiguration | NA | NA |
| Log | logConfiguration | ConsoleOutputProvider | NA |
| FileProvider | logFileProviderConfiguration |
| ServiceProvider | logServiceProviderConfiguration |
| SQLServerProvider | logSQLServerProviderConfiguration |
| Security | securityConfiguration | SignatureProvider | NA |
| CetrtificateProvider | NA |

Tabela 4 Nomes default das seções de configuração

Os próximos tópicos irão detalhar cada um dos blocos, mostrar suas dependências, principais métodos e propriedades.

* + 1. Audit

O bloco **Audit** dá suporte à gravação de eventos de auditoria. Conta com dois estágios de persistência:

1. **Local** - de alta desempenho que faz o armazenamento temporário dos dados;
2. **Remoto** - responsável pelo armazenamento de longo prazo.

Além disso, possui um serviço que pode ser ligado ou desligado que é capaz de receber eventos de uma fonte externa à aplicação.

A criação de mais de uma instância do bloco é permitida, porém somente um serviço de recebimento de eventos pode existir por aplicação.

Possui a seguinte estrutura física:

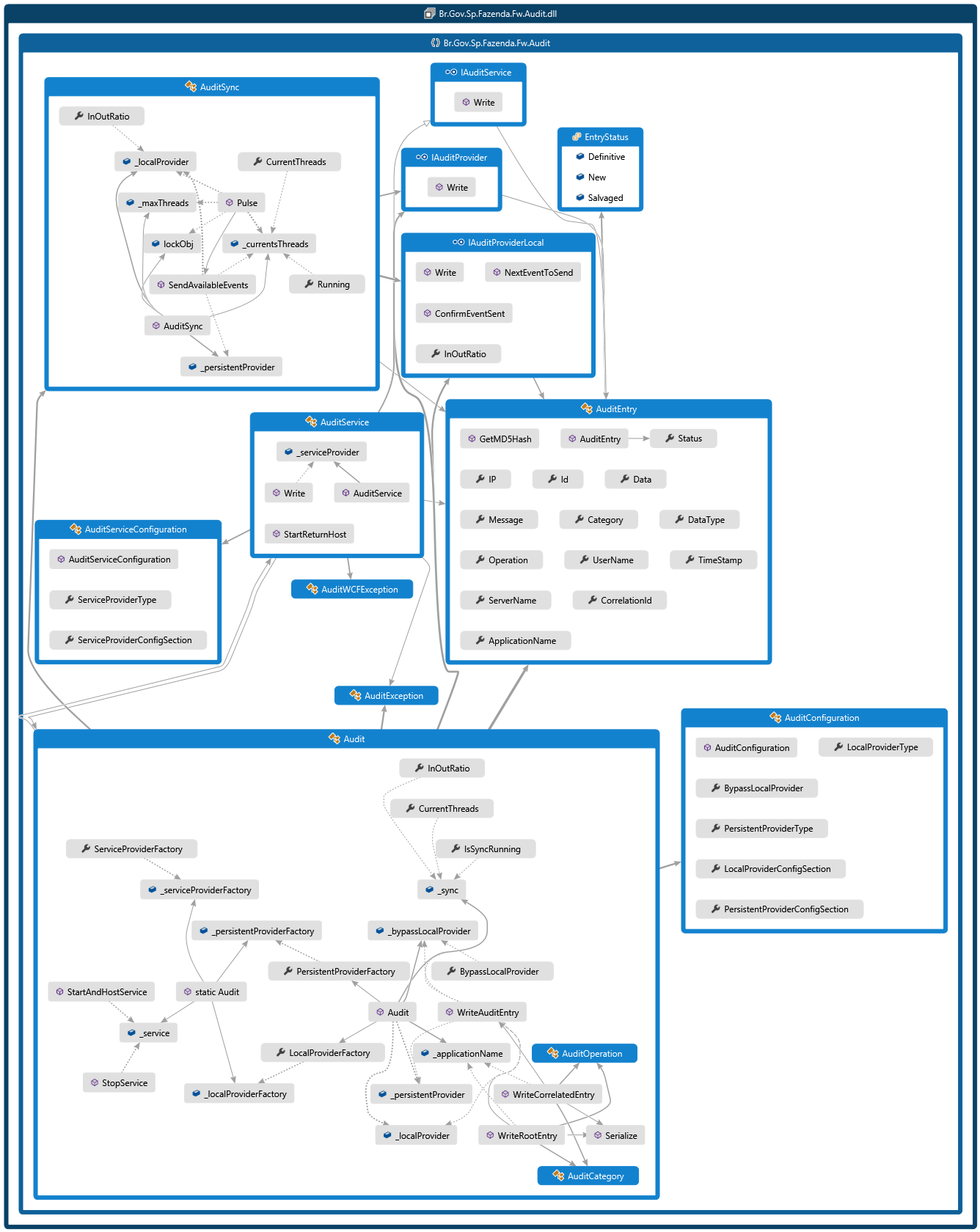


Figura 10 Arquitetura física do bloco Audit

* **Audit**: classe abstrata que deve ser estendida pela aplicação e que fornece implementações para a escrita dos eventos de auditoria.
  + **InOutRatio:** propriedade somente de leitura que mostra a relação entre a entrada de eventos e saída do *provider* local. Utilizada para iniciar novas threads de sincronização, caso necessário;
  + **CurrentThreads:** propriedade somente de leitura que mostra quantas threads estão fazendo a sincronização entre o *provider* local e definitivo;
  + **IsSyncRunning**: propriedade somente de leitura que indica se a sincronização está em execução;
  + **BypassLocalProvider**: propriedade somente de leitura que indica se o *provider* local está em uso;
  + **ServiceProviderFactory:** propriedade que armazena a fábrica que constrói o *provider* de persistência definitivo utilizado pelo serviço;
  + **PersistentProviderFactory:** propriedade que armazena a fábrica que constrói o *provider* de persistência definitivo;
  + **LocalProviderFactory:** propriedade que armazena a fábrica que constrói o *provider* de persistência local;
  + **StartAndHostService**: inicia e hospeda uma instância do serviço definido na classe *AuditService;*
  + **StopService**: para a instância do serviço definido na classe **AuditService***;*
  + **WriteRootEntry**e **WriteCorrelatedEntry**: responsáveis por montar o objeto de auditoria **AuditEntry**e chamar o método privado **WriteAuditEntry***;*
  + **WriteAuditEntry**: repassa o objeto de auditoria aos respectivos *providers* e se for o caso, sinalizar a classe **AuditSync** que existem dados a serem sincronizados através de uma chamada ao método **Pulse***;*
  + **Audit.AuditCategory**: subclasse que serve de base para a padronização dos termos utilizados como categorias nos métodos **WriteRootEntry**e **WriteCorrelatedEntry***.* Na classe derivada deve ser utilizada como tipo de propriedade: *public static readonly*;
  + **Audit.AuditOperation**: subclasse que serve de base para a padronização dos termos utilizados como operações nos métodos **WriteRootEntry**e **WriteCorrelatedEntry***.* Na classe derivada deve ser utilizada como tipo de propriedade: *public static readonly*;
  + **Serialize:** Serializa o objeto de negócio a ser passado aos *providers*;
* **AuditSync**: classe responsável por sincronizar os dados recebidos pelo *provider* local com o *provider* definitivo.
  + **InOutRatio:** propriedade somente de leitura que mostra a relação entre a entrada de eventos e saída do *provider* local. Utilizada para iniciar novas threads de sincronização, caso necessário;
  + **CurrentThreads:** propriedade somente de leitura que mostra quantas threads estão fazendo a sincronização entre o *provider* local e definitivo;
  + **Running**: propriedade que indica se a sincronização está em execução;
  + **Pulse**: inicia uma **Task**que chama o método **SendAvailableEvents**. Analisa a relação de entrada e saída de eventos e inicia novas threads de sincronização;
  + **SendAvailableEvents**: percorre a fila de envio o final enviando para o *provider* definitivo os itens armazenados. O método só termina ao encontrar o final da fila;
* **AuditService**: classe responsável por criar um serviço utilizando **WCF** com o objetivo de receber eventos de auditoria. Utilizado no cenário em que o aplicativo tem uma parte na DMZ e não possui acesso direto ao repositório definitivo de auditoria. Independentemente da quantidade de blocos **Audit** criados, somente um serviço de recebimento pode ser iniciado.
  + **Write**: envia objeto de auditoria recebido de maneira síncrona para o*provider* de persistência definitiva;
  + **StartAndReturnHost**: inicia e retorna uma instância do serviço definido na classe **AuditService**;
* **AuditEntry**: classe responsável por definir a entidade que será usada para a persistência dos eventos de auditoria.
  + **GetMD5Hash**: calcula o *hash* do objeto baseado na sua versão serializada.
* **AuditStatus**: *enumeration* que define o estado do evento de auditoria:
  + **New:** o objeto acabou de ser criado e inserido na fila;
  + **Salvaged:** o objeto foi recuperado do repositório local e inserido na fila de envio;
  + **Definitive:** o objeto foi recuperado do repositório definitivo;
* **AuditConfiguration**: classe responsável por definir a configuração do bloco:
  + **LocalProviderType:** propriedade do tipo *string* que define a classe e *assembly* do *provider* de persistência local. Deve ser informado no seguinte formato: “nome\_da\_classe\_com\_namespace, strong\_name\_do\_assembly”;
  + **LocalProviderConfigSection:** propriedade do tipo string que define o nome da seção de configuração usada para inicializar o **LocalProviderType**; É passado para o método **ApplyConfiguration** definido na interface **IUtilProvider**;
  + **PersistentProviderType:** propriedade do tipo *string* que define a classe e *assembly* do *provider* de persistência definitiva. Deve ser informado no seguinte formato: “nome\_da\_classe\_com\_namespace, strong\_name\_do\_assembly”;
  + **PersistentProviderConfigSection:** propriedade do tipo string que define o nome da seção de configuração usada para inicializar o **PersistentProviderType**; É passado para o método **ApplyConfiguration** definido na interface **IUtilProvider**;
  + **BypassLocalProvider:** propriedade booleana que define se o *provider* local e o mecanismo de sincronização entre os *providers* local e definitivo será usado. Se verdadeiro, o *provider* local não é utilizado e os eventos são gravados de maneira síncrona com o *provider* definitivo;
* **AuditServiceConfiguration**: classe responsável por definir a configuração do serviço de auditoria:
  + **ServiceProviderType:** propriedade do tipo *string* que define a classe e *assembly* do *provider* utilizado pelo serviço de recebimento de eventos de auditoria. Deve ser informado no seguinte formato: “nome\_da\_classe\_com\_namespace, strong\_name\_do\_assembly”;
  + **ServiceProviderConfigSection:** propriedade do tipo string que define o nome da seção de configuração usada para inicializar o **ServiceProviderType**; É passado para o método **ApplyConfiguration** definido na interface **IUtilProvider**;
* **IAuditService:** define o contrato utilizado pelo **AuditService***;*
* **IAuditProvider**: define a interface para os *providers* de persistência definitiva; Implementa a interface **IUtilProvider**;
* **IAuditProviderLocal**: define a interface para os *providers* de persistência local; Implementa a interface **IUtilProvider**;
* **AuditException:** classe que define uma exceção de negócio para o bloco e *providers*.
* **AuditWCFException:** classe que define uma exceção de negócio para o serviço.

Exemplo do arquivo de configuração:

<configSections>

<section name="auditConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Audit.AuditConfiguration, Br.Gov.Sp.Fazenda.Fw.Audit, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

<section name="auditLocalSQLCompactProviderConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Audit.LocalSQLCompactProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Audit.LocalSQLCompactProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

<section name="auditSQLServerProviderConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Audit.SQLServerProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Audit.SQLServerProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

</configSections>

<auditConfiguration bypassLocalProvider="False"

persistentProviderType="Br.Gov.Sp.Fazenda.Fw.Audit.SQLServerProvider, Br.Gov.Sp.Fazenda.Fw.Audit.SQLServerProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"

localProviderType="Br.Gov.Sp.Fazenda.Fw.Audit.LocalSQLCompactProvider, Br.Gov.Sp.Fazenda.Fw.Audit.LocalSQLCompactProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d" />

<connectionStrings>

<add name="TesteDB"

connectionString="Data Source=.\SQLExpress;Initial Catalog=Teste;Integrated Security=True;"

providerName="System.Data.SqlClient"/>

</connectionStrings>

<auditSQLServerProviderConfiguration procedureName="INSERT\_AUDIT" connectionStringName="TesteDB"/>

<auditLocalSQLCompactProviderConfiguration repositoryFullPath="c:\temp\data.sdf" safeMode="False" />

Listagem 2 Detalhe do arquivo de configuração do bloco Audit

A configuração necessária para a hospedagem do serviço WCF também reside neste arquivo, porém não é específica do bloco. Trata-se de uma configuração nativa do WCF. O artigo abaixo possui a descrição completa dessa configuração:

**Windows Communication Foundation Configuration Schema**  
<http://msdn.microsoft.com/en-us/library/ms731734.aspx>

* + 1. Cache

O bloco de **Cache** dá suporte ao mecanismo de cache de objetos com base em uma chave e por um período especificado de tempo. Permite criar mais de uma instância do bloco com configurações diferentes nos casos onde se é necessário o uso de diferentes mecanismos de cache.

Principais componentes do bloco de **Cache**:

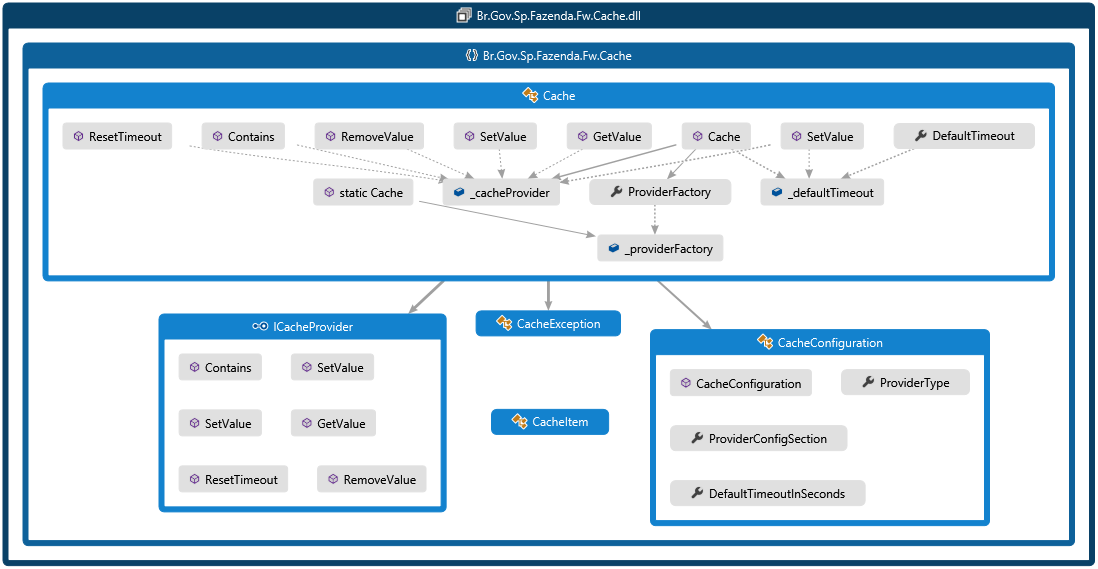


Figura 11 Principais componentes do bloco Cache

* **Cache**: contém os métodos do bloco de cache que serão usados pela aplicação. Repassa as chamadas para o *provider* de cache
  + **Contains:** verifica a existência de um item no cache;
  + **GetValue:** retorna um item do cache;
  + **RemoveValue:** remove um item do cache;
  + **SetValue:** insere um item no cache. Possui *overloads* que permitem definir um tempo relativo ou absoluto de expiração ou se a expiração é reiniciada à cada acesso ao item;
  + **ResetTimeout:** Reinicia o tempo de expiração do item;
  + **ProviderFactory:** propriedade que armazena a fábrica que constrói o *provider* do bloco;
* **CacheConfiguration**: classe responsável por definir a configuração do bloco:
  + **ProviderType:** propriedade do tipo *string* que define a classe e *assembly* do *provider*. Deve ser informado no seguinte formato: “nome\_da\_classe\_com\_namespace, strong\_name\_do\_assembly”;
  + **ProviderConfigSection:** propriedade do tipo string que define o nome da seção de configuração usada para inicializar o *ProviderType*; É passado para o método *ApplyConfiguration* definido na interface *IUtilProvider*;
  + **DefaultTimeoutInSeconds:** propriedade do tipo inteiro que define o timeout padrão em segundos.
* **CacheItem:** Classe que define um item de cache
  + **Value:** Propriedade do tipo *object* para manter no cache
* **Timeout:** Propriedade para definição do tempo de duração cache
* **ResetTimeoutOnEveryAccess**: Reinicia o valor do timeout todas as vezes que o objeto é acessado no cache;
* **ICacheProvider:** define a interface para o *provider* do cache; Implementa a interface *IUtilProvider*;
* **CacheException:** classe que define uma exceção de negócio para o bloco e providers.

Exemplo do arquivo de configuração:

<configSections>

<section name="cacheConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Cache.CacheConfiguration, Br.Gov.Sp.Fazenda.Fw.Cache, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

</configSections>

<cacheConfiguration defaultTimeoutInSeconds="100"

providerType="Br.Gov.Sp.Fazenda.Fw.Cache.MemoryCacheProvider, Br.Gov.Sp.Fazenda.Fw.Cache.MemoryCacheProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"

providerConfigSection="" />

Listagem 3 Detalhe do arquivo de configuração do bloco de Cache

* + 1. Config

O bloco de **Config** dá suporte ao mecanismo de persistência de objetos de configuração. Os dados são persistidos e recuperados com o uso de um *provider*. Utiliza o bloco de **Cache** para manter os objetos recuperados na memória por um tempo pré-definido.

Possui a seguinte estrutura física:

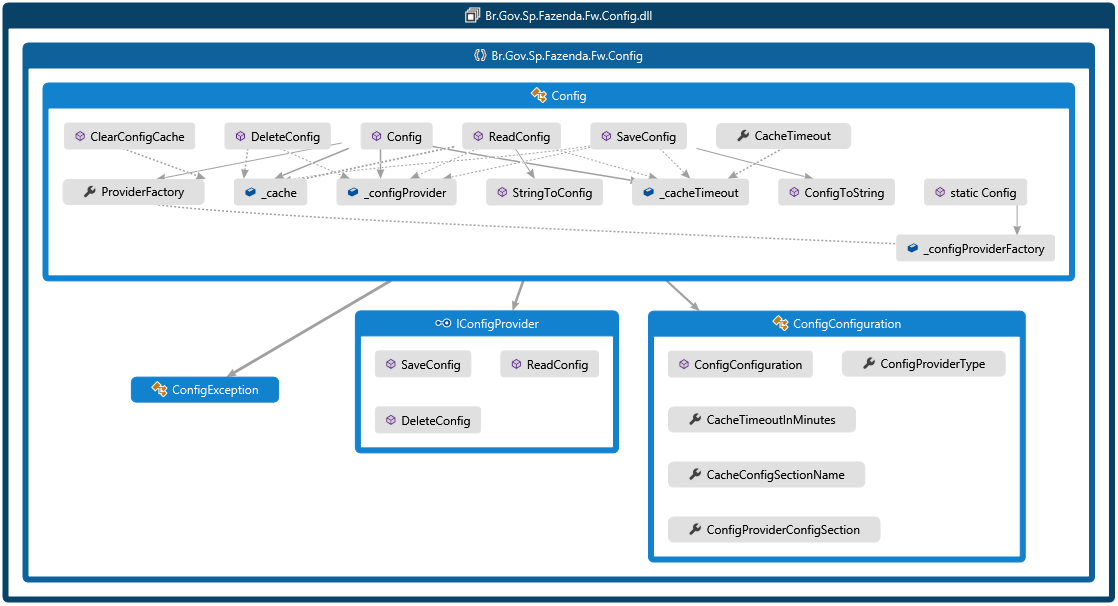


Figura 12 Estrutura física do bloco Config

* **Config**: contém os métodos para manipulação de objetos de configuração que serão lidos e persistidos pelo *provider* do config.
  + **SaveConfig:** Recebe umobjeto de configuração e uma chave de busca. Salva o objeto usando o *provider* do bloco config, além de colocar o objeto no cache utilizando o bloco de cache;
  + **ReadConfig:** Lê um objeto de configuração usando uma chave de busca. Primeiro tenta recuperá-lo do cache em seguida do provider do bloco config;
  + **DeleteConfig:** Remove um objeto do cache e do repositório definitivo;
  + **CacheTimeout**: propriedade que contém o timeout dos objetos colocados no cache;
  + **ClearConfigCache:** Limpa todos os objetos do cache, mas não altera os objetos amazenados pelo provider do config;
  + **Config2String:** Converte a instância de um objeto de configuração em uma string contendo sua versão serializada;
  + **String2Config:** Converte uma string contendo uma versão serializada de um objeto de configuração novamente em uma instância;
  + **ProviderFactory:** propriedade que armazena a fábrica que constrói o *provider* do bloco.
* **ConfigConfiguration**: classe responsável por definir a configuração do bloco:
  + **CacheTimeoutInMinutes:** propriedade do tipo inteiro que define o tempo de vida das configurações uma vez que são inseridas no cache;
  + **ConfigProviderType:** propriedade do tipo *string* que define a classe e *assembly* do *provider* do config. Deve ser informado no seguinte formato: “nome\_da\_classe\_com\_namespace, strong\_name\_do\_assembly”;
  + **ConfigProviderConfigSection:** propriedade do tipo string que define o nome da seção de configuração usada para inicializar o *ConfigProviderType*; É passado para o método *ApplyConfiguration* definido na interface *IUtilProvider*;
  + **CacheConfigSection:** propriedade do tipo string que define o nome da seção de configuração usada para inicializar o bloco Cache;
* **IConfigProvider**: define a interface para o *provider* do config; Implementa a interface *IUtilProvider*;
* **ConfigException:** classe que define uma exceção de negócio para o bloco e providers.

Exemplo do arquivo de configuração:

<configSections>

<section name="cacheConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Cache.CacheConfiguration, Br.Gov.Sp.Fazenda.Fw.Cache, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

<section name="cfgConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Config.ConfigConfiguration, Br.Gov.Sp.Fazenda.Fw.Config, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

<section name="cfgSQLServerProviderConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Config.SQLServerProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Config.SQLServerProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

</configSections>

<connectionStrings>

<add name="TesteDB" connectionString="Data Source=.\SQLExpress;Initial Catalog=Teste;Integrated Security=True;" providerName="System.Data.SqlClient"/>

</connectionStrings>

<cacheConfiguration providerType="Br.Gov.Sp.Fazenda.Fw.Cache.MemoryCacheProvider, Br.Gov.Sp.Fazenda.Fw.Cache.MemoryCacheProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d" />

<cfgConfiguration cacheTimeoutInMinutes="1"

cfgProviderType="Br.Gov.Sp.Fazenda.Fw.Config.SQLDataProvider, Br.Gov.Sp.Fazenda.Fw.Config.SQLDataProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"

cacheConfigSectionName="cacheConfiguration" />

<cfgSQLServerProviderConfiguration cfgTableName="Config" connectionStringName="TesteDB"/>

Listagem 4 Detalhe do arquivo de configuração do bloco de Config.

* + 1. Exception Handling

Bloco que dá suporte ao log e tratamento de exceções. Utiliza o bloco de **Log** para gravar os detalhes da exceção e deve ser utilizado pela camada **Business** nas classes **Business Process**.

Através de políticas definidas no arquivo de configuração, permite tomar diferentes ações dependendo do tipo de exceção encontrada. Não utiliza *providers*, pois as suas funcionalidades são providas pelo próprio .Net.

O *rethow* de uma exceção só pode acontecer no *catch* referente ao *try* onde ela ocorreu. Em função disso toda vez que for utilizado é necessário usar o seguinte padrão:

* Caso esteja usando o método fornecido pela classe **BusinessProcessBase** o padrão é:

if (this.HandleException(<EXCEPTION>, <NOME DA POLITICA>, <MENSAGEM DE ERRO>))

throw;

Listagem 5 Uso do método de tratamento de exceção - BPBase

* Caso esteja usando o bloco diretamente:

if (exceptionHandlerInstance.HandleException(<EXCEPTION>, <NOME DA POLITICA>, <MENSAGEM DE ERRO>))

throw;

Listagem 6 Uso do método de tratamento de exceção – Uso direto do bloco

Arquitetura física:

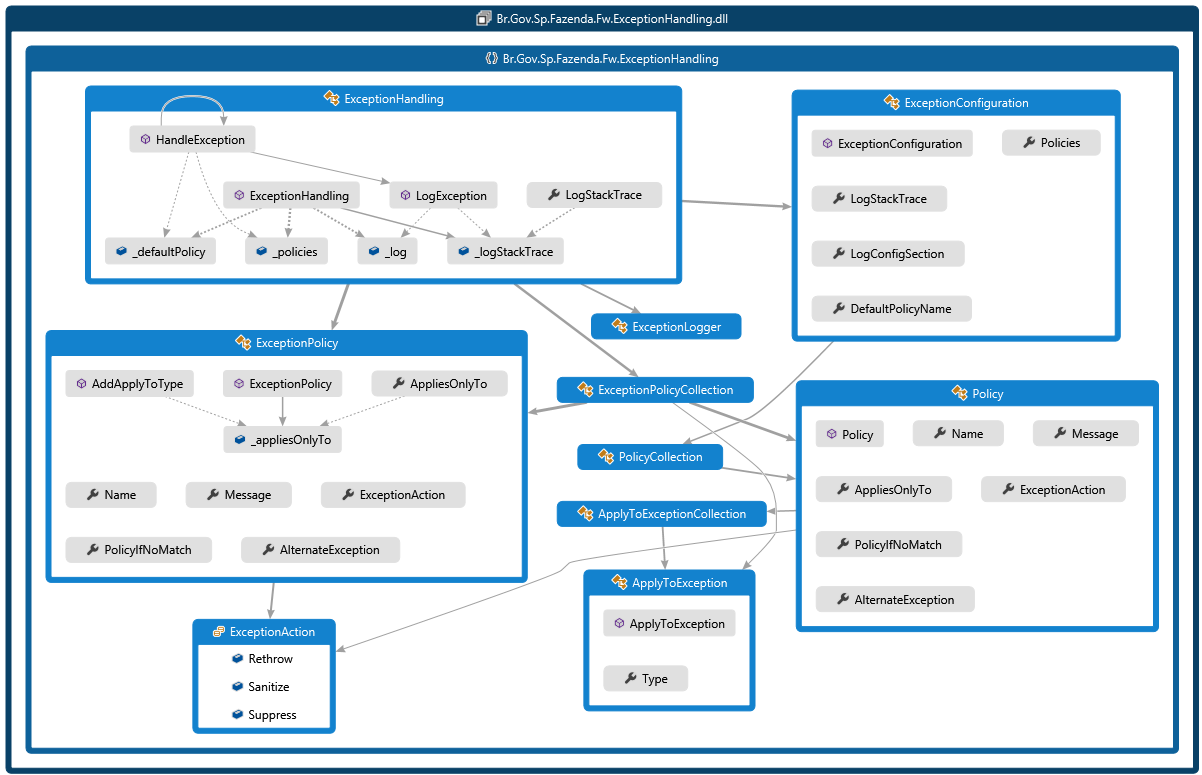


Figura 13 Estrutura física do bloco Exception Handling

* **ExceptionHanding**: contém o método para tratamento e log das exceções
  + **HandleException:** Recebe a exceção, política e mensagem. Loga os detalhes e depois toma as ações definidas pela política;
  + **LogException**: chamado pelo método *HandleException* utiliza o bloco de Log para gravar os detalhes da exceção;
  + **LogStackTrace**: propriedade que indica se o log deve incluir o stack do método que gerou a exceção
* **ExceptionLogger:** instância do bloco log responsável por gravar os detalhes das exceções;
* **ExceptionConfiguration:** classe responsável por definir a configuração do bloco:
  + **LogStackTrace**: propriedade do tipo boolean que indica se o *stacktrace* deve ser logado;
  + **LogConfigSection**: propriedade do tipo string que contém no nome da seção de configuração do bloco Log;
  + **DefaultPolicyName**: propriedade do tipo string que contém o nome da política default;
  + **Policies**: coleção de políticas;
* **PolicyCollection e Policy:** classe que define as políticas de tratamento de exceção e uma coleção dessas políticas:
  + **Name**: propriedade do tipo string que contém no nome da política;
  + **Message**: mensagem padrão, caso nenhuma seja passada no método de tratamento;
  + **ExceptionAction**: propriedade do tipo *ExceptionAction* que contém a ação de tratamento
  + **AppliesOnlyTo**: coleção de objetos *ApplyToException* que limitam a aplicação dessa política aos tipos definidos nessa lista;
  + **PolicyIfNoMatch**: caso a política tenha o atributo *AppliesOnlyTo* e a exceção recebida não seja de um desses tipos, essa propriedade do tipo *string* define o nome da nova política a ser usada;
  + **AlternateException**: propriedade do tipo *string* que contém o tipo da nova exceção a ser lançada, caso o tipo de ação escolhida seja *Sanitize*;
* **ApplyToExceptionCollection e ApplyToException:** que define o tipo da exceção à qual determinada política se aplica e uma coleção de tipos:
  + **Type**: propriedade do tipo string que contém o tipo da exceção, no formato: “nome\_da\_classe\_com\_namespace, strong\_name\_do\_assembly”. Caso o tipo esteja em um assembly já carregado (blocos ou o próprio .Net) pode-se omitir no strong\_name\_do\_assembly;
* **ExceptionPolicyCollection e ExceptionPolicy:** classes que armazenam as políticas lidas do config;
* **ExceptionAction**: *enumeration* que contém as possíveis ações a serem tomadas após o log da exceção:
  + **Rethrow:** lança a mesma exceção recebida;
  + **Suppress:** não lança nenhuma exceção;
  + **Sanitize:** lança uma nova exceção contendo a mensagem informada.

No momento da criação do bloco as seguintes regras são validadas:

* É necessário existir uma política default;
* A política default não pode ter o atributo appliesOnlyTo;
* Se uma política tem o atributo appliesOnlyTo, ela deve ter o atributo policyIfNoMatch.

Exemplo do arquivo de configuração:

<configSections>

<section name="exceptionConfiguration" type="Br.Gov.Sp.Fazenda.Fw.ExceptionHandling.ExceptionConfiguration,Br.Gov.Sp.Fazenda.Fw.ExceptionHandling, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

<section name="logConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Log.LogConfiguration, Br.Gov.Sp.Fazenda.Fw.Log, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

<section name="fileLogProviderConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Log.FileLogProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Log.FileLogProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

</configSections>

<exceptionConfiguration logConfigSection="logConfiguration" logStackTrace="true" defaultPolicyName="Rethrow" >

<policies>

<policy name="Suppress" exceptionAction="Suppress" policyIfNoMatch="Sanitize">

<appliesOnlyTo>

<exception type="System.Exception" />

</appliesOnlyTo>

</policy>

<policy name="Rethrow" exceptionAction="Rethrow" />

<policy name="Sanitize" exceptionAction="Sanitize" alternateException ="System.InvalidOperationException" message="Mensagem padrao para sanitizacao (originada em App.Config)"/>

</policies>

</exceptionConfiguration>

<logConfiguration providerType="Br.Gov.Sp.Fazenda.Fw.Log.FileLogProvider,Br.Gov.Sp.Fazenda.Fw.Log.FileLogProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d" />

<fileLogProviderConfiguration logFileDirectory="c:\temp\"

maxSizeMB="2"

filesToCreate="5" />

* + 1. Log

O bloco **Log** dá suporte à gravação de eventos de log. Utiliza *providers* de persistência síncronos com o aplicativo que solicita a gravação de um registro de log. Possui um serviço que pode ser ligado ou desligado que é capaz de receber eventos de uma fonte externa à aplicação.

Permite a criação de mais de uma instância do bloco, porém somente um serviço de recebimento de eventos pode existir por aplicação.

Possui a seguinte estrutura física:

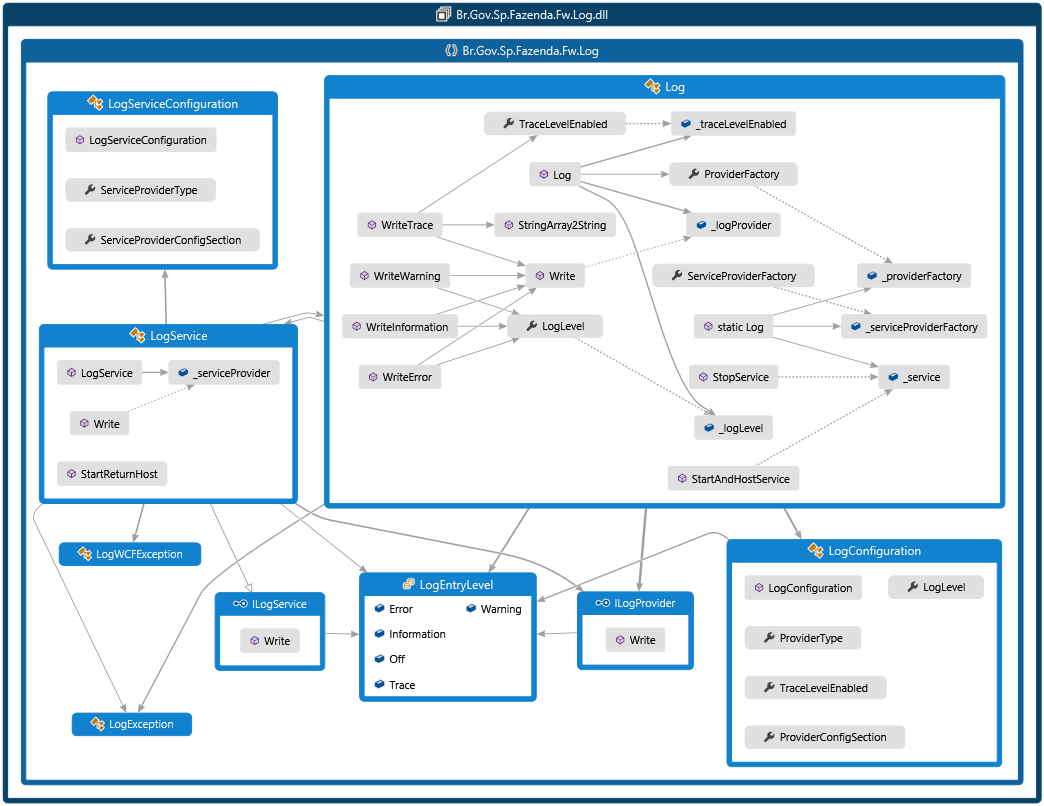


Figura 14 Arquitetura física do bloco Log

* **Log**: classe abstrata que deve ser estendida pela aplicação e fornece métodos que fazem a gravação dos eventos de log.;
  + **WriteError, WriteInformation e WriteWarning**: verificam o nível de log configurado no bloco e se for o caso, repassam a chamada para o método *Write*;
  + **WriteTrace**: verifica se o nível *trace* está habilitado no bloco e se estiver repassa a chamada para o método *Write*. Antes de repassar a chamada inclui o *stack trace* no parâmetro *longText*;
  + **Write**: Envia as informações para o *provider* configurado;
  + **StartAndHostService**: inicia e hospeda uma instância do serviço definido na classe *LogService*;
  + **StopService**: termina a instância do serviço definido na classe *LogService*;
  + **TraceLevelEnabled**: propriedade do tipo *boolean* que indica se o nível *trace* está habilitado;
  + **LogLevel**: altera o nível de log configurado;
  + **ServiceProviderFactory:** propriedade que armazena a fábrica que constrói o *provider* utilizado pelo serviço;
  + **ProviderFactory:** propriedade que armazena a fábrica que constrói o *provider* utilizado pelo bloco;
* **LogService**: classe responsável por criar um serviço utilizando WCF com o objetivo de receber eventos de log. Utilizado no cenário em que o aplicativo tem uma parte na DMZ e não possui acesso direto ao repositório definido pelo *provider*. Independentemente da quantidade de blocos Log criados, somente um serviço de recebimento pode ser iniciado.
  + **Write**: envia as informações recebidas de maneira síncrona para o *provider* de persistência.
  + **StartAndReturnHost**: inicia e retorna uma instância do serviço definido na classe *LogService*;
* **LogLevel**: enumeration que define os possíveis níveis da informação sendo logada;
* **ILogService**: define o contrato utilizado pelo *LogService*;
* **ILogProvider**: define a interface para os *providers* de log; Implementa a interface *IUtilProvider*;
* **LogConfiguration**: classe responsável por definir a configuração do bloco:
  + **LogLevel**: propriedade enumeration do tipo *LogLevel* que define o nível mínimo a ser logado, respeitando a seguinte ordem: *Off*, *Error*, *Warning* e *Information*;
  + **TraceLevelEnabled**: propriedade do tipo booleana que define se o nível *trace* está ativado*;*
  + **ProviderType**: propriedade do tipo string que define a classe e assembly do provider a ser utilizado. Deve ser informado no seguinte formato: “nome\_da\_classe\_com\_namespace, strong\_name\_do\_assembly”;
  + **ProviderConfigSection:** propriedade do tipo string que define o nome da seção de configuração usada para inicializar o *ProviderType*; É passado para o método *ApplyConfiguration* definido na interface *IUtilProvider*
* **LogServiceConfiguration**: classe responsável por definir a configuração do bloco:
  + **ServiceProviderType:** propriedade do tipo *string* que define a classe e *assembly* do *provider* utilizado pelo serviço de recebimento dos eventos de log. Deve ser informado no seguinte formato: “nome\_da\_classe\_com\_namespace, strong\_name\_do\_assembly”;
  + **ServiceProviderConfigSection:** propriedade do tipo string que define o nome da seção de configuração usada para inicializar o *ServiceProviderType*; É passado para o método *ApplyConfiguration* definido na interface *IUtilProvider*;
* **LogException**: classe que define uma exceção de negócio para o bloco e *providers;*
* **AuditWCFException:** classe que define uma exceção de negócio para o serviço.

Exemplo do arquivo de configuração:

<configSections>

<section name="logConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Log.LogConfiguration, Br.Gov.Sp.Fazenda.Fw.Log"/>

<section name="logFileProviderConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Log.FileProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Log.FileProvider"/>

<section name="logSQLServerProviderConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Log.SQLServerProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Log.SQLServerProvider"/>

<section name="logServiceProviderConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Log.LogServiceProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Log.ServiceProvider"/>

</configSections>

<connectionStrings>

<add name="TesteDB" connectionString="Data Source=.\SQLExpress;Initial Catalog=Teste;Integrated Security=True;" providerName="System.Data.SqlClient"/>

</connectionStrings>

<logServiceProviderConfiguration endpointConfigurationName="netTcpLogServiceConfig" />

<logConfiguration providerType="Br.Gov.Sp.Fazenda.Fw.Log.ServiceProvider,

Br.Gov.Sp.Fazenda.Fw.Log.ServiceProvider"

logLevel="Information" traceLevelEnabled="True" />

<logFileProviderConfiguration logFileDirectory="c:\temp\" maxSizeMB="20" filesToCreate="10" />

<logSQLServerProviderConfiguration procedureName="INSERT\_LOG" connectionStringName="TesteDB" />

Listagem 7 Detalhe do arquivo de configuração do bloco Log

A configuração necessária para a hospedagem do serviço WCF também reside neste arquivo, porém não é específica do bloco. Trata-se de uma configuração nativa do WCF. O artigo abaixo possui a descrição completa dessa configuração:

**Windows Communication Foundation Configuration Schema**<http://msdn.microsoft.com/en-us/library/ms731734.aspx>

* + 1. Security

O bloco **Security** dá suporte à manipulação e validação de certificados e assinaturas digitais.

Possui a seguinte estrutura física:

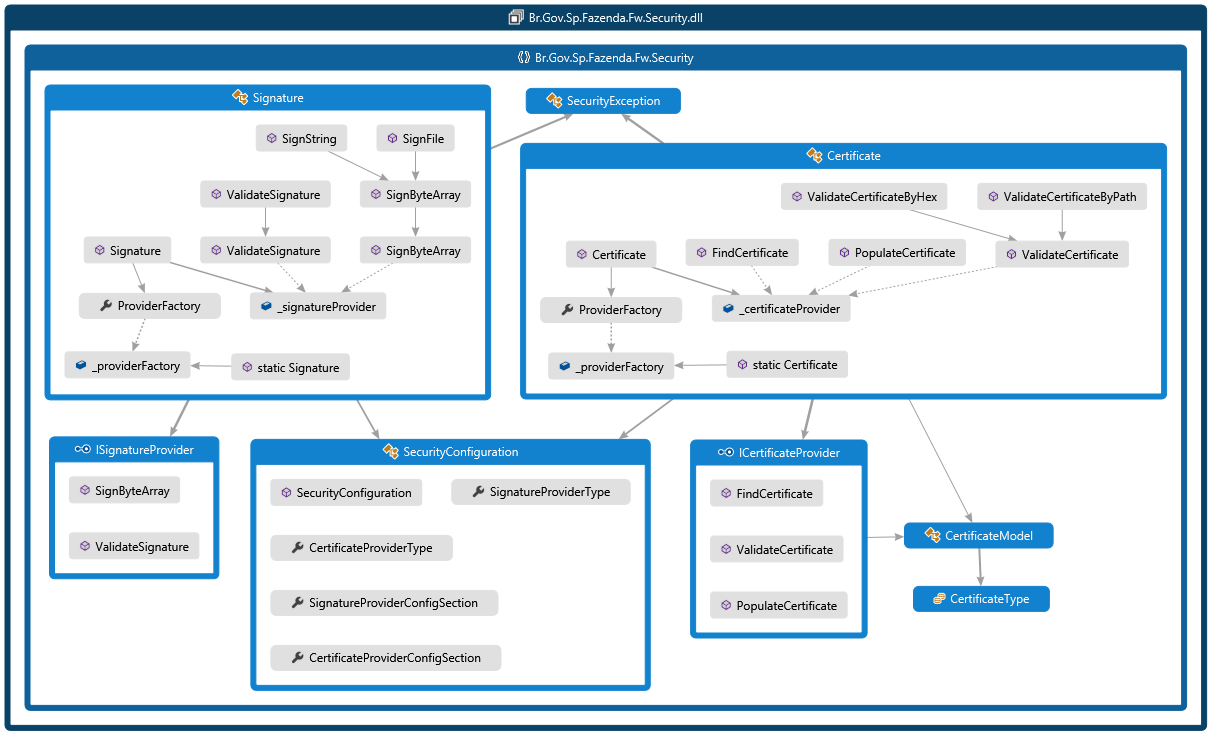


Figura 15 Arquitetura física do bloco Security

* **Certificate**: contém os métodos para manipulação de certificados digitais através de um *provider;*
  + **PopulateCertificate**: recebe um certificado digital x509 em formato hexadecimal e retorna uma classe *CertificateModel* com as propriedades do mesmo preenchidas.
  + **ValidateCertificateByHex:** recebe uma string que representa um certificado digital x509 em formato hexadecimal, faz a conversão para um certificado e chama o método *ValidateCertificate*;
  + **ValidateCertificateByPath:** recebe uma string que contém o path de um certificado digital, faz a conversão para um certificado e chama o método *ValidateCertificate*
  + **ValidateCertificate**: recebe um certificado digital para validação quanto a: Validade, Cadeia, Policy e Revogação;
  + **FindCertificate**: encontra e retorna um certificado armazenado na certificate store da máquina;
  + **ProviderFactory:** propriedade que armazena a fábrica que constrói o *provider* utilizado pela classe *Certificate*;
* **CertificateModel:** classe a ser preenchida pelo método *PopulateCertificate*;
* **Signature**: Classe que oferece métodos para manipulação de assinaturas digitais no formato PKCS#7 através de um provider:
  + **ValidateSignature**: verifica se um arquivo ou array de bytes possui uma assinatura digital válida;
  + **SignString**: recebe uma string e um critério de busca. Converte a string em um array de bytes e chama o método *SignByteArray*;
  + **SignFile**: recebe uma string com o path de um arquivo e um critério de busca. Lê o conteúdo do arquivo em um array de bytes e chama o método *SignByteArray*
  + **SignByteArray**: realiza a assinatura digital PKCS#7. Possui overloads que permitem passar um certificado ou um critério de busca na certificate store da máquina;
  + **ProviderFactory:** propriedade que armazena a fábrica que constrói o *provider* utilizado pela classe *Signature*;
* **ICertificateProvider:** define a interface para o *provider* de manipulação de certificados digitais. Implementa a interface *IUtilProvider*;
* **ISignatureProvider:** define a interface para o *provider* de assinatura digital. Implementa a interface *IUtilProvider*;
* **SecurityConfiguration**: classe responsável por definir a configuração do bloco:
  + **CertificateProviderType:** propriedade do tipo *string* que define a classe e *assembly* do *provider* que manipula os certificados digitais. Deve ser informado no seguinte formato: “nome\_da\_classe\_com\_namespace, strong\_name\_do\_assembly”;
  + **ServiceProvidereConfigSection:** propriedade do tipo string que define o nome da seção de configuração usada para inicializar o *CertificateProviderType*; É passado para o método *ApplyConfiguration* definido na interface *IUtilProvider*;
  + **SignatureProviderType:** propriedade do tipo *string* que define a classe e *assembly* do *provider* que manipula as assinaturas digitais. Deve ser informado no seguinte formato: “nome\_da\_classe\_com\_namespace, strong\_name\_do\_assembly”;
  + **SignatureProviderConfigSection:** propriedade do tipo string que define o nome da seção de configuração usada para inicializar o *SignatureProviderType*; É passado para o método *ApplyConfiguration* definido na interface *IUtilProvider*;
* **SecurityException:** exceção retornada pelo bloco em caso de erro;

Exemplo do arquivo de configuração:

<configSections>

<section name="securityConfiguration"

type="Br.Gov.Sp.Fazenda.Fw.Security.SecurityConfiguration, Br.Gov.Sp.Fazenda.Fw.Security"/>

</configSections>

<securityConfiguration

certificateProviderType="Br.Gov.Sp.Fazenda.Fw.Security.CertificateProvider,

Br.Gov.Sp.Fazenda.Fw.Security.SecurityProvider"

certificateProviderConfigSection=""

signatureProviderType="Br.Gov.Sp.Fazenda.Fw.Security.SignatureProvider,

Br.Gov.Sp.Fazenda.Fw.Security.SecurityProvider"

signatureProviderConfigSection="" />

Listagem 8 Arquivo de configuração do bloco de Security

* 1. Providers

Os *providers* são bibliotecas auxiliares que contém a implementação concreta de uma interface de comunicação, que pode estar no ***Core*** ou em um bloco ***Util***. A configuração de cada bloco indica qual a classe do *provider* deve ser instanciada*.* Opcionalmente é possível indicar uma seção de configuração específica para cada *provider*, caso essa propriedade seja omitida um valor pré-definido é utilizado, conforme a tabela no tópico 7.2:

Para exemplificar se uso e enriquecer a prova de conceito, foram desenvolvidos *providers* para os seguintes blocos: Audit; Cache; Config; Data Access; Log e Security.

* + 1. Audit

Para mostrar o funcionamento do bloco de auditoria foram criados os seguintes *providers*:

#### LocalSQLCompactProvider

Tem a responsabilidade de gravar o evento de auditoria localmente em um banco de dados do tipo **SQL Server Compact**. Este banco tem a vantagem de não requerer instalação, possuir conectores nativos para o .Net e abstrair as operações de leitura e escrita.

O *provider* trabalha com uma fila em memória e é a partir dela que os objetos de auditoria são sincronizados com o *provider* de persistência definitivo. O objeto de auditoria é gravado primeiramente no disco (para garantir o armazenamento) e depois inserido na fila de envio. Ao receber um objeto de auditoria do tipo *AuditEntry* ele é imediatamente criptografado e convertido em um objeto do tipo *SecureAuditEntry* que então é gravado no disco e enviado para a fila.

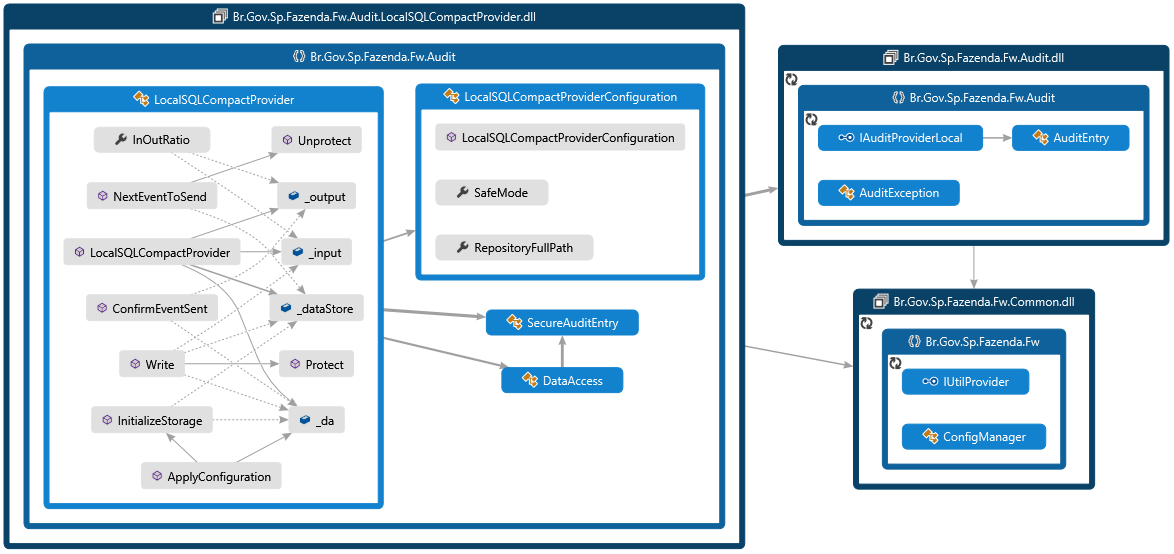


Figura 16 Arquitetura física do provider LocalSQLCompactProvider (Audit)

* **LocalSQLCompactProvider**: classe que implementa a interface **IAuditProviderLocal** e possui as regras de negócio relativas ao funcionamento do *provider*:
  + **ApplyConfiguration**: método que termina a inicialização do *provider*. Faz a inicialização lendo as propriedade definidas em uma seção de configuração;
  + **InOutRatio**: propriedade do tipo int que mostra a relação de entrada e saída de eventos desse provider. A classe **AuditSync** do bloco **Audit** utiliza esse valor para criar novas threads de sincronização;
  + **InitializeStorage**: recupera os objetos que eventualmente estão armazenados no banco de dados **SQLCompact** e envia para a fila de sincronização. Esse método é executado somente uma vez na inicialização do *provider*. Todo objeto recuperado nesse momento tem o status de *Salvaged*;
  + **Write**: criptografa o objeto recebido utilizando o método *Protect*, armazena no disco e envia para a fila;
  + **ConfirmEventSent**: remove determinado objeto do banco de dados. Deve ser chamado após a confirmação de sucesso a operação de escrita no repositório definitivo.
  + **NextEventToSend**: consome e retorna o próximo objeto da fila. Uma vez retornado o objeto é retirado da fila;
  + **Protect**: converte um objeto **AuditEntry** em um **SecureAuditEntry** utilizando a DPAPI do .Net. Os dados são criptografados só podem ser recuperados pelo usuário que executou a criptografia;
  + **Unprotect**: converte um objeto **SecureAuditEntry**em um**AuditEntry**;
* **DataAccess**: classe que possui os métodos para acesso físico ou banco de dados;
* **LocalSQLCompactProviderConfiguration**: classe responsável por definir a configuração do provider:
  + **SafeMode:** boolean – Se verdadeiro as operações de escrita no banco de dados são realizadas de modo síncrono o que causa uma redução na capacidade de armazenar eventos, porém existe a garantia que o dado foi salvo no disco antes de enviar para a fila. Se falso os eventos são gravado no disco à cada 10ms.
  + **RepositoryFullPath:** string – Path para o arquivo do banco de dados.
* **SecureAuditEntry**: Entidade de negócio que possui uma objeto do tipo **AuditEntry** criptografado.

Detalhe do arquivo de configuração:

<configSections>

<section name="localSQLCompactProviderConfiguration"

type="Br.Gov.Sp.Fazenda.Fw.Audit.LocalSQLCompactProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Audit.LocalSQLCompactProvider"/>

</configSections>

<localSQLCompactProviderConfiguration repositoryFullPath="c:\temp\data.sdf" safeMode="False" />

Listagem 9 Arquivo de configuração do provider LocalSQLCompactProvider (Audit)

#### SQLServerProvider

Tem a responsabilidade de gravar o evento de auditoria em um banco de dados **SQL Server**. Ao receber o objeto, a gravação é feita de maneira síncrona em uma tabela com *schema* pré-definido.

Por questões de segurança a inserção se dá através da execução de uma *stored procedure*, flexibilizando as possibilidades de permissionamento da tabela onde os dados serão efetivamente armazenados.

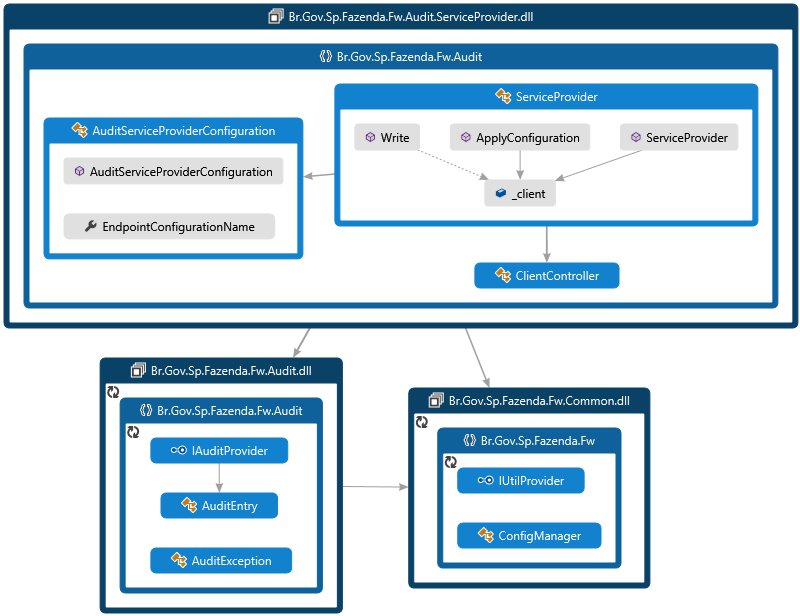


Figura 17 Arquitetura física do provider SQLServerProvider (Audit)

* **SQLServerProvider**: classe que implementa a interface **IAuditProvider** e possui as regras de negócio relativas ao funcionamento do provider:
  + **ApplyConfiguration**: método que termina a inicialização do *provider*. Faz a inicialização lendo as propriedade definidas em uma seção de configuração;
  + **Write**: executa a gravação do objeto no banco de dados através da execução de uma procedure;
* **DataAccess**: classe que possui os métodos para acesso físico ou banco de dados;
* **SQLServerProviderConfiguration**: classe responsável por definir a configuração do provider:
  + **ProcedureName**: string – nome da procedure de inserção a ser executada;
  + **ConnectionStringName**: string – nome da conexão com o banco onde a tabela e procedure foram previamente criados.

Detalhe do arquivo de configuração:

<configSections>

<section name="auditSQLServerProviderConfiguration"

type="Br.Gov.Sp.Fazenda.Fw.Audit.SQLServerProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Audit.SQLServerProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

</configSections>

<connectionStrings>

<add name="TesteDB" connectionString="Data Source=.\SQLExpress;Initial Catalog=Teste;Integrated Security=True;" providerName="System.Data.SqlClient"/>

</connectionStrings>

<auditSQLServerProviderConfiguration

procedureName="INSERT\_AUDIT"

connectionStringName="TesteDB"/>

Listagem 10 Arquivo de configuração do provider SQLServerProvider (Audit)

#### ServiceProvider

Tem a responsabilidade de interagir com o serviço disponibilizado pelo bloco **Audit** através da classe ***AuditService***. As classes proxy para a comunicação com o serviço já fazem parte do *provider*, bastando ajuste do WCF no arquivo de configuração. Não possui configuração própria, sendo apenas necessário configurar o cliente do WCF.

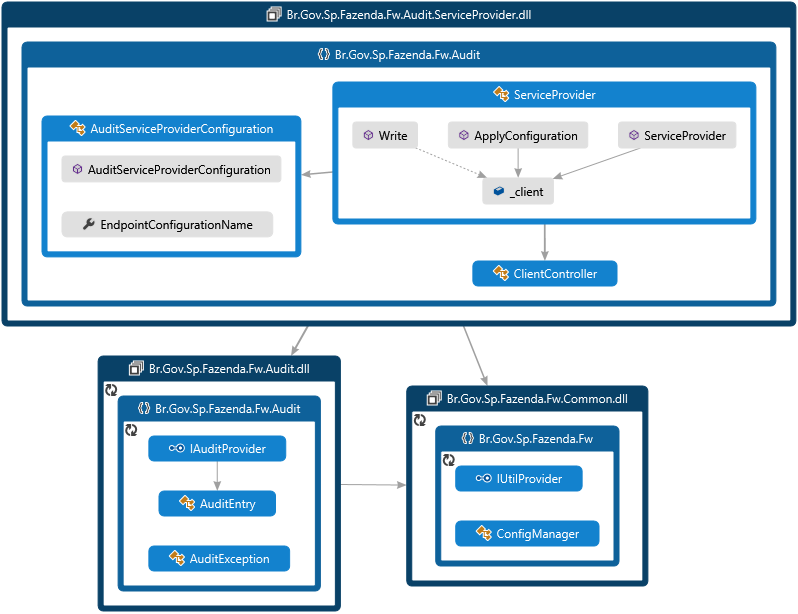


Figura 18 Arquitetura física do provider ServiceProvider (Audit)

* **ServiceProvider**: classe que implementa a interface **IAuditProvider** e possui as regras de negócio relativas ao funcionamento do provider:
  + **ApplyConfiguration**: método que termina a inicialização do *provider*. Faz a inicialização lendo as propriedade definidas em uma seção de configuração;
  + **Write**: obtém um cliente com canal de comunicação aberto do WCF e executa o método *write.*
* **ClientController**: classe responsável por gerenciar o ciclo de vida do canal de comunicação do WCF;
* **AuditServiceProviderConfiguration**: classe responsável por definir a configuração do provider:
  + **EndpointConfigurationName**: nome do endpoint que configura o serviço.

Detalhe do arquivo de configuração:

<configSections>

<section name="auditServiceProviderConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Audit.AuditServiceProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Audit.ServiceProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

</configSections>

<auditServiceProviderConfiguration endpointConfigurationName="AuditServiceConfig" />

Listagem 11 Arquivo de configuração do provider ServiceProvider (Audit)

* + 1. Cache

#### MemoryCacheProvider

Esse provider é baseado no mecanismo de cache em memória nativo do .net framework e permite o armazenamento de um objeto serializável na memória por um período de tempo pré-definido.

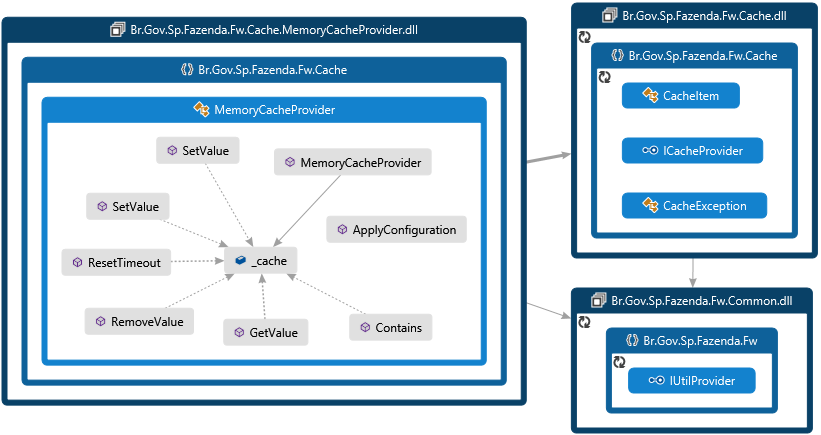


Figura 19 Arquitetura física do provider MemoryCacheProvider (Cache)

* **MemoryCacheProvider:** classe que implementa a interface **ICacheProvider** e possui as regras de negócio do *provider:*
  + **ApplyConfiguration**: método que termina a inicialização do *provider*. Faz a inicialização lendo as propriedade definidas em uma seção de configuração;
  + **Contains:** Verifica se existe um objeto em cache para a chave informada
  + **RemoveValue:** Remove um objeto em cache utilizando a chave informada
  + **GetValue:** Obtém um objeto em cache para utilizando a chave informada
  + **SetValue:** Atribui o valor especificado para a chave fornecida. Possui overloads que permitem definir o tempo de expiração absoluto ou relativo;
  + **ResetTimeout:** Atribui um novo período de duração para a chave informada

Esse *provider* não possui arquivo de configuração.

* + 1. Config

#### SQLServerProvider

Este provider oferece a capacidade de persistir um objeto serializado do .Net framework diretamente em um banco de dados **Microsoft SQL Server**.

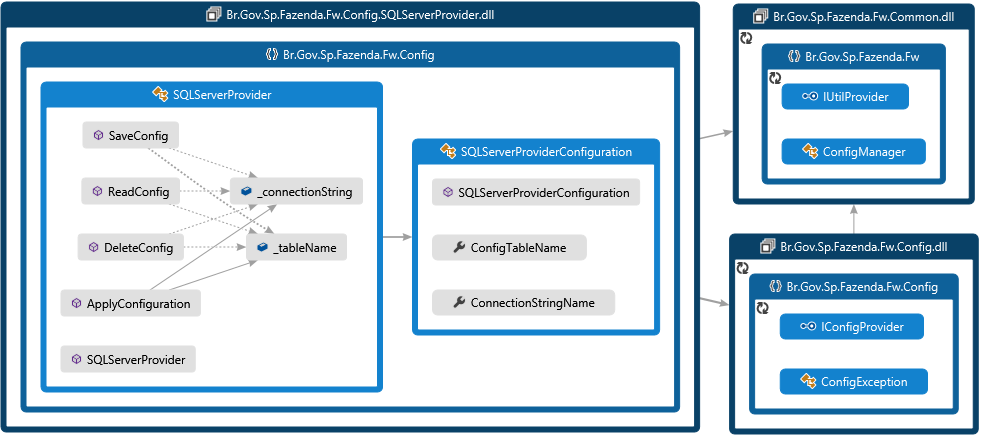


Figura 20 Arquitetura física do provider SQLServerProvider (Config)

* **SQLServerProvider**: classe que implementa a interface **IConfigProvider** oferece métodos de negócio para o provider:
  + **ApplyConfiguration**: método que termina a inicialização do *provider*. Faz a inicialização lendo as propriedade definidas em uma seção de configuração;
  + **ReadConfig:** recupera um objeto serializado usando uma chave de busca;
  + **SaveConfig:** salva um objeto serializado associando-o a uma chave de busca;
  + **DeleteConfig:** exclui um objeto serializado usando uma chave de busca;
* **SQLServerProviderConfiguration:** classe responsável por definir a configuração do *provider*:
  + **ConfigTableName:** propriedade do tipo string que armazena o nome da tabela a ser usado para persistência dos objetos de configuração.
  + **ConnectionStringName:** propriedade do tipo string que armazena o nome da conexão com o banco onde a tabela foi previamente criada.

Detalhe do arquivo de configuração:

<configSections>

<section name="cfgSQLServerProviderConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Config.SQLServerProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Config.SQLServerProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

</configSections>

<connectionStrings>

<add name="TesteDB" connectionString="Data Source=.\SQLExpress;Initial Catalog=Teste;Integrated Security=True;" providerName="System.Data.SqlClient"/>

</connectionStrings>

<cfgSQLServerProviderConfiguration

cfgTableName="Config" connectionStringName="TesteDB"/>

Listagem 12 Arquivo de configuração do provider SQLServerProvider (Config)

* + 1. Data Access

#### ADOProvider

Executa comandos **CRUD** (Create, Read, Update, Delete) numa base de dados usando o **Microsoft ADO.NET**. Este provider requer que seja informado o nome da *Stored Procedure* responsável pela execução de cada uma destas operações.

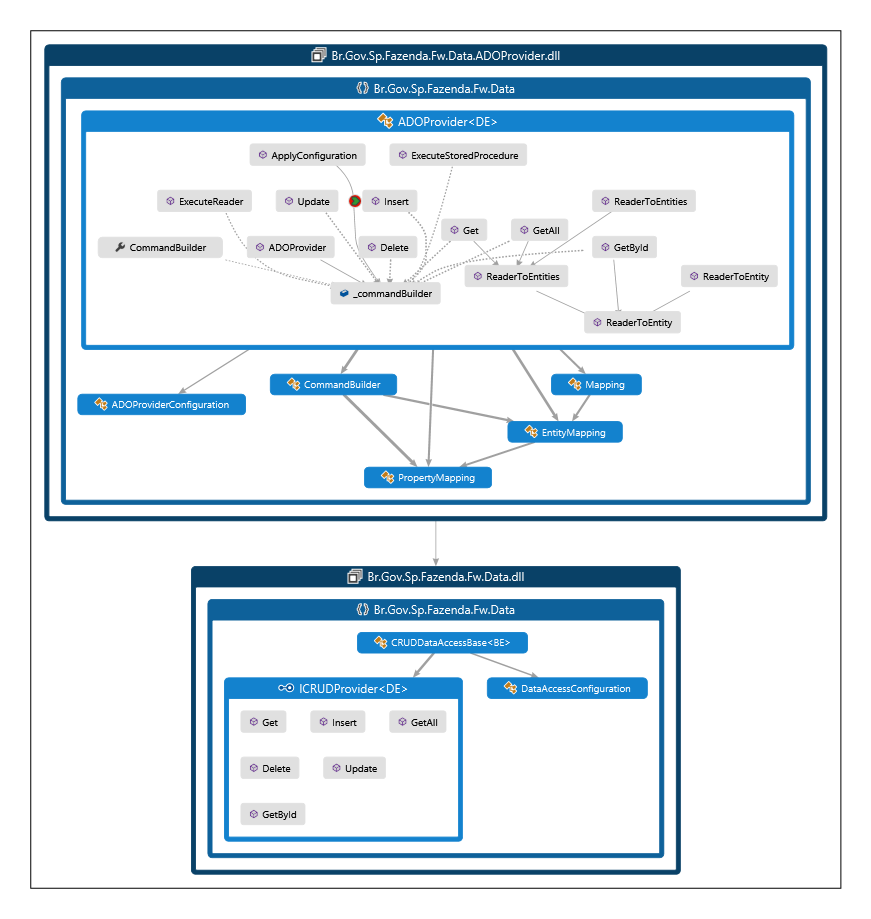


Figura 21 Arquitetura física do provider ADOProvider (Data Access)

* **ADOProvider<DE>**: Classe que implementa a interface **ICRUDProvider<DE>**:
  + **DE**: Entidade a ser manipulada pelo *provider* de dados;
  + **ApplyConfiguration**: Método que termina a inicialização do *provider*. Faz a inicialização lendo as propriedade definidas em uma seção de configuração;
  + **CommandBuilder**: Retorna a instância do tipo CommandBuilder cotendo métodos como:
    - **GetDbConnection:** Obtém o DbConnection retornado pela fábrica.
    - **GetDbCommand:** Obtém DbCommand retornado pela fábrica.
    - **GetDbParameter**: Obtém DbParameter retornado pela fábrica.
  + **ReaderToEntities**: Converte as linhas contidas no DbDataReader numa lista de entidades.
  + **ReaderToEntity**: Converte uma linha contida no DbDataReader numa entidade.
  + **Insert**: Insere entidades no repositório através de uma *Stored Procedure*.
  + **Update**: Atualiza entidades no repositório através de uma *Stored Procedure*.
  + **Delete**: Deleta entidades do repositório através de uma *Stored Procedure*.
  + **GetById**: Obtém uma entidade do repositório através de seu ID, usando uma *Stored Procedure*.
  + **Get**: Obtém um conjunto de entidades do repositório.
  + **GetAll**: Obtém todas as entidades do repositório.
  + **ExecuteStoredProcedure**: Executa uma *Stored Procedure* no repositório sem retorno de linhas.
  + **ExecuteReader**: Executa uma *Stored Procedure* no repositório com retorno de linhas.
* **ADOProviderConfiguration:** Classe responsável por definir a configuração do *provider*:
  + **ConnectionStringName:** nome da string de conexão existente no arquivo de configuração a ser usada para acessar o banco de dados.

Detalhe do arquivo de configuração:

<configSections>

<section name="dataADOProviderConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Data.ADOProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Data.ADOProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

</configSections>

<connectionStrings>

<add name="ADONorthwind" providerName="System.Data.SqlClient" connectionString="Data Source=(LocalDB)\v11.0;AttachDbFilename=E:\NORTHWND.MDF;integrated security=True;MultipleActiveResultSets=True" />

</connectionStrings>

<dataADOProviderConfiguration connectionStringName="ADONorthwind" />

<system.data>

<DbProviderFactories>

<add name="SqlClient Data Provider" invariant="System.Data.SqlClient" description=".Net Framework Data Provider for SqlServer" type="System.Data.SqlClient.SqlClientFactory, System.Data, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />

</DbProviderFactories>

</system.data>

Listagem 13 Arquivo de configuração do provider ADOProvider (Data Access)

**Mapeamento das entidades:**

Para poder mapear entre as entidades e as tabelas/*stored procedures*, e entre as propriedades e as colunas/parâmetros, são necessários metadados de mapeamento. O *provider* **ADOProvider** usa suas próprias classes de mapeamento:

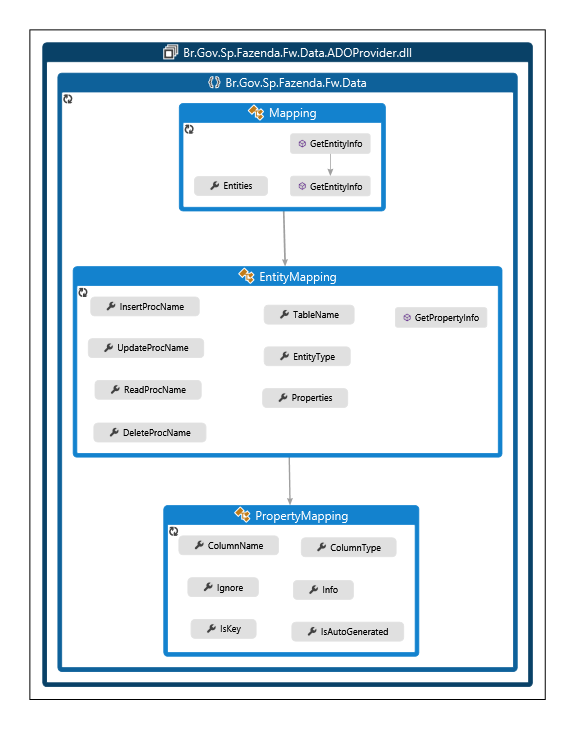


Figura 22 Classes para mapeamento usado pelo provider ADOProvider (Data Access)

* **Mapping**: classe que mantém as informações de mapeamento de todas as entidades:
  + **GetEntityInfo**: Retorna uma instância de **EntityMapping** com as informações de mapeamento de uma determinada entidade;
* **EntityMapping**: Classe que mantém as informações de mapeamento de uma entidade:
  + **GetPropertyInfo**: Retorna uma instância de **PropertyMapping** com as informações de mapeamento de uma determinada propriedade;
  + **TableName**: Nome da tabela à qual a entidade está mapeada;
  + **InsertProcName**: Nome da *stored procedure* para executar uma inclusão da entidade mapeada;
  + **UpdateProcName**: Nome da *stored procedure* para executar uma atualização da entidade mapeada;
  + **DeleteProcName**: Nome da *stored procedure* para executar uma exclusão da entidade mapeada;
  + **ReadProcName**: Nome da *stored procedure* para executar uma leitura da entidade mapeada;
* **PropertyMapping**: Classe que mantém as informações de mapeamento de uma propriedade:
  + **ColumnName**: Nome da coluna à qual a propriedade está mapeada;
  + **ColumnType**: Tipo da coluna à qual a propriedade está mapeada;
  + **IsKey**: Informa se a coluna faz parte da chave primária da tabela;
  + **IsAutoGenerated**: Informa se a coluna é gerada automaticamente pelo banco de dados;
  + **Ignore**: Informa se a propriedade deve ser desconsiderada ao mapear as colunas/parâmetros;

Além das propriedades, as classes **EntityMapping** e **PropertyMapping** possuem métodos que permitem uma configuração de suas propriedades no estilo ***fluent***.

#### SQLServerProvider

Executa comandos **CRUD** (Create, Read, Update, Delete) numa base de dados **SQL Server** usando o **SQLClient** do **Microsoft ADO.NET**. Este provider requer que seja informado o nome da *Stored Procedure* responsável pela execução de cada uma destas operações. Caso o nome da *Stored Procedure* não seja informado, o provider pode gerar dinamicamente os comandos necessários para as operações de **CRUD** da entidade específica.

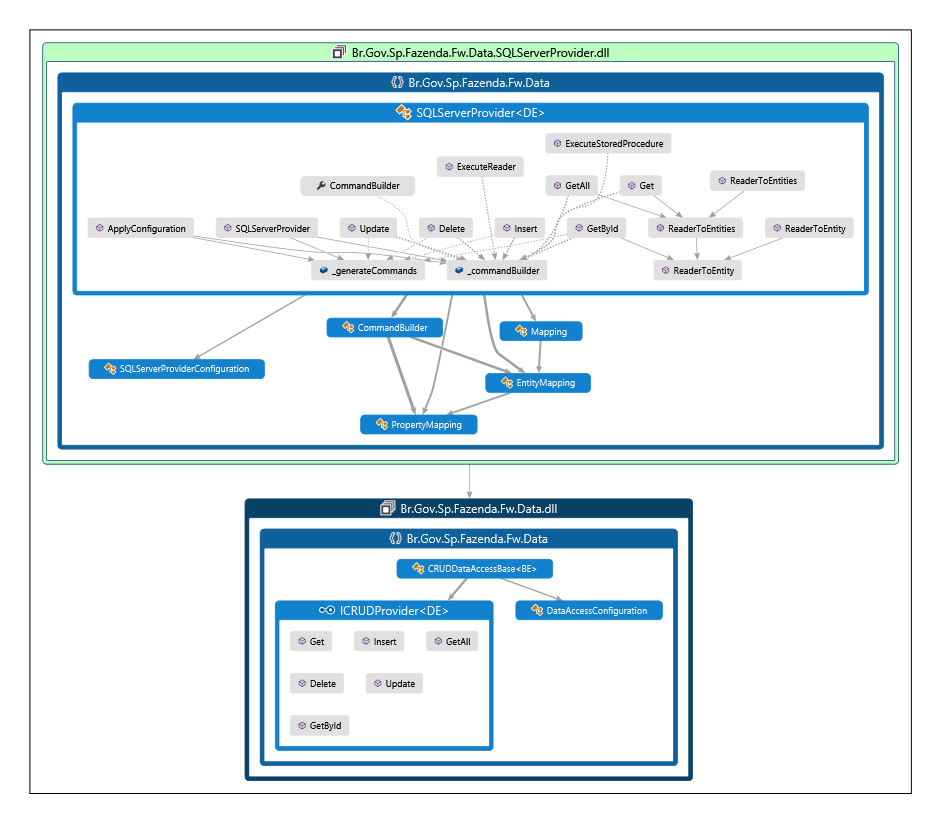


Figura 23 Arquitetura física do provider SQLServerProvider (Data Access)

* **SQLServerProvider<DE>**: Classe que implementa a interface **ICRUDProvider<DE>**:
  + **DE**: Entidade a ser manipulada pelo *provider* de dados;
  + **ApplyConfiguration**: Método que termina a inicialização do *provider*. Faz a inicialização lendo as propriedade definidas em uma seção de configuração;
  + **CommandBuilder**: Retorna a instância do tipo CommandBuilder cotendo métodos como:
    - **GetDbConnection:** Obtém o DbConnection retornado pela fábrica.
    - **GetDbCommand:** Obtém DbCommand retornado pela fábrica.
    - **GetDbParameter**: Obtém DbParameter retornado pela fábrica.
  + **ReaderToEntities**: Converte as linhas contidas no DbDataReader numa lista de entidades.
  + **ReaderToEntity**: Converte uma linha contida no DbDataReader numa entidade.
  + **Insert**: Insere entidades no repositório através de uma *Stored Procedure* ou do comando dinamicamente gerado.
  + **Update**: Atualiza entidades no repositório através de uma *Stored Procedure* ou do comando dinamicamente gerado.
  + **Delete**: Deleta entidades do repositório através de uma *Stored Procedure* ou do comando dinamicamente gerado.
  + **GetById**: Obtém uma entidade do repositório através de seu ID, usando uma *Stored Procedure* ou do comando dinamicamente gerado.
  + **Get**: Obtém um conjunto de entidades do repositório.
  + **GetAll**: Obtém todas as entidades do repositório.
  + **ExecuteStoredProcedure**: Executa uma *Stored Procedure* no repositório sem retorno de linhas.
  + **ExecuteReader**: Executa uma *Stored Procedure* no repositório com retorno de linhas.
* **ADOProviderConfiguration:** Classe responsável por definir a configuração do *provider*:
  + **ConnectionStringName:** Nome da string de conexão existente no arquivo de configuração a ser usada para acessar o banco de dados.
  + **GenerateCommands:** Indica se a geração dinâmica de comandos SQL para CRUD está habilitada.

Detalhe do arquivo de configuração:

<configSections>

<section name="dataSQLServerProviderConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Data.SQLServerProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Data.SQLServerProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

</configSections>

<connectionStrings>

<add name="ADONorthwind" providerName="System.Data.SqlClient" connectionString="Data Source=(LocalDB)\v11.0;AttachDbFilename=E:\NORTHWND.MDF;integrated security=True;MultipleActiveResultSets=True" />

</connectionStrings>

<dataSQLServerProviderConfiguration connectionStringName="ADONorthwind" generateCommands="true" />

<system.data>

<DbProviderFactories>

<add name="SqlClient Data Provider" invariant="System.Data.SqlClient" description=".Net Framework Data Provider for SqlServer" type="System.Data.SqlClient.SqlClientFactory, System.Data, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />

</DbProviderFactories>

</system.data>

Listagem 14 Arquivo de configuração do provider SQLServerProvider (Data Access)

**Mapeamento das entidades:**

Para poder mapear entre as entidades e as tabelas/*stored procedures*, e entre as propriedades e as colunas/parâmetros, são necessários metadados de mapeamento. O *provider* **SQLServerProvider** usa suas próprias classes de mapeamento:

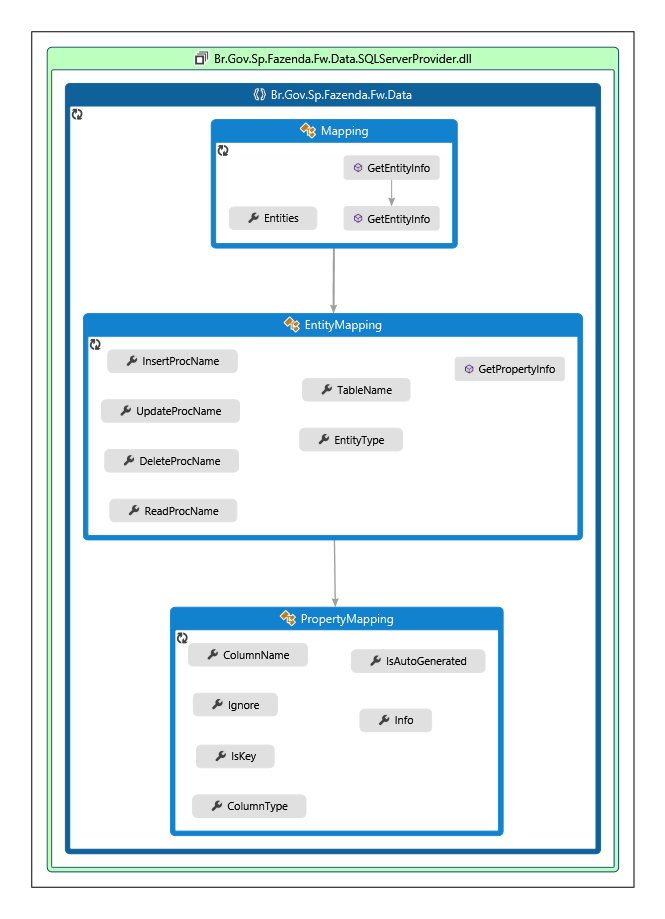


Figura 24 Classes para mapeamento usado pelo provider SQLServerProvider (Data Access)

* **Mapping**: Classe que mantém as informações de mapeamento de todas as entidades:
  + **GetEntityInfo**: Retorna uma instância de **EntityMapping** com as informações de mapeamento de uma determinada entidade;
* **EntityMapping**: Classe que mantém as informações de mapeamento de uma entidade:
  + **GetPropertyInfo**: Retorna uma instância de **PropertyMapping** com as informações de mapeamento de uma determinada propriedade;
  + **TableName**: Nome da tabela à qual a entidade está mapeada;
  + **InsertProcName**: Nome da *stored procedure* para executar uma inclusão da entidade mapeada;
  + **UpdateProcName**: Nome da *stored procedure* para executar uma atualização da entidade mapeada;
  + **DeleteProcName**: Nome da *stored procedure* para executar uma exclusão da entidade mapeada;
  + **ReadProcName**: Nome da *stored procedure* para executar uma leitura da entidade mapeada;
* **PropertyMapping**: Classe que mantém as informações de mapeamento de uma propriedade:
  + **ColumnName**: Nome da coluna à qual a propriedade está mapeada;
  + **ColumnType**: Tipo da coluna à qual a propriedade está mapeada;
  + **IsKey**: Informa se a coluna faz parte da chave primária da tabela;
  + **IsAutoGenerated**: Informa se a coluna é gerada automaticamente pelo banco de dados;
  + **Ignore**: Informa se a propriedade deve ser desconsiderada ao mapear as colunas/parâmetros;

Além das propriedades, as classes **EntityMapping** e **PropertyMapping** possuem métodos que permitem uma configuração de suas propriedades no estilo ***fluent***.

#### EFProvider

Executa comandos **CRUD** (Create, Read, Update, Delete) numa base de dados usando o **Microsoft Entity Framework 5.0**.

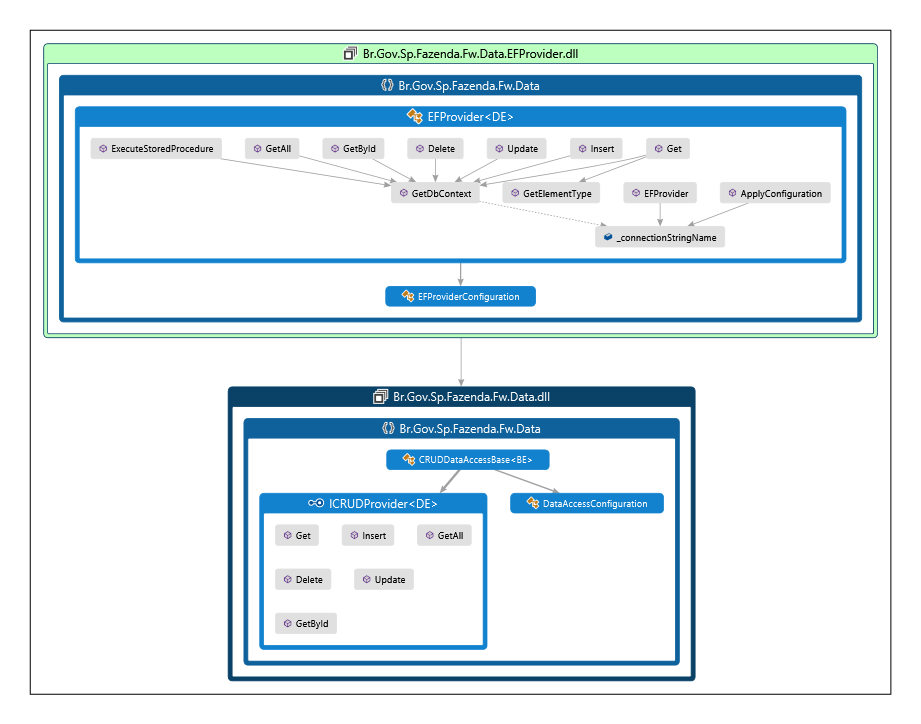


Figura 25 Arquitetura física do provider EFProvider (Data Access)

* **EFProvider<DE>**: Classe que implementa a interface **ICRUDProvider<DE>**:
  + **DE**: Entidade a ser manipulada pelo *provider* de dados;
  + **ApplyConfiguration**: Método que termina a inicialização do *provider*. Faz a inicialização lendo as propriedade definidas em uma seção de configuração;
  + **GetDbContext:** Obtém o DbContext associado à string de conexão informada no config.
  + **Insert**: Insere entidades no repositório.
  + **Update**: Atualiza entidades no repositório.
  + **Delete**: Deleta entidades do repositório.
  + **GetById**: Obtém uma entidade do repositório através de seu ID.
  + **Get**: Obtém um conjunto de entidades do repositório.
  + **GetAll**: Obtém todas as entidades do repositório.
  + **ExecuteStoredProcedure**: Executa uma *Stored Procedure* no repositório sem retorno de linhas.
* **EFProviderConfiguration:** Classe de configuração do *provider* do Entity Framework para Data Access:
  + **ConnectionStringName:** Nome da string de conexão existente no arquivo de configuração a ser usada para acessar o banco de dados.

Detalhe do arquivo de configuração:

<configSections>

<section name="dataEFProviderConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Data.EFProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Data.EFProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

</configSections>

<connectionStrings>

<add name="EFNorthwind" providerName="System.Data.EntityClient" connectionString="metadata=res://\*/Model.Northwind.csdl|res://\*/Model.Northwind.ssdl|res://\*/Model.Northwind.msl;provider=System.Data.SqlClient;provider connection string=&quot;Data Source=(LocalDB)\v11.0;AttachDbFilename=E:\NORTHWND.MDF;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework&quot;" />

</connectionStrings>

<dataEFProviderConfiguration connectionStringName="EFNorthwind" />

Listagem 15 Arquivo de configuração do provider EFProvider (Data Access)

* + 1. Log

Para mostrar o funcionamento do bloco de log foram criados os seguintes *providers*:

Todos os *providers* do bloco log geram uma entrada de erro no event viewer local do servidor, caso ocorra um erro na gravação.

#### SQLServerProvider

Tem a responsabilidade de gravar o evento de log em um banco de dados SQL Server. Ao receber o objeto, a gravação é feita de maneira síncrona em uma tabela com *schema* pré-definido.

Por questões de segurança a inserção se dá através da execução de uma procedure, flexibilizando as possibilidades de permissionamento da tabela onde os dados serão efetivamente armazenados.

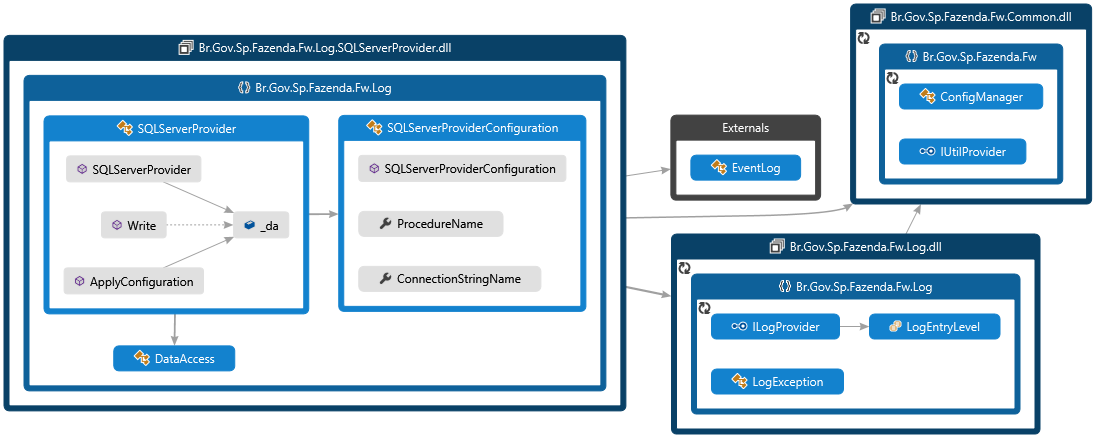


Figura 26 Arquitetura física do provider SQLServerProvider (Log)

* **SQLServerProvider**: classe que implementa a interface **ILogProvider** e possui as regras de negócio relativas ao funcionamento do *provider*:
  + **ApplyConfiguration**: método que termina a inicialização do *provider*. Faz a inicialização lendo as propriedade definidas em uma seção de configuração;
  + **Write**: executa a gravação dos dados no banco através da execução de uma procedure;
* **DataAccess**: classe que possui os métodos para acesso físico ao banco de dados;
* **SQLServerProviderConfiguration**: classe responsável por definir a configuração do provider:
  + **ProcedureName**: propriedade do tipo string que armazena o nome da procedure de inserção a ser executada;
  + **ConnectionStringName**: propriedade do tipo string que armazena o nome da conexão com o banco onde a tabela e procedure foram previamente criados.

Detalhe do arquivo de configuração:

<configSections>

<section name="logSQLServerProviderConfiguration"

type="Br.Gov.Sp.Fazenda.Fw.Log.SQLServerProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Log.SQLServerProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

</configSections>

<connectionStrings>

<add name="TesteDB" connectionString="Data Source=.\SQLExpress;Initial Catalog=Teste;Integrated Security=True;" providerName="System.Data.SqlClient"/>

</connectionStrings>

<logSQLServerProviderConfiguration

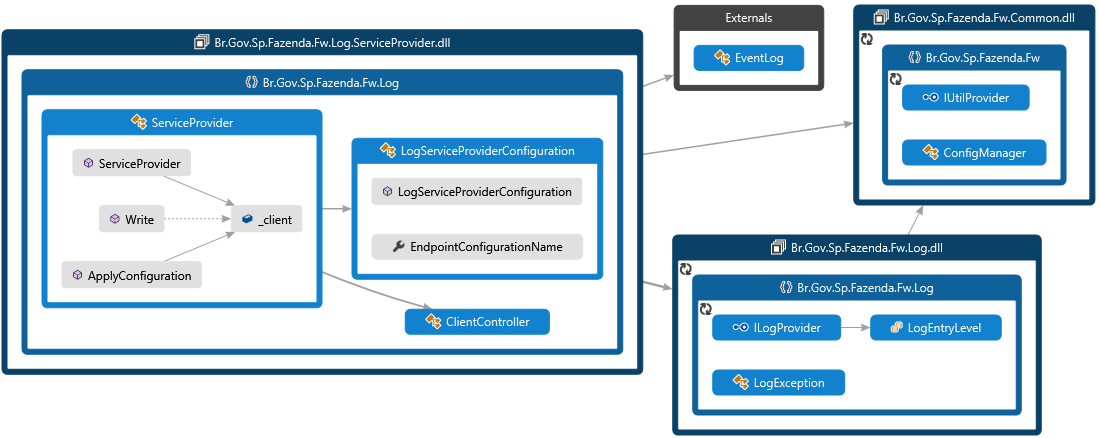
procedureName="INSERT\_LOG"

connectionStringName="TesteDB"/>

Listagem 16 Arquivo de configuração do provider SQLServerProvider (Log)

#### ServiceProvider

Tem a responsabilidade de interagir com o serviço disponibilizado pelo bloco **Log** através da classe **LogService**. As classes proxy para a comunicação com o serviço já fazem parte do *provider*, bastando ajuste do WCF no arquivo de configuração. Não possui configuração própria, sendo apenas necessário configurar o cliente do WCF.



Listagem 17 Arquitetura física do provider ServiceProvider (Log)

* **ServiceProvider**: classe que implementa a interface **ILogProvider** e possui as regras de negócio relativas ao funcionamento do *provider*:
  + **ApplyConfiguration**: método que termina a inicialização do *provider*. Faz a inicialização lendo as propriedade definidas em uma seção de configuração;
  + **Write**: utiliza a instância do **ClientController** com canal de comunicação aberto do WCF e executa o método *write.*
* **ClientController**: classe responsável por gerenciar o ciclo de vida do canal de comunicação do WCF;
* **LogServiceProviderConfiguration**: classe responsável por definir a configuração do provider:
  + **EndpointConfigurationName**: nome do endpoint que configura o serviço.

#### FileLogProvider

Tem a responsabilidade de gravar o evento de log em um conjunto de arquivos texto. Deve-se definir o tamanho máximo do log e quantidade de arquivos a serem gerados. Quando o tamanho ultrapassa o máximo, o arquivos mais antigos são apagados.

A verificação do tamanho dos arquivos é feita por uma *thread* de limpeza que roda à cada **500ms**, portanto o tamanho máximo é uma aproximação.

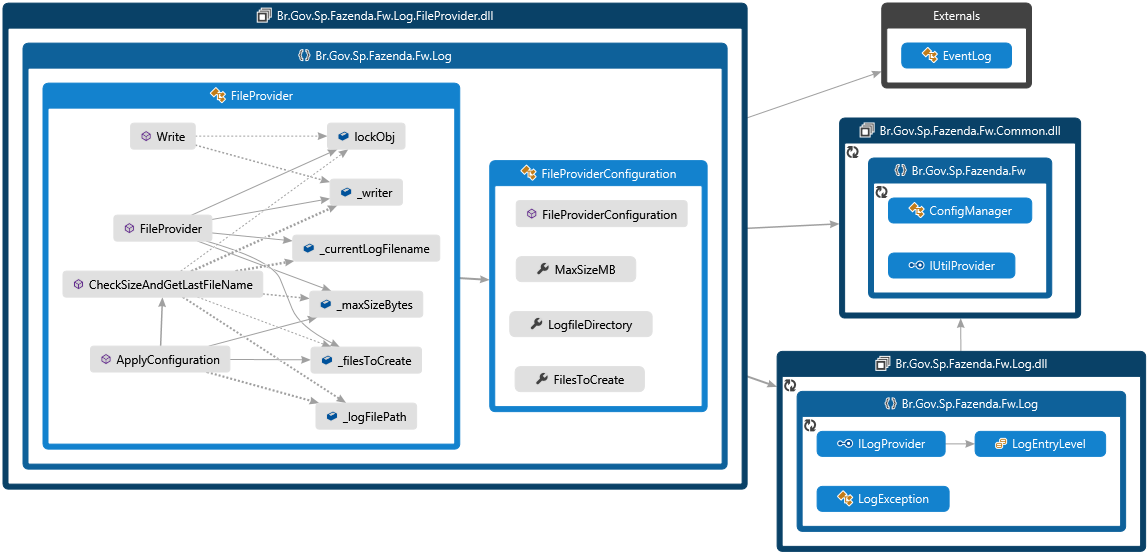


Figura 27 Arquitetura física do provider FileLogProvider

* **FileLogProvider:** classe que implementa a interface **ILogProvider** e possui as regras de negócio relativas ao funcionamento do *provider*:
  + **ApplyConfiguration**: método que termina a inicialização do *provider*. Faz a inicialização lendo as propriedade definidas em uma seção de configuração;
  + **Write:**.Escreve a entrada de log no arquivo utilizando um stream que fica permanentemente aberto;
  + **CheckSizeAndGetLastFileName:**.Verifica o tamanho, cria ou apaga arquivos do conjunto. Caso necessário abre um novo stream;
* **FileLogProviderConfiguration:** classe responsável por definir a configuração do provider:
  + **MaxSizeMB:** int – tamanho máximo do conjunto de arquivos em MB;
  + **FilesToCreate:** int – quantidade de arquivos a serem criados. O tamanho de cada arquivo será aproximadamente MaxSizeMB / FilesToCreate;
  + **LogfileDirectory:** string – diretório onde serão gravados os arquivos

Detalhe do arquivo de configuração:

<configSections>

<section name="logFileProviderConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Log.FileProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Log.FileProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

</configSections>

<logFileProviderConfiguration logFileDirectory="c:\temp\" maxSizeMB="2" filesToCreate="5" />

Listagem 18 Arquivo de configuração do provider FileLogProvider

#### ConsoleOutputProvider

Tem a responsabilidade de escrever o evento de log diretamente em uma tela de console.

Não possui configuração específica.

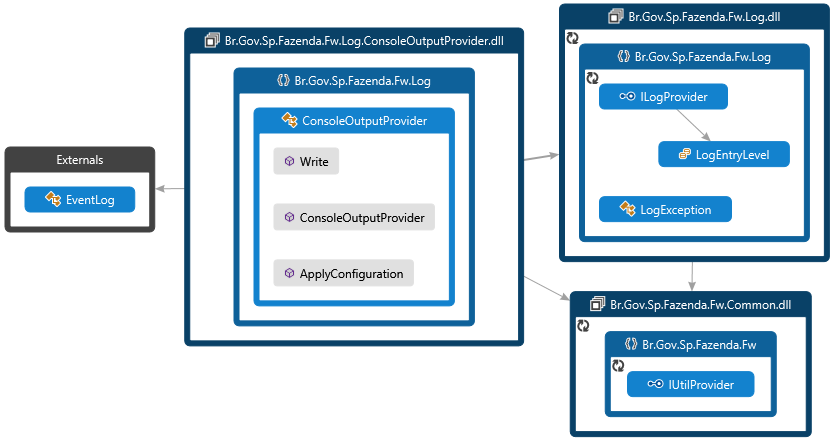


Figura 28 Arquitetura física do provider ConsoleOutputProvider

* **ConsoleOutputProvider:** classe que implementa a interface **ILogProvider** e possui as regras de negócio relativas ao funcionamento do *provider*:
  + **ApplyConfiguration**: método que termina a inicialização do *provider*. Faz a inicialização lendo as propriedade definidas em uma seção de configuração;
  + **Write:**.Prepara os dados em um formato amigável para a tela e envia para o console.
    1. Security

Disponibiliza dois *providers* com as seguintes funcionalidades:

* **CertificateProvider:** Manipulação de certificados digitais **x509**.
* **SignatureProvider:** Assinatura digital baseada em **PKCS#7** usando bibliotecas nativas do .net framework.

#### CertificateProvider

Tem a responsabilidade de encapsular as funcionalidades nativas do .Net de manipulação de certificados.

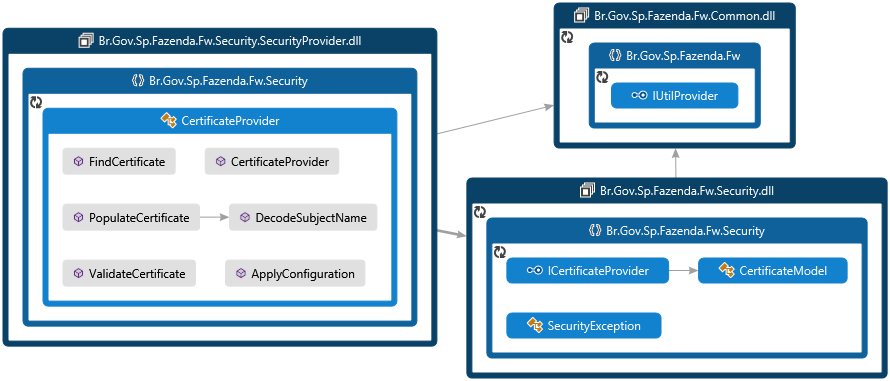


Figura 29 Arquitetura física do CertificateProvider

* **CertificateProvider:** classe que implementa a interface *ICertificateProvider* para manipulação de certificados digitais x509
  + **ApplyConfiguration**: método que termina a inicialização do *provider*. Faz a inicialização lendo as propriedade definidas em uma seção de configuração;
  + **PopulateCertificate:** recebe um certificado em formato de string hexadecimal e preenche a classe CertificadoDigitalEntidade com as propriedades mais comuns do mesmo.
  + **ValidateCertificate:** recebe um certificado e políticas de validação. Este método realiza validações de:
* Data de validade
* Revogação
* OIDs
* Cadeia

#### SignatureProvider

Tem a responsabilidade de encapsular as funcionalidades nativas do .Net de manipulação de certificados.

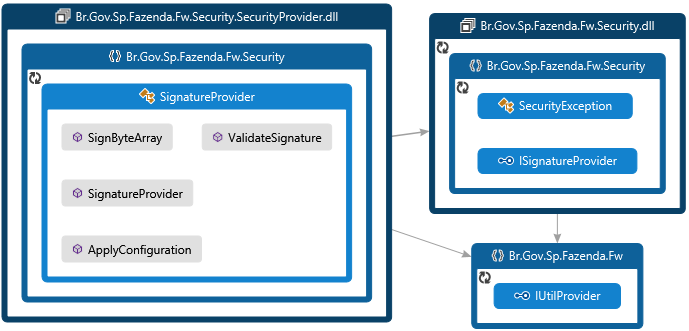


Figura 30 Arquitetura física do SignatureProvider

* **DigitalSignatureProvider:** classe que implementa a interface *ISignatureProvider* para manipulação de assinaturas digitais PCKS#7
  + **ApplyConfiguration**: método que termina a inicialização do *provider*. Faz a inicialização lendo as propriedade definidas em uma seção de configuração;
  + **SignByteArray:** recebe um array de bytes e um certificado digital para assinatura;
  + **ValidateSignature:** método booleano para realizar a validade de um documento assinado digitalmente.

1. Guia de Uso

Este tópico contém informações relativas ao uso do **Sefaz.Fw** como instalação e exemplos de utilização das suas principais funcionalidades.

* 1. Instalação

O **Sefaz.Fw** deve ser disponibilizado aos desenvolvedores como bibliotecas compiladas (sem o código-fonte) instaladas em suas máquinas. Em reunião com o time da **SEFAZ**, decidiu-se que os binários não serão armazenados no GAC. Dessa maneira o programa de instalação fornecido apenas copia os arquivos para o diretório: **%Program Files%\Secretaria da Fazenda\Sefaz.Fw\Assemblies** .

Os ***templates*** que fazem uso do **Sefaz.Fw** tem suas referências apontando para uma pasta local na ***solution*** e o *wizard* de criação dos projetos faz a cópia do diretório de instalação para essa pasta.

* + 1. Providers

Os *providers* fornecidos em conjunto com o **Sefaz.Fw** serão copiados para a pasta:

**%Program Files%\Secretaria da Fazenda\Sefaz.Fw\Assemblies\Providers** . Cada provider é armazenado em uma pasta separada com todas as suas dependências.

Durante a criação de uma ***solution*** a partir do *template* fornecido, os *providers* não serão copiados, pois nesse momento não se sabe quais serão utilizados.

Não existe a necessidade de fazer referência aos *providers* no projeto, pois eles são carregados dinamicamente pelos blocos no momento em que são utilizados, porém é necessários que eles sejam copiados para o diretório ***bin*** ou ***bin\providers*** da aplicação

* 1. Exemplos – How-to

Este tópico contém alguns exemplos de uso do **Sefaz.Fw** e seus blocos.

* + 1. Audit

#### Referências

Para utilizar o bloco, adicione uma referência para as bibliotecas: **Sefaz.Fw.Util.Audit** e **Sefaz.Fw.Common**.

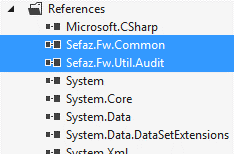


Figura 31 Referências para o uso do bloco Audit

#### Classe que herda de **Br.Gov.Sp.Fazenda.Fw.Audit.Audit**

Crie uma classe que herda de **Br.Gov.Sp.Fazenda.Fw.Audit.Audit**.O seu objetivo é:

* Encapsular todas as chamadas de auditoria feitas pela aplicação;
* Definir os nomes das categorias de auditoria. Para isso deve-se criar uma sub-classe e herdar da base **Br.Gov.Sp.Fazenda.Fw.Audit.Audit.AuditCategory**;
* Definir os nomes das operações (ou casos de uso). Para isso criar uma sub-classe e herdar da base **Br.Gov.Sp.Fazenda.Fw.Audit.Audit.AuditOperation***;*

class AuditExample : Audit

{

public AuditExample() : base("Example") { }

class CategoryExample : AuditCategory

{

public static readonly AuditCategory Cat1 = new AuditCategory("Categoria 1");

public static readonly AuditCategory Cat2 = new AuditCategory("Categoria 2");

public static readonly AuditCategory Cat3 = new AuditCategory("Categoria 3");

}

class OperationExample : AuditOperation

{

public static readonly AuditOperation Op1 = new AuditOperation("Operação 1");

public static readonly AuditOperation Caso1 = new AuditOperation("Caso de Uso 1");

}

public Guid InitBusinessOperation1(string userName, string ip)

{

return WriteRootEntry(CategoryExample.Cat1, OperationExample.Caso1, userName, ip, "Iniciando operação");

}

public Guid ContinueBusinessOperation1(Guid correlationId, string userName, string ip)

{

return WriteCorrelatedEntry(correlationId, CategoryExample.Cat2, OperationExample.Caso1, userName, ip, "Continuando operação");

}

public Guid EndBusinessOperation1(Guid correlationId, string userName, string ip)

{

return WriteCorrelatedEntry(correlationId, CategoryExample.Cat3, OperationExample.Caso1, userName, ip, "Finalizando operação");

}

}

Listagem 19 Classe AuditExample

A vantagem desta construção é que todas as chamadas ao bloco de auditoria se originam do mesmo local, tornando simples sua localização em qualquer parte do código.

A definição das sub-classes *CategoryExample* e *OperationExample* garantem uma padronização nos valores que serão armazenados nos repositórios de auditoria, uma vez que não podem ser passadas string livres para os métodos.

#### Config

As linhas no arquivo de configuração dependem dos *providers* utilizados. No arquivo abaixo, serão utilizados o **LocalSQLCompactProvider** e **SQLServerProvider**.

Seções de configuração:

<configSections>

<section name="auditConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Audit.AuditConfiguration, Br.Gov.Sp.Fazenda.Fw.Audit, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

<section name="auditLocalSQLCompactProviderConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Audit.LocalSQLCompactProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Audit.LocalSQLCompactProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

<section name="auditSQLServerProviderConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Audit.SQLServerProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Audit.SQLServerProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

</configSections>

<connectionStrings>

<add name="TesteDB" connectionString="Data Source=.\SQLExpress;Initial Catalog=SampleDB;Integrated Security=True;" providerName="System.Data.SqlClient"/>

</connectionStrings>

<auditConfiguration bypassLocalProvider="False"

persistentProviderType="Br.Gov.Sp.Fazenda.Fw.Audit.SQLServerProvider, Br.Gov.Sp.Fazenda.Fw.Audit.SQLServerProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"

localProviderType="Br.Gov.Sp.Fazenda.Fw.Audit.LocalSQLCompactProvider, Br.Gov.Sp.Fazenda.Fw.Audit.LocalSQLCompactProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d" />

<auditLocalSQLCompactProviderConfiguration repositoryFullPath="c:\temp\data.sdf" safeMode="False" />

<auditSQLServerProviderConfiguration

procedureName="INSERT\_AUDIT"

connectionStringName="TesteDB"/>

Listagem 20 Configuração do bloco Audit

A descrição de cada parâmetro está detalhada em um sub-tópico do item tópico 7.2 referente ao bloco ou *provider*

Utilizando o ServiceProvider:

O **ServiceProvider** foi construído para interagir com um serviço WCF disponibilizado pelo bloco **Audit**. Ele funciona como o cliente de um serviço que é fornecido diretamente pelo bloco.

Ele possui uma seção de configuração para definir o nome do *endpoint* que configura o serviço.

<configSections>

<section name="auditServiceProviderConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Audit.AuditServiceProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Audit.ServiceProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

</configSections>

<auditServiceProviderConfiguration endpointConfigurationName="httpAuditServiceConfig" />

Listagem 21 Configuração do ServiceProvider

Além disso, o *provider* precisa de uma seção que configura um cliente WCF:

<system.serviceModel>

<bindings>

<basicHttpBinding>

<binding name="httpNoSecurity">

<security mode="None" />

</binding>

</basicHttpBinding>

</bindings>

<client>

<endpoint address="http://localhost:53150/Audit.svc" binding="basicHttpBinding"

bindingConfiguration="httpNoSecurity" contract="Br.Gov.Sp.Fazenda.Fw.Audit.IAuditService"

name="httpAuditServiceConfig">

</endpoint>

</client>

</system.serviceModel>

Listagem 22 Configuração do WCF para o ServiceProvider

É importante lembrar que essa configuração precisa ser feita em sincronia com a configuração do serviço no bloco pois elas precisam ser compatíveis entre si. Os próximos tópicos irão mostrar como configurar o bloco para ativar o serviço.

Como dito anteriormente, o **ServiceProvider** funciona como um cliente. Para que ele possa funcionar corretamente, é necessário que um servidor esteja apto a receber os eventos enviados por ele. Cada bloco audit pode hospedar apenas uma instância de serviço e pode ser hospedado tanto em um executável quanto no IIS.

A configuração do bloco para ambos é a mesma e está exemplificada abaixo:

<configSections>

<section name="auditServiceConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Audit.AuditServiceConfiguration, Br.Gov.Sp.Fazenda.Fw.Audit, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

<section name="auditSQLServerProviderConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Audit.SQLServerProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Audit.SQLServerProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

</configSections>

<connectionStrings>

<add name="TesteDB" connectionString="Data Source=.\SQLExpress;Initial Catalog=SampleDB;Integrated Security=True;" providerName="System.Data.SqlClient"/>

</connectionStrings>

<auditServiceConfiguration

serviceProviderType="Br.Gov.Sp.Fazenda.Fw.Audit.SQLServerProvider, Br.Gov.Sp.Fazenda.Fw.Audit.SQLServerProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d" />

<auditSQLServerProviderConfiguration procedureName="INSERT\_AUDIT" connectionStringName="TesteDB"/>

Listagem 23 Configuração do serviço para o bloco Audit

Para a hospedagem em um executável, o WCF pode ser configurado da seguinte maneira:

<system.serviceModel>

<bindings>

<netTcpBinding>

<binding name="NoSecurity">

<security mode="None" />

</binding>

</netTcpBinding>

</bindings>

<behaviors>

<serviceBehaviors>

<behavior name="">

<serviceMetadata httpGetEnabled="false" />

<serviceThrottling maxConcurrentCalls="100" maxConcurrentSessions="300" maxConcurrentInstances="200" />

</behavior>

</serviceBehaviors>

</behaviors>

<services>

<service name="Br.Gov.Sp.Fazenda.Fw.Audit.AuditService">

<endpoint address="net.tcp://0.0.0.0:8500/AuditService" binding="netTcpBinding"

bindingConfiguration="NoSecurity" name="AuditServiceConfig" contract="Br.Gov.Sp.Fazenda.Fw.Audit.IAuditService" />

<endpoint address="net.tcp://0.0.0.0:8501/mex" binding="mexTcpBinding"

bindingConfiguration="" name="MetadataExchange" contract="IMetadataExchange" />

</service>

</services>

</system.serviceModel>

Listagem 24 Configuração do WCF para hospedagem no executável

A configuração acima cria um serviço WCF sendo suas principais características:

* Protocolo: net.tcp;
* Endereço: net.tcp://0.0.0.0:8500/AuditService  
  (O IP 0.0.0.0 indica que o serviço estará disponível em todos os IPs do servidor).
* Classe que define o serviço: Br.Gov.Sp.Fazenda.Fw.Audit.AuditService
* Interface do serviço: Br.Gov.Sp.Fazenda.Fw.Audit.IAuditService
* Os metadados do serviço estão expostos no endereço: net.tcp://0.0.0.0:8501/mex

Para a descrição e referência completa das configuração do WCF:

**Windows Communication Foundation Configuration Schema**  
<http://msdn.microsoft.com/en-us/library/vstudio/ms731734.aspx>

Iniciando o serviço em um executável:

Para iniciar o serviço, basta executar a seguinte linha de código:

Audit.StartAndHostService();

Listagem 25 Iniciando o serviço WCF hospedado no executável

Quando hospedado através do IIS, não é necessário código ou a criação de um arquivo “.svc”. O WCF possui um mecanismo chamado ***file-less activation****,* que cria um arquivo virtual utilizando somente o arquivo de configuração:

<system.serviceModel>

<bindings>

<basicHttpBinding>

<binding>

<security mode="None" />

</binding>

</basicHttpBinding>

</bindings>

<behaviors>

<serviceBehaviors>

<behavior name="">

<serviceMetadata httpGetEnabled="true" />

<serviceThrottling maxConcurrentCalls="100" maxConcurrentSessions="300" maxConcurrentInstances="200" />

</behavior>

</serviceBehaviors>

</behaviors>

<services>

<service name="Br.Gov.Sp.Fazenda.Fw.Audit.AuditService">

<endpoint binding="basicHttpBinding" bindingNamespace="" contract="Br.Gov.Sp.Fazenda.Fw.Audit.IAuditService"/>

</service>

</services>

<serviceHostingEnvironment>

<serviceActivations>

<add relativeAddress="./Audit.svc" service="Br.Gov.Sp.Fazenda.Fw.Audit.AuditService"/>

</serviceActivations>

</serviceHostingEnvironment>

</system.serviceModel>

Listagem 26 Configuração do WCF para hospedagem no IIS

A configuração acima cria um serviço WCF sendo suas principais características:

* Protocolo: HTTP;
* Endereço: relativo ao servidor web: ./Audit.svc  
  (O IP 0.0.0.0 indica que o serviço estará disponível em todos os IPs do servidor).
* Classe que define o serviço: Br.Gov.Sp.Fazenda.Fw.Audit.AuditService
* Interface do serviço: Br.Gov.Sp.Fazenda.Fw.Audit.IAuditService
* Os metadados do serviço estão expostos através de HTTP no mesmo endereço

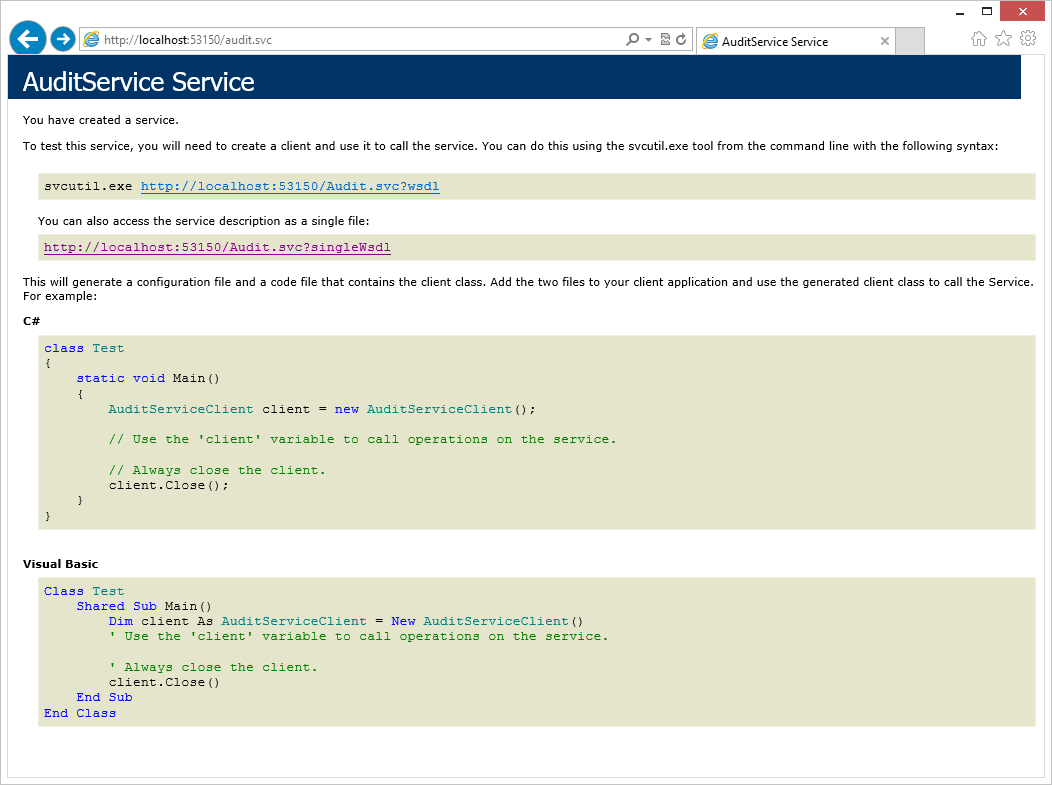


Figura 32 WCF hospedado no IIS

#### Exemplo de utilização

O exemplo abaixo mostra como criar uma instância da classe derivada do bloco **Audit** e chamar três métodos que realizam a auditoria:

AuditExample audit = new AuditExample();

string user = WindowsIdentity.GetCurrent().Name;

string ip = Dns.GetHostAddresses(Environment.MachineName).Where((i) => i.AddressFamily == System.Net.Sockets.AddressFamily.InterNetwork).FirstOrDefault().ToString();

Guid auditId = audit.InitBusinessOperation1(user, ip);

audit.ContinueBusinessOperation1(auditId, user, ip);

audit.EndBusinessOperation1(auditId, user, ip);

Listagem 27 Exemplo de uso do bloco Audit

Como o *provider* definitivo informado no arquivo de configuração é o **SQLServerProvider**, o destino final da auditoria é uma tabela no banco de dados:

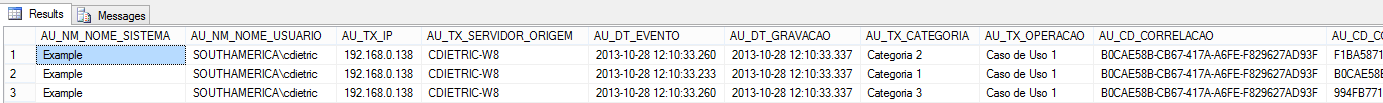


Figura 33 Eventos de auditoria armazenados em banco

* + 1. Cache

#### Referências

Para utilizar o bloco, adicione uma referência para as bibliotecas: **Sefaz.Fw.Util.Cache** e **Sefaz.Fw.Common**:

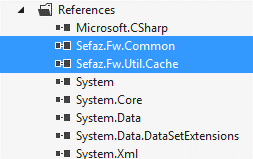


Figura 34 Referências para o uso do bloco Cache

#### Config

O trecho abaixo define a configuração do bloco **Cache** e seu *provider* (**MemoryCacheProvider**):

<configSections>

<section name="cacheConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Cache.CacheConfiguration, Br.Gov.Sp.Fazenda.Fw.Cache, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

</configSections>

<cacheConfiguration

providerType="Br.Gov.Sp.Fazenda.Fw.Cache.MemoryCacheProvider, Br.Gov.Sp.Fazenda.Fw.Cache.MemoryCacheProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d" />

Listagem 28 Configuração do bloco Cache

O *MemoryCacheProvider* não possui configuração.

A descrição de cada parâmetro está detalhada em um sub-tópico do item 7.2 referente ao bloco ou *provider*

#### Exemplo de utilização

O exemplo abaixo utiliza o bloco **Cache** para executar a seguinte sequência de operações:

1. Coloca um valor no cache com tempo de expiração relativo de 2s;
2. Verifica se o item foi inserido;
3. Aguarda 2s;
4. Verifica se o item ainda está no cache
5. Coloca um valor no cache com tempo de expiração absoluto 3s no futuro;
6. Verifica se o item foi inserido;
7. Aguarda 2s;
8. Verifica se o item ainda está no cache;
9. Coloca um valor no cache com tempo de expiração relativo de 2h;
10. Verifica se o item foi inserido;
11. Remove o item do cache antes da expiração;
12. Verifica se o item ainda está no cache

Cache cache = new Cache();

int number = 1;

TimeSpan timeout2Segundos = new TimeSpan(0,0,2);

Console.WriteLine("Armazenado valor no cache - Expiração em 2 segundos");

cache.SetValue("Numero", number, timeout2Segundos);

Console.WriteLine(String.Format("Valor em cache? - {0}", cache.Contains("Numero")));

if (cache.Contains("Numero"))

Console.WriteLine(String.Format("Valor da variável em cache é: {0}", cache.GetValue("Numero").ToString()));

Console.WriteLine("Aguardando expiração...");

Thread.Sleep(timeout2Segundos.Add(new TimeSpan(0,0,1)));

Console.WriteLine(String.Format("Valor em cache? - {0}", cache.Contains("Numero")));

Console.WriteLine();

DateTime expiration = DateTime.Now.AddSeconds(3);

Console.WriteLine("Armazenado valor no cache - Expiração: {0}", expiration.ToString("dd/MM/yyyy HH:mm:ss"));

cache.SetValue("Numero", number, expiration);

Console.WriteLine(String.Format("Valor em cache? - {0}", cache.Contains("Numero")));

if (cache.Contains("Numero"))

Console.WriteLine(String.Format("Valor da variável em cache é: {0}", cache.GetValue("Numero").ToString()));

Console.WriteLine("Aguardando expiração...");

Thread.Sleep(timeout2Segundos.Add(new TimeSpan(0, 0, 4)));

Console.WriteLine(String.Format("Valor em cache? - {0}", cache.Contains("Numero")));

Console.WriteLine();

TimeSpan timeout2Horas = new TimeSpan(2, 0, 0);

Console.WriteLine("Armazenado valor no cache - Expiração em 2 horas");

cache.SetValue("Numero", number, timeout2Horas);

Console.WriteLine(String.Format("Valor em cache? - {0}", cache.Contains("Numero")));

if (cache.Contains("Numero"))

Console.WriteLine(String.Format("Valor da variável em cache é: {0}", cache.GetValue("Numero").ToString()));

Console.WriteLine(String.Format("Limpando cache..."));

cache.RemoveValue("Numero");

Console.WriteLine(String.Format("Valor em cache? - {0}", cache.Contains("Numero")));

Console.WriteLine();

Console.ReadKey();

Listagem 29 Exemplo de uso do bloco Cache

**Saída no console:**

Armazenado valor no cache - Expiração em 2 segundos

Valor em cache? - True

Valor da variável em cache é: 1

Aguardando expiração...

Valor em cache? - False

Armazenado valor no cache - Expiração: 28/10/2013 13:18:41

Valor em cache? - True

Valor da variável em cache é: 1

Aguardando expiração...

Valor em cache? - False

Armazenado valor no cache - Expiração em 2 horas

Valor em cache? - True

Valor da variável em cache é: 1

Limpando cache...

Valor em cache? - False

* + 1. Config

#### Referências

Para utilizar o bloco, adicione uma referência para as bibliotecas: **Sefaz.Fw.Util.Config** e **Sefaz.Fw.Common**:

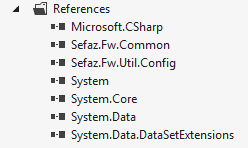


Figura 35 Referências para o uso do bloco Config

#### Config

O trecho abaixo define a configuração do bloco **Config**, seu *provider* (**SQLServerProvider**) e bloco dependente (**Cache**):

<configSections>

<section name="cacheConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Cache.CacheConfiguration, Br.Gov.Sp.Fazenda.Fw.Cache, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

<section name="cfgConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Config.ConfigConfiguration, Br.Gov.Sp.Fazenda.Fw.Config, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

<section name="cfgSQLServerProviderConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Config.SQLServerProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Config.SQLServerProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

</configSections>

<connectionStrings>

<add name="TesteDB" connectionString="Data Source=.\SQLExpress;Initial Catalog=SampleDB;Integrated Security=True;" providerName="System.Data.SqlClient"/>

</connectionStrings>

<cacheConfiguration providerType="Br.Gov.Sp.Fazenda.Fw.Cache.MemoryCacheProvider, Br.Gov.Sp.Fazenda.Fw.Cache.MemoryCacheProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d" />

<cfgConfiguration cacheTimeoutInMinutes="1"

cfgProviderType="Br.Gov.Sp.Fazenda.Fw.Config.SQLServerProvider, Br.Gov.Sp.Fazenda.Fw.Config.SQLServerProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d" cacheConfigSectionName="cacheConfiguration" />

<cfgSQLServerProviderConfiguration cfgTableName="Config" connectionStringName="TesteDB"/>

Listagem 30 Configuração do bloco Config

Importante observar que existe uma dependência entre os blocos **Config** e **Cache**, portando um bloco **Cache** também precisa ser configurado.

A descrição de cada parâmetro está detalhada em um sub-tópico do item 7.2 referente ao bloco ou *provider*

#### Exemplo de utilização

O exemplo abaixo utiliza o bloco **Config** para executar a seguinte sequência de operações:

1. Cria os objetos que serão guardados pelo bloco;
2. Grava o objeto em banco;
3. Efetua a leitura do objeto;
4. Altera uma configuração;
5. Efetua a leitura do objeto;
6. Remove uma configuração;
7. Tenta ler a configuração removida e recebe uma exceção.

ConfigExample configExample1 = new ConfigExample

{ Configuration1 = "Configuração A 1", Configuration2 = "Configuração A 2", Configuration3 = "Configuração A 3" };

ConfigExample configExample2 = new ConfigExample

{ Configuration1 = "Configuração B 1", Configuration2 = "Configuração B 2", Configuration3 = "Configuração B 3" };

Cfg.Config cfgExample = new Cfg.Config();

cfgExample.SaveConfig<ConfigExample>("Exemplo1", configExample1);

cfgExample.SaveConfig<ConfigExample>("Exemplo2", configExample2);

WriteObj(cfgExample.ReadConfig<ConfigExample>("Exemplo1"));

WriteObj(cfgExample.ReadConfig<ConfigExample>("Exemplo2"));

configExample1.Configuration1 = "Configuração A 1 - Atualizada";

cfgExample.SaveConfig<ConfigExample>("Exemplo1",configExample1);

WriteObj(cfgExample.ReadConfig<ConfigExample>("Exemplo1"));

cfgExample.DeleteConfig("Exemplo2");

WriteObj(cfgExample.ReadConfig<ConfigExample>("Exemplo2")); //Exception

Listagem 31 Exemplo de uso do bloco Config

Como o *provider* informado no arquivo de configuração foi o **SQLServerProvider**, os objetos de configuração são armazenados em uma tabela no banco de dados:

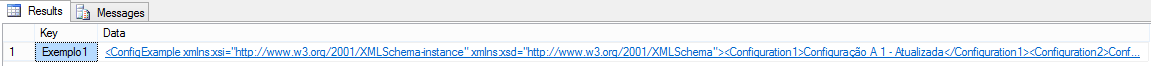


Figura 36 Objeto de configuração armazenado no banco de dados

* + 1. Exception Handling

#### Referências

Para utilizar o bloco, adicione uma referência para as bibliotecas: **Sefaz.Fw.Util.ExceptionHandling** e **Sefaz.Fw.Common**:

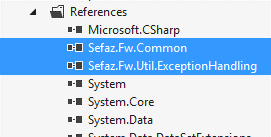


Figura 37 Referências para o uso do bloco ExceptionHandling

#### Config

O trecho abaixo define a configuração do bloco **ExceptionHandling** e bloco dependente (**Log**):

<configSections>

<section name="exceptionConfiguration" type="Br.Gov.Sp.Fazenda.Fw.ExceptionHandling.ExceptionConfiguration, Br.Gov.Sp.Fazenda.Fw.ExceptionHandling, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

<section name="logConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Log.LogConfiguration, Br.Gov.Sp.Fazenda.Fw.Log, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

<section name="logFileProviderConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Log.FileProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Log.FileProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

</configSections>

<exceptionConfiguration logConfigSection="logConfiguration" logStackTrace="true" defaultPolicyName="Rethrow" >

<policies>

<policy name="Suppress" exceptionAction="Suppress" policyIfNoMatch="Sanitize">

<appliesOnlyTo>

<exception type="System.ArgumentException" />

</appliesOnlyTo>

</policy>

<policy name="Rethrow" exceptionAction="Rethrow" />

<policy name="Sanitize" exceptionAction="Sanitize" alternateException="System.InvalidOperationException" message="Mensagem padrao para sanitizacao (originada em App.Config)" policyIfNoMatch="Rethrow" >

<appliesOnlyTo>

<exception type="System.ApplicationException" />

</appliesOnlyTo>

</policy>

</policies>

</exceptionConfiguration>

<logConfiguration providerType="Br.Gov.Sp.Fazenda.Fw.Log.FileProvider,Br.Gov.Sp.Fazenda.Fw.Log.FileProvider"

logLevel="Error"

traceLevelEnabled="false" />

<logFileProviderConfiguration logFileDirectory="c:\temp\"

maxSizeMB="2"

filesToCreate="5" />

Listagem 32 Configuração do bloco ExceptionHandling

#### Exemplo de utilização

O exemplo abaixo utiliza o bloco **ExceptionHandling** para executar a seguinte sequência de operações:

1. Cria o bloco de tratamento de exceções;
2. Trata uma exceção do tipo *ArgumentException* com a política “Suppress”;
3. Trata uma exceção do tipo *ApplicationException* com a política “Suppress”. Pelo arquivo de configuração essa política não se aplica a esse tipo de exceção que é roteada para a política “Sanitize”;
4. Trata uma exceção do tipo *ApplicationException* com a política “Sanitize”;
5. Trata uma exceção do tipo *ApplicationException* com a política “Rethrow” e recebe a exceção;

Todos os tratamentos der erro são logados pelo **FileProvider** conforme definido no arquivo de configuração.

Ex.ExceptionHandling exception = new Ex.ExceptionHandling();

try

{

throw new ArgumentException("Error 001");

}

catch (Exception ex)

{

if (exception.HandleException(ex, "Suppress"))

throw;

}

try

{

try

{

throw new ApplicationException("Error 002");

}

catch (Exception ex)

{

if (exception.HandleException(ex, "Suppress"))

throw;

}

}

catch (InvalidOperationException sanitizedException)

{

Console.WriteLine(sanitizedException.Message);

}

try

{

try

{

throw new ApplicationException("Error 003");

}

catch (Exception ex)

{

if (exception.HandleException(ex, "Sanitize"))

throw;

}

}

catch (InvalidOperationException sanitizedException)

{

Console.WriteLine(sanitizedException.Message);

}

try

{

throw new ApplicationException("Error 004");

}

catch (Exception ex)

{

if (exception.HandleException(ex, "Rethrow"))

throw;

}

Listagem 33 Exemplo de uso do bloco ExceptionHandling

**Resultado no arquivo criado pelo bloco Log:**

Level: Error

Date/Time: 28/10/2013 18:00:12

Server: CDIETRIC-W8

Severity: 0

Long Text:

--------------------------------

System.ArgumentException

Error 001

--------------------------------

--Stack Trace-------------------

at ExceptionHandling.Example.Program.Main(String[] args) in c:\TFS\SEFAZ\_ARQ\Sefaz.Fw2\Examples\ExceptionHandling\ExceptionHandling.Example\Program.cs:line 14

--------------------------------

Level: Error

Date/Time: 28/10/2013 18:00:12

Server: CDIETRIC-W8

Severity: 0

Long Text:

--------------------------------

System.ApplicationException

Error 002

--------------------------------

--Stack Trace-------------------

at ExceptionHandling.Example.Program.Main(String[] args) in c:\TFS\SEFAZ\_ARQ\Sefaz.Fw2\Examples\ExceptionHandling\ExceptionHandling.Example\Program.cs:line 26

--------------------------------

Level: Error

Date/Time: 28/10/2013 18:00:12

Server: CDIETRIC-W8

Severity: 0

Long Text:

--------------------------------

System.ApplicationException

Error 003

--------------------------------

--Stack Trace-------------------

at ExceptionHandling.Example.Program.Main(String[] args) in c:\TFS\SEFAZ\_ARQ\Sefaz.Fw2\Examples\ExceptionHandling\ExceptionHandling.Example\Program.cs:line 45

--------------------------------

Level: Error

Date/Time: 28/10/2013 18:00:12

Server: CDIETRIC-W8

Severity: 0

Long Text:

--------------------------------

System.ApplicationException

Error 004

--------------------------------

--Stack Trace-------------------

at ExceptionHandling.Example.Program.Main(String[] args) in c:\TFS\SEFAZ\_ARQ\Sefaz.Fw2\Examples\ExceptionHandling\ExceptionHandling.Example\Program.cs:line 60

--------------------------------

* + 1. Log

#### Referências

Para utilizar o bloco, adicione uma referência para as bibliotecas: **Sefaz.Fw.Util.Log** e **Sefaz.Fw.Common**.

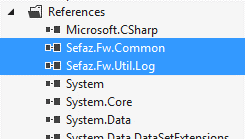


Figura 38 Referências para o uso do bloco Log

#### Classe que herda de Br.Gov.Sp.Fazenda.Fw.Log.Log

Crie uma classe que herda de **Br.Gov.Sp.Fazenda.Fw.Log.Log**. O seu objetivo é:

* Encapsular todas as chamadas de log feitas pela aplicação;
* Fornecer base para implementação de tratamentos específicos de eventos de log.

class LogExample : Log

{

public ExemploLog() : base("Example") { }

public void FileNotFound(string fileName)

{

WriteError(0, "Arquivo não encontrado", "O arquivo não pode ser encontrado no caminho especificado");

}

public void OperationComplete(string operationName)

{

WriteInformation(0, "Operação concluída", string.Format("A operação {0} foi concluída",operationName));

}

public void LowDiskSpace()

{

WriteWarning(0, "Disco com pouco espaço", "Disco com pouco espaço, liberar antes de realizar a operação");

}

}

Listagem 34 Classe LogExample

#### Config

As linhas no arquivo de configuração dependem dos *providers* utilizados. No arquivo abaixo, será utilizado o **FileProvider**.

Seções de configuração:

<configSections>

<section name="logConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Log.LogConfiguration, Br.Gov.Sp.Fazenda.Fw.Log"/>

<section name="logFileProviderConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Log.FileProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Log.FileProvider"/>

</configSections>

<logConfiguration providerType="Br.Gov.Sp.Fazenda.Fw.Log.FileProvider, Br.Gov.Sp.Fazenda.Fw.Log.FileProvider" logLevel="Warning" traceLevelEnabled="True" />

<logFileProviderConfiguration logFileDirectory="c:\temp\" maxSizeMB="20" filesToCreate="10" />

Listagem 35 Configuração do bloco Log

A descrição de cada parâmetro está detalhada em um sub-tópico do item tópico 7.2 referente ao bloco ou *provider*

Utilizando o ServiceProvider:

O **ServiceProvider** foi construído para interagir com um serviço WCF disponibilizado pelo bloco **Log**. Ele funciona como o cliente de um serviço que é fornecido diretamente pelo bloco.

Ele possui uma seção de configuração para definir o nome do *endpoint* que configura o serviço.

<configSections>

<section name="logServiceProviderConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Log.LogServiceProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Log.ServiceProvider"/>

</configSections>

<logServiceProviderConfiguration endpointConfigurationName="netTcpLogServiceConfig" />

Listagem 36 Configuração do ServiceProvider

Além disso, o *provider* precisa de uma seção que configura um cliente WCF:

<system.serviceModel>

<bindings>

<netTcpBinding>

<binding name="netTcpNoSecurity">

<security mode="None" />

</binding>

</netTcpBinding>

</bindings>

<client>

<endpoint address="net.tcp://localhost:8800/LogService" binding="netTcpBinding"

bindingConfiguration="netTcpNoSecurity" contract="Br.Gov.Sp.Fazenda.Fw.Log.ILogService"

name="netTcpLogServiceConfig">

</endpoint>

</client>

</system.serviceModel>

Listagem 37 Configuração do WCF para o ServiceProvider

É importante lembrar que essa configuração precisa ser feita em sincronia com a configuração do serviço no bloco pois elas precisam ser compatíveis entre si. Os próximos tópicos irão mostrar como configurar o bloco para ativar o serviço.

Como dito anteriormente, o **ServiceProvider** funciona como um cliente. Para que ele possa funcionar corretamente, é necessário que um servidor esteja apto a receber os eventos enviados por ele. Cada bloco **Log** pode disponibilizar apenas uma instância de serviço, que pode ser hospedado tanto em um executável quanto no IIS.

A configuração do bloco para ambos é a mesma e está exemplificada abaixo:

<configSections>

<section name="logServiceConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Log.LogServiceConfiguration, Br.Gov.Sp.Fazenda.Fw.Log, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

<section name="logSQLServerProviderConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Log.SQLServerProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Log.SQLServerProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

</configSections>

<connectionStrings>

<add name="TesteDB" connectionString="Data Source=.\SQLExpress;Initial Catalog=Teste;Integrated Security=True;" providerName="System.Data.SqlClient"/>

</connectionStrings>

<logServiceConfiguration serviceProviderType="Br.Gov.Sp.Fazenda.Fw.Log.SQLServerProvider, Br.Gov.Sp.Fazenda.Fw.Log.SQLServerProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d" />

<logSQLServerProviderConfiguration procedureName="INSERT\_LOG" connectionStringName="TesteDB"/>

Listagem 38 Configuração do serviço para o bloco Log

Para a hospedagem em um executável, o WCF pode ser configurado da seguinte maneira:

<system.serviceModel>

<bindings>

<netTcpBinding>

<binding>

<security mode="None" />

</binding>

</netTcpBinding>

</bindings>

<behaviors>

<serviceBehaviors>

<behavior name="logBehavior">

<serviceMetadata httpGetEnabled="false" />

<serviceThrottling maxConcurrentCalls="100" maxConcurrentSessions="300" maxConcurrentInstances="200" />

</behavior>

</serviceBehaviors>

</behaviors>

<services>

<service name="Br.Gov.Sp.Fazenda.Fw.Log.LogService" behaviorConfiguration="logBehavior">

<endpoint address="net.tcp://0.0.0.0:8800/LogService" binding="netTcpBinding" name="LogServiceConfig" contract="Br.Gov.Sp.Fazenda.Fw.Log.ILogService"/>

<endpoint address="net.tcp://0.0.0.0:8801/mex" binding="mexTcpBinding" name="MetadataExchange" contract="IMetadataExchange" />

</service>

</services>

</system.serviceModel>

Listagem 39 Configuração do WCF para hospedagem no executável

A configuração acima cria um serviço WCF sendo suas principais características:

* Protocolo: net.tcp;
* Endereço: net.tcp://0.0.0.0:8800/LogService  
  (O IP 0.0.0.0 indica que o serviço estará disponível em todos os IPs do servidor).
* Classe que define o serviço: Br.Gov.Sp.Fazenda.Fw.Log.LogService
* Interface do serviço: Br.Gov.Sp.Fazenda.Fw.Audit.ILogService
* Os metadados do serviço estão expostos no endereço: net.tcp://0.0.0.0:8501/mex

Para a descrição e referência completa das configuração do WCF:

Windows Communication Foundation Configuration Schema  
<http://msdn.microsoft.com/en-us/library/vstudio/ms731734.aspx>

Iniciando o serviço em um executável:

Para iniciar o serviço, basta executar a seguinte linha de código:

Log.StartAndHostService();

Listagem 40 Iniciando o serviço WCF hospedado no executável

Quando hospedado através do IIS, não é necessário código ou a criação de um arquivo “.svc”. O WCF possui um mecanismo chamado *file-less activation,* que cria um arquivo virtual utilizando somente o arquivo de configuração:

<system.serviceModel>

<bindings>

<basicHttpBinding>

<binding>

<security mode="None"/>

</binding>

</basicHttpBinding>

</bindings>

<behaviors>

<serviceBehaviors>

<behavior name="">

<serviceMetadata httpGetEnabled="true"/>

<serviceThrottling maxConcurrentCalls="100" maxConcurrentSessions="300" maxConcurrentInstances="200" />

</behavior>

</serviceBehaviors>

</behaviors>

<services>

<service name="Br.Gov.Sp.Fazenda.Fw.Log.LogService">

<endpoint binding="basicHttpBinding" contract="Br.Gov.Sp.Fazenda.Fw.Log.ILogService"/>

</service>

</services>

<serviceHostingEnvironment>

<serviceActivations>

<add relativeAddress="./Log.svc" service="Br.Gov.Sp.Fazenda.Fw.Log.LogService" factory=""/>

</serviceActivations>

</serviceHostingEnvironment>

</system.serviceModel>

Listagem 41 Configuração do WCF para hospedagem no IIS

A configuração acima cria um serviço WCF sendo suas principais características:

* Protocolo: HTTP;
* Endereço: relativo ao servidor web: ./Log.svc  
  (O IP 0.0.0.0 indica que o serviço estará disponível em todos os IPs do servidor).
* Classe que define o serviço: Br.Gov.Sp.Fazenda.Fw.Log.LogService
* Interface do serviço: Br.Gov.Sp.Fazenda.Fw.Audit.ILogService
* Os metadados do serviço estão expostos através de HTTP no mesmo endereço

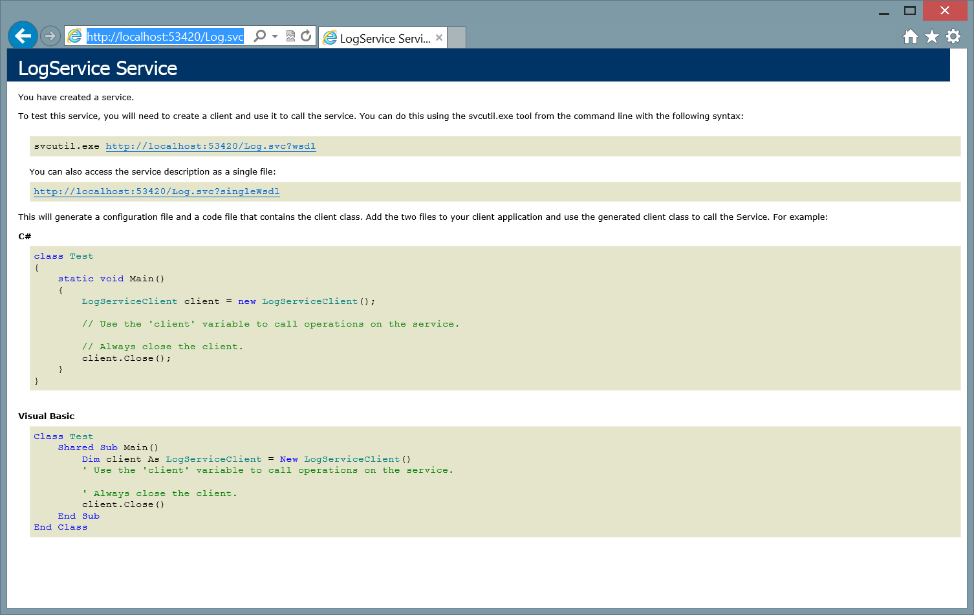


Figura 39 WCF hospedado no IIS

#### Exemplo de utilização

O exemplo abaixo mostra como criar uma instância da classe derivada do bloco **Log** e chamar três métodos que realizam o log. Como o bloco está configurado com o nível “Warning” somente as chamadas com o nível “Warning” e “Error” serão armazenadas:

ExemploLog log = new ExemploLog();

log.FileNotFound("C:\\temp\\dados.txt");

log.LowDiskSpace();

log.OperationComplete("Exemplo bloco log");

Listagem 42 Exemplo de uso do bloco Log

Como o *provider* informado no arquivo de configuração foi o **FileProvider**, o destino final do log é arquivo texto:

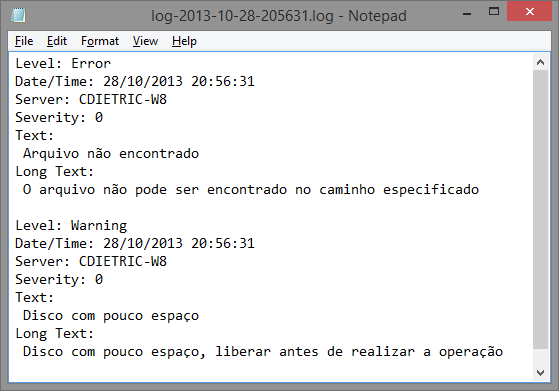


Figura 40 Log armazenado em arquivo

* + 1. Security

#### Referências

Para utilizar o bloco, adicione uma referência para as bibliotecas: **Sefaz.Fw.Util.Security** e **Sefaz.Fw.Common**:

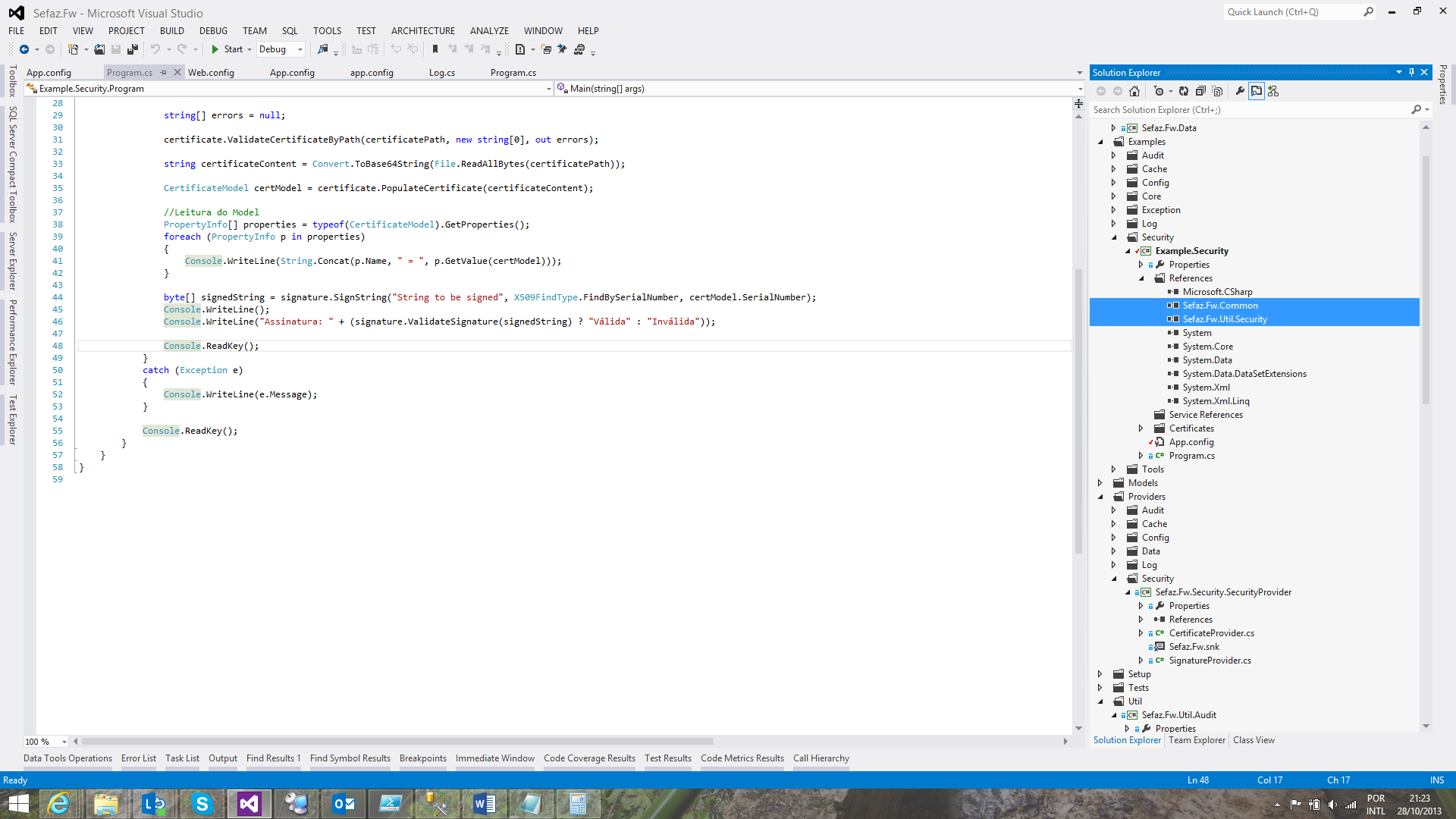


Figura 41 Referências para uso do bloco Security

#### Config

O trecho abaixo define a configuração do bloco **Security** e seus *providers* (**CertificateProvider**e **SignatureProvider**):

<configSections>

<section name="securityConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Security.SecurityConfiguration, Br.Gov.Sp.Fazenda.Fw.Security, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

</configSections>

<securityConfiguration

certificateProviderType = "Br.Gov.Sp.Fazenda.Fw.Security.CertificateProvider, Br.Gov.Sp.Fazenda.Fw.Security.SecurityProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"

signatureProviderType="Br.Gov.Sp.Fazenda.Fw.Security.SignatureProvider, Br.Gov.Sp.Fazenda.Fw.Security.SecurityProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d" />

Listagem 43 Configuração do bloco Security

#### Exemplo de Utilização

No exemplo a seguir, o bloco é utilizado para realizar seguinte sequência de operações:

1. Cria instâncias dos blocos Certifiate e Signature;
2. Verifica se o certificado de teste está armazenado na store “Trusted Root Certification Authorities”;
3. Valida um certificado armazenado em disco;
4. Cria um classe *CertificateModel* a partir de um certificado armazenado em disco;
5. Assina digitalmente uma string;
6. Valida a assinatura digital feita anteriormente.

Para o exemplo, é necessário que o certificado “Certificate - Password - SEFAZ.pfx” seja importado na “Trusted Root Certification Authorities” da máquina onde as operações de assinatura serão realizadas. Ele está localizado na pasta “Certificates” do projeto de exemplo e a senha para importação é SEFAZ.

string certificatePath = AppDomain.CurrentDomain.BaseDirectory + "\\Certificates\\Certificate.cer";

Certificate certificate = new Certificate();

Signature signature = new Signature();

if (certificate.FindCertificate(X509FindType.FindByThumbprint, "16C95C87D5BBEA4C6FA0F09761D534F20B599A14").Count == 0)

throw new Exception("The certificate necessary for this test is not installed on this machine.\nPlease import 'Certificate - Password - SEFAZ.pfx' on 'Trusted Root Certification Authorities'.\nThe certificate can be found on the project.");

string[] errors = null;

certificate.ValidateCertificateByPath(certificatePath, new string[0], out errors);

string certificateContent = Convert.ToBase64String(File.ReadAllBytes(certificatePath));

CertificateModel certModel = certificate.PopulateCertificate(certificateContent);

//Leitura do Model

PropertyInfo[] properties = typeof(CertificateModel).GetProperties();

foreach (PropertyInfo p in properties)

{

Console.WriteLine(String.Concat(p.Name, " = ", p.GetValue(certModel)));

}

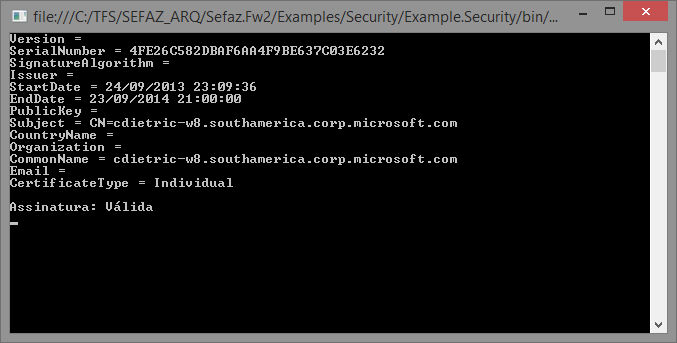
byte[] signedString = signature.SignString("String to be signed", X509FindType.FindBySerialNumber, certModel.SerialNumber);

Console.WriteLine();

Console.WriteLine("Assinatura: " + (signature.ValidateSignature(signedString) ? "Válida" : "Inválida"));

Listagem 44 Exemplo de uso do bloco Security

**Saída no console:**



* + 1. Data Access

#### Referências

Para utilizar o bloco, adicione uma referência para as bibliotecas: **Sefaz.Fw.Data** e **Sefaz.Fw.Common**:

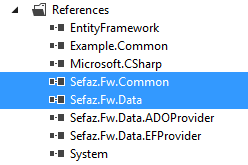


Figura 42 Referências para o uso do bloco Data Access

#### Classe que herda de **CRUDDataAccessBase<BE>**

Crie uma classe que herda de **CRUDDataAccessBase<BE>**, onde BE é a entidade de negócio. O seu objetivo é:

* Encapsular todas as chamadas relacionadas à persistência de **BE** feitas pela aplicação;
* Herdar funcionalidades para **CRUD** de **BE** já prontas;

public class DAOrder : CRUDDataAccessBase<Order>

{

public bool MyProc(int orderId)

{

EFProvider<Order> prov = (EFProvider<Order>)base.DAProvider;

int i = prov.ExecuteStoredProcedure("MyProc", new SqlParameter[] { new SqlParameter("OrderID", orderId) });

return (i > 0);

}

}

Listagem 45 Classe DAOrder

#### Config

O trecho abaixo define a configuração do bloco **Data Access** e seu *provider* (**EFProvider**):

<configSections>

<section name="dataAccessConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Data.DataAccessConfiguration, Br.Gov.Sp.Fazenda.Fw.Data, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

<section name="dataEFProviderConfiguration" type="Br.Gov.Sp.Fazenda.Fw.Data.EFProviderConfiguration, Br.Gov.Sp.Fazenda.Fw.Data.EFProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d"/>

</configSections>

<connectionStrings>

<add name="EFNorthwind" providerName="System.Data.EntityClient" connectionString="metadata=res://\*/Model.Northwind.csdl|res://\*/Model.Northwind.ssdl|res://\*/Model.Northwind.msl;provider=System.Data.SqlClient;provider connection string=&quot;Data Source=(LocalDB)\v11.0;AttachDbFilename=E:\NORTHWND.MDF;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework&quot;" />

</connectionStrings>

<dataAccessConfiguration providerType="Br.Gov.Sp.Fazenda.Fw.Data.EFProvider`1, Br.Gov.Sp.Fazenda.Fw.Data.EFProvider, Version=2.0.0.0, Culture=neutral, PublicKeyToken=6307dcc39942f68d" providerConfigSection="" />

<dataEFProviderConfiguration connectionStringName="EFNorthwind" />

</system.data>

Listagem 46 Configuração do bloco Data Access

A descrição de cada parâmetro está detalhada em um sub-tópico do item 7.2 referente ao bloco ou *provider*

#### Exemplo de utilização

O exemplo abaixo utiliza o bloco **Data Access** para executar a seguinte sequência de operações:

1. Cria uma instância de **DAOrder** (classe que herda de **CRUDDataAccessBase<Order>**);
2. Cria uma instância de **QueryParams<Order>** e configura que a ordenação de ser por OrderID ascendente e depois por CustomerID descendente, e deve recuperar a primeira página, onde o tamanho da página é 20;
3. Executa o método ***Get*** já existente na classe **CRUDDataAccessBase** para obter o resultado da consulta;

public QueryResult<Order> MyQuery()

{

QueryResult<Order> orders = null;

DAOrder da = new DAOrder();

QueryParams<Order> parameters = new QueryParams<Order>();

parameters.SortBy(o => o.OrderID).SortThenByDesc(i => i.CustomerID).GetPage(20, 0);

orders = da.Get(parameters);

return orders;

}

Listagem 47 Exemplo de uso do bloco Data Access

* + 1. Business

#### Referências

Para utilizar o bloco, adicione uma referência para as bibliotecas: **Sefaz.Fw.Business**, **Sefaz.Fw.Data** e **Sefaz.Fw.Common**:

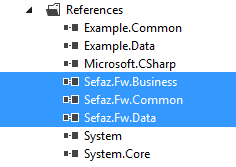


Figura 43 Referências para o uso do bloco Business

#### Classe que herda de **CRUDEntityManagerBase<BE, DA>**

Crie uma classe que herda de **CRUDEntityManagerBase<BE, DA>**, onde BE é a entidade de negócio e DA é a classe de acesso a dados. O seu objetivo é:

* Encapsular toda regra de negócio relacionada à **BE**;
* Herdar funcionalidades para **CRUD** de **BE** já prontas.

public class EMOrder : CRUDEntityManagerBase<Order, DAOrder>

{

public bool MyProc(int orderId)

{

return DataAccessComponent.MyProc(orderId);

}

public List<Order> MyQueryProc(DateTime startDate, DateTime endDate)

{

return DataAccessComponent.MyQueryProc(startDate, endDate);

}

}

Listagem 48 Classe EMOrder

#### Classe que herda de **BusinessProcessBase**

Crie uma classe que herda de **BusinessProcessBase**. O seu objetivo é:

* Ser a API da aplicação, contendo todos os métodos que estão disponíveis para a camada de apresentação e de serviços;
* Encapsular toda regra de negócio da aplicação. Orquestrar a regra de negócio contida em uma ou mais *Entity Managers*;
* Realizar o tratamento e log das exceções ;

public class BPOrder : BusinessProcessBase

{

private static Ex.ExceptionHandling \_exceptionHandler = new Ex.ExceptionHandling();

/// <summary>

/// Trata uma exceção ocorrida num método de um Business Process.

/// Internamente pode chamar o bloco ExceptionHandling ou implementar sua própria lógica de tratamento/log de exceções.

/// </summary>

/// <param name="ex">Exceção a ser tratada.</param>

/// <param name="policyName">Política a ser aplicada.</param>

/// <param name="message">Mensagem a ser logada.</param>

/// <returns>True -> Deve ser dado um rethrow.</returns>

protected override bool HandleException(Exception ex, string policyName = "", string message = null)

{

return \_exceptionHandler.HandleException(ex, policyName, message);

}

public QueryResult<Order> GetOrders(int pageSize, int pageNumber)

{

QueryResult<Order> orders = null;

try

{

EMOrder em = new EMOrder();

QueryParams<Order> parameters = new QueryParams<Order>();

parameters.SortBy(o => o.OrderID).SortThenByDesc(i => i.CustomerID).GetPage(pageSize, pageNumber);

orders = em.Get(parameters);

}

catch (Exception ex)

{

if (HandleException(ex)) throw;

}

return orders;

}

public Order CreateOrder(Order newOrder)

{

try

{

using (TransactionScope ts = new TransactionScope(TransactionScopeOption.Required))

{

EMOrder em = new EMOrder();

newOrder = em.Insert(newOrder);

ts.Complete();

}

}

catch (Exception ex)

{

if (HandleException(ex, "", "Error creating order.")) throw;

}

return newOrder;

}

public void UpdateOrder(Order order)

{

try

{

using (TransactionScope ts = new TransactionScope(TransactionScopeOption.Required))

{

EMOrder em = new EMOrder();

em.Update(order);

ts.Complete();

}

}

catch (Exception ex)

{

if (HandleException(ex, "", "Error updating order.")) throw;

}

}

public void DeleteOrder(Order order)

{

try

{

using (TransactionScope ts = new TransactionScope(TransactionScopeOption.Required))

{

EMOrder em = new EMOrder();

em.Delete(order);

ts.Complete();

}

}

catch (Exception ex)

{

if (HandleException(ex, "", "Error deleting order.")) throw;

}

}

public Order GetOrder(int orderId)

{

Order order = null;

try

{

EMOrder em = new EMOrder();

order = em.GetById(new Order() { OrderID = orderId });

}

catch (Exception ex)

{

if (HandleException(ex, "", "Error getting order by id.")) throw;

}

return order;

}

}

Listagem 49 Classe BPOrder

#### Config

O bloco **Business** não possui configuração.

* 1. Teste Integrado

Para exemplificar o uso de vários *providers*, e executar um teste integrado, foi criada uma solução partir do template de aplicação .Net.

Ela simula uma situação hipotética da criação de um pedido de compra e usa um teste unitário para representar a camada do sistema que está na DMZ. Esse desenho permite utilizar a infraestrutura do Visual Studio para a execução dos testes.

O projeto tem a seguinte arquitetura física:

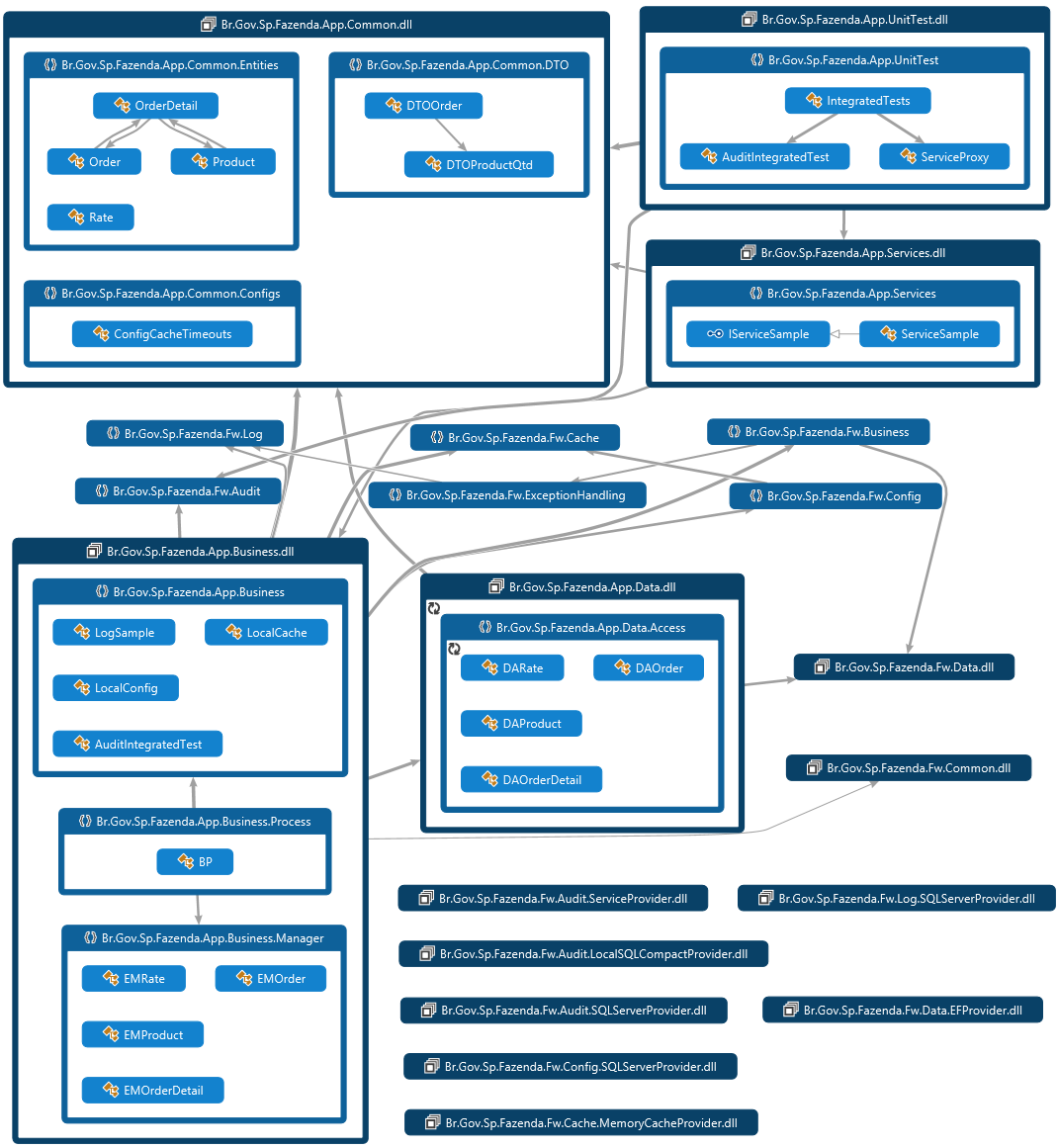


Figura 44 Arquitetura física do projeto de teste integrado

#### Infraestrutura utilizada

Para a execução dos testes, foi criada uma infraestrutura de servidores utilizando o IaaS disponível no Windows Azure com a seguinte configuração:



Configuração dos servidores:

* Agentes de teste: 2 servidores com 2 CPUs 3,5GB de memória;
* Web: 3 servidores com 1 CPU e 1,75 GB de memória;
* SQL: 1 servidor com 4 CPUs, 7GB de memória e 4 discos em RAID-0;
* Client: 1 servidor com 1 CPU e 1,75 GB de memória;
* Controller: 1 servidor com 1 CPU e 1,75 GB de memória;
* Active directory: 1 servidor com 1 CPU e 1,75 GB de memória;

#### Cenários de teste

Foram criados seis métodos de teste:

1. TestGetAllProducts: utiliza a camada de serviço para recuperar todos entidades do tipo Product. Ao ler todos os produtos, ele são colocados um em cache na memória;
2. TestGetProduct: utiliza a camada de serviço e recupera um produto. Registra (log) se a leitura foi feita através do cache ou banco;
3. TestGetRate: utiliza a camada de serviços o obtém uma taxa para conversão de um valor em dólares para reais. A cotação é válida por um minuto;
4. TestOrderProcessing: utiliza a camada de serviços e prepara uma ordem completa: lê os produtos, lê a taxa de conversão, escolhe os produtos e submete a ordem;
5. TestOrderProcessing: utiliza a camada de serviços e prepara uma ordem, porém aguarda 1 minuto para que a cotação expire. Espera receber uma exceção;
6. TestOrderProcessingWithAudit: utiliza o mesmo processo do método *TestOrderProcessing*, porém faz auditoria no lado do cliente além da auditoria no serviço;

Utilizando esses métodos, foi criado um teste de carga e um de longa duração.

O teste de carga utiliza um cenário *goal based* com um alvo de 70% de CPU nos servidores Web. O tempo de execução do teste é de 10 minutos.

O teste de longa duração utiliza uma carga constante de 20 usuários e foi executado por um período de 10h.

Cenário 1 – Utilizando somente 1 servidor web

Nesse cenário, as máquinas SEFAZ-WEB02 e SEFAZ-WEB03 foram desligadas e o teste iniciado.

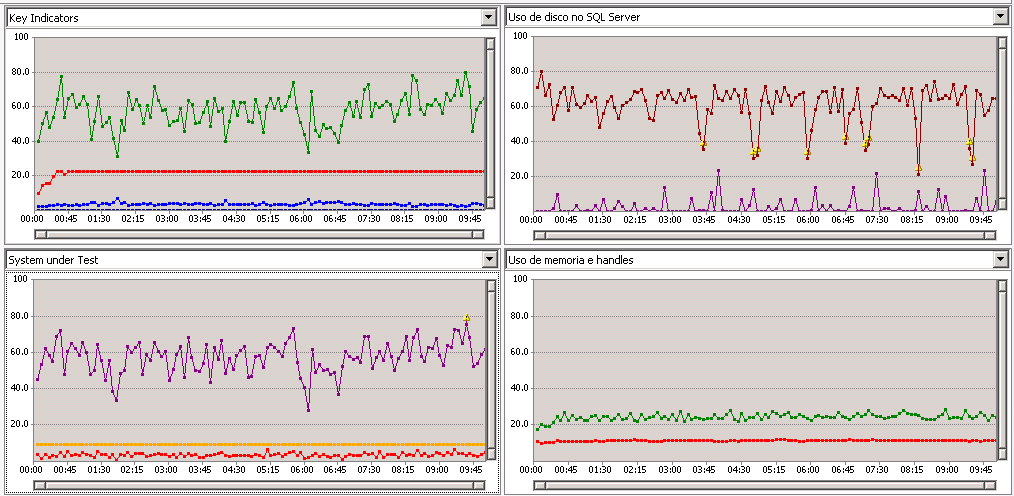
Após a execução os seguintes resultados foram obtidos:

* A quantidade de usuários estabilizou-se em 23 usuários simultâneos;
* Esses usuários criaram uma carga média de 55 testes/segundo; (pico: 99 testes/s)
* Durante os 10min foram executados aproximadamente 33.000 testes;
* O tempo médio de execução dos testes foi de 0,38s; (máx: 0,78s)
* O uso do disco no SQL Server foi moderado gerando alguns alertas;
* O uso de memória e handles se manteve estável não havendo indicação de vazamento desses recursos.

Quantidade de testes/s

% do tempo inativo do disco no SQL Server

Quantidade de usuários



CPU do servidor web

Fila de espera do disco no SQL Server

Tempo médio da execução dos testes

Uso de memória (private bytes) no servidor web

CPU do SQL Server

Quantidade de handles em uso no servidor web

Memória livre no servidor web

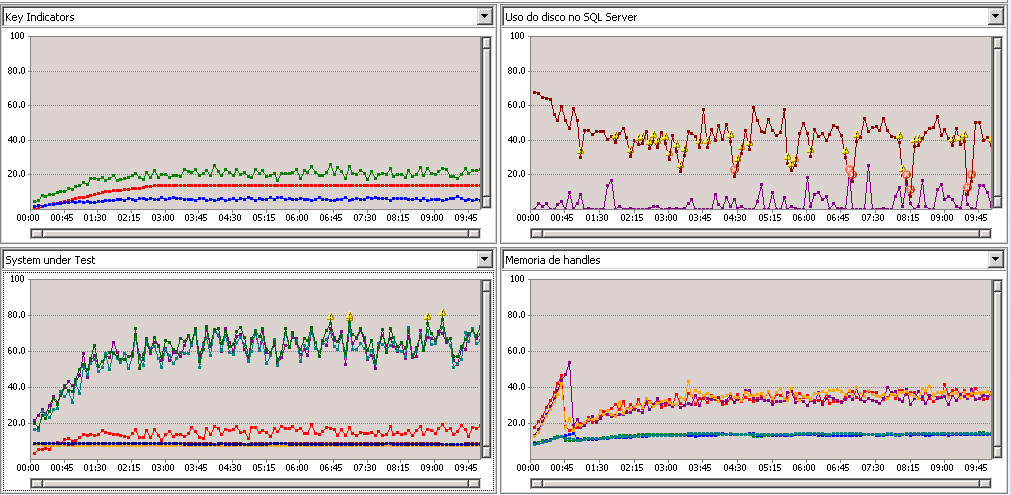
Cenário 2 – Utilizando 3 servidores Web

Após a execução os seguinte resultados foram obtidos:

* A quantidade de usuários estabilizou-se em 140 usuários simultâneos;
* Esses usuários criaram uma carga média de 197 testes/segundo; (pico: 263 testes/s)
* Durante os 10min foram executados aproximadamente 118.000 testes;
* O tempo médio de execução dos testes foi de 0,61s; (máx: 0,80s)
* O uso do disco no SQL Server foi intenso várias vezes gerando alertas;
* O uso de memória e handles se manteve estável não havendo indicação de vazamento desses recursos.

% do tempo inativo do disco no SQL Server

Quantidade de testes/s



Uso de memória (private bytes) nos servidores Web

CPU dos servidores web

Quantidade de usuários

Tempo médio da execução dos testes

Fila de espera do disco no SQL Server

Quantidade de handles em uso nos servidores web

CPU do SQL Server

Memória livre nos servidores web

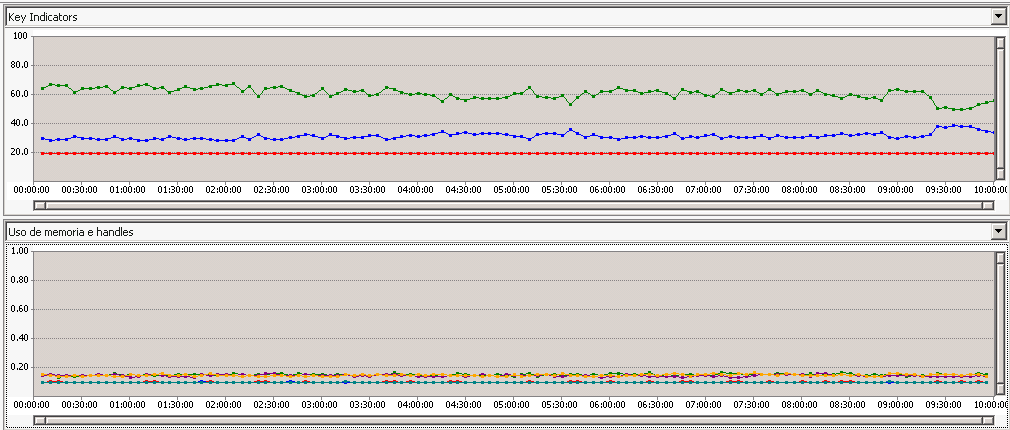
Cenário 3 – Utilizando 3 servidores Web – Longa duração

Após a execução os seguinte resultados foram obtidos:

* A quantidade de usuários foi de 20 usuários simultâneos;
* Esses usuários criaram uma carga média de 61 testes/segundo; (pico: 68 testes/s)
* Durante os 10h foram executados aproximadamente 2.200.000 testes;
* O tempo médio de execução dos testes foi de 0,32s; (máx: 0,39s)
* O uso de memória e handles se manteve estável não havendo indicação de vazamento desses recursos.

Quantidade de testes/s

Tempo médio da execução dos testes



Uso de memória (private bytes) nos servidores Web

Quantidade de usuários

Quantidade de handles em uso nos servidores web

1. Governança

Uma vez que o time de desenvolvimento tenha um framework à disposição, sugerimos que a **SEFAZ** estabeleça um grupo de pessoas que ficará responsável por mantê-lo.

O documento ***“Governança dos Guias e Templates de Desenvolvimento”*** sugere a criação de um *time de arquitetura de software*, acreditamos que esse seja o time ideal para manter o framework proposto, pelos seguintes motivos:

* Alta capacitação técnica dos envolvidos
* Estabelecimento de responsabilidade
* Controle de evolução e atualização
* Controle do código-fonte do framework
* Manutenção do ciclo de vida do framework
* Suporte ao time de desenvolvimento (e este time, suportado pelo Premier da Microsoft)
* Revisão dos sistemas a serem promovidos a fim de se certificar o correto e adequado uso do framework
* Deployment
* Treinamento
  1. Versionamento

Novas versões do framework serão versionadas e assinadas digitalmente a fim de que sejam unicamente idenfiticadas, possibilitando o uso simultâneo sem que os novos sistemas, que usam as novas versões do framework, interfiram nas aplicações legado.

1. Referências

|  |  |
| --- | --- |
| Assunto | Url |
| How to: Use Data Protection | <http://msdn.microsoft.com/en-us/library/ms229741.aspx/html> |
| Windows Communication Foundation Configuration Schema | <http://msdn.microsoft.com/en-us/library/ms731734.aspx> |
| Enterprise Library | <http://msdn.microsoft.com/en-us/library/dn169621.aspx> |
| Entity Framework | <http://msdn.microsoft.com/en-us/data/ef.aspx> |
| Designing Service Applications | <http://msdn.microsoft.com/en-us/library/ee658114.aspx> |
| Design Guidelines for Exceptions | <http://msdn.microsoft.com/en-us/library/ms229014(v=vs.100).aspx> |
| Hosting and Consuming WCF Services | <http://msdn.microsoft.com/en-us/library/bb332338.aspx> |