Imran Chowdhury

Imran.chowdhury@gdit.co.uk

Abstract

Dev documentation copied from one note for safe keeping. This deals with developer notes while creating the api, call structure naming standards etc

Dev Notes for crm portal api

This document holds the all the dev documentation generated for the project.

Table of Contents

[Introduction: 2](#_Toc461721647)

[Solution Structure 2](#_Toc461721648)

[Portal - Business Logic 2](#_Toc461721649)

[Interface 2](#_Toc461721650)

[Data Model 3](#_Toc461721651)

[Worker 3](#_Toc461721652)

[Explanation of Responses 3](#_Toc461721653)

[Request Object Definition 4](#_Toc461721654)

[Portal - CRM Service 4](#_Toc461721655)

[Service Tracing Class 4](#_Toc461721656)

[Web Config File 5](#_Toc461721657)

[Logging - Log DB 5](#_Toc461721658)

[Logging 6](#_Toc461721659)

[Possible Upgrades 6](#_Toc461721660)

[Glossary 7](#_Toc461721661)

[Appendix 1 – SQL statement for the table used in the log db 7](#_Toc461721662)

# Introduction:

The purpose of the project is to standardise GDIT’s API structure. Although a lot of the standards has not been applied but enough provision has made within the solution to implement good ALM practices. The source of the solution is located in under TFS.

$/GDIT/GDIT.Portal.CRMService

# Solution Structure

The solution has several projects. The namespace for the project defined as follows

* **GDIT.Portal.CRMService** : The main WCF project that contains all the operation contracts. This project contains no and/or no business logic
* **GDIT.Portal.BusinessLogic** : Contains business logic and data contracts for the WCF project. For the separation of dependency it is recommended that this project should be sub divided into two more project for business logic and data contract definition.
* **GDIT.Portal.Authenticator** : Contains place holder class structure for authentication of the calling user. There is no scope in the current project. Hence this is not implemented.
* **GDIT.Portal.Schema** : Contains the Xrm.Cs file for CRM. It is recommended that the developer should use organization proxy rather than strongly typed class structure. For this project this has not been altered.
* **GDIT.Logging** : Optional project that will manage the different layers of tracing. As the name space suggest, the project will be created with a view to be consumed by all/any project that needs logging. For this reason this project should sit in a different section in source tree in TFS. The project has connector for SQL and windows event log for now but it is created in a way that can incorporate other types of logging.
* **GDIT.Portal.Logging.LogDB** : A database project that contains schema definition for the sql logging.
* **GDIT.Logging.UnitTests** : Unit test project for logging.
* **GDIT.Portal.CRMService.UnitTests** : Unit test for CRM service

A console application named testwcf is created as test harness for the WCF. This is not recommended approach. No further work has done on the project.

# Portal - Business Logic

This project has three main folder

* Interface - Interfaces that are used by internal business logic only
* DataModel - Containing the data contracts for the WCF only
* Worker - Implementation of various interface and main business logic

## Interface

Interface has will hold IDisposable instances of BL classes. At the moment each .cs class contains business logic concerning one entity. This does not have to be a hard and fast rule. Developers should create interface based on the specific business and technical needs.

## Data Model

Apart from standard contracts, this folder has some classes that are crucial for portal SVC to work. They are,

* **ICRMEntity (Interface)**: Has common property for every entity namely Id, name and logical entity. Every data contract that represents a crm entity should implement this interface.
* **CRMDataDefinition (Class)**: Serializable class for complex crm data type. At the moment it holds the definition for CRMLookup, some other obvious choice of classes will be Money, option values etc.
* **Service Request (Implements IServiceRequest)** works as a super class for all the request. This class contains and object that holds properties to identify a caller. This object can be used to authenticate a caller.
* **Service Execution Response (implements IServiceExecutionResponse)**: this class works as a super class for the responses. This class holds properties like execution status, type of error etc. There can be three types of response
  + Scaler Response - That responds with a single (scaler object) i.e. one entity.
  + Collection Response - that responds with a collection of scaler objects. I.e. a collection of entity.
  + Void response - This response contains nothing up execution status.

This class also holds a property for the context id. Context id is the reference number for the log it creates. Logging mechanism is discussed in detail in later section.

* **Scaler Response (Extends Service Execution Response)** : Apart from the properties inherited from the super class, it has but one property - EntityResult. This is a generic object of ICRMEntity. Meaning entity result can only be an object that implements ICRMEntity interface.
* **Collection Response (Extends Service Execution Response)**: Similar to Scaler Response class. The only difference is instead of just one ICRMEntity generic object, it has a solitary property that represent a collection of ICRMEntity implementation.

## Worker

This folder holds the information of implementation of logical interfaces. For now it only holds Incidents and contact related logic.

## Explanation of Responses

Below is an example of a method that returns a scaler response object

public ScalerResponse<\_ICRMEntity> GetMeCrmEntity(Param){…}

It means the consumer expects a scaler response to be returned by the method. The scaler response object contains an object of type = \_crmEntity. An example of the result should look like below

|  |  |
| --- | --- |
| Success | Error |
| {  EntityResult = \_crmEntity {EntityName = "contact", EntityId = ABC1243 .. .. ..}  ExecutionStatus = Success  ContextId="6e9c8356-048e-4768-90ae-2a8844b0a294"  ErrorResult = <null>  } | {  EntityResult = <null>  ExecutionStatus = Error  ContextId="6e9c8356-048e-4768-90ae-2a8844b0a294"  ErrorResult = {  ErrorCode = 123546,  ClientMessage = "Please Check with Admin",  InterTrace = <stack trace>,  TypeOfError = Error  }  } |

## Request Object Definition

By default the WCF will except a solitary request object that implements IServiceRequest. As discussed above request object will hold caller information from the super class and number of other objects that defines the method query.

# Portal - CRM Service

This project has only the interfaces and classes required by window communication foundation. At the moment the following WCF methods are implemented

* ScalerResponse<UserProfile> Get User Profile - Get just one contact object for the portal.
* CollectionResponse<Case> Get Recent Activity - Get the customer's recent cases raised. By default it will return top 50 latest customer cases. If the consumer prefers more result it can pass in select top object (default is null) any integer value.
* ServiceExecutionResponse UpdateUserProfile - Updates contact information by crm guid.

## Service Tracing Class

Service tracing encapsulates lots of the log in, API authentication. Side by side, it also has some method that can access configuration elements like SQL connection level of logging etc. Apart from request validation all the methods in the class is vert self-explanatory. Request validation is called on initiation of every web API call. It validates the request object and then tries to authenticate the API user. When the authentication successfully passes, the result object is given back to the called where upon specific business logic is called.

An upgrade of the current code should include further encapsulation of the business logic class. It should have a delegate method to be called inside the business logic class. In this way we can get rid of all the try catch and exception handling per API call.

## Web Config File

Web config file is contains mostly standard settings. It contains additional settings to connect to crm organization service. These information is can be found on msdn [walkthrough for CRM web application](https://msdn.microsoft.com/en-us/library/gg695790(v=crm.7).aspx). MSDN also contain information for [Portal Creation](https://msdn.microsoft.com/en-us/library/gg695806(v=crm.7).aspx) in general. Apart from CRM setting only one additional app setting is used.

<AppSettings>

<!--expected params are - OFF, FATAL, ERROR, WARN, INFO, DEBUG, ALL-->

<add key="LogLevel" value="ALL"/>

</appSettings>

The setting name is 'Log Level'. Expected values are given below.

OFF = 1,

FATAL = 2,

ERROR = 3,

WARN = 4,

INFO = 5,

DEBUG = 6,

ALL = 7

The expected value is literal with fixed casing. The system the converts the literal into enum internally. Note that one log level will log all of the log levels before it. Meaning. If the log level is set to 'WARN' it means logs in the form of FATAL, ERROR and WARN will be recorded but anything listed as INFO and DEBUG will not be recorded. Also note that at the moment if the system finds the log level is anything but 'OFF' it will expect a valid sql connection to be present. If the SQL connection is not present the application would break. This is a major drawback of the current logging mechanism.

## Logging - Log DB

This project contains deployable code for the database used for logging. Below is the ERD for the database.

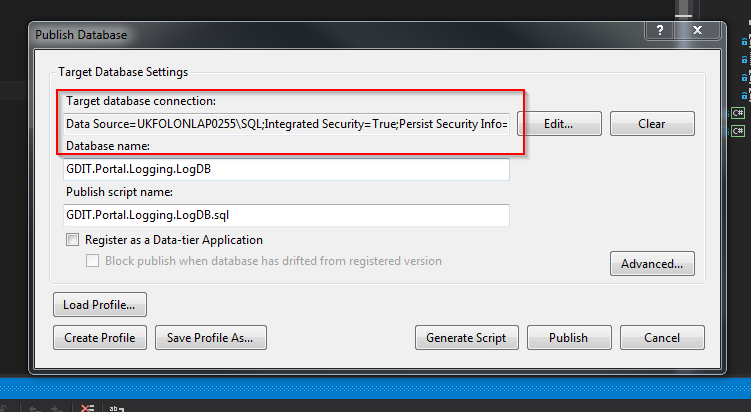


The data definition can be found in the glossary.

The ERD is very much self-explanatory. The API should have an authenticated user. This user has to be securely will only be allowed to make calls. At the moment the functionality has not been implemented.

As soon as a call is made an entry is created in the API context table. Cases where the user is not authenticated API context table will also log failure attempts. Once the context is created its unique reference is given back and stored in the result object. Subsequent logs (i.e. debug log, warning, and error) are stored in the trace log with the context id. This will help the developer to pin point exact logs per call.

Once the database is published the connection string needs to be copy pasted to the web config file of the API. Easiest way to find out the connection string is to go back to the published menu and selecting the correct connection.



# Logging

Simple library project to connect to log database and storing context and trace. The mechanism for logging is discussed in the previous section. The only additional class is DIagTool.cs. This class contains functionality to produce logs in the windows event. Windows event is very limited to the way is structures to event. Every single event is a log entry. Therefore as a default only context errors and log error, warning, fatal error is recorded in the windows log. All the other types of log are not included to save server disk space.

# Possible Upgrades

* The project is a WCF project. GDIT should consider moving away from WCF and start introducing web API.
* Naming convention is not maintained while creating the project. One example of the project name would be GDIT.Portal.CRMService where is should only have been CRMService while keeping the name space as GDIT.Portal.CRMService.
* GDIT.Portal.Authencator is not implemented.
* Logging is very much closed coupled with the API. We should consider coming up with more loosely coupled logging mechanism so that same code base can be used for logging third party application like plugin, workflow, and web application.
* The project depends on XRM.cs as ORM model. This dependency should be removed and organization service proxy needs to be used.

# Glossary

## Appendix 1 – SQL statement for the table used in the log db

USE [master]

GO

/\*\*\*\*\*\* Object: Database [GDIT.Portal.Logging.LogDB] Script Date: 15/09/2016 16:50:32 \*\*\*\*\*\*/

CREATE DATABASE [GDIT.Portal.Logging.LogDB]

CONTAINMENT = NONE

ON PRIMARY

( NAME = N'GDIT.Portal.Logging.LogDB', FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL12.SQL\MSSQL\DATA\GDIT.Portal.Logging.LogDB\_Primary.mdf' , SIZE = 4288KB , MAXSIZE = UNLIMITED, FILEGROWTH = 1024KB )

LOG ON

( NAME = N'GDIT.Portal.Logging.LogDB\_log', FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL12.SQL\MSSQL\DATA\GDIT.Portal.Logging.LogDB\_Primary.ldf' , SIZE = 1024KB , MAXSIZE = 2048GB , FILEGROWTH = 10%)

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET COMPATIBILITY\_LEVEL = 120

GO

IF (1 = FULLTEXTSERVICEPROPERTY('IsFullTextInstalled'))

begin

EXEC [GDIT.Portal.Logging.LogDB].[dbo].[sp\_fulltext\_database] @action = 'enable'

end

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET ANSI\_NULL\_DEFAULT ON

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET ANSI\_NULLS ON

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET ANSI\_PADDING ON

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET ANSI\_WARNINGS ON

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET ARITHABORT ON

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET AUTO\_CLOSE OFF

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET AUTO\_SHRINK OFF

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET AUTO\_UPDATE\_STATISTICS ON

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET CURSOR\_CLOSE\_ON\_COMMIT OFF

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET CURSOR\_DEFAULT LOCAL

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET CONCAT\_NULL\_YIELDS\_NULL ON

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET NUMERIC\_ROUNDABORT OFF

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET QUOTED\_IDENTIFIER ON

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET RECURSIVE\_TRIGGERS OFF

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET DISABLE\_BROKER

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET AUTO\_UPDATE\_STATISTICS\_ASYNC OFF

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET DATE\_CORRELATION\_OPTIMIZATION OFF

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET TRUSTWORTHY OFF

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET ALLOW\_SNAPSHOT\_ISOLATION OFF

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET PARAMETERIZATION SIMPLE

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET READ\_COMMITTED\_SNAPSHOT OFF

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET HONOR\_BROKER\_PRIORITY OFF

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET RECOVERY FULL

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET MULTI\_USER

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET PAGE\_VERIFY NONE

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET DB\_CHAINING OFF

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET FILESTREAM( NON\_TRANSACTED\_ACCESS = OFF )

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET TARGET\_RECOVERY\_TIME = 0 SECONDS

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET DELAYED\_DURABILITY = DISABLED

GO

EXEC sys.sp\_db\_vardecimal\_storage\_format N'GDIT.Portal.Logging.LogDB', N'ON'

GO

USE [GDIT.Portal.Logging.LogDB]

GO

/\*\*\*\*\*\* Object: Table [dbo].[ApiContext] Script Date: 15/09/2016 16:50:33 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[ApiContext](

[Id] [uniqueidentifier] NOT NULL DEFAULT (newsequentialid()),

[UserId] [uniqueidentifier] NULL,

[CallerName] [nvarchar](200) NULL,

[CreatedOn] [datetime] NULL DEFAULT (getdate()),

[MachineName] [nvarchar](100) NULL,

[MethodType] [nvarchar](50) NULL,

[RequestXML] [nvarchar](max) NULL,

[FatalErrorText] [nvarchar](500) NULL,

PRIMARY KEY CLUSTERED

(

[Id] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

GO

/\*\*\*\*\*\* Object: Table [dbo].[TraceLog] Script Date: 15/09/2016 16:50:33 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[TraceLog](

[Id] [uniqueidentifier] NOT NULL DEFAULT (newsequentialid()),

[CreatedOn] [datetime] NOT NULL DEFAULT (getdate()),

[Thread] [nvarchar](300) NULL,

[Level] [nvarchar](50) NULL,

[PortalMessage] [nvarchar](max) NULL,

[TraceExceptionMessage] [nvarchar](max) NULL,

[InnerException] [nvarchar](max) NULL,

[StackTrace] [nvarchar](max) NULL,

[ApiContextId] [uniqueidentifier] NOT NULL,

PRIMARY KEY CLUSTERED

(

[Id] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

GO

/\*\*\*\*\*\* Object: Table [dbo].[User] Script Date: 15/09/2016 16:50:33 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[User](

[Id] [uniqueidentifier] NOT NULL,

[UserName] [nvarchar](200) NOT NULL,

[Password] [nvarchar](50) NOT NULL,

PRIMARY KEY CLUSTERED

(

[Id] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

GO

ALTER TABLE [dbo].[User] ADD DEFAULT (newsequentialid()) FOR [Id]

GO

ALTER TABLE [dbo].[ApiContext] WITH CHECK ADD CONSTRAINT [FK\_ApiContext\_ToTable] FOREIGN KEY([UserId])

REFERENCES [dbo].[User] ([Id])

GO

ALTER TABLE [dbo].[ApiContext] CHECK CONSTRAINT [FK\_ApiContext\_ToTable]

GO

ALTER TABLE [dbo].[TraceLog] WITH CHECK ADD CONSTRAINT [FK\_TraceLog\_ToTable] FOREIGN KEY([ApiContextId])

REFERENCES [dbo].[ApiContext] ([Id])

GO

ALTER TABLE [dbo].[TraceLog] CHECK CONSTRAINT [FK\_TraceLog\_ToTable]

GO

USE [master]

GO

ALTER DATABASE [GDIT.Portal.Logging.LogDB] SET READ\_WRITE

GO